Ph.D. Dissertation in Engineering

Decision Support Tool for IoT Service Providers

Cost-Performance Optimization for IoT-based Sensor-Actor Systems

February 2017

Mohammad Mahdi Kashef

Technology Management, Economics, and Policy Program

College of Engineering

Seoul National University

Decision Support Tool for IoT Service Providers

Cost-Performance Optimization for IoT-based Sensor-Actor Systems

지도교수 황준석

이 논문을 공학박사 학위논문으로 제출함

2017년 2월

서울대학교 대학원
협동과정 기술경영경제정책전공
Mohammad Mahdi Kashef

모함마드 마흐디 카쉐프의 공학박사학위논문을 인준함

2017년 2월

위 원 장 이 종 수 (인)

부위원장 황 준 석 (인)

위 원 김 의 직 (인)

위 원 윤 현 영 (인)

위 원 이 대 호 (인)

Decision Support Tool for IoT Service Providers

Cost-Performance Optimization for IoT-based Sensor-Actor Systems

Mohammad Mahdi Kashef

Abstract

The Internet of Things (IoT) refers to the uniquely identifiable objects (things) and their virtual representations in an Internet-like structure. IoT has appeared on the stage, interconnecting a variety of physical objects over the Internet, enabling the objects to communicate and cooperate with each other to achieve predefined goals. It is predicted to be an integral component of the Future Internet (FI); therefore, IoT should be seamlessly and smoothly integrated with other FI integrated services. Nevertheless, IoT devices are location dependent, and expensive to develop and deploy, because the IoT supporting infrastructure such as computing power, storage and networks are resource constrained.

To fulfill these shortcomings, another recent phenomenon named Cloud computing can be the most promising and cost-effective solution. Indeed, Cloud offers relatively cheap, ubiquitous, unlimited and elastic solution for the supporting infrastructure. Therefore, to connect, manage and track the IoT-based devices and provide feasible access to a set of multitude computing resources, many IoT Service Providers (IoTSP) may utilize Clouds to offer their users certain services. Aiming to offer services to globally distributed users, an IoTSP will deploy its Virtual Machines (VMs) on multi Clouds that is consisting of various Cloud Service Providers (CSPs) to have satisfactory coverage and performance for the users.

Integration of IoT with the multi Clouds raises new challenges among which the economic challenges are from the most critical factors for success of the IoTSP business. One of the major problems in this context is to minimize the overall cost of the IoT system while keeping satisfactory level of performance in order for the business to be profitable. To this end, the IoTSP has to maintain its IoT-devices cost-optimally and to place its VMs on the available CSPs cost-optimally. In other words, the problem for IoTSPs is finding the cost optimum placement for their IoT devices and VMs.

This dissertation addresses the problem by proposing a decision support tool for

IoTSPs to find their cost-optimum devices and VM placement on multi Clouds. Sound system architecture is designed for the tool to carry out the tasks. The tool comprises a heuristic algorithm for IoT device placement, a cost estimation module as well as VM placement optimization algorithm. The cost estimation module estimates the total infrastructure costs for any given VM placements considering multi Cloud environment. Applying the proposed optimization algorithms on the estimated cost and the expected performance returns the cost-optimum IoT device and VM placement for the IoTSP. The proposed decision support tool is examined by several simulation scenarios and the results demonstrate the working of the tool.

Keywords: Internet of Things, IoT Service Providers, Cloud computing, multi Clouds, Decision Support Tool, cost estimation, cost optimization, VM placement

Student Number: 2010-30812

Technology Management, Economics, and Policy Program

College of Engineering

Seoul National University

Table of Contents

List of tables

List of Figures

Chapter 1. Introduction

1.1 Introduction and Background

The Internet-of-Things (IoT) has burst onto the stage, interconnecting every type of objects over the Internet acting as endless sources of data and information. The phenomenon has required a combination of three developments. The first is miniaturization with technology being available in hands of users through smart devices any-where and any time; second, is the overcoming of limitations of the mobile telephony infrastructure that brings ubiquity. And the third is the intelligence embedded in the applications and services that makes use of the vast amount of data created via the IoT Sensor networks (Alcaraz, Najera et al. 2010).

IoT is an integral component of the future Internet (FI); hence, it should be seamlessly and smoothly integrated within the other FI integrated services in which the emerging delivery models, such as the Internet-of-Services (IoS) and their associated utility computing paradigms offer the computing resources as a metered service (Armbrust, Fox et al. 2009, Soldatos, Serrano et al. 2012). Despite the IoT characteristics of high heterogeneity of devices, technologies, and protocols, it lacks different important properties such as scalability, interoperability, flexibility, reliability, efficiency. Since Cloud has

proved to provide the above features (Distefano, Merlino et al. 2012, Fox, Kamburugamuve et al. 2012, Suciu, Vulpe et al. 2013), those are drivers for IoTSP to utilize Clouds.

More specifically, IoT devices are location dependent, and the required supporting infrastructure i.e. computing power, storage and networks are resource constrained and expensive to develop and deploy. On the other hand, Cloud computing supports virtually unlimited storage capacity and processing power as well as pervasive infrastructure (A Vouk 2008, Armbrust, Fox et al. 2009, Hilley 2009). Moreover, Cloud computing infrastructures are location independent and elastic and provide access to a multitude of computing resources (Soldatos, Serrano et al. 2012). These features and characteristics of Clouds imply that most of the IoT issues can be solved (at least to some extent) getting benefitted from Cloud computing (Rao, Saluia et al. 2012, Lumpkins 2013, Botta, de Donato et al. 2015). As a result, it is believed that Cloud computing can be one of the best fitting choices to connect the *things* around us so that we can access to them in a ubiquitous environment; i.e. anything at any time and any place (Botta, de Donato et al. 2015). In other words, Cloud can be the most promising and cost-effective solution to connect, manage and track IoT. That is why many works such as (Chang, Chen et al. 2011, Parwekar 2011, Rao, Saluia et al. 2012, Zhou, Leppanen et al. 2013) proposed the Cloud Computing model

as the best fit for IoT applications with the dynamic computational requirements.

It is predicted that various service providers will utilize IoT to offer novel services to their customers. It might include the start-up companies as well as the innovative ones that are trying to generate revenue out of the novelty services offered on IoT. In this research, the Internet-of-Things Service Providers (IoTSP) refer to the companies who intend to offer particular services based on IoT. Figure 1-1 depicts the general concept of the IoTSP that offers some IoT-based services, and how they are benefited from integration between IoT and Clouds.

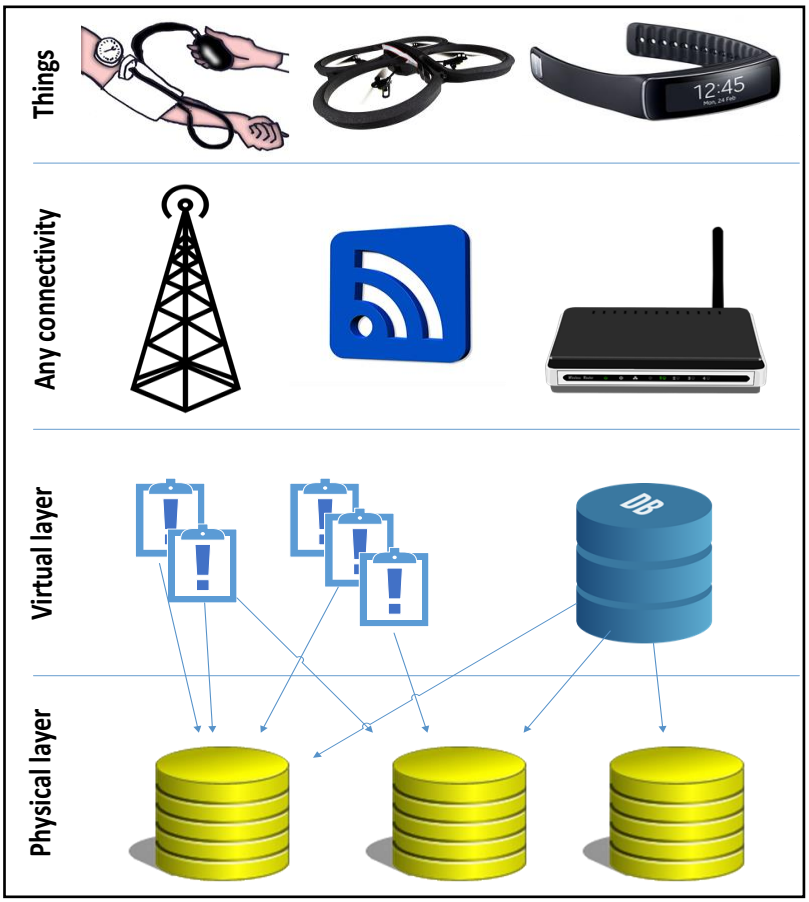


Figure 1-1 A diagram which shows the model for IoTSP utilizing Clouds

The IoTSP will require maintaining its services on minimum-possible level of costs to establish a reasonable business. On the other hand, it needs to offer satisfactory level of service to its customers so that they get more motivation to use the services and be loyal to the IoTSP. This makes a trade-off situation in which the IoTSP has to decide an optimum status in which it minimizes the overall costs while maintaining the satisfactory level of

service.

1.2 Problem Description

The Internet-of-Things Service Providers (IoTSP) intend to offer particular services based on IoT attributes. Many of these services are combined from IoT-sensors and IoT-actors, that is to say, there are physical sensors and physical actors that are smart and connected to a ubiquitous network.

Some examples of this can be a variety of IoT applications from “smart fire extinguisher” to “cardiac arrest/heart attack care system”. The common features of these cases include: 1) the *smart sensor* supposed to detect specific threat, danger or disaster 2) the *smart actor* which can be triggered by IoT system to tackle the danger without human intervention, and 3) a *ubiquitous network* which lets the IoT devices to be connected, the required calculations to be processed and work as infrastructure of the IoT system.

The system should work autonomously, i.e. the danger should be discovered, the processes should be executed and the proper reaction should be decided and commanded by system itself. Besides, there should be *agents* to receive the command and perform it to address the danger with no need of any human interference, management or decision-makings.

To achieve the goals, the IoTSP needs to manage and maintain the *sensors* and the *actors* as well as some *computer servers* to support the required computing and storage power for them. In addition, the IoTSP should keep them connected through suitable infrastructure.

Moreover, the success of the IoTSP relies on maintaining its system cost optimally, i.e. to achieve high service level and user satisfaction while spending minimum costs. Obviously, the IoTSP business can be profitable only if the IoTSP makes more revenue than what it costs. To this end, the major costs of the IoTSP should be studied well and then IoTSP needs to optimize them. Here, the cost of sensors can be overlooked because it is relatively cheap and in many scenarios, the sensors are paid by the users (not by the service providers). Nevertheless, the actors and the supporting servers are costly modules and the IoTSP needs to consider them well to avoid extra costs.

The actors might have different prices due to their features and qualities. The more actor types and classes are, the more complicated decision-making will be. The reason is the possible situations in the combination of the decision factors that become abundant. Choosing the best of them requires so many calculations so that it ensures that the IoTSP has picked the optimum case.

Besides, the IoTSP has to consider all of the costs incurred by utilizing Cloud for its infrastructure, as this is another major part of its costs. Therefore, one of the critical challenges for IoTSP is to minimize the overall cost of utilizing Cloud infrastructure because the less IoTSP costs, the more the profit increases. Finding the cheapest Cloud service provider (CSP) cannot be a proper solution to the problem. There are multiple reasons, e.g. geographically distributed demand causes the IoTSP to distribute its services over various CSPs, which is called *multi Clouds* in this paper. Considering the multi Clouds to be the infrastructure for IoTSP, one new problem is the cost optimization of using the Cloud resources. Hence, IoTSP cost optimization on the multi Cloud environment is of high importance.

In other words, to maximize the profit, the IoTSPs need to maintain two major costs at their optimum level: 1) the cost of the actors, and 2) the cost of the infrastructure. Alternatively, from other perspective, the IoTSP needs to utilize the costs used for the actors and for the infrastructure in an optimum way.

The cost can be optimality defined as a trade-off between cost and another objective; like maximum performance, user satisfaction level, and maximum coverage on the globe. Hence, the problem of this research is considered to be multi-objective optimization.

As mentioned in the above, there are two optimization problems to be addressed by this research. The first is cost optimality of the actors and the second is the cost optimality of the infrastructure (utilized multi Clouds). The optimality is defined to have the maximum performance while having certain level of cost (budget). For the former, the performance is measured by the covered regions and populations living in them by the service. For the latter, it is estimated by the average responding time, i.e. the average time VMs respond to the system requests.

In any danger-related scenario, *the reaction time factor* is critical. To be able to react in time and tackle the danger, there should be physically close enough actors to the danger spots. This is the critical condition for the IoTSP, i.e. only if this condition is satisfied, the costs for IoTSP and efforts to establish the IoT system will be worthy.

Therefore, the chosen performance factors for the actors optimization are “the covered population by the service” (CP) and “Time-to-Reaction” (TtR). The first one indicates the total population covered by the deployed actors. The second one includes the time of discovery of the danger, time to reach the danger spot and time to tackle the problem. Obviously, in many cases, the major part is time to reach because it involves the physical movement from the base to the danger spot. However, to keep the problem general, we

consider all of the three parts.

Here the IoTSP decision-making problem of this research is divided into two optimization problems: 1) to maximize CP (the covered population) and minimize TtR (Time-to-Reaction) while cost of the actors and their bases is maintained under the given budget. 2) To minimize the cost of the VMs while the overall delay of the system is at an acceptable level.

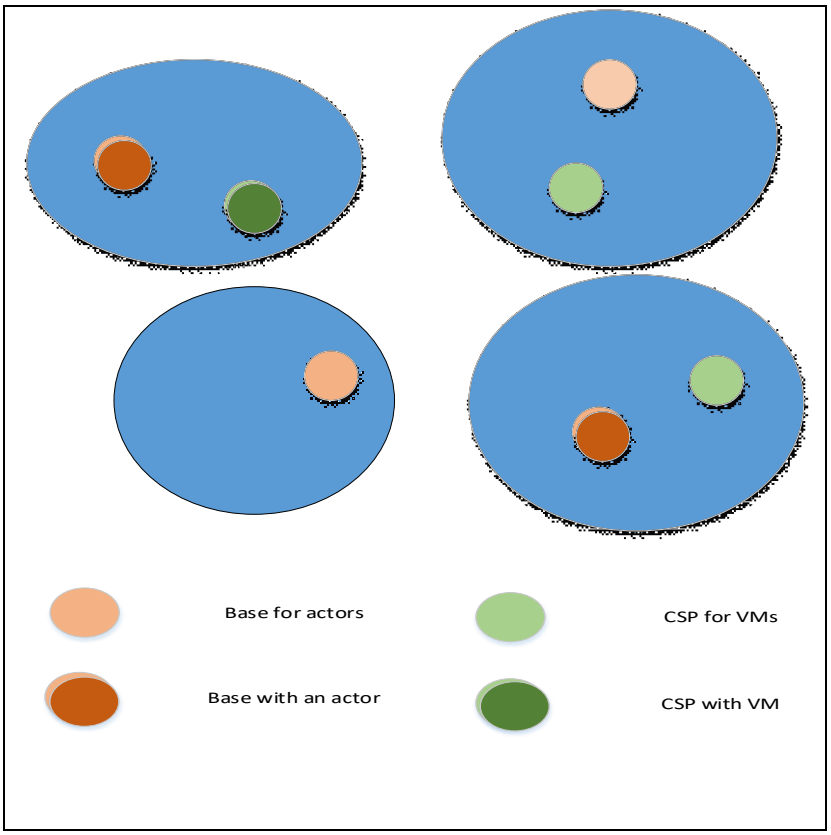


Figure 1-2 Schematic view of regions, bases and actors

For the first, the IoTSPs require deciding how many actors are affordable due to the assumed budget and then to determine where are the optimum bases to be placed for the actors so that IoTSP can achieve the highest covered areas and more population under its service as well as gaining the lowest TtR (Time-to-Reaction) for the potential danger. A schematic actor placement on the actor bases and the regions is depicted in Figure 1-2.

This might be considered a simple decision to be made. Nevertheless, understanding the nature of the problem and the decision to be taken assures that the situation might be too complicated to decide simply. One fact is that the larger the number of IoT actors is, the more the cost of establishment and maintenance of the system are. This implies that the cost is not a linear function of the number of actors. Hence, the IoTSP really needs sophisticated calculations to find the optimum places. Deployment of the actors on the optimum placement ascertains the highest performance while incurring the minimum possible cost.

For the latter, the most fitting infrastructure is considered as multi Cloud, where the IoTSP can enjoy the flexibility of the demand and pay-as-you-go feature as well as global coverage and widely distributed servers wherever needed.

Therefore, the IoTSPs have to place their servers in form of Virtual Machines[1] (VMs) on different Cloud Service Providers (CSPs) to support and process the data sent by the sensors. One may think that finding optimum placement for the actors and VMs of the IoTSPs is a simple job, but it is not the case.

Considering the global demand (or even a local demand for the IoTSPs which intend to serve locally) various options of actor placement could be imagined. The combination of the actors on the available bases will make a huge set of options among which the IoTSPs should choose the best.

On the other hand, similar situation of the actor placement is applied for the VM placement decision; i.e. the larger the number of VMs and their features, available Cloud Service Providers, their available types of VMs and price differences, the more calculation complexity to find the optimum placement (Altmann and Kashef 2014).

Considering the complexity of the situation and variety of the decision factors, it is clear that in real cases, the IoTSPs require proper decision-making tool to find out the optimum placement among the various available options.

---

[1] The definition of VM in this research is explained in 2.2.

1.3 Research Objectives

The aim of this paper is to support Internet-of-Things Service Providers (IoTSP) in both of their placement decisions: 1) to decide the number of affordable actors and the optimum places for them; and 2) to find out optimum VM placement on different CSPs (Multi Clouds). These two objectives help the IoTSP to establish and maintain its system, devices and infrastructure optimally so that it may enjoy high efficiency and performance.

For the first objective, the IoTSP should find the affordable number of the actors based on the given budget, actors' price, actors' maintenance cost as well as the bases' maintenance cost.

Then based on the number of the actors, the total number of possible placements can be known. Among them, the IoTSP requires to find the optimum placement that leads to the highest performance, i.e. lowest Time-to-Reaction with its limited resources.

The second objective deals with the placement of the Virtual Machines (VMs) supporting the IoT devices and the offered services as well as computing their required power and processes.

Utilizing different Cloud service providers, there are bunches of possible VM-placements. Amongst the various possible placements, the IoTSP needs to find the optimum one. Hence, there is need for a tool to facilitate the VM-placement optimization

task.

In particular, we intend to construct a *decision support tool* that can be used by IoTSPs to find out the optimum placement for their *actors* and their *VMs*. The tool is supposed to address the two research problems, i.e. the actors' placement and the VMs' placement. Hence, the decision support tool is constructed from two major units:

The first unit helps the IoTSP to find out how many actors should be placed and where to place them in such a placement that maximizes the covered population (CP) and minimizes its Time-to-Reaction (TtR).

The second unit considers the *total cost* incurred by the VMs. Total cost includes cost of running VMs on different Clouds, the traffic cost from users to the VMs as well as the traffic cost among VMs of different Clouds. Then, an *optimization algorithm* enables the second unit to help the IoTSP in their *VM placement decision-makings*.

1.4 Research Questions

Seeking the mentioned objectives, this work is intended to answer three questions: 1) How to find out the cost-optimum actors' placement on the regions? 2) How to estimate VMs' cost for an IoTSP on multi Clouds? 3) How the IoTSP can find out its cost-optimum VM placement on different Cloud service providers?

1.5 Methodology

To answer the research questions, the following steps are conducted: first, a systematic literature review is performed on cost estimation and cost optimization of IoT and multi Clouds. Second, all types of costs incurred by IoTSP are studied and collected. Based on the results of the second step, a model for cost estimation is proposed as long with a novel optimization algorithm for IoTSP actors' placement and a nice VM placement on multi Clouds. Based on these results, a decision support tool is made to help the IoTSP find its cost-optimum placement. A sound system architecture is designed to operate the proposed algorithms. Finally, multiple simulation scenarios are suggested and the simulation is conducted to demonstrate working of the proposed decision support tool for IoTSPs.

1.6 Contribution of the Research

This work proposes a *decision support tool* to be globally used by the IoTSPs to determine their optimum situation for utilizing the resources and devices. In particular, the IoTSP may find out its optimum actors' placement and the optimum VM placement on the multi Clouds. This tool comprises estimation of the total infrastructure cost, a heuristic algorithm called iCELF for optimization of actors' placement as well as an optimization algorithm for VM placement of IoTSP. The cost estimator considers the total costs incurred

by the IoTSP. It should address the total infrastructure cost that includes cost of running VMs and the traffic cost among the VMs. The optimizer examines the total cost of every possible placement, and suggests the most cost-effective one for the VM placement. A novel architecture is designed to contain the proposed modules as a working system.

1.7 Research Outline

The structure of this dissertation research is illustrated in Figure 1-3 and explained as follows: the second chapter gives an overview on the state-of-the-art and related work concerning Internet-of-things, Cloud computing, integration of them and resource allocation. In the third chapter, we explain our methodology in details. First, we detailed the systematic literature review process. Then the novel decision support system is explained including the problem scenario, the proposed architecture, cost estimation model and cost optimization algorithm. The last part of this chapter presents simulation method used to evaluate the suggested tool. Chapter 4 details the simulation environment, the assumptions and input factors as well as the simulation scenario for a typical problem. Then it divides the problem into two folds namely: actor placement problem and VM placement problem. The fifth chapter examines the workings of the proposed decision support tool with simulation experiments through explaining the simulation experiments and results. Chapter 6 reviews the results of this research and concludes the dissertation with a brief discussion.

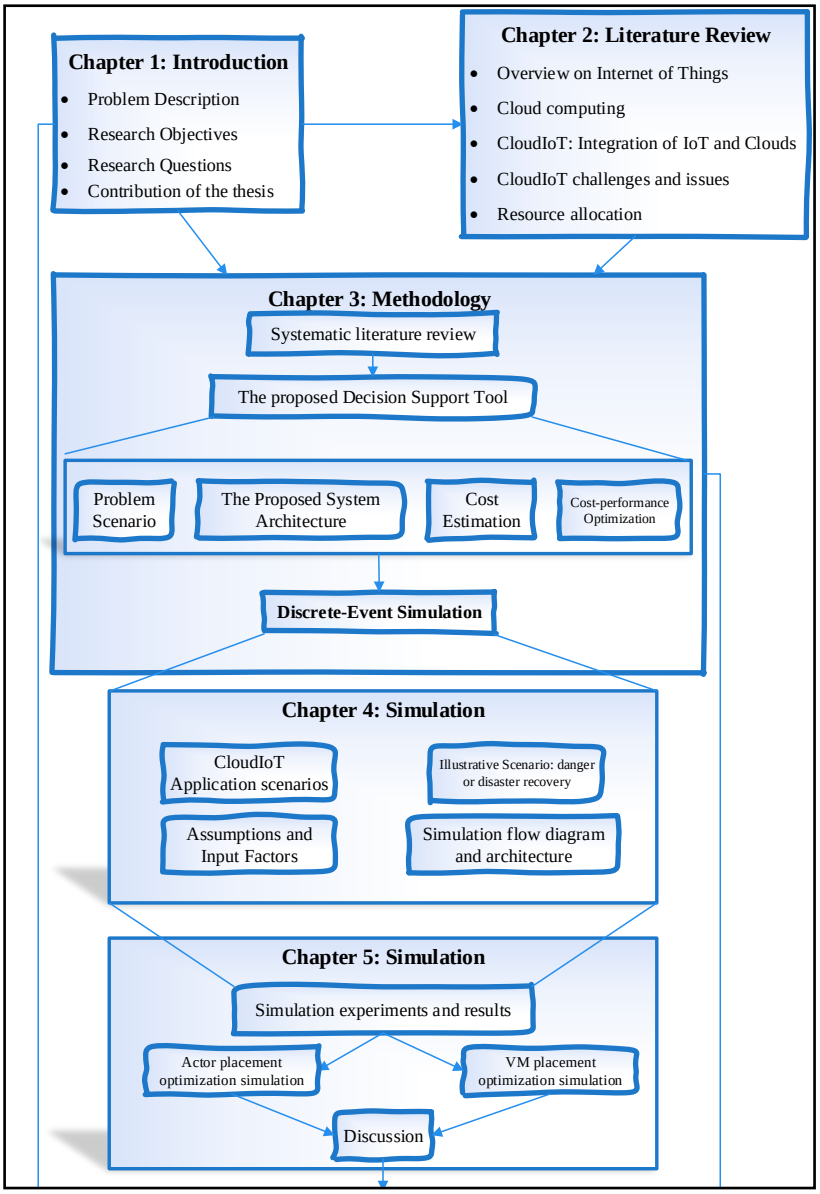


Figure 1-3: The structure of this dissertation

Chapter 2. Literature Review

To proceed this thesis, we have conducted a systematic literature review to cover all related works. In this chapter, we illustrate the state-of-the-art under categorized sections: Firstly, we review Internet-of-Things. Secondly, we look into the wide subject of Cloud computing briefly. Thirdly, we study integration of Internet-of-Things with Cloud computing. Then the challenges and issues of the integration will be discussed. Finally, the related literature about resource allocation on Clouds is presented.

2.1 Internet of Things: IoT

The Internet of Things is an integrated part of Future Internet and could be defined as a dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual “things” have identities, physical attributes, virtual personalities and use intelligent interfaces as well as being seamlessly integrated into the information network (Vermesan, Friess et al. 2011).

In this phenomenon, things should be understood with new definition: *In the context of “Internet of Things” a “thing” could be defined as a real/physical or digital/virtual entity that exists and moves in space and time and is capable to be*

identified. Things are commonly identified either by the assigned identification numbers, names and/or location addresses (Vermesan, Friess et al. 2011).

In the IoT, "things" are expected to become active participants in business, and social processes where they are enabled to interact and communicate with each other and with the environment by exchanging data and information "sensed" about the environment, while autonomously reacting to the "real/physical world" events and influencing it by running processes that trigger actions and create services not based on direct human intervention.

In recent years, the Internet-of-Things (IoT) has received a significant research attention. IoT is based on intelligent objects (things) (Miorandi, Sicari et al. 2012) which are dynamically interconnected (Li, Da Xu et al. 2014) empowering ubiquitous scenario with the real objects of the world (Tan and Wang 2010, Miorandi, Sicari et al. 2012). From the business perspective, there is a huge amount of investment in this field showing great interest of industry in IoT. Hence, major research projects and studies are expected to focus on it and elaborate on its applications, and address the research gaps.

One of the limitations in IoT is that the *things* have limited storage and processing capacity and they need certain infrastructure for their connectivity (Miorandi, Sicari et al.

2012). Therefor reliability, performance and connectivity are among the concerns about the utilizing IoT mentioned by the researchers (Botta, de Donato et al. 2015). There are some solutions suggested by academia as well as industry to address the mentioned problem discussed in section 2.3.

2.2 Cloud Computing

Although there are still many Internet forums and blog discussions on what Cloud computing is and is not, the NIST definition seems to have captured the commonly agreed Cloud computing aspects that are mentioned in most of the academic papers published in this area (Mell and Grance 2009). The NIST definition states that Cloud computing is A model for on-demand network access to a shared pool of resources that can be rapidly provisioned and released with minimal management effort (Mell and Grance 2009). As Cloud computing is still in its infancy, the definition of Cloud computing is likely to be improved later when new ideas, services, and developments in Cloud computing will be explored. Nonetheless, for the purpose of our paper, the definition given by NIST is sufficient.

From all of different *things* to be offered as a service, this work is limited to the IaaS, i.e. Infrastructure-as-a-Service. If the consumer does not manage the underlying

Cloud infrastructure in spite of having control over operating systems, storage, and deployed applications, we refer to it as IaaS (Mell and Grance 2009).

Thanks to the well-developed concept of virtualization, the service providers offer a different type of infrastructure on a single physical machine (server). For that the virtualization engine assigns certain amount of processing power (CPU), memory (RAM) and storage (Hard Disk) to which (Smith and Ravi 2005). This is called Virtual Machine, which is contrary to the concept of Physical Machine (PM). Therefore, the scope of this research is limited to the IaaS; in other words, it is about optimization of VM placement.

From other perspective, the research boundary of this thesis is given with respect to *the federation of Clouds;* i.e. when there are some competing Cloud providers (Mell and Grance 2009) to offer the infrastructure.

Through the creation of the hybrid Clouds, the company may enjoy more reliability and QoS by employing the public Cloud to cope with the peaks while capitalizing on the investment made for the data center (Van den Bossche, Vanmechelen et al. 2013). Besides, to truly fulfill the promise of Cloud computing, there should be technological capabilities to federate disparate data centers, including those owned by separate organizations. Only through federation and interoperability can infrastructure providers take the advantage of

their aggregated capabilities to provide a seemingly infinite service computing utility. Specifically, we refer to the infrastructure that supports this paradigm as a federated Cloud (Rochwerger, Breitgand et al. 2009).

This also eases the adoption of companies to Cloud, as one of the most important obstacles to Cloud adoption is the well-known vendor lock-in (Hilley 2009). Figure 2-1 illustrates above definitions showing case of an assumptive company, which runs its private Cloud (i.e., it owns data center) to host its corresponding VMs, and because some of its services are using two different Cloud providers (i.e., Provider 1 and Provider 2) as infrastructure of remaining VMs.

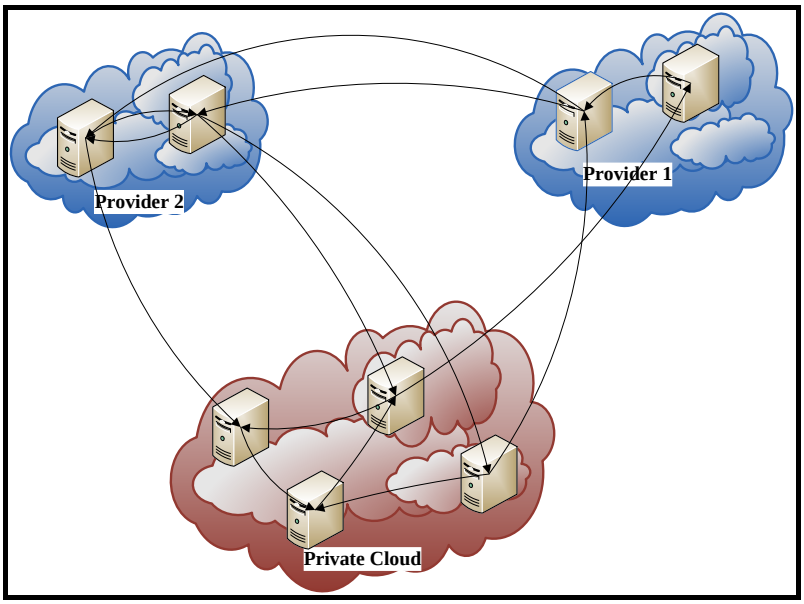


Figure 2-1: Schematic view of a federated hybrid Cloud

Based on these definitions, as shown in Table 2-1, we can also distinguish the four categories of Clouds:

Table 2-1. Cloud categories

		Cloud Interconnection	
		Pure Cloud	Hybrid
Number of Cloud Providers	One Cloud Provider	Category 1: Public Cloud	Category 3: Hybrid Cloud
	Two or more Cloud providers	Category 2: Federated Clouds	Category 4: Federated Hybrid Clouds

While all four categories fit the definition of Clouds, the first and second categories

describe Clouds, in which users do not own a private Cloud (i.e., data center) and their services are all on public Clouds. Users of the first category use one public Cloud while users of the second category use a group of interconnected Cloud providers (federated Clouds). The third and fourth categories describe interconnected Clouds, in which one of the Clouds is owned by the user (private Cloud). In particular, the third category defines a combination of a private Cloud and a public Cloud. The fourth category represents the situation, in which a user runs some of its VMs on its private Cloud and some others on a federation of public Clouds.

2.3 CloudIoT: Integration of IoT and Clouds

Although these two phenomena are independently developed, but due to certain reasons, they seem to complete each other (Lumpkins 2013, Botta, de Donato et al. 2015). From a very simple and untightened integrated model between IoT and Clouds proposed by (Hassanein 2011) until fully related integration level. The minimum level of integration might be the case of using Clouds only for analyzing the huge data generated by IoT. Many other researches have noted about this fact, and among which (Mehdipour, Noori et al.), they have emphasized that the Cloud-based datacenters are being used to carry out the heavy weight of big data generated by IoT.

Among the drivers for integration of IoT and Clouds, we may highlight communication (Data and application sharing) (Soldatos, Serrano et al. 2012), storage and need for big data (Jiang, Da Xu et al. 2014), and computation (Botta, de Donato et al. 2015). The result of merging IoT and Cloud is said to be a very powerful disruptive technology which is going to shape the major part of future Internet (Tan and Wang 2010). The term CloudIoT is suggested for the paradigm by (Botta, de Donato et al. 2015) emphasizing on the novel application of this convergence.

Moreover, there are various applications of IoT and Cloud integration. For instance (Li, Zhong et al. 2013) mentioned *Cloud logistics* being promoted by Cloud computing and Internet-of-Things. The second example is (Chen and Hu 2013) in which the Internet of *intelligent* things is studied in connection to Clouds. The newly emerged idea of Robot as a Service which can be categorized as a subset of IoT is being studied. They also tried to extend the idea of centralized Clouds to be decentralized. We referred to this as federated Clouds in 2.2. A more popular application of the well-studied IoT is smart home. (Kirkham, Armstrong et al. 2014) has proposed an integrating architecture to Cloud computing. The proposed architecture facilitates sending data and taking management from remote Cloud-based networks of services.

In this work we don't consider other approaches which can be seen as integration of IoT and Clouds such as (Asimakopoulou, Sotiriadis et al. 2013) in which the authors proposed utilization of Micro-clouds getting the inspiration of IoT and share the computing power of the small devices i.e. the *things.*

2.4 CloudIoT Challenges and Issues

Nevertheless, on the path of this integration there are still problems and issues to be studied and addressed. Many researchers have put their efforts to tackle them in various methods. We categorize them into three general types: 1) technical approaches, 2) security and privacy-driven approaches, and 3) economic and business approaches.

The major focus of the literature is on the technical issue. As the objective of this work is non-technical issues, we only mention few instance of the technical approach. The authors of (Soldatos, Serrano et al. 2012) proposes the building blocks of a middleware framework for the convergence of IoT and Clouds. Their framework utilizes Linked Sensor Data and W3C Semantic Sensor Networks standard to the proposed framework that enables the IoT service providers to deploy their IoT services onto Cloud infrastructure. Another research which proposes middleware for IoT and mobile Cloud computing integration is (Le Vinh, Bouzefrane et al. 2015). Albeit the authors have mainly focused on the smart

home rather than considering other types of IoT applications. The novel work of (Persson and Angelsmark 2015) addresses the fragmentation of the IoT and Cloud area by proposing a framework to unify programming in the integrated environment.

About the security-driven approaches, we may mention (Henze, Hermerschmidt et al. 2015) which focuses on the privacy concerns of individual users and data in the integrated system. They present UPECSI, which is their proposed solution for User-driven Privacy Enforcement for Cloud-based Services in the IoT. UPECSI takes a comprehensive approach to privacy for the Cloud-based IoT by providing an integrated solution for privacy enforcements that focuses on individual end-users and developers of Cloud services at the same time. Another example is the works on SensorCloud project (Hummen, Henze et al. 2012, Catrein and QSC AG 2013, Henze, Hummen et al. 2014) aimed at protecting the sensor data inside the sensor network, and before being uploaded to the Cloud. Similarly, (Henze, Bereda et al. 2014, Pooja, Pai et al. 2014) also realize the protection of sensor data already within the sensor network. On the other hand, some works propose securing data when it is outsourced to the Clouds. We may mention (Lounis, Hadjidj et al. 2012, Li, Yu et al. 2013, Liu, Huang et al. 2014) as a few examples. A Software-Defined-Networking (SDN)-based architecture is proposed by (Olivier, Carlos et al. 2015) to achieve security in

IoT network. Another instance is (Kalra and Sood 2015) which has studied the authentication schemes for this integration and proposed a mutual authentication protocol for secure communication between embedded devices and Cloud servers.

Unfortunately, the research community has not paid significant attention to the last category; i.e. economic and business issues of IoT and Cloud integration. We could mention that (Li and Xu 2013) has put the first steps in clarification of IoT business models. The research community is seemed to be in its infancy about economic issues of the integration and much progress is expected to happen in near future. Nevertheless, when it comes to the cost-effectiveness and other economic aspects, the academia has not contributed in anything. Hence, proper research should study and address the economic challenges of IoT and Cloud integration. Our published work is also one of the first researches covering this concern (Mohammad Mahdi Kashef 2016).

It is noteworthy that both proprietary and open source platforms have stepped forward and implemented the CloudIoT applications. Just to mention an example of each, SAP HANA is a novel system based on Clouds which offers IoT services (Mathew 2015), while OpenIoT from GitHub is an open source project developing a middleware for integrating IoT solutions.

2.5 Resource Allocation

2.5.1 General Resource Allocation Strategies

Since the purpose of this work is to address some issues of the integration of IoT and Clouds, which means using Clouds for IoT Service Provider as its infrastructure, in this section we review some related works about resource allocation on Cloud computing. In addition, as the Grid computing has emerged earlier than Clouds, some inspirations from Grid might be learned for resource allocation. Although the Grid and Cloud are similar in many aspects, the task scheduling has basic differences. The Grid was designed to handle large sets of limited duration jobs. In the Grid, mainly the tasks (jobs) are defined and are supposed to be calculated and finished, i.e. there is a total processing time for them (Dong 2007). This is because in many cases there is some experiments which should be assigned to computational resources to run it completely and give the results (Buyya, Abramson et al. 2000, Buyya, Abramson et al. 2005). Once the calculation is done, the need for computational power is over. There are many researches considering this aspect of Grid for the resource allocation problem. For e.g. (Ai, Tang et al. 2011) considers deadline-constrained resource allocation by which they try to solve the problem of composite web-services onto Grid.

However, for the case of Clouds, some services are considered to be deployed and run on different physical machines. Probably, the physical machines might be from different Cloud providers. Nevertheless, the same demand of Grid could be done by Clouds. There are many works dealing with scheduling or resource allocation considering the workloads (Wang, Von Laszewski et al. 2009, Van den Bossche, Vanmechelen et al. 2010, Van den Bossche, Vanmechelen et al. 2011, Van den Bossche, Vanmechelen et al. 2013). In fact, there are couples of disparate resource allocation methods that are well studied. For example, we may mention: First-Come-First-Served (FCFS) (Sotomayor, Montero et al. 2009), Last-In-First-Out (LIFO) (Wenhong 2008, Chang, Chang et al. 2012), round robin (Sotomayor, Montero et al. 2009, Mahajan, Makroo et al. 2013), priority based scheduling (Malawski, Juve et al. 2012), etc.

Nonetheless, in the scope of this research, we only focus on deployment of IoT services (in the form of VMs) on Clouds. This means that there is no concept of deadline for the services (VMs) in our work. More specifically, when an IoTSP intends to deploy its services on Cloud it means the IoTSP needs infrastructure to launch them; and there are some different users who needs to connect to certain services to fulfill their demands. The other point about Cloud is that it is widely assumed that Cloud resources are unlimited.

This is also highlighted for integration of IoT with Clouds. So the scheduling algorithms which are considering the limited number of resources (e.g. (Buyya, Yeo et al. 2008)) doesn't fit to the Cloud users neither (Dong 2007).

Among the available scheduling algorithms, this research has selected the optimization algorithms. Hence, we only concentrate on the algorithms trying to optimize certain criterion (criteria). Moreover, we focus on the placement problems on the Cloud. Therefore, we do not study other available optimization algorithms as they are beyond the scope of this work. Different suggested criteria for optimizing the resource allocation is studied in 2.5.3 to 2.5.6.

2.5.2 Economy-based Placement Optimizations

In this field, placement optimization is defined as "finding the most suitable service placement, which maximizes or minimizes one or several criteria and adheres to the constraints" (Grozev and Buyya 2014). Apparently, there are different optimization models using various criteria for optimizing the VMs. Among them, in an economy-based approach, placement decisions are made dynamically at runtime and are driven and directed by the end-users requirements. This means the resource management systems need to provide mechanisms that allow Cloud consumers to define their requirements and facilitate

the realization of their goals (Buyya, Abramson et al. 2005). These goals are called optimization criteria by which we categorize optimization algorithms in 2.5.

There are several types of economic models introduced for resource allocation, as per each categorization of (Buyya 2002), several economic models are used in computational environment. They covered: 1) commodity market models, 2) posted price models, 3) bargaining models, 4) tendering, or contract- net models, 5) auction models, 6) bid-based proportional resource sharing models, 7) cooperative bartering models, and 8) monopoly and oligopoly.

The above-mentioned models are studied in the context of Cloud computing as well as IoT more or less. For example, in the auction model, each provider and consumer acts independently and privately agrees on the selling price. There are different works considering auction models to address resource allocation problem like (Lazar and Semret 1997, Buyya, Abramson et al. 2005, Izakian, Abraham et al. 2010, Wei-Yu, Guan-Yu et al. 2010, Toosi, Van Mechelen et al. 2014, Goyal 2015, Moon, Kim et al. 2015). Despite those works considering auction strategies into the Clouds, some other works suggests that the commodity markets are better choice for resource allocation for computational economies (Wolski, Plank et al. 2001, Chawla and Chana 2015, Chawla and Chana 2015).

Hence, for the scope of this research, we concentrate on the *commodity market models*. In the commodity market model, providers specify their resource price and charge users according to the amount of resource they consume. It can be flat or variable price depending on the resource supply and demand (Buyya 2002). Of course, the flat rating is more used in the Clouds, and this is the reason that in our modeling, we assume the prices would be flat over a certain time. Once the provider decides to change any of the prices, it will be posted so that all Cloud users may know the differences.

In academia, different aspects for the optimization of the resource allocation have been proposed. Before going into the details, it is important to be noted that optimization unit, i.e. the unit or criterion by which the optimization is being conducted, may vary among different optimization algorithms. Basically the optimization unit can be VM as a certain amount of computational power, memory and storage (Singer, Livenson et al. 2010), or can be the amount of VM properties (Deelman, Singh et al. 2008). More specifically, one may optimize VM placement on different CSPs, considering each VM as a fixed package. On the other hand, one may consider elasticity for VMs. This urges to try to optimize the computing power, memory or storage for different given services.

In this work, we only deal with the first concept and consider certain given VM

specifications. We classify them into four groups according to their optimization criteria. More details are explained as follows in 2.5.3 to 2.5.6:

2.5.3 Performance-related Optimization

Several factors may represent performance. Also In the field of Cloud placements, different factors are considered by research community to evaluate the performance of the system. (Nan, Yifeng et al. 2011) consider service response time as criterion of performance, and they try to minimize this in their study. (Wang and Lu 2008) proposed an optimization model to keep the *average response time* close to desirable amount; while minimizing the energy consumption. To be brief, we just mention some of the other suggested criteria to optimize the placement like: delay (Zamanifar, Nasri et al. 2012), data transfer time consumption (which can be considered as special type of delay) (Jing Tai and Jun 2010), Quality of Service (Li, Zhuo et al. 2013), reliability (Zhao and Sakurai 2013, Tian and Zhao 2014), and load balancing (Mahajan, Makroo et al. 2013).

Another work which is special in some sense is (Lee, Jeong et al. 2014) which has defined a detailed way of measuring the low-level performance of the system. They have designed a performance analysis scheme of each host node in Clouds, considering the number of cores and specification of CPU and memory size. Then, the resource allocation

system provides to allocate virtual machine on the optimal node to supply the service considering user's needs and to effectively use the high-performance and low-performance of node considering each performance. The work maximizes the performance based on the resource allocation scheme.

Performance of the system fluctuates over time and per different network conditions incur the risk and make the system unreliable for Cloud users. But the fluctuation has been considered only in some of the works, while others only consider the system static (Nan, Yifeng et al. 2011). On the other side, there are many works noting the dynamicity of the performance. Basically, these researchers tackled this problem by two strategies: 1) Performance prediction-based strategies. 2) Rescheduling-based strategies. In the former, the known prediction methods are used to enable system to maintain the desired level of performance on peaks. There are also some proposed simulation-based methods. Some examples of this type are (Buyya 2002, Chen, Soundararajan et al. 2006, Verboven, Hellinckx et al. 2008, Li, Zong et al. 2011, Lucas Simarro, Moreno-Vozmediano et al. 2011). On the other hand, the latter suggests to consider the new status as input factor that recalculates and gives the new optimum placement according to the new situation. A few examples are (Wang and Lu 2008, Hao, Yen et al. 2009, Jing Tai and Jun 2010, Zamanifar,

Nasri et al. 2012, Lee, Jeong et al. 2014).

2.5.4 Cost-related Optimization

There are many cost related algorithms suggested for resource allocation like the performance measures. This subject also is a well-studied problem in computer science. Meantime there are disparate factors introduced as measures for "economy related" or "cost related" Optimization. The payment system used for Clouds is either pay-per-use or dynamic (variable) pricing (Hilley 2009). Obviously, the free services (like Google docs[2]) are out of this kind of optimizations. The dynamic pricing vendors are also considered beyond the scope of this research.

Some works have suggested optimization algorithms to minimize the operational costs. Different methods are proposed to capture these criteria. For example, (Nan, Yifeng et al. 2011) addressed the problem in two steps. First, they propose a cost model to capture the costs incurred by the service placement. Then they run a minimization algorithm by optimizing the capacities of the used resources. They solve the latter problem via a Lagrange multiplier method.

---

[2] https://docs.google.com/

To decrease the penalty of server failure, (Zhao and Sakurai 2013) proposed a scheduling model which is claimed to improve the cost-efficiency. It has also proved that the model increases the revenue of the Cloud provider. However, there are other costs not related to the penalty of server failure. Hence, this work cannot be considered as overall cost optimizer.

Minimizing the overall operational costs is the goal of (Woo and Mirkovic 2014). This is accomplished by evaluation and comparison of some scenarios on multi providers. They also provided a low-level comparison on services provided by CSPs considering cost and performance. Nevertheless, as their major focus is comparing single provider with multi providers, this is not enough to evaluate two placements on multi Clouds. Moreover, there is no explicit algorithm that can be used for general cases.

For optimizing virtual machine placements across different Clouds, (Lucas Simarro, Moreno-Vozmediano et al. 2011) proposed a scheduling model. The scheduler counts variables such as average prices or Cloud prices for suggesting an optimal deployment. They use integer programming to minimize total cost infrastructure. Assuming each hour as scheduling period, the algorithm gives the best possible price for each period. This means that the algorithm needs estimation for the next hour for the best calculations. The

accuracy of the results relies on the estimation algorithm. The other problem of this work is that the overhead of re-deployment is ignored which has a significant impact on the overall cost.

The replication is used for many purposes in network industry. Despite the technical issues of replicatino, (Mansouri, Toosi et al. 2013) tried to minimize the replication cost of VMs placed on the public Clouds while maintaining expected availability. They categorised the VMs into two groups: popular objects and non-popular ones. They suggested to place the popular ones on more available data centers. And then the rest to be deployed on lower QoS supporting CSPs. It is claimed that this heuristic approach keeps the cost lower. Nevertheless, in many cases one cannot differentiate the reliability and availability of CSPs as they are really competitive.

Considering the deadline-constrained tasks, there are some works addressing the optimizatino problems like (Van den Bossche, Vanmechelen et al. 2010, Van den Bossche, Vanmechelen et al. 2011, Malawski, Juve et al. 2012, Van den Bossche, Vanmechelen et al. 2013). As an example, (Van den Bossche, Vanmechelen et al. 2013) have proposed a set of cost-efficient scheduling algorithm. The domain is public and private Clouds. They have designed a public Cloud scheduler as well as a hybrid Cloud scheduler.

Maximization of the utilization of the data center and minimization of the cost of VMs running on the public Clouds are the goals of (Van den Bossche, Vanmechelen et al. 2010). They are trying to achieve the aforementioned goals while observing QoS constraints. To address the problem, they proposed a binary integer program formulation of the scheduling problem.

As two last examples which are close to the problem of this research, (AO, XU et al. 2013, Altmann and Kashef 2014) would be explained as follows: Both of them tried to find out the cost structure. But (AO, XU et al. 2013) considers it on the distributed Clouds; while (Altmann and Kashef 2014) considers it for hybrid federated Clouds. Then, based on the proposed cost models, (AO, XU et al. 2013) conducts a cost-minimization algorithm. As the problem domain is the distributed Clouds, the work tries to optimize the number of edge nodes that minimize the total cost. They performed the suggested model for three different cases: Services like Web caching, CDN-like applications and Applications like Cloud storage. As the cases are generic, they might be used for several purposes. Nevertheless, (Altmann and Kashef 2014) have developed a comprehensive cost model which considers all of the cost factors for hybrid federated Clouds. Besides, a brute-force optimization is applied for placing the VMs on disparate private or public Clouds.

2.5.5 Energy Consumption Optimization

The other perspective absorbed much attention is energy consumption. As the environmental concerns are getting hot and serious, all of the data centers and Cloud providers have to care this as well as academia. According to many reports, the other important point is energy consumption that is one of the major sources of data center costs. (Poess and Nambiar 2008, Li, Qian et al. 2012, Horri, Mozafari et al. 2014) reported that about 30% of the monthly expenditure of data centers is caused by electricity charges. As a few examples, (Bianchini and Rajamony 2004, Patel and Shah 2005, Tang, Gupta et al. 2008, Vouk, Averitt et al. 2008, Wang and Lu 2008, Diaz, Castro et al. 2013, Horri, Mozafari et al. 2014, Kessaci, Melab et al. 2014, Wang, Wang et al. 2014) tried to address it by optimizing the energy consumption. The common point among them is that they try to minimize the energy consumption of a Cloud data center considering maintenance of some other factors at the desirable situation.

As an example of some similar approached works, (Wang, Von Laszewski et al. 2009) has suggested a thermal aware optimization. Considering the fact that heat mainly comes from the processes, and by proper placement of services, the Cloud providers can reduce the incurred heat as well as the energy consumed.

An optimization-based algorithm is proposed in (Song, Huang et al. 2014) for virtual machine placements. Their algorithm considers servers and VM dependencies. They minimized the number of physical hosts to reduce the energy consumption.

2.5.6 Combined Scheduling (multi-objective optimization)

Majority of the above three sections reviewed the placements which only considers one criteria. This gives a simple objective function. One step beyond this is to combine some scheduling goals. Multi-criteria objective functions are the results of that combination. Here, we mention some of the works within which at least two criteria are being compromised together.

Budgeted server placement is the problem addressed by (Yang, Fang et al. 2011). The authors are trying to maximize the number of servable clients while the budget level seems to be fixed.

There are some researches combining performance or economic related criteria with energy consumption factors. For example (Zhao and Sakurai 2013) proposed scheduling model combining revenue maximization and energy consumption minimization. The reliability maximization is assumed the key to maximize the revenue. (Nan, Yifeng et al. 2011) formulated and solved the responding time minimization problem and resource

cost minimization problem, respectively. Maximization of the utilization of private Cloud while minimizing the cost of running VMs on public is the objective of (Van den Bossche, Vanmechelen et al. 2010). The work solves the problem observing the Quality of Service constraints.

Chapter 3. Methodology

In this chapter, the steps of this research are explained in details. First, the method of systematic literature review is explained. The second section describes the contribution of this dissertation. The components of the decision support tool are depicted: 1) the proposed system architecture, 2) the cost estimation and 3) the cost optimization method suggested by this work. The last section details the simulation method used to examine the performance of the proposed tool.

3.1 The proposed Decision Support Tool

In this section, the main contribution of this dissertation is explained which is the Decision Support Tool for IoT Service Providers. First, the problem scenario is discussed and the input factors of the problem are detailed. Then, the problem statement of this dissertation is described. Then the proposed architecture for the decision support tool depicts and explains its components. After that, our placement cost estimation and cost optimization algorithms are discussed in details.

3.1.1 Problem Scenario

As explained in the first chapter, the IoTSP intends to offer particular services to its customers. The services are based on some IoT devices that are supposed to be smart and

connected to the network. The services are assumed to be based on IoT-sensors and IoT-actors, in which the *smart sensor* is supposed to detect specific danger in time while the *smart actor* should urgently tackle the danger. The sensors and actors are assumed to utilize the multi Cloud environment, which is a proper infrastructure for this case. Moreover, the whole IoT services should be designed and executed as autonomous, i.e. without need to any human intervention. Hence, appropriate computing and storage powers should be involved in the system.

Based on the above, IoTSPs have to establish appropriate system architecture and proper infrastructure offering the services for their users. According to the distribution of the users, the service should cover different spots of the globe; hence, there should be close-enough actors to every aggregation of the users. Otherwise, those users will not be covered by this danger-recovery service. Nevertheless, the physical place of the actors can have various choices, which is called *actor placement.* This is very critical for the success of the IoTSP to have a proper and efficient actor placement. An illustrative of the actor placement is shown in Figure 3-1.

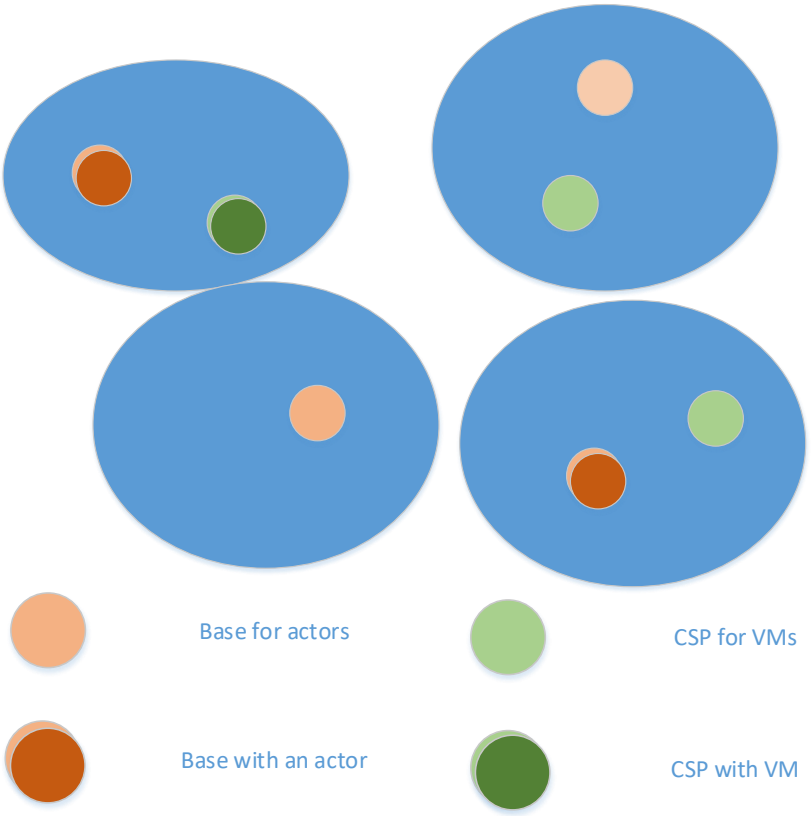


Figure 3-1 schematic view of actor and VM placement on the regions

From the other perspective, in the form of Virtual Machines, the computing servers are to be placed on proper physical spots. This implies several possible placements for the VMs, among which the IoTSP should decide and select one of them. A general snapshot of the VM placement is depicted in Figure 3-2.

Obviously, different actor placements will incur various overall *service time*, i.e. the time which is required for their reaction. In addition, the total covered population of

each placement varies and is important factor for the placement decision making. On the other hand, different VM placements incur various placement costs (due to price differences of different CSPs and class difference of the VMs). This also complicates the decision making for the IoTSP.

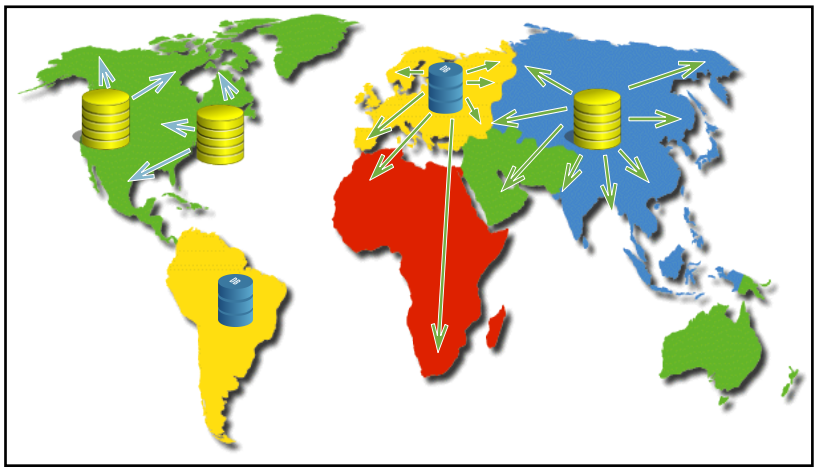


Figure 3-2 schematic view of a placement

The following assumptions are considered in this work to remove the unnecessary complications. It is notable that these assumptions will not cause this work lose its generality.

Firstly, the *budget* for *fixed cost* (*FC*) is given, that is to say that the amount considered by the IoTSP for establishment of the system (actors, VMs, infrastructure etc.). This excludes the *variable cost* (*VC*), e.g. monthly cost, and the budget can be extracted

from the ‘investment power’ of IoTSP or its ‘cash capacity’ in the real cases.

Each region is assumed to have one *actor base* on which the IoT actor can be deployed. Nevertheless, according to the distance between the actor’s base and the neighboring regions, each actor can cover some neighbor regions as well. In other words, each actor will be responsible for not only its region, but also for some surrounding regions as well. It is notable that as the speed of the actors may differ, the service coverage of actors may vary, too. This factor complicates the problem more.

Table 3-1: example of the VM tiers and their attributes as input factors

Provider	Tier	CPU	Memory	storage	price
A - USA	T1	16	14 GB	240 GB	4.99 G
A - EU	A1	8	14 GB	240 GB	2.99 G
B – USA	T2	8	7 GB	120 GB	3.99 G
C - USA	T3	2	3.5 GB	60 GB	1.99 G

The total number of VMs is assumed as *v*, and available types of VM spec that are supported by all of the CSPs are given. The spec identifies these features of the VM: processor, memory, storage size and bandwidth. Obviously, each spec might have its own

price. In addition, the price of the same tier might vary on different CSPs. An example of these data about the VM tiers is shown in Table 3-1. These all are supposed to be the input factors. Besides, it is assumed that the globe is divided into *n* parts (like continents, but not necessarily always) with their own given number of users, Internet connectivity rates and available CSPs. Each *part* is considered to have an actor base, and the average distance of each base to all of the spots of the *part* is given. Besides, the distance among center points of the *parts* is given. An example of the input factors of the parts, and the data provided about them is depicted in Table 3-2.

Table 3-2: example of the input factors of the parts and their data

	part I – North & South America	part II – Europe	part III – East Asia
Population	10 M	3.7 M	25 M
Available CSP(s)	4	3	3
Distance to other parts	Vector VI including all of the distance s of this part	Vector VII including all of the distance s of this part	Vector VIII including all of the distance s of this part
Traffic to other parts	Vector TI including all of the distance s of this part	Vector TII including all of the distance s of this part	Vector TII including all of the distance s of this part

It is assumed that the traffic generated by each VM is Θ portion of its bandwidth. The average required bandwidth for each user is assumed as β. In addition, the traffic cost from any *part* to all other *parts* is one of the input factors. If the IoTSP does not place any VM in a particular part, the users of that *part* will be connected to the closest CSP for them. Finally, the capacity of all of the CSPs is assumed unlimited.

3.1.2 The Problem Statement

The main problem this research intends to address is IoTSP optimization of actors and VMs problems. This can be modeled as G (N, A, V) in which N represents the nodes (regions), A represents the actors and V represents the VM (Virtual Machines).

As mentioned above, this dissertation focuses on two problems: 1) Actor placement optimization 2) VM placement optimization. Using graph theory notations, the first problem can be formulated as a given network $G'(V, E)$ representing the nodes V as the regions and the edges E as the connections between the regions. In other words, E represents the subsets of G that are the nodes; there is possible access from each node to some of the other nodes. Any actor placement on G may be presented by subset $A \subseteq V$; i.e. the nodes on which an actor is placed. Hence, the problem is to optimize the IoTSP cost by choosing the proper N, A, and V.

Utilizing the notation of reward and cost, the first problem can be formulated as:

$$\max_{A \subseteq \mathrm{N}} R(A) subject\ to\ C(A) \leq B$$

Where *B* is the budget the IoTSP can spend for its actors in total.

Then the second problem can be written as:

$$\max_{V \subseteq \mathrm{N}} R(V) subject\ to\ C(V) \leq B'$$

Where B' is the total budget to be spent by the IoTSP for the VMs.

3.1.3 The First Part: Actor Placement Algorithm

As explained in the previous section, making decision of how to place the actors is very critical for IoTSP success. Efficient placements will not simply result in "low performance" and "late reaction time" of tackling the dangers. This might lead to a situation that the whole IoT system becomes void and useless.

The easiest method to find the optimum placement is to generate all of the possible placement sets and then calculate the objective function for all of the possible placements. Finally, the placement that generates minimum/maximum of the decision factor can be found. Nevertheless, optimizing this algorithm is NP-hard.

To improve the scalability of the algorithm, the attributes of the objective function

are studied. One of the observations is that the objective function is submodular.

In 3.1.3.1 the Submodularity attribute is studied and the mathematical background is briefly explained. In addition, some of the simple methods to utilize the Submodularity are discussed. Then, in 3.1.3.3, the proposed algorithm of this dissertation named iCELF is discussed which is an improved version of CELF in details.

3.1.3.1 Submodularity and diminishing return

The observation of the objective function shows that it is submodular. Submodular functions exhibit diminishing returns: attribute which can be defined as: when we have only placed a few actors, placing another actor provides greater effect, i.e. covered area compared to when placing it after we have placed many actors (the covered area grows slower and slower with actor placement size).

The diminishing return can be formulated as follows (R is a set function and A, B and V are sets):

For all placements $A \subseteq B \subseteq V$ and actor $a \in V \backslash B$, it holds that $R(A \cup \{a\}) - R(A) \geq R(B \cup \{a\}) - R(B)$ (Leskovec, Krause et al. 2007). A set function R with this property is called submodular.

Therefore, the actor placement problems can be reduced to the issue of maximizing

a non-decreasing submodular function, subject to a constraint on the budget to be spent for the actors. More generally, any objective function that can be viewed as an expected penalty reduction is submodular. Submodularity of R is the key property exploited by the proposed algorithm.

But optimization of submodular functions is also NP-hard problem (Khuller, Moss et al. 1999). The reason is that it takes to find the optimum placement out of all combinations. The more the number of actors and bases are, the more complicated problem is. Hence, some improvements are required for the algorithm. Applying some heuristics may scale up the algorithm while the result will be near optimal (Leskovec, Krause et al. 2007).

Simpler case is one of the commonly used heuristic methods, where the *greedy algorithm* takes the most gaining node as next step. Basically, submodularity guarantees decreases in the marginal benefits as iterations passes. In other words, the later actors that are added by the greedy algorithm can increase the total gain of the function *less* than the gain from the sooner added actors.

3.1.3.2 Multi-criterion optimization

In many of the practical applications that require decision-making, simultaneous

optimization of multiple objectives is needed. Similarly in case of this problem, each decision option (i.e. each placement) will have a vector of scores like: R(A) = (R1(A), ..., Rm(A)). Here, various situations can arise that two placements A1 and A2 are incomparable, e.g., R1(A1)>R1(A2), but at the same time R2(A1)<R2(A2). Thus, all we can hope for the optimization are *Pareto-optimal solutions* (Boyd and Vandenberghe 2004). The placement A is called Pareto-optimal, if there does not exist another placement A' such that for all i: Ri(A')≥Ri(A), and for some j: Rj(A')≥Rj(A) (i.e., there is no placement A' which is at least as good as A in all objectives Ri, and strictly better in at least one objective Rj).

One common approach for finding such Pareto-optimal solutions is scalarization (Boyd and Vandenberghe 2004). Here, one may pick positive weights λ1 >0, ... λm>0, and optimize the objective R(A) = $\sum_i$ λiRi(A). Any solution which is maximizing R(A) is guaranteed to be Pareto-optimal (Boyd and Vandenberghe 2004), and when the weights λi vary, different Pareto-optimal solutions can be achieved. One might be concerned that, even if optimizing the individual objectives Ri is easy (i.e., can be approximated well), optimizing the sumR= $\sum_i$ λiRi might be hard. However, submodularity is closed under nonnegative linear combinations and thus, the new scalarized objective is submodular as

well. Therefore, the algorithms developed in the following section can be applicable.

3.1.3.3 The proposed algorithm: iCELF (improved Cost-Effective Lazy Forward selection) algorithm

To scale up the algorithm for solving the optimization problem that is NP-hard, some heuristic methods can be applied. As mentioned in 3.1.3.1 one simple heurist is the greedy algorithm, but there are few problems with greedy algorithm. Firstly, it is still very slow algorithm. Secondly, it only works for unit cost and this is an important limitation. In the real cases, the regions area may differ dramatically; hence, the actors to be used for large areas should be faster and stronger as well as being more expensive. Thirdly, as mentioned in Chapter 13.1.3.1, it may fall arbitrarily bad. In this section, the developed algorithm namely iCELF (improved Cost-Effective Lazy Forward) is presented which is a two-pass greedy algorithm to improve the shortcomings of the first greedy algorithm while the results are still near optimal.

First to address the non-constant prices for the actors, instead of comparing R(A), another parameter, i.e. *marginal gain* is suggested:

$$s_k = \underset{s \in V \backslash A_{k-1}}{argmax} \frac{R(A_{k-1} \cup \{s\}) - R(A_{k-1})}{c(s)}$$

This means the greedy algorithm should pick up the element that maximizes the

reward/cost ratio. This solves the problem for non-unit cost for the actors as well. The algorithm may finish once there is no more actor to be selected without exceeding the assumed budget.

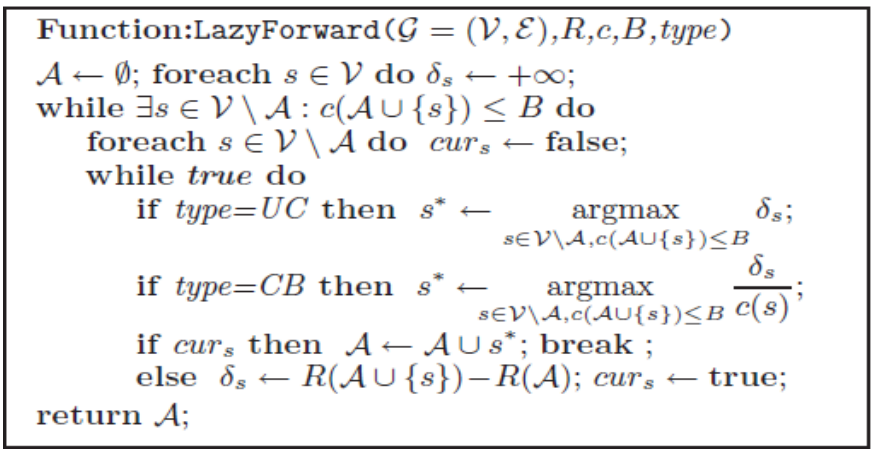
Therefor if the list of the actors is kept ordered based on the marginal benefit S_k for each iteration, the algorithm does not need to calculate the reward of all of the nodes, and then to decide which is the max (Leskovec, Krause et al. 2007).

Nevertheless, this may fall arbitrarily very bad. For example, if the a1 is next node to be picked up having very little reward but also little cost and a2 has a great cost as well as a great reward, the greedy algorithm will pick a1. In that case, the algorithm may not be able to afford any more actors due to the shortage of the budget. Hence, it will stop. However, the optimal solution was to pick a2 due to its high gain in reward.

As proved by (Khuller, 1999) if the two parameters are considered, i.e. unit cost and the marginal gain, then their maximum will guarantee to be always a near optima solution.

Now, considering the nature of the problem, it can be understood that normally adding an actor to the set of selected actors will add very little amount to the covered population (which maximizing it is the objective of this issue). This is more effective in the

cases with sparser regions.



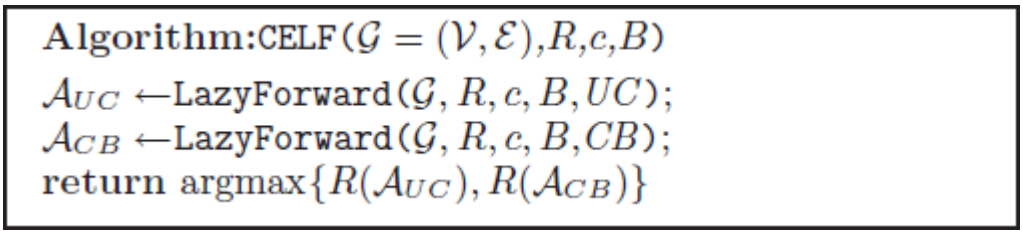


Figure 3-3 pseudo-code of iCELF algorithm

The last improvement on this algorithm is to decrease the consumed calculation time. It is called lazy algorithm due to the skipping of the recalculation of majority of nodes and picking the next actor without it. Figure 3-3 depicts pseudo-code for the iCELF algorithm.

Considering the marginal improvement or gain to be achieved by each actor (S_k),

the vector which holds all of them is ordered descending. In addition, all of the actors are initially marked invalid. In each step, if the top node is invalid, the algorithm recalculates S_k. In many cases after recalculation, the amount will only decrease a little; so that the top node will stay the same. In this case, there is no need to recalculate other nodes and the algorithm will select the top node for the next actor.

Utilizing the above-mentioned rule, an algorithm can be improved to avoid checking all of the unselected actors. It will be sufficient if the algorithm has a fast lookup for those whose marginal gain is higher than that of the others.

3.1.4 Second Part: VM Placement Algorithm

3.1.4.1 Cost Estimation

Selection of a placement amongst all possible ones can be done based on different criteria. In this paper, the decision criterion is the cost optimality of the placement. In other words, this research intends to help the IoTSP to find the cost optimum placement so that it maximizes the profit of its business.

The cost estimation of this dissertation is getting inspiration of (Altmann and Kashef 2014) in which the authors have collected all types of infrastructure costs on multi Clouds. The costs include cost of running VMs on multiple Clouds as well as the traffic

cost among VMs that are placed on different Clouds. For the case of this research, i.e. IoTSP, another cost factor is added called the traffic costs from users to the VMs. As the company has to provide Internet access for the devices given to the users, it is the responsibility of the IoTSP to let the devices to be connected. Hence, Internet costs should be considered in this study as well.

For further calculations, the following definitions are used for the mathematical modeling:

1. *Traffic Cost Rate Matrix (TCR_{P*P})*: this is proposed by (Altmann and Kashef 2014) which is a *P×P-dimensional* matrix in which the *(i, j)* element indicates the rate per unit traffic from Cloud provider *i* to Cloud provider *j*. That means it is the sum of the cost for one unit of outgoing traffic from provider *i* to provider *j* and the cost for one unit of incoming traffic to provider *j* from provider *i*. As Cloud providers usually charge only for in-bound and out-bound traffic of a service (i.e., for traffic that is related to outside their Cloud), all main diagonal elements of TCR are zero. However, in case that a provider charges users for their internal data traffic, the corresponding main diagonal elements will not be equal to zero.
2. *VM Spec List ($Spec_{4*S}$)*: it is a *4*S* matrix which keeps the details of possible specs.

The total number of possible specs is *S*. The matrix rows represent the 1) VM processor, 2) memory, 3) storage size and 4) bandwidth.

3. *VM Spec Vector* ($Spec_{1*V}$): it is a *1*V* vector which records the spec number (referring to *SL* matrix) of each VM. It is notable that some of the specs are common among the CSPs.
4. *Cost of Provider for SpecMatrix* ($CostPSpec_{S*P}$): it is an *S*P* matrix whose rows refer to *Spec* of a particular VM and the columns refer to the CSP on which the VM is placed. The matrix will return the cost of the given spec on the given CSP. This matrix would be extracted from CSPs' price list.
5. *VM Placement Matrix* (Plc_{D*V})*: it is a D*V* matrix in which D represents all of possible placements, and *V* is the total number of the VMs. The matrix holds details of all of placements giving each of them a code for further references. While using the matrix, *Plc* (m,*) gives a vector which represents placement # *m,* i.e. the element *(m, i)* returns the code of CSP on which the VM # *m* is placed.
6. *Internet Connectivity price* (IC_{1*n}): it is a 1*n vector holding the Internet cost per month for each of the *n* assumed regions.
7. *User* $_{1*}n$*:* it is a 1*n vector for the population (number of users) living in each of

the *n* regions.

Based on the proposed definitions and notations, the required calculations can be properly performed as follows:

To capture the total infrastructure cost, we should consider three cost factors: 1) cost of running VMs, 2) traffic cost among VMs (for synchronization and replication purposes), and 3) Internet connectivity costs for the users. Equation 1 formulates them:

$$TotalCostofPlacement: TCP_m(j) =$$

$$CVM_m + TrC_{VM_m} + CIC_m \qquad (1)$$

Which returns to the total cost of a given placement of IoTSP as a function of time (*m*). Normally (but not necessarily always) the time unit is considered to be month. The total cost of placement is sum of the three factors: cost of running VMs, traffic cost among VMs and cost of internet connectivity. The details of the equation are explained in equations 2, 3 and 4:

$$CostofrunningVMs: CVM_m(j) = m\sum_{i=1}^{V}\Big(CostPSpec\big(Spec(i), Plc(j,i)\big)\Big)$$

(2)

Which calculates the computing costs of IoTSP incurred by a given placement *j*. *Spec(i)* returns the specification of the i^{th} VM. *CostPSpec* gives the price of running VM #*i*

on its CSP. The corresponding CSP could be known from *Plc (j, i)* where *j* refers to the given placement. The sigma considers all of VMs of the placement. Hence, the parameter *i* refers to all of the VMs starting from 1 to *V* which is the number of VMs.

$$TrafficCostofVM: TrC_{VM_m}(j) = m \sum_{k=1}^{M} \sum_{i=1}^{M} \Big(TCR\big(Plc(j,i), Plc(j,k)\big) * \Big(\theta * SL\big(4, Spec(i)\big)\Big)\Big) \quad (3)$$

Which calculates traffic cost among VMs of the IoTSP in a given placement *j* over time. In the equation *Plc (j, i)* and *Plc (j, k)* return the CSP corresponding to i^{th} and k^{th} VM accordingly. Then, *TCR* gives the traffic cost rate between these two CSPs. Multiplying the cost rate to $\theta * SL(4, Spec(i))$, i.e. the amount of traffic generated by the VM # *i* calculates the traffic cost of the VM.

Note that it is assumed that the amount of bandwidth dedicated to any VM is related to its usage and defined in the spec selected for that particular VM. Besides θ is the ratio of the real usage of dedicated bandwidth to the total bandwidth. Hence, multiplication of θ to the bandwidth dedicated to the VM returns the real usage of the corresponding VM.

Obviously, the sigma accumulates the total traffic cost of all of the VMs in the given placement and returns the total traffic cost at the end.

$$Costof InternetConnectivity: CIC_m =$$

$$m * \sum_{i=1}^{n} IC(i) * User(i) \qquad (4)$$

Which calculates the accumulative Internet cost incurred by the usage of the *n* given regions. *IC (i)* gives the Internet price per each region per month while *User (i)* shows the assumed number of the users of the region.

3.1.4.2 Cost-Performance Optimization

In this step, another model is proposed for the second problem faced by IoTSP, i.e. the cost optimization of VM placement on Multi Cloud providers (federated Clouds).

Objective Function: The objective function of the optimization is to minimize the overall cost of VM placement while offering satisfactory network coverage and performance for the users of the IoTSP, which is defined as the Quality of experience (QoE). QoE is the service level targeted and guaranteed by the IoTSP to its customers. This can be measured by delay time from the node to servers (which are on the relevant VM).

Based on the above, the problem is modeled as:

Minimize

(overall cost)

Subject to:

(Network coverage for all of the users)

Each of the nodes has access to VM within less than "*a* seconds" delay.

Each of the user requests should be processed within less than "*b* seconds".

The first constraint stated that for each of the regions in which the IoTSP has customers, there should be close-enough CSP to the region that responds the requests. Obviously, here, closeness is defined by network measures, not by physical distance. The second constraint indicates the "Quality of Experience" (QoE) or the "service level" which the IoTSP intends to guarantee that level for its customers seeking their satisfaction.

To model the problem, two mediatory functions are defined as follow:

Delay: this indicates the delay between a given *node* or average and the closest VM. The function output is in *millisecond (ms)* unit.

$Process\_Time_{Request}$: this shows the required time to process a given *request* from

users.

Using these two definitions and the mathematical modeling proposed in 3.1.4.1, the problem is rewritten as:

Minimize

(TCP_m)

subject to

(Network coverage for all of the users)

$Delay_{node} \leq \alpha$ seconds

$Request\_Time_{Process} \leq \beta$ seconds

In which TCP is the abbreviation of Total Cost of Placement.

Optimization Algorithm: an optimization algorithm is proposed for the raised problem scenario. The algorithm returns the cost optimum VM placement among all of the possible VM placements. The steps of the algorithm are illustrated in Figure 3-4.

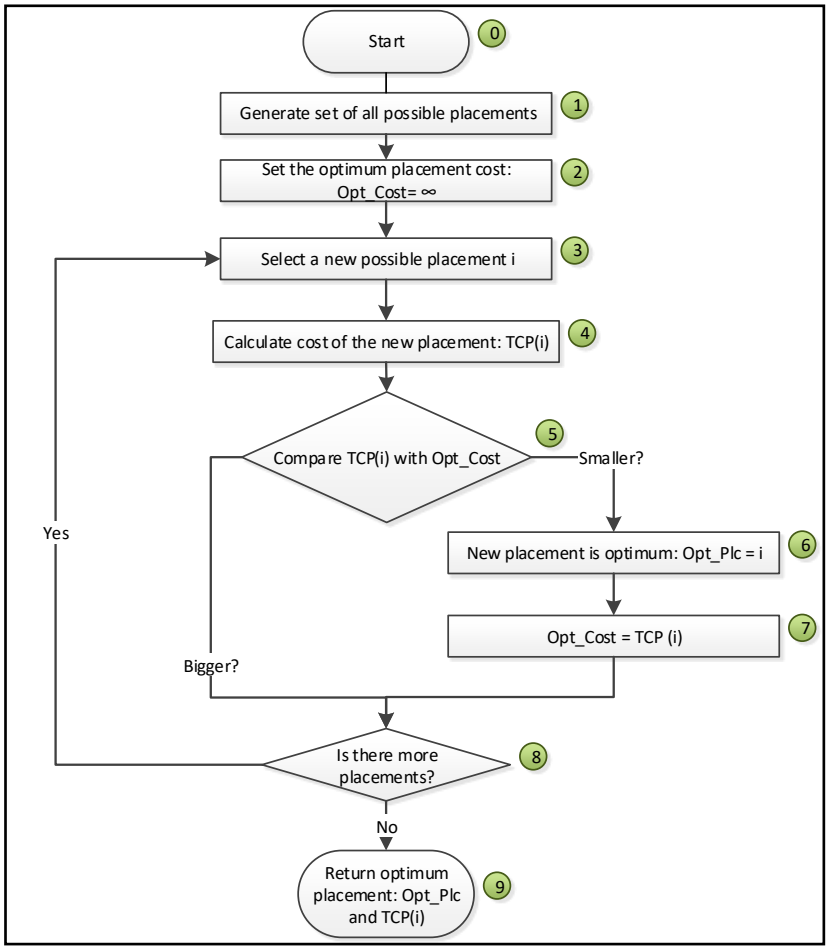


Figure 3-4: The proposed optimization algorithm

In details, the algorithm works as follows: First, it generates the list of all possible VM placements. Step three picks one of the placements and step 4 calculates the total cost of this placement (TCP) using equation 1. The next step compares the TCP with the latest optimum placement. If the current placement incurs less cost, the algorithm picks it (step 6) and substitute the TCP to be the optimum cost (step 7). Then the algorithm checks

whether there are more placements to be calculated and evaluated or not. If still there are remaining placements, it goes to step three to pick one of them and continue the loop until all of the possible placements are traced. Finally, the algorithm returns the optimum placement and its total cost.

3.1.5 The Proposed System Architecture

For the sake of cost optimizing the placement of one IoTSP, the system architecture is proposed as shown in Figure 3-5. This is suggested for a tool that is to be used for IoT service provider who intends to offer particular services dealing with danger/treat to its clients based on IoT and needs to decide about placement of some IoT actors as well as some VMs.

The proposed architecture for the decision support tool to be used by IoTSP includes a coordinator, Optimization Module (OM), Placement Cost Estimation Module (PCEM), monitoring module, and a Database. The user of the proposed decision support tool is IoTSP. The Database is being fed by the monitoring module that keeps capturing the situation of different regions and Cloud service providers, their offered VMs, and their prices. The sequence of utilizing the tool by IoTSP is as follows: The coordinator communicates with the IoTSP to receive the requests as well as the required data and to

send back the reports. OM is triggered by the request from the coordinator. Then it sets the given data from IoTSP as well as the Database. Based on the collected data, OM runs the optimization algorithm to find out the optimum placement for IoTSP. To achieve the optimum placement, OM triggers the PCEM that calculates three parts of the cost explained in 3.3. Therewith, PCEM returns the total cost of the given placement to OM to be used in the optimization algorithm. Finally, OM results the optimum placement for the IoTSP and sends it to the coordinator so that it is delivered to the user.

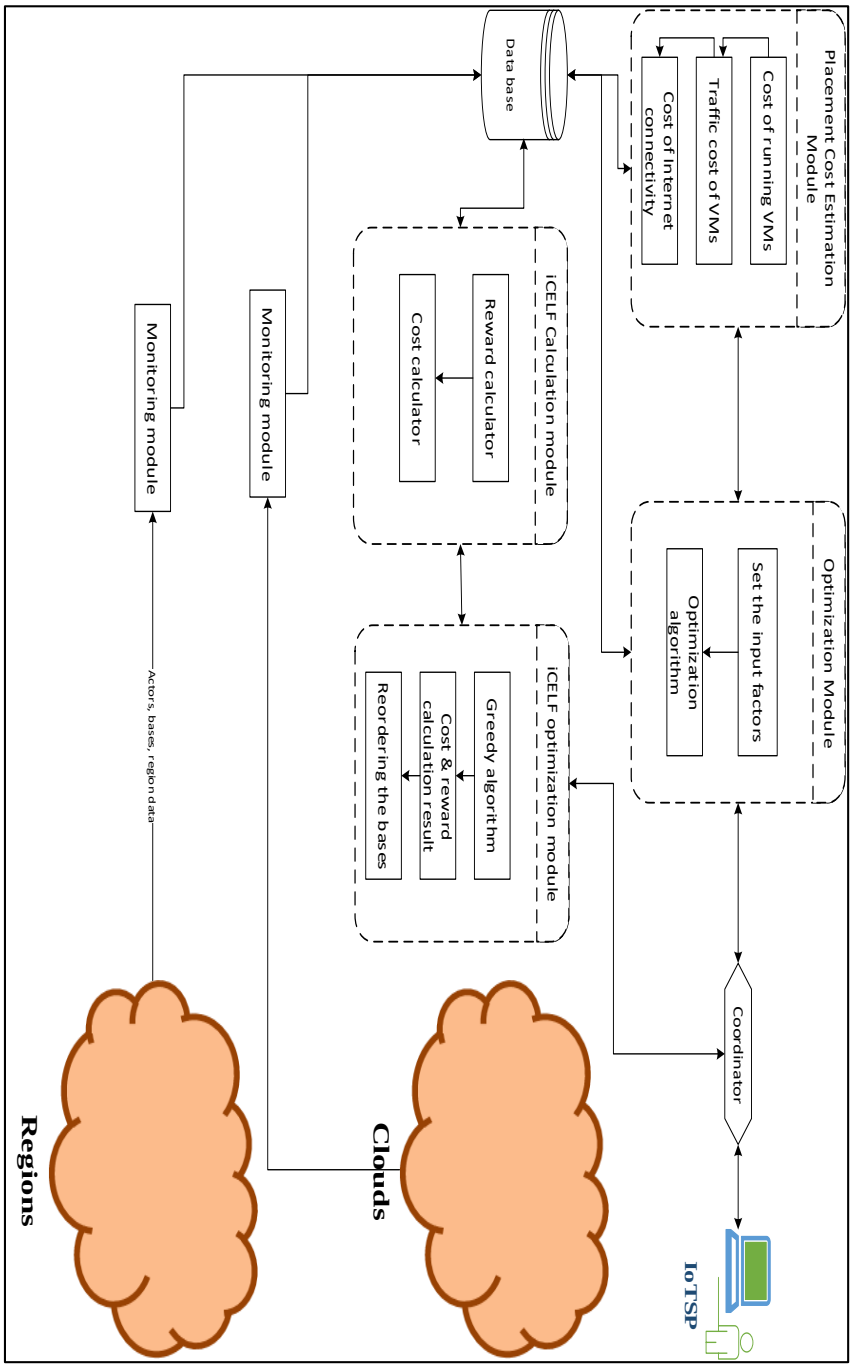


Figure 3-5: the proposed system architecture for the decision support tool

3.2 Chapter Summary

In this chapter, the methodology of this research has been reviewed. First, the method used for systematic literature review is briefly discussed. Then, the steps conducted to construct the decision support tool are explained. To this end, the problem scenario is detailed, and the very problem of the research is stated. Here the problem is divided into two sections namely: actor placement and VM placement. For the former, the submodularity feature and diminishing return are described. Considering these two properties, iCELF that is the proposed algorithm for actor placement optimization is discussed. Then, for the later, two mathematical models namely cost estimation function and cost optimization algorithm are detailed. Finally, the proposed system architecture for IoTSP decision support tool is illustrated in which the functions and algorithms are set up as modules.

Chapter 4. Simulation

In this chapter the simulation related topics are discussed whilst the next chapter explains the simulation experiments and analyzes the results.

First in 4.1, two real application cases are explained which clarify the necessity of the IoT-based danger recovery systems. These application cases are used for the simulation scenarios. As mentioned in 1.2, the research problem is divided into two major problems, and this dissertation suggests two separate solutions for each. Therefore, it is necessary to evaluate the proposed solutions separately. Also as the two problems, namely *actor placement optimization* and *VM placement optimization* are completely independent, two different simulation methods (which are proper for each of them) were suggested and performed.

Hence, the sections 4.2 and 4.3 are dedicated for the two evaluations. For the former, in 4.2 the proposed actor placement model (which has been described in 3.1.3) is evaluated using the two application cases of 4.1.1 and 4.1.2. The assumptions for the simulation configuration are detailed in 4.2.1. Then, the scenarios for actor placement simulation are detailed in 4.2.3 (four different types of situations) and 4.2.4 (in four parts).

For the latter, it is considered that for the two application cases mentioned in 4.1.1

and 4.1.2, after the IoTSP decides about its actors, the infrastructure for the system should be determined. For that, the assumptions and simulation configuration to utilize CloudIoT are explained in 4.3.1. Then, four different cases of the simulation scenarios are detailed in 4.3.2. The proposed VM placement algorithm (3.1.4) is assessed by these various simulation scenarios.

It is notable that as the nature of the two problems, are very different and the problems are completely independent, there was no use of merging or combining the experiments' results. Figure 4-1 illustrates the conceptual structure of the fourth and fifth chapter of this dissertation, which covers the simulation, the experiments, and their results.

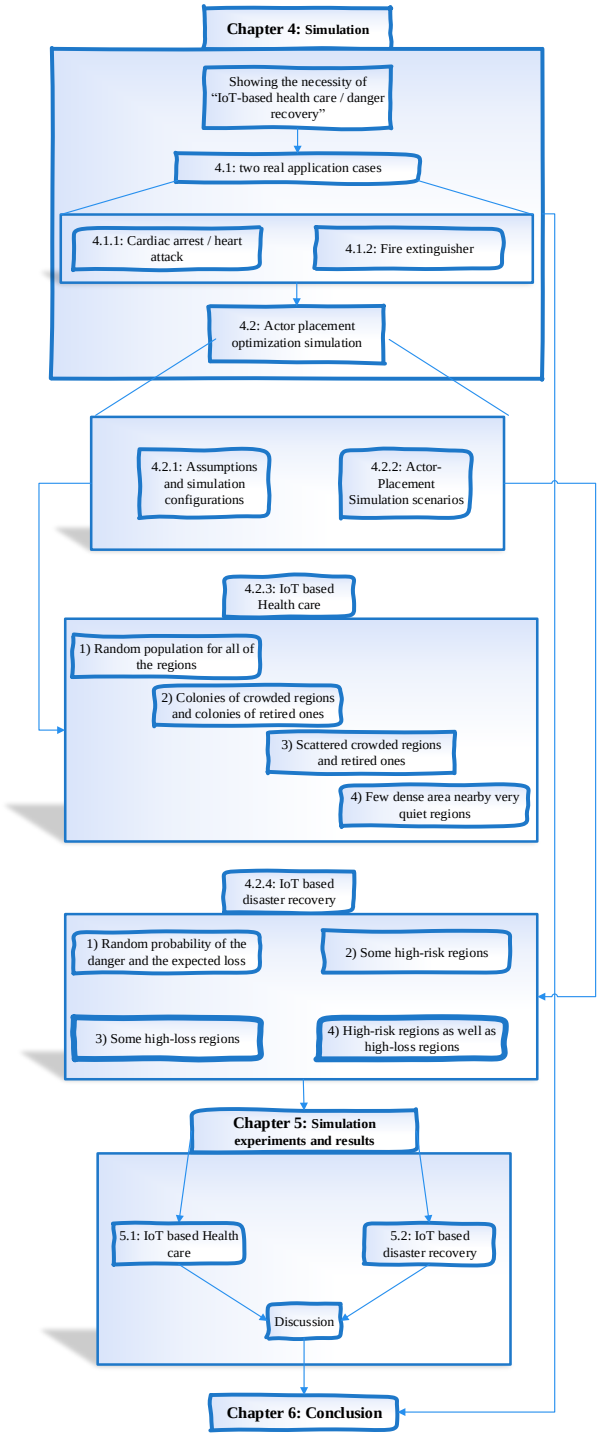


Figure 4-1: the conceptual structure of the fourth and fifth chapters

4.1 Suggested IoT-based Danger Recovery Scenarios

The main problem of this dissertation is very generic and can cover so many industries with different purposes. In other words, it is an abstract form of so many real cases that all share something. The common point is that there is a danger; that is to say a heart attack from health care industry, or fire or fluid in disaster recovery industry. In this section, two special instances of IoT services are explained: 1) Cardiac arrest/heart attack 2) Fire extinguisher. These two can be considered as real cases of IoT based system. The common point of the scenarios is that the IoT systems are made of IoT sensors (health measure sensors and fire detectors) as well as IoT actors (Ambulance Drone and Fire Fighter Drone). Hence, the IoTSP needs to plan for nice management and maintenance of both types of devices.

It is worthy to emphasize that the main problem of this thesis is not limited to these two scenarios. Rather, it can be applied on all of the problems having the common feature that is smart sensors and smart actors. Obviously, the proposed solution of this research is applicable for all of these problems. This section evaluates the proposed solutions in the two application cases with some simulation scenarios.

Finally, it should be noted that this dissertation has addressed two separate problems. Thence, to evaluate each of the proposed algorithms, two independent simulation

experiments have been set up. The first simulation is about the actor placement and evaluates the iCELF algorithm. The second is to examine the proposed VM machine placement algorithm.

It is remarkable that even though both of the algorithms are about cost-performance optimization, but because the scales and types of the optimization factors vary, one cannot combine the results. The other point is that the objectives of the optimizations are also separate, i.e. one is about the physical placement of the actors and the other is about the VM and its placement on multi clouds. Hence, the two mentioned simulations of this dissertation are designed and performed independently and the results are discussed separately.

4.1.1 Cardiac Arrest/Heart Attack

According to statistics, more than 600,000 people die every year due to heart attack; this is 25% of all deaths. In USA, every 34 seconds someone has heart attack. Similarly, the stat from other places of the world shows the huge loss due to heart attacks or cardiac arrest.

Fortunately, after these dangers still there is chance of survival. However, the rescue should take place within only 15 minutes. In other words, within 15 minutes the danger

should be reported, the patient should be reached and the rescue process should be taken place so that the patient can survive.

In many cases, the attack happens with no prior symptoms or signs, so calling the emergency and arrival of ambulance takes a long time. Especially, considering the traffic jam of the cities, there is high probability of late arrival.

One possible solution is utilizing *Ambulance Drone* instead of the ambulance cars. The drones should be smart and autonomous and should enjoy good connectivity to the center. The drone can reach the patient faster by flying direct paths instead of following the roads; and it does not get stuck by traffic jam neither. If the drone carries the required stuff from the center, a doctor/nurse can talk to the people nearby the patient and teach them how to save the patient.

Suggested simulation scenario: a company intends to offer an IoT-based health service to its clients. The purpose of the system is two features: 1) to collect all of the essential health data of patients and keep the records, and 2) to react to dangerous situation (e.g. heart attack) which is detected or reported by sending drones.

To achieve the goals of this system, there should be ubiquitous IoT system to support it by computing and storage servers. Besides, proper coverage of the drones should

be designed in a way that they can reach to all of the spots of the city in appropriate time.

Nevertheless, the IoTSP has to decide about the level of cost-performance of its offered services. In one hand, the number and places of computing and storage servers should be decided. As running any VM involves some cost, the decision should be well studied so that an optimized placement is chosen. This is the VM-placement decision.

On the other hand, similar decision should be taken for the drones, i.e. their number and their places. In addition, drones are expensive and maintaining them ready on their bases is costly. The placement decision to be made by the IoTSP is to find out the optimum bases to place the drones. The number of drones is limited due to the budget constraint. The optimization objective is to maximize the coverage of the whole drones or to minimize the “Time-to-Reaction” (TtR) including the time of discovery of the danger, time to reach the danger spot and time to tackle the problem.

The proposed model can address this problem and support the IoTSP to decide properly. Related simulation scenarios are designed to evaluate the proposed algorithm and compare its results to the other algorithms to show its working.

4.1.2 Fire Extinguisher

According to the published statistics, every 20 seconds a fire department responds

to a fire somewhere in the US. Fires in jungles cause huge environmental damage and the speed of fire growth is very high there. The relationship between the time of reaching to the fire and the damage happened is not linear. Hence, the sooner the fire is controlled, the lower the damage is.

To be ready to fight with fire, one idea is to place fire-extinguisher drones. Utilizing their feature of fast moving to reach the destination as well as the ability of flying over the roads and skipping the traffic jam, the fire could be controlled faster and wider.

The effectiveness of this system is based on the condition that there should be a reasonable coverage of the drones and accessible distance to the areas from the dronebases. In other words, there should be available bases equipped with a fire-extinguisher drone to the spot of the fire. Otherwise, before the drone arrives to the fireplace, it would have spread and burned so that controlling it will be impossible or so difficult. Besides, the loss that will happen is an exponential function of time.

The suggested simulation scenario for this case is an IoTSP that intends to offer an IoT-based fire extinguisher service to its clients. The system is to monitor the fire detectors and to react to any fire attacks by commanding the closest drone to fly to the fire spot and tackle it.

To achieve the goals of the IoT system, some supporting infrastructure should be available which is on multi-Clouds. Moreover, sufficient number of the drones should be set to reach to all of the probable fire spots in time.

Nonetheless, the IoTSP has to consider its service level according to cost-performance of the services. IoTSP needs to decide the number and places of its VMs and find its optimum VM placement.

Similarly, fire-extinguisher drones' number and places should be decided. In addition, drones are expensive and maintaining them ready on their bases is costly. The placement decision to be taken by the IoTSP is to find out the optimum bases to place the drones. There will be maximum number of drones according to the given budget. The optimization objective is to minimize "Time-to-Reaction" (TtR) which includes the time of discovery of the fire, time to reach the fire spot, and time to extinguish it.

The proposed model of this dissertation addresses the problem and supports the IoTSP in their decision-makings. Through some simulation scenarios, the proposed algorithms are evaluated and their results are compared to the other algorithms.

4.2 Actor Placement Optimization Simulation

In this section, the simulation environment for the *actor placement optimization* is

illustrated and few scenarios are discussed to assess the working of the proposed model.

4.2.1 Assumptions and Simulation Configurations

For the actor placement, we assume that the given regions for each center are considered and the distance among centers of all regions is known: G (V, E). In addition, the population of each region is assumed given as P (V). The IoTSP has decided its budget as B. The probability of the danger in each region is assumed as Prob (V).

The speed of actors is another given factor. The acceptable reaction time to the danger is assumed; i.e. the duration in which saving or protecting from the danger is possible.

The simulation tool for this section is MATLAB software. Using the matrix-based programming in MATLAB, we setup the environment for the simulation. To ascertain the working of the proposed model for different cases, these input factors are considered random: population of regions and the distances among them. Based on the randomness of these two factors, various simulation experiments are executed.

Utilizing MATLAB software, three different algorithms are programmed: random placement, greedy placement, and iCELF (proposed by this research). Generating random input factors, each algorithm is being executed and the results are analyzed. Details of the

experiments and results area were discussed in 05.1. The analysis shows that iCELF outperforms the other algorithms.

4.2.2 Actor-Placement Simulation Scenarios

Two main scenarios are considered in this dissertation to evaluate the working of the proposed *actor placement algorithm*; the first one is from the health care industry: cardiac arrest or heart attack rescue, while the second one is to help firefighters job. Each scenario is tested by various simulation experiments.

The details of the simulation experiments for the above-mentioned situations are mentioned in 5.1.

4.2.3 IoT-based Healthcare

In this section, the general scenario for healthcare is explained. The simulator has generated two input factors, namely population of the regions and distances among the regions.

To analyze the performance of the algorithm in different situations, the region populations are generated for four states: 1) Random population for all of the regions 2) Colonies of crowded regions and colonies of retired ones 3) Scattered crowded regions and retired ones 4) Few dense areas and so many quiet regions nearby them (like New York or

Washington, etc.).

One more factor that is changed on different simulation experiments is the actor speed. It has not been considered as new situation, but for the above-mentioned situations, three types of actor speed are considered: fast, medium, and slow. These three are relatively defined due to the distance and the urgency of the need for the recovery.

4.2.3.1 Random population for all of the regions

Considering *V=1000* regions (nodes) on which the IoTSP is supposed to cover for the service, in this series of simulation experiments, all of the regions will get random population between a very wide ranges: minimum is 1000 and maximum is 500,000.

The simulator repeats the input factor generation for 100 times and the given parameters are used by the three algorithms to calculate and compare the results. The experiment results are explained in 5.1.1.1.

4.2.3.2 Colonies of crowded regions and colonies of retired ones

The second series of simulation experiments is when the population is not distributed equally random among all of the regions, rather the simulator will generate some colonies of very crowded regions, with random population between 10,000 to 500,000 and some colonies of retired regions with random population between 1000 and

6,000. The number of crowded regions and retired regions is a given factor and this is used for some sensitivity analysis of the simulation.

Again, some rounds of random generation are executed and the three algorithms are tested by this. The results are discussed in 05.1.1.2.

4.2.3.3 Scattered crowded regions and retired ones

The third series of simulation experiments is also considers the population not to be equally random distributed among all of the regions, rather, the simulator will generate some very crowded regions (with given percentage of them among all regions) and the rest are retired.

Again, some rounds of random generation are executed and the three algorithms are tested by this. The results are discussed in 5.1.1.3.

4.2.3.4 Few dense area nearby very quiet regions

This time the situation is very similar to the previous one, except the few very dense areas area surrounded by so many quiet places. In other words, here the simulator plays with the distances among the crowded and retired regions.

The simulation will clarify whether it matter where the crowded places are or not, and how the proposed model and algorithms may address these different situations.

The results is depicted and discussed in 5.1.1.4.

4.2.4 IoT-based Disaster Recovery

In this section, the general scenario for disaster recovery is detailed. Different situations may occur considering two factors: probability of the danger in regions and expected loss of the regions due to the disaster. Four main possible situations are discussed in this section and will be taken as simulation cases.

4.2.4.1 Random probability of the danger and the expected loss

The first considered situation is when the probability of danger in all of the regions is assumed equal. In addition, the expected loss follows the random nature. Hence, a random function generates the probability and the expected loss and those are given to the tool as input factor.

4.2.4.2 Some high-risk regions

In the real world, the probability of happening of any specific danger might vary in different regions. In this scenario, some of the regions are considered to have high risk. It is notable that in this scenario the expected loss of the danger in all of the regions equally follows a random function.

4.2.4.3 Some high-loss regions

In this scenario, the probability of the danger is assumed randomly equal among all of the regions. Nevertheless, some of the regions are considered to have high-expected loss. As a real example, the jungles will have much higher loss in case of fire comparing to an industrialized complex.

4.2.4.4 High-risk regions as well as high-loss regions

This scenario is mixture of the both above cases: there are assumed some regions with high risk and probability of the danger and with high-expected loss. The rest of the regions are considered to have random probability of the danger and random expected loss.

4.3 VM Placement Optimization Simulation

In this section, considering the scenario of cardiac arrest/heart attack that is explained in 4.1.1, the simulation environment for the VM placement optimization is discussed and the simulation scenarios are explained.

4.3.1 Assumptions and Simulation Configurations

The world is divided into 6 regions called *user bases*. For the sake of simplicity (without losing generality), each user base is contained within a single time zone. The number of simultaneous connected users is assumed high at the working hours, because the

patients may wake up in the morning and start normal daytime activities. It is also assumed that 5% of the registered users of IoTSP are simultaneously connected to the servers during the peak time and only one tenth of that number of users is connected during the off-peak hours. Furthermore, each user's device (*thing*) makes a new request every 5 minutes while connected. The amount of the above-mentioned assumed data is shown in Table 4-1 and Table 4-2.

Table 4-1: Different regions and their timings

User base	Region	Time zone	Peak hours (local time)	Peak hours (GMT)
UB1	0 – N. America	GMT – 6.00	8-18	2-12
UB2	1 – S. America	GMT – 4.00	8-18	4-14
UB3	2 - Europe	GMT + 1.00	8-18	9-19
UB4	3 - Asia	GMT + 6.00	8-18	14-24
UB5	4 - Africa	GMT + 2.00	8-18	10-20
UB6	5 - Oceania	GMT + 10.00	8-18	18-4

Table 4-2: Number of users for peak and off-peak times

User base	Region	Simultaneous connected users during peak hrs	Simultaneous connected users during off-peak hrs
UB1	0 – N. America	500,000	50,000
UB2	1 – S. America	450,000	45,000
UB3	2 - Europe	300,000	30,000
UB4	3 - Asia	1,000,000	100,000
UB5	4 - Africa	50,000	5,000
UB6	5 - Oceania	70,000	7,000

In terms of the cost of hosting applications in a Cloud, a pricing plan is assumed which somehow follows the actual pricing plan of Amazon EC2 (2015) and Microsoft Azure (2015). The VM price and data traffic rates differ for the regions. In addition, the VMs are assumed to have three tiers, which are shown in Table 4-3. The details of the VM tier prices and data traffic rates for each of the six regions are given in

	# of processors	Memory	Storage	Bandwidth
A1	8	14 GB	240 GB	1 G
A2	4	7 GB	120 GB	1 G
A3	2	3.5 GB	60 GB	1 G

Table 4-4.

Table 4-3: VM classes (Tiers)

	# of processors	Memory	Storage	Bandwidth
A1	8	14 GB	240 GB	1 G
A2	4	7 GB	120 GB	1 G
A3	2	3.5 GB	60 GB	1 G

Table 4-4: Available VMs in regions and their prices

Region	VM A1 ($/hr)	VM A2 ($/hr)	VM A3 ($/hr)	Traffic rate ($/GB)
0 – N. America	0.6	0.3	0.15	50
1 – S. America	0.7	0.35	0.175	100
2 - Europe	0.6	0.3	0.15	60

3 - Asia	0.9	0.45	0.225	70
4 - Africa	NA	NA	NA	NA
5 - Oceania	0.8	0.4	0.2	70

Size of the virtual machines used to host the applications in the experiment is 100MB. Virtual machines have 1GB of RAM memory and 10MB of available bandwidth. Simulated hosts have x86 architecture, virtual machine monitor Xen, and Linux operating system. Each simulated data center (DC) hosts 50 dedicated virtual machines. Machines have 2 GB of RAM and 100GB of storage. Each machine has four CPUs, and each CPU has a capacity power of 10,000 MIPS[3]. A time-shared policy is used to schedule resources to VMs. Users are grouped by a factor of 1000, and requests are grouped by a factor of 100. Each user request requires 250 instructions to be executed.

4.3.2 VM-Placement Simulation Scenarios

Several scenarios are considered in our research to evaluate the working of the proposed model. Each scenario may have various instances by placing the VMs in different

[3] Million Instructions per Second: a unit of computing speed

regions. Hence, in each scenario, we run the simulation for all of the possible placements and record the results of each experiment. The considered parameters for this simulation are *overall cost* which consist of *Virtual Machine cost (VM cost*) and *traffic cost* as well as the *overall response time (RT)*. In this section, we describe the experiments in details; and the next section (4.5) explains the results.

4.3.2.1 Single powerful data center

The simplest scenario consists of modeling the case where a single centralized Cloud data center is used to host the IoT-based application. In this model, all requests from all of the users around the world are processed by this single data center. The data center has 50 virtual machines allocated to the application and it is from A1 class of the specifications shown in Table 4-3. Generally, this scenario is not considered as cloud federation, but only a single data center that can be on any cloud service provider.

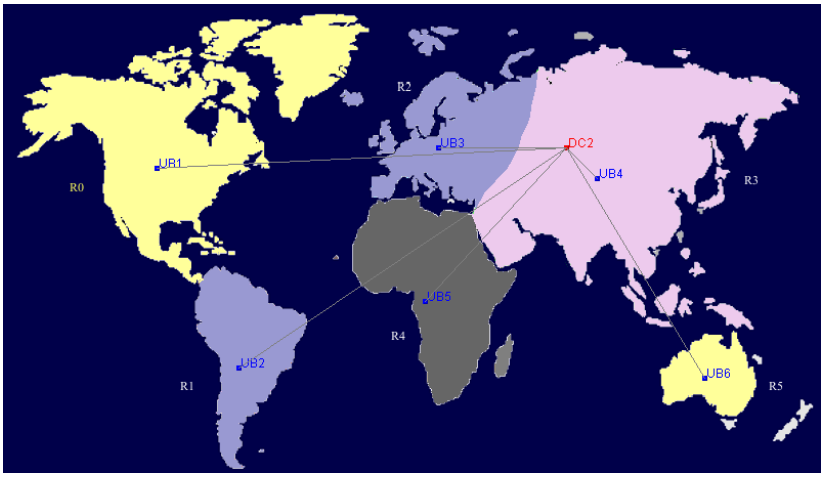


Figure 4-2: An example of one-DC-scenario

For the experiments, we change the location of the data center and run the experiment for every region, and calculate the results of all different cases. The results of the first scenario are explained in 5.2.1.

4.3.2.2 Two small size data centers

The second scenario consists of the use of two data centers at the same time, each one from the A3 class (Table 4-3). Data centers have 50 virtual machines allocated to the application. These two are placed in different regions; therefore, combination of C(5,2) will be the number of required experiments. The results of the 10 experiments are discussed in 5.2.2.

4.3.2.3 Two powerful data centers

The previous scenario is repeated with two VMs from A1 class that is the first tier,

and different possible placements of them are simulated and recorded. Figure 4-3 demonstrates the placement. The result is illustrated in 5.2.3.

4.3.2.4 Full data center coverage

The last scenario to be simulated is deploying VMs in all of the available DCs. As shown in

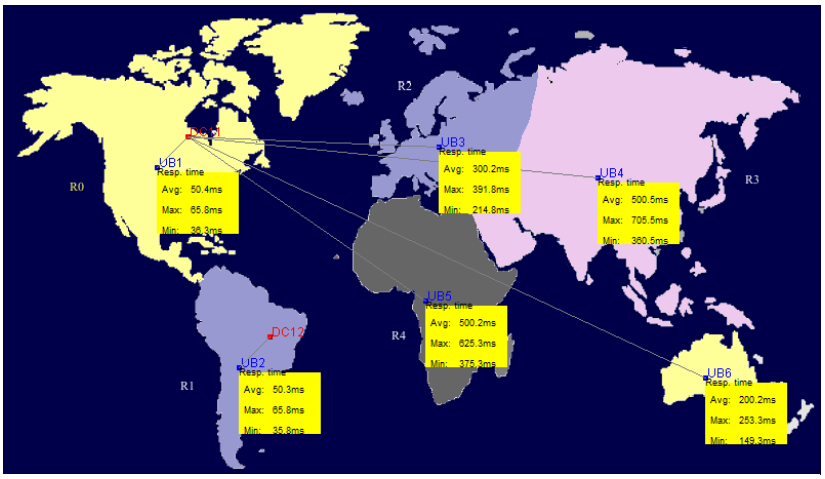


Figure 4-3 Placement of two DCs over globe

	# of processors	Memory	Storage	Bandwidth
A1	8	14 GB	240 GB	1 G
A2	4	7 GB	120 GB	1 G

A3	2	3.5 GB	60 GB	1 G

Table 4-4 all of the six regions have a regional data center, except for the case of Africa (region 4). The case is run for having DC of tier 1 on all of five regions, and then the experiment is repeated for tier 2 and tier 3. The last try is the case of running all of the three tiers at the same time on all of the regions (15 DCs). The results of the four above-mentioned experiments are explained in 5.2.4.

4.4 Chapter summary

In this chapter, three illustrative scenarios are explained and considered for the simulation, in which an IoT service provider offers its IoT-based service to its customers locally or globally. The assumptions' details and simulation's configurations are explained to give exact simulation situation to the readers. The scenarios are in the domain of health-care. The first one deals with the IoT level, i.e. actor placement; while the second one deals with the infrastructure level, i.e. VM placement. The chapter explains the simulation scenarios in detail.

Chapter 5. Experiments and Results

Multiple simulation scenarios are conducted to evaluate the working of the proposed model of this research. Various simulation experiments are performed as explained inChapter 4. In this chapter, the details of each experiment and their results are described.

5.1 Actor Placement Optimization Simulation

As discussed in 4.2.3, two general cases of usage for actor placement optimization are considered. Each of these cases represents a variety of instances; hence, the result of the simulation experiments can be easily generalized. The first case (cardiac arrest/health care) represents “IoT-based health care system”; while the second one (fire extinguisher) represents IoT-based disaster recovery.

To evaluate the proposed algorithm for actor placement, a group of simulation experiments has been executed for each case. To this end, two other algorithms are used as baselines: Random selection and greedy algorithm. In this section, the simulation environment and assumptions are firstly explained. Then, the results of each experiment are detailed in following parts.

5.1.1 IoT-based Healthcare

The simulation for IoT-based healthcare is performed using MATLAB R2016a application. The code for this simulation is depicted in Appendix A.

The main purpose of the simulation is to evaluate the proposed algorithm of this dissertation. Hence, two other algorithms are considered as references and the results of the three algorithms, namely 1) random selection, 2) greedy algorithm, and 3) iCELF proposed by this dissertation are calculated and measured from different perspectives. The random algorithm just selects random set of possible nodes (due to budget limitation) which are assumed equal to300 nodes. The greedy algorithm tries to add the nodes in which the probability of attack occurrence is the most. The iCELF is explained in 3.1.3.3.

The simulation environment consists of 1000 nodes (regions) considered for the simulation. The population of each node is assumed as the input factor (generated from a random function that gives the population for each node). The simulation is run for 100 times so that the random data can represent the reality. On the other hand, the feasible number of the actors is assumed 300 ea. that means that according to the budget limitation and other conditions of the IoTSP, 300 actors can be purchased, supported and maintained. Therefore, the optimization problem is to find the optimum placement for the 300 actors.

For simplicity of the simulation, without losing the generality, the probability of the danger (in this example cardiac arrest/heart attack) is assumed equal for all of the people. Hence, the probability of its occurrence in any region is a function of the population of the region. In further studies, the accuracy can be improved by considering the age ratio of regions and calculation of the probability of the attack according to the age composition of each region.

The actor speed is another given data, which affects the covered regions. The reason is that after the attack occurs, there is limited time within which the saving efforts can be useful, i.e. if the actor reaches the attacked person later than the *Accepted Time* this is not acceptable. This can be calculated based on the actor speed and the distance between the region in which the actor is placed, and the regions of the patient. Besides, to be more similar to the reality, three different types of the actors are considered with three various prices. The main difference of the actors is in their speed that affects the objective of the research objective, i.e. the average service time and the covered population.

Final input factor of this simulation is the distances among the regions. This is a bit more complicated and by combination of population and distance factor, four different types are considered to evaluate the working of the proposed algorithm. These four cases

are described in 5.1.1.1 to 5.1.1.3. Also for each of the cases, three different actor speeds are considered.

5.1.1.1 Random population for all of the regions

As it is explained in 4.2.3.1, three hundreds of simulation experiments are performed and the results are detailed in this section. First, considering the actor speed medium and normal, i.e. 100 km/h, the results of this set of experiments is shown in Table 5-1.

Table 5-1: Comparison of the three algorithms by different measures for actors with speed of 100 km/h

	iCELF	**Greedy**	**Random**
Unique Nodes	72	60	52
Redundant Nodes	21	48	30
Served Population	79.98%	67.77%	52.11%
Average service time	4.59	5.43	6.33

The values of the table are the average of the 100 rounds of experiments. The first measure indicates how many nodes (i.e. regions) are covered by each of the algorithms; the

second measure, the redundant nodes shows the number of nodes covered by more than one actor. It can be the index of wastage of the resources, and obviously the more redundant node means the less efficient the placement is. The third measure deals with the covered population. It is similar to the first one, but the first was about the number of regions while the third is about the real population whom are under support of the placed actors. The comparison of results of the table is also depicted in Figure 5-1.

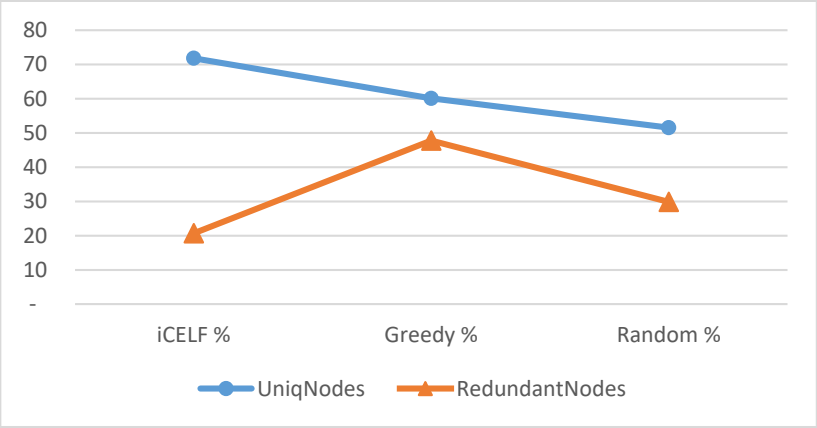


Figure 5-1: Comparison diagram of unique nodes and population covered by the three algorithms for random population

The diagram depicts that the proposed algorithm of this dissertation outperforms the two others, i.e. random and even greedy algorithms. As clear, the population covered

by the iCELF is more than 70% of the total population. This happened with only 300 actors hence the rest of 700 regions are without any actors. Nevertheless, the iCELF algorithm has optimized the placement and decided the most efficient nodes for the actors.

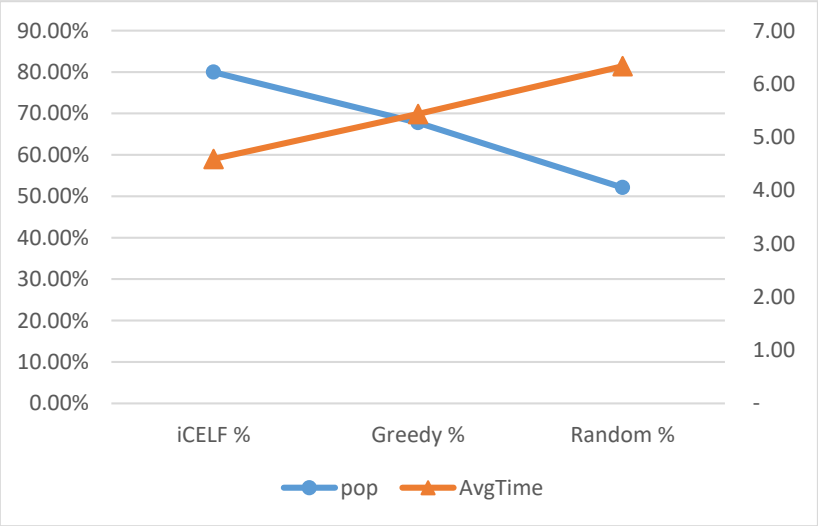


Figure 5-2: Comparison diagram of the redundant nodes by the three algorithms for 100 km/h speed

From another perspective, Figure 5-2 shows how many redundant and unnecessary nodes each algorithm has selected. This can be another factor to rank the algorithms. Obviously, the less the redundant nodes are, the more efficient the algorithm is.

5.1.1.2 Colonies of crowded regions and colonies of retired ones

As it is explained in 4.2.3.1, first a hundred simulation experiments are performed for the situation of an environment of scattered regions mixed of crowded and retired areas. This simulation experiment has been performed in few rounds considering the percentage of the crowded regions to all. The results of the three algorithms in the mentioned rounds of experiments are discussed in this section. First, the crowded percentage is considered equal to 50%. For a hundred times, the random population of crowded and retired regions is generated and the random distances among them are determined. Then, the three algorithms are executed and the average results of this set of experiments is shown in Table 5-2.

Table 5-2 Comparison of the three algorithms by different measures for the case of two crowded and two retired in each colony

	iCELF	**Greedy**	**Random**
Unique Nodes	77	64	54
Redundant Nodes	18	45	31
Served Population	84.34%	71.01%	54.12%
Average service time	4.58	5.50	6.50

The first measure indicates how many nodes (i.e. regions) are covered by each of the algorithms; while the second measure, redundant nodes shows the number of nodes covered by more than one actor. It can be index of wastage of the resources, because the more redundant node means the less efficient the placement is.

The third measure deals with the percentage of the covered population to all the population of the whole regions. It is similar to the first one, but the first was about the number of regions while the third is about the real population whom is under the support of the placed actors. This can be considered as the most important factor to decide about the placements.

The fourth measure indicates the average service time to the regions, caused by the suggested placement of each algorithm. The comparison diagram of the results is also depicted in Figure 5-5 and Figure 5-6.

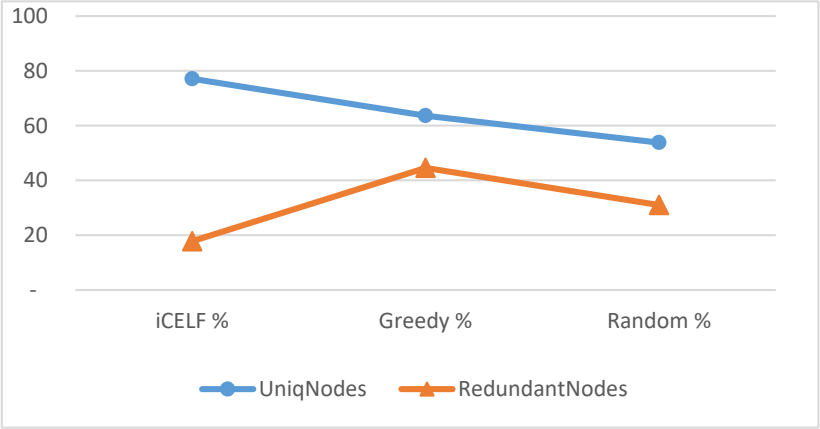


Figure 5-3: Comparison diagram of the covered unique nodes and redundant nodes by the three algorithms for the case of 50% crowded regions

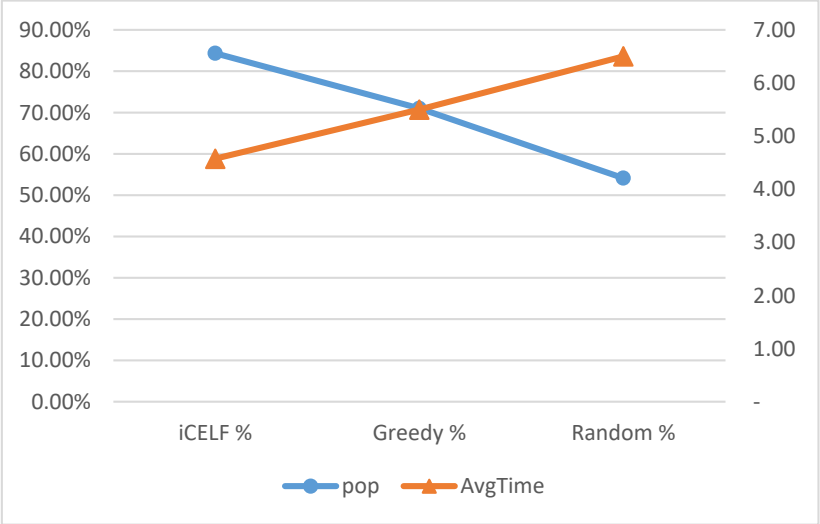


Figure 5-4: Comparison diagram of the covered population and average service time by the three algorithms for the case of 50% crowded regions

The diagram depicts that iCELF outperforms two others, i.e. random and even

greedy algorithms. As it is clear, the population covered by the iCELF is almost 85% of the total population. This happened with only 300 actors hence the rest 700 regions are out of any actors. Besides, the average service time to all of the regions is less than 5.0; while it is 5.5 for greedy algorithm.

Nevertheless, the iCELF algorithm has optimized the placement and decideds the most efficient nodes for the actors. Figure 5-3 shows how many redundant and unnecessary nodes each algorithm has selected. This can be another factor to rank the algorithms. Obviously, the less the redundant nodes are, the more efficient the algorithm is.

5.1.1.3 Scattered crowded regions and retired ones

As it is explained in 4.2.3.10, first a hundred simulation experiments are performed for the situation of an environment of scattered regions mixed of crowded and retired areas. This simulation experiment has been performed in few rounds considering the percentage of the crowded regions to all. The results of the three algorithms in the mentioned rounds of experiments are discussed in this section.

First, the crowded percentage is considered 50%. For a hundred times the random population of crowded and retired regions is generated and the random distances among them are determined. Then the three algorithms are executed and the average results of this

set of experiments are shown in Table 5-3.

Table 5-3: Comparison of the three algorithms by different measures for the case of 50% crowded regions

	iCELF	**Greedy**	**Random**
Unique Nodes	72	58	52
Redundant Nodes	20	32	30
Served Population	91.46%	85.40%	52.91%
Average service time	4.50	5.50	6.09

The first measure indicates how many nodes (i.e. regions) are covered by each of the algorithms; while the second measure, redundant nodes, shows the number of nodes covered by more than one actor. It can be index of wastage of the resources, because the more redundant node means the less efficient the placement is.

The third measure deals with the percentage of the covered population to all the population of the whole regions. It is similar to the first one, but the first was about the number of regions; while the third one is about the real population whom are under support of the placed actors. This can be considered as the most important factor to decide about the placements.

The fourth measure indicates the average service time to the regions, caused by the suggested placement of each algorithm. The comparison diagram of the results is also depicted in Figure 5-5 and Figure 5-6.

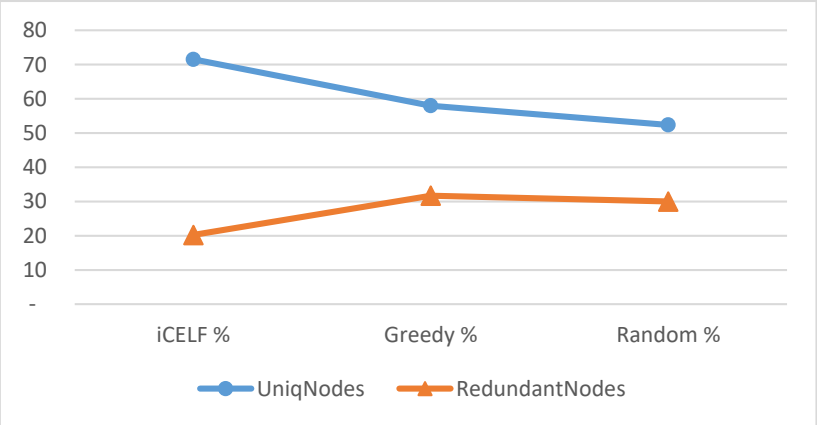


Figure 5-5: Comparison diagram of the covered unique nodes and redundant nodes by the three algorithms for the case of 50% crowded regions

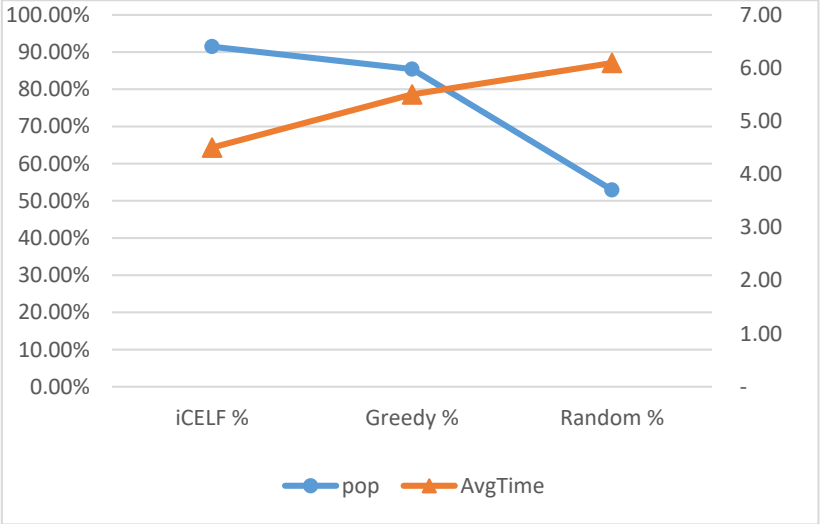


Figure 5-6: Comparison diagram of the covered population and the average service time by the three algorithms for the case of 50% crowded regions

The diagram depicts that iCELF outperforms two others, i.e. random and even greedy algorithms. As it is clear, the population covered by the iCELF is more than 90% of the total population. This happened with only 300 actors; hence, the rest of 700 regions are out of any actors. Besides, the average service time to all of the regions is around 4.5 while it is 5.5 for greedy algorithm.

Nevertheless, the iCELF algorithm has optimized the placement and decided the most efficient nodes for the actors. Figure 5-5 shows how many redundant and unnecessary nodes each algorithm has selected. This can be another factor to rank the algorithms.

Obviously, the less the redundant nodes are, the more efficient the algorithm is.

5.1.1.3.1 Sensitivity analysis: different actors' speed

The same simulation scenario has been done for the slow actors, i.e the actor speed was 10. A hundred experiments are generated and evaluated for the slow actors and the results are shown in Table 5-4.

Table 5-4: Comparison of the three algorithms by different measures for actors with speed of 10 km/h

	iCELF	**Greedy**	**Random**
Unique Nodes	30.00%	29.84%	26.08%
Redundant Nodes	0.10%	0.26%	4.02%
Served Population	50.76%	50.53%	26.27%

In this case, because the speed is too slow to reach to the patients, the results of iCELF are very close to those of the greedy algorithm. Still iCELF outperforms the other two algorithms from all the three perspectives (with less difference).

On the other hand, the same simulation scenarios are executed considering the 500 km/h for actors' speed. The summary of the results of hundred rounds of experiments is

illustrated in Table 5-5.

Table 5-5: Comparison of the three algorithms by different measures for actors with speed of 500 km/h

	iCELF	**Greedy**	**Random**
Unique Nodes	79.28%	64.88%	54.26%
Redundant Nodes	21.32%	52.38%	34.56%
Served Population	85.14%	70.01%	53.91%

The same conclusion can be regarded considering this. In this case, as the speed of the actor has increased, the covered regions and populations of the greedy algorithm have improved and become around 65%. Nevertheless, and the redundant nodes have increased and become around 52%, but iCELF makes much better results than the greedy algorithm which averagely is about 79% and the redundant nodes are around 21% which is still very low. This means iCELF saves the resources with its nice distribution up to 30%.

Finally, the results of iCELF on the three different speeds are compared in Table 5-6.

Table 5-6: Comparison of iCELF algorithms by different measures for three actors' speed

	10	**100**	**500**
Unique Nodes	30.00%	79.58%	79.28%
Redundant Nodes	0.10%	14.99%	21.32%
Served Population	50.76%	88.56%	85.14%
Average service time	4.50	5.50	6.09

The final comparing measure is the average response time, i.e. the duration of reaching to the patient with the selected actor placement. Figure 5-7 depicts the average rate over 100 times of the random experiments.

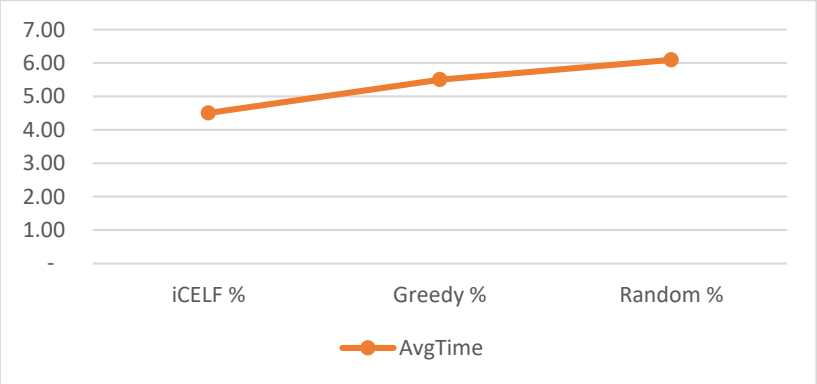


Figure 5-7: Comparison diagram of the covered unique nodes and redundant nodes by the three algorithms for the case of two crowded regions

5.1.1.4 Few dense area nearby very quiet regions

As it is explained in 4.2.3.1, few hundreds of simulation experiments are performed for the situation of an environment of few dense areas that is surrounded by very quiet regions. This is the most fitting situation to the reality of many countries; like in Korea where Seoul metropolitan is surrounded by so many small cities like Ansan, Suwon etc. The results of the three algorithms in the mentioned rounds of experiments are discussed in this section.

First, only two very crowded regions are assumed with few surrounding nodes. For hundred times, the random population of crowded and retired regions is generated and random distances among them are determined. Then, the three algorithms are executed and

the average results of this set of experiments are shown in Table 5-7.

Table 5-7: Comparison of the three algorithms by different measures for the case of two very crowded regions surrounded by quiet areas

	iCELF	**Greedy**	**Random**
Unique Nodes	75	64	52
Redundant Nodes	21	44	32
Served Population	83.76%	71.22%	53.13%
Average service time	4.59	5.37	6.57

The first measure indicates how many nodes (i.e. regions) are covered by each of the algorithms; while the second measure, redundant nodes shows the number of nodes covered by more than one actor. It can be index of wastage of the resources, because the more redundant node means the less efficient the placement is.

The third measure deals with the percentage of the covered population to all the population of the whole regions. It is similar to the first one, but the first was about the number of regions, while the third is about the real population whom are under support of

the placed actors. This can be considered as the most important factor to decide about the placements.

The fourth measure indicates the average service time to the regions, which is caused by the suggested placement of each algorithm. The comparison diagram of the results is also depicted in Figure 5-8 and Figure 5-9.

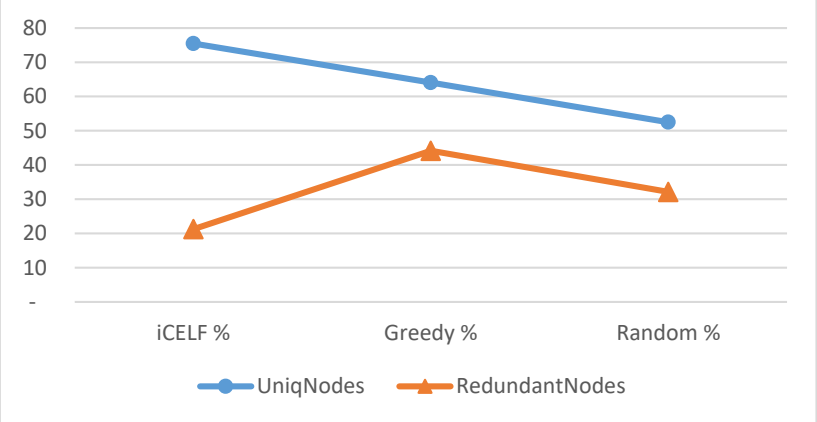


Figure 5-8: Comparison diagram of the covered unique nodes and redundant nodes by the three algorithms for the case of two very crowded regions

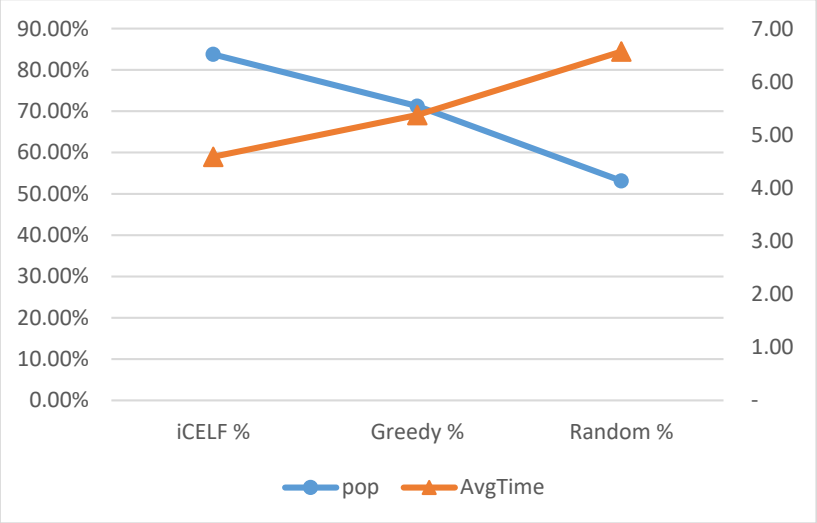


Figure 5-9: Comparison diagram of the covered population and average service time by the three algorithms for the case of two very crowded regions

The diagram depicts that iCELF outperforms two others, i.e. random and even greedy algorithms. As it is clear, the population covered by the iCELF is more than 80% of the total population. This happened with only 300 actors; hence, the rest of 700 regions are out of any actors. Besides, the average service time to all of the regions is around 4.5; while it is 5.5 for greedy algorithm.

Nevertheless, the iCELF algorithm has optimized the placement and decided the most efficient nodes for the actors. Figure 5-8 shows how many redundant and unnecessary nodes each algorithm has selected. This can be another factor to rank the algorithms.

Obviously, the less the redundant nodes are, the more efficient the algorithm is.

5.1.1.4.1 Sensitivity analysis: different actors' speed

The same simulation scenario has been done for the slow actors, i.e. the actor speed is 10. A hundred experiments are generated and evaluated for the slow actors and the results are shown in Table 5-8.

Table 5-8: Comparison of the three algorithms by different measures for actors with speed of 10 km/h

	iCELF	**Greedy**	**Random**
Unique Nodes	30.00%	29.84%	26.08%
Redundant Nodes	0.10%	0.26%	4.02%
Served Population	50.76%	50.53%	26.27%

In this case, because the speed is too slow to reach to the patients, the results of iCELF are very close to those of the greedy algorithm. Still iCELF outperforms the other two algorithms from all the three perspectives (with less difference).

On the other hand, the same simulation scenarios are executed considering the 500 km/h for actors' speed. The summary of the results of hundred rounds of experiments is

illustrated in Table 5-9.

Table 5-9: Comparison of the three algorithms by different measures for actors with speed of 500 km/h

	iCELF	**Greedy**	**Random**
Unique Nodes	79.28%	64.88%	54.26%
Redundant Nodes	21.32%	52.38%	34.56%
Served Population	85.14%	70.01%	53.91%

The same conclusion can be regarded considering this. In this case, as the speed of the actor has increased, the covered regions and populations of the greedy algorithm have improved and become around 65%. Nevertheless, and the redundant nodes have increased and become around 52%, but iCELF makes much better results than the greedy algorithm which averagely is about 79% and the redundant nodes are around 21% which is still very low. This means iCELF saves the resources with its nice distribution up to 30%.

Finally, the results of iCELF on the three different speeds are compared in Table 5-10.

Table 5-10: Comparison of iCELF algorithms by different measures for three actors' speed

	10	100	500
Unique Nodes	30.00%	79.58%	79.28%
Redundant Nodes	0.10%	14.99%	21.32%
Served Population	50.76%	88.56%	85.14%
Average service time	4.50	5.50	6.09

5.1.1.4.2 Sensitivity analysis: different combinations of crowded and retired areas

Another factor for sensitivity analysis is the ratio of mixture of crowded regions with retired ones. The combination has been tested for four cases: 1) when the ratio is 10% - 90%, 2) when the ratio of crowded regions to retired ones is 50%-50%, 3) when the ratio is 80% - 20%, and finally 4) when the ratio is 90% - 10%. Each of the four cases is executed for 100 rounds of experiments to assure unbiased results. The results of these four cases plus the previous case (30% - 70%) are depicted in Table 5-11 and Figure 5-10.

Table 5-11: Comparison table of *percentage of the covered population* by the three algorithms in the five different population distribution in the regions

Crowded to retired ratio	iCELF	Greedy	Random
10%-90%	77.01%	82.81%	22.86%
30%-70%	61.02%	57.05%	23.42%
50%-50%	49.80%	44.81%	20.81%
80%-20%	42.89%	37.19%	22.40%
90%-10%	42.39%	36.36%	22.48%

As seen in the table in four of the five cases, iCELF outperforms the two other algorithms. It can be said that iCELF produces between 6~14% better results than the greedy algorithm and between 45~70% than the random selection. The only case where the greedy algorithm produces better results than iCELF is the first one in which the crowded regions are very few and the retired regions are 90% of all. This is an exceptional case, but it should be notified for the IoTSP that in case the crowded regions are very low, iCELF is not recommended. The results shown in the diagram also suggest that after iCELF passes the greedy algorithm, there is no meaningful difference once the “distribution of the

crowded regions" is changed. The difference between iCELF and greedy algorithm becomes stable and almost fixed.

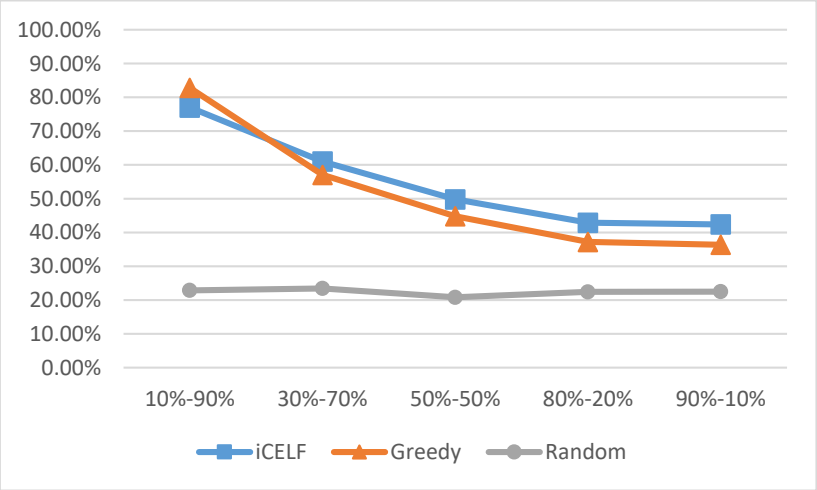


Figure 5-10: Sensitivity analysis results in the five crowded regions ratio

Table 5-12: Comparison table of the total covered population by the three algorithms in the five different population distributions in the regions

Crowded to retired ratio	iCELF	Greedy	Random	Total population
10%-90%	22,202,672	23,876,106	6,589,890	28,832,327
30%-70%	47,363,776	44,284,188	18,182,746	77,622,596
50%-50%	63,798,897	57,398,728	26,657,128	128,100,255
80%-20%	87,311,184	75,722,396	45,611,187	203,586,145
90%-10%	98,118,094	84,174,088	52,037,056	231,476,545

Table 5-13: Comparison of the covered percentage on the three algorithms in five population distributions

Crowded to retired ratio	iCELF	Greedy	Random
10%-90%	**77.01%**	**82.81%**	**22.86%**
30%-70%	**61.02%**	**57.05%**	**23.42%**
50%-50%	**49.80%**	**44.81%**	**20.81%**
80%-20%	**42.89%**	**37.19%**	**22.40%**
90%-10%	**42.39%**	**36.36%**	**22.48%**

The result shows that the proposed algorithm by this dissertation (iCELF) outperforms the other two mentioned algorithms in the selection of the nodes for actor placement. It also points that iCELF can well handle the crowded areas and suggest very efficient placement so that greater results are achieved with the same number of actors.

5.1.2 IoT-based Disaster Recovery

Similar to the case of healthcare, the simulation for IoT-based disaster recovery is also performed using MATLAB R2016a application. The code for this simulation is depicted in 0.

The simulation is supposed to evaluate the proposed algorithm, iCELF. Hence, the results of the three algorithms, namely 1) random selection, 2) greedy algorithm, and 3) iCELF that is proposed by this dissertation are processed and measured from different perspectives.

The simulation environment consisting of 1000 nodes (regions) is considered for the simulation. The probability of the danger and the expected loss of the danger (in case of happening) are assumed as the input factors (generated from a random function that gives the population for each node). The simulation is run for 100 times for each of the considered situations so that the random data can represent the normal case.

Besides, to be more similar to the reality, three different types of the actors are considered with three various prices. The main difference of the actors is in their speed that affects the objective of the research objective, i.e. the average service time and the covered population.

At the end, some sensitivity analysis has been performed to evaluate the working of the algorithm in the changes and the rate of environmental changes that may affect their results.

5.1.2.1 Random probability of the danger and random loss amounts in the regions

As it is explained, in 4.2.3.1 and 4.2.4.1, one hundred of simulation experiments are performed and the results are detailed in this section. The results of this set of experiments are shown in Table 5-14.

Table 5-14: Comparison of the three algorithms by different measures

	iCELF	**Greedy**	**Random**
Unique Nodes	**81**	**66**	**55**
Redundant Nodes	**22**	**62**	**34**
Average Time	**4.70**	**5.74**	**6.89**

The values of the table are the average of the 100 rounds of experiments. The first measure indicates how many unique nodes (i.e. regions) are covered by each of the algorithms, i.e. the number of the nodes that have only one actor to support them in case of danger. The second measure, redundant nodes shows the number of nodes covered by more than one actor. It can be index of wastage of the resources or inefficiency of the actor

placement. Obviously, the more redundant node means the less efficient the placement is. The third measure deals with the average service time, i.e. in overall, the average time the actors reach to the danger spot. The comparison of results of the table is also depicted in Figure 5-11.

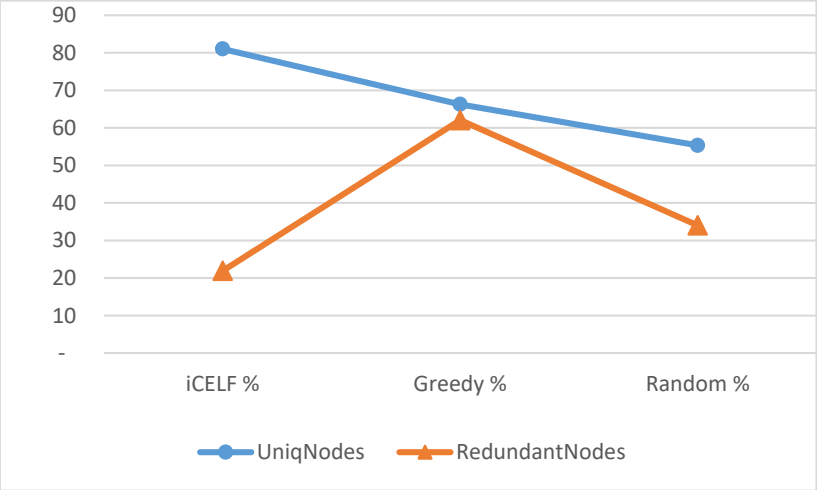


Figure 5-11: Comparison diagram of unique nodes and redundant nodes by the three algorithms

The diagram depicts that the proposed algorithm of this dissertation outperforms the two others, i.e. random and even greedy algorithms. As it is clear, the unique nodes covered by the iCELF are almost 80% of the total nodes, while for greedy algorithm they are around 70%. Nevertheless, considering the minimum redundant nodes of the iCELF

algorithm (around 20%) while the greedy algorithm suggests more than 60% and the random placement incurs more than 30%, it can be said that iCELF has optimized the placement and decided the most efficient nodes for the actors.

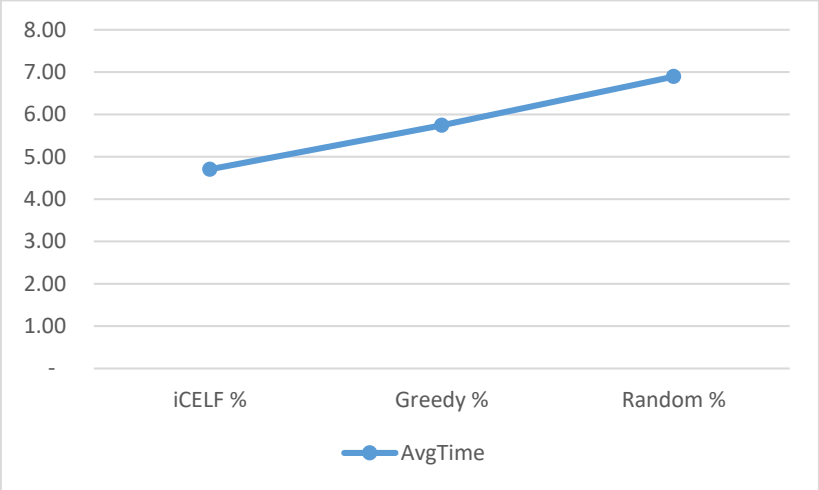


Figure 5-12: Comparison diagram of the average time for the placements suggested by the three algorithms

From another perspective, to compare the three algorithms, considering the third row of the

Table 5-18 the average time the suggested placement incurs is compared in Figure 5-12. In addition, the overall reward of each placement is evaluated in the same figure.

5.1.2.2 Some high-risk regions

As it is explained in 4.2.4.2, one hundred of simulation experiments are executed and the results are discussed in this section. The main assumption of these hundred rounds is that few regions are assumed to be in high risk, i.e. high probability of danger occurrence. Nevertheless, the expected loss of the danger is assumed random. The results of this set of experiments are shown in Table 5-15.

Table 5-15: Comparison of the three algorithms by different measures

	iCELF	**Greedy**	**Random**
Unique Nodes	81	66	55
Redundant Nodes	22	62	34
Average Time	4.70	5.74	6.89

The values of the table are the average of the 100 rounds of simulation experiments. The first measure indicates how many unique nodes (i.e. regions) are covered by each of the algorithms, i.e. the number of the nodes with only one actor to support them in case of danger. The second measure, redundant nodes shows the number of nodes covered by more

than one actor. It can be index of wastage of the resources or inefficiency of the actor placement. Obviously, the more redundant node means the less efficient the placement is. The third measure deals with the average service time, i.e. in overall, the average time the actors reach to the danger spot. The comparison of results of the table is also depicted below in Figure 5-13.

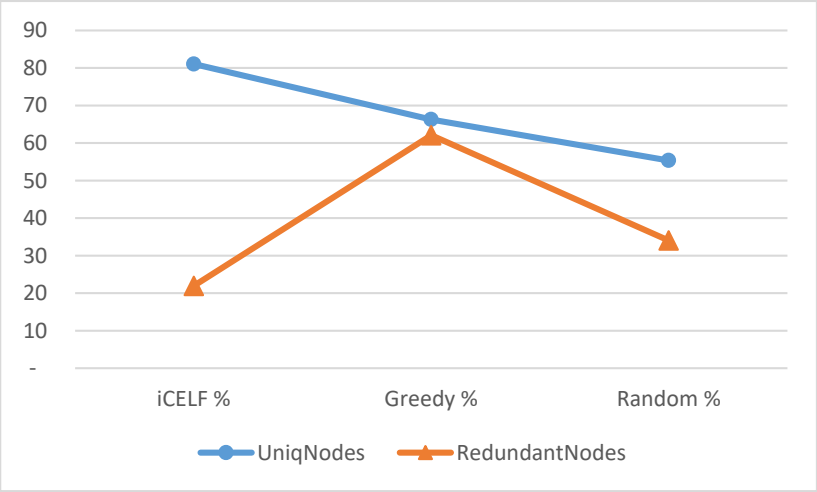


Figure 5-13: Comparison diagram of unique nodes and redundant nodes by the three algorithms

The diagram depicts that the proposed algorithm of this dissertation outperforms the two others, i.e. random and even greedy algorithms. As it is clear, the unique nodes covered by the iCELF are more than 80% of the total nodes; while for the greedy algorithm they are almost 70%. Nevertheless, considering the minimum redundant nodes of the

iCELF algorithm (little bit more than 20%) while the greedy algorithm suggests more than 60% and the random placement more than 30%, it can be said that iCELF has optimized the placement and suggested the most efficient nodes for the actors.

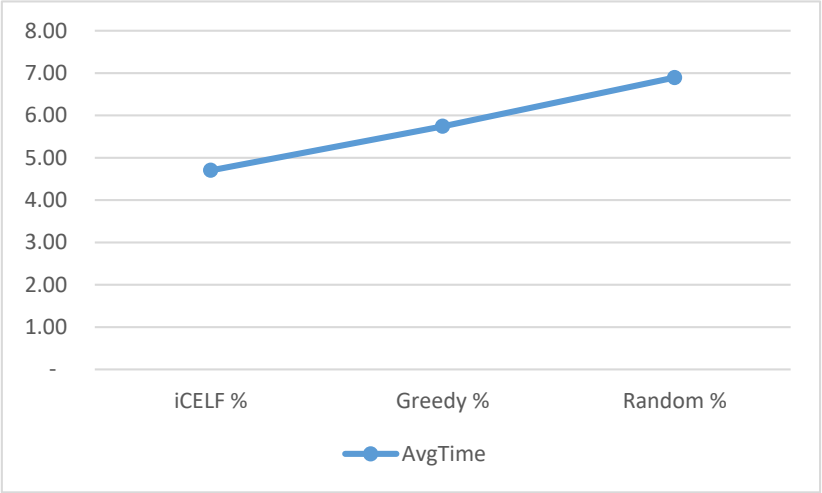


Figure 5-14: Comparison diagram of the average time for the placements suggested by the three algorithms

From another perspective, to compare the three algorithms, considering the third row of the Table 5-15, the average time the suggested placement incurs is compared in Figure 5-14.

5.1.2.3 Some high-loss regions

As it is explained in 4.2.4.2 and 4.2.4.3, one hundred of simulation experiments are

executed and the results are discussed in this section. The main assumption of these hundred rounds is that some regions are assumed to have expected loss, i.e. huge amount of loss in case of danger occurrence. Nevertheless, the probability of the danger in the regions is assumed random. The results of this set of experiments are shown in Table 5-16.

Table 5-16: Comparison of the three algorithms by different measures

	iCELF	**Greedy**	**Random**
Unique Nodes	85	69	56
Redundant Nodes	20	55	34
Average Time	4.63	5.69	6.94

The values of the table cells are the average of the 100 rounds of simulation experiments. The first measure indicates how many unique nodes (i.e. regions) are covered by each of the algorithms, i.e. the number of the nodes with only one actor to support them in case of danger. The second measure, redundant nodes shows the number of nodes covered by more than one actor. It can be index of wastage of the resources or inefficiency of the actor placement. Obviously, the more redundant node means the less efficient the

placement is. The third measure deals with the average service time, i.e. in overall, the average time the actors reach to the danger spot. The comparison of results of the table is also depicted in Figure 5-15.

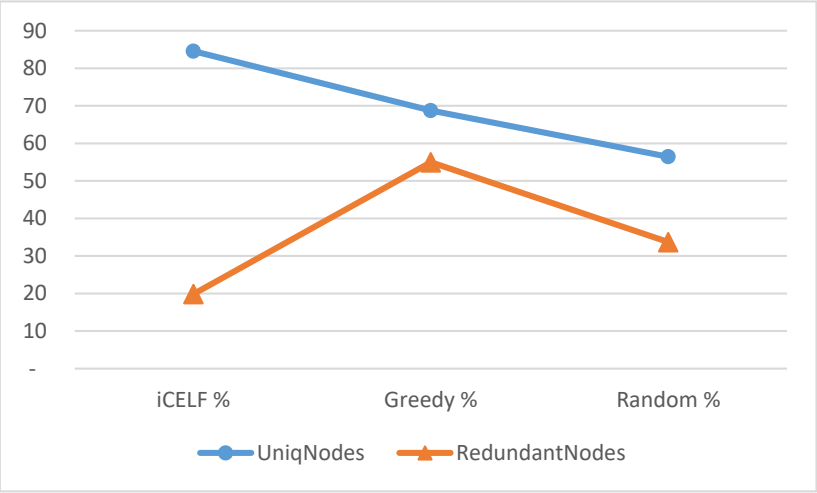


Figure 5-15: Comparison diagram of unique nodes and redundant nodes by the three algorithms

The diagram depicts that the proposed algorithm of this dissertation outperforms the two others, i.e. random and even greedy algorithms. As it is clear, the unique nodes covered by the iCELF are more than 85% of the total nodes; while they are less than 70% for greedy algorithm. Nevertheless, considering the minimum redundant nodes of the iCELF algorithm (less than 20%), while the greedy algorithm suggests 55% and the random

placement incurs more than 30%, it can be said that iCELF has optimized the placement and suggested the most efficient nodes for the actors.

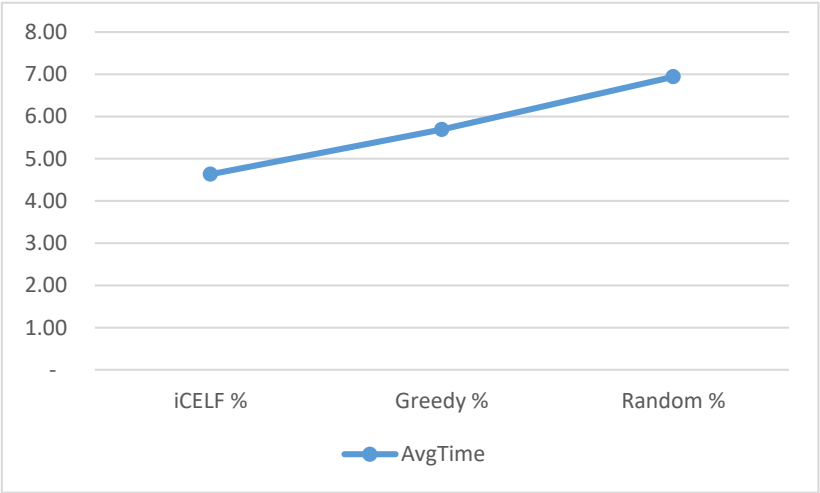


Figure 5-16: Comparison diagram of the average time for the placements suggested by the three algorithms

From another perspective, to compare the three algorithms, considering the third row of the Table 5-16, the average time the suggested placement incurs is compared in Figure 5-16.

5.1.2.4 High-risk regions versus high-loss regions

As it is explained in 4.2.4.2, 4.2.4.3, and 4.2.4.4, one hundred of simulation experiments are executed and the results are discussed in this section. The main assumption

of these hundred rounds is that some regions are assumed to be in high risk (probability of the danger) and high expected loss, i.e. huge amount of loss in case of danger occurrence. The results of this set of experiments are shown in Table 5-17.

Table 5-17: Comparison of the three algorithms by different measures

	iCELF	**Greedy**	**Random**
Unique Nodes	81.73	65.61	55.87
Redundant Nodes	20.50	61.69	33.50
Average Time	4.63	5.76	6.78

The values of the table cells are the average of the 100 rounds of simulation experiments. The first measure indicates how many unique nodes (i.e. regions) are covered by each of the algorithms, i.e. the number of the nodes with only one actor to support them in case of danger. The second measure, redundant nodes shows the number of nodes covered by more than one actor. It can be index of wastage of the resources or inefficiency of the actor placement. Obviously, the more redundant node means the less efficient the placement is. The third measure deals with the average service time, i.e. in overall, the average time the actors reach to the danger spot. The comparison of results of the table is

also depicted in Figure 5-17.

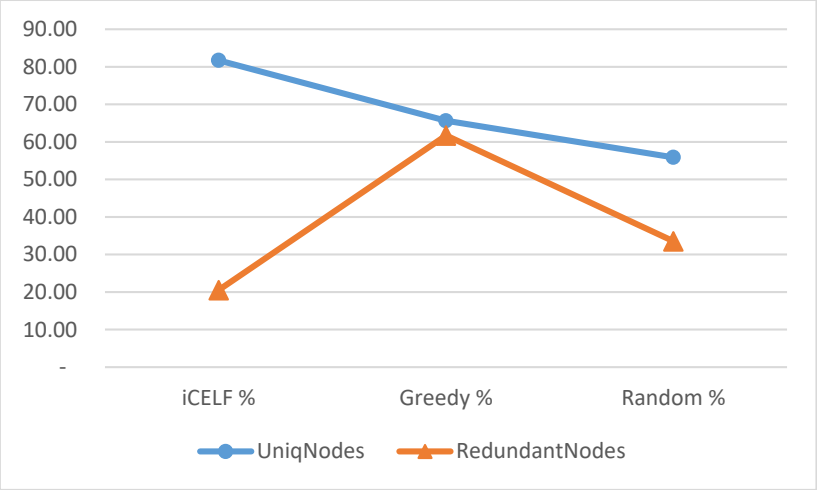


Figure 5-17: Comparison diagram of unique nodes and redundant nodes by the three algorithms

The diagram shows that the proposed algorithm of this dissertation outperforms the two others, i.e. random and the greedy algorithms. As it is clear, the unique nodes covered by the iCELF are more than 80% of the total nodes; while they are around 65% for the greedy algorithm. Nevertheless, considering the minimum redundant nodes of the iCELF algorithm (around 20%), while the greedy algorithm suggests 60% and the random placement incurs more than 30%, it can be said that iCELF has optimized the placement and suggested the most efficient nodes for the actors.

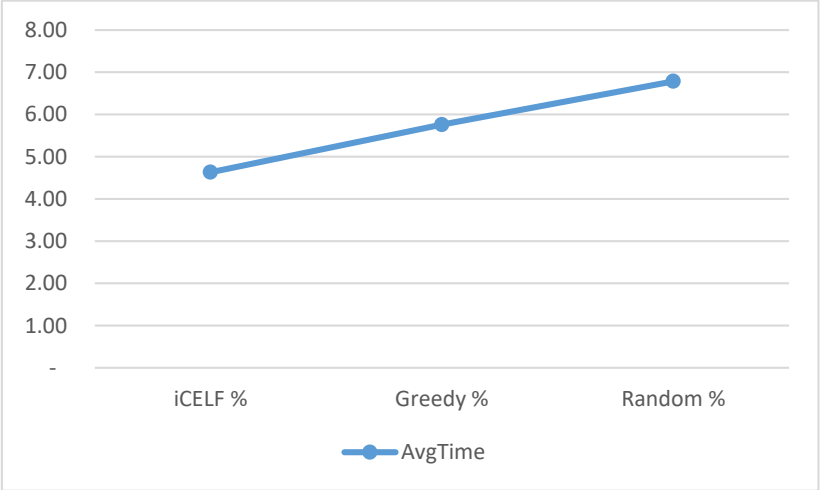


Figure 5-18: Comparison diagram of the average time for the placements suggested by the three algorithms

From another perspective, to compare the three algorithms, considering the third row of the Table 5-17, the average time the suggested placement incurs is compared in Figure 5-18

5.2 VM Placement Optimization Simulation

In this section, the simulation experiments and their results of the scenario that are detailed in 4.3 are explained. There were four situations that the simulation has been performed and their results are explained in the following sections.

5.2.1 Single Powerful Data Center

The first scenario is about the placement of VM on one DC. This includes the combinations of C(5,1) or $\binom{5}{1}$, i.e. 5 experiments. The results of the 5 experiments are illustrated in

Table 5-18 and compared in Figure 5-19.

Table 5-18: Detail results of the first scenario

code	average RT[4] (ms)	VM cost ($)	Traffic cost ($)	Total ($)
1028	614.29	518	249.99	767.99
1027	405.08	583.26	249.9	833.16
1026	309.56	388	214.2	602.2
1025	564.49	453.65	357	810.65
1024	314.72	388	178.5	566.5

[4] Response time

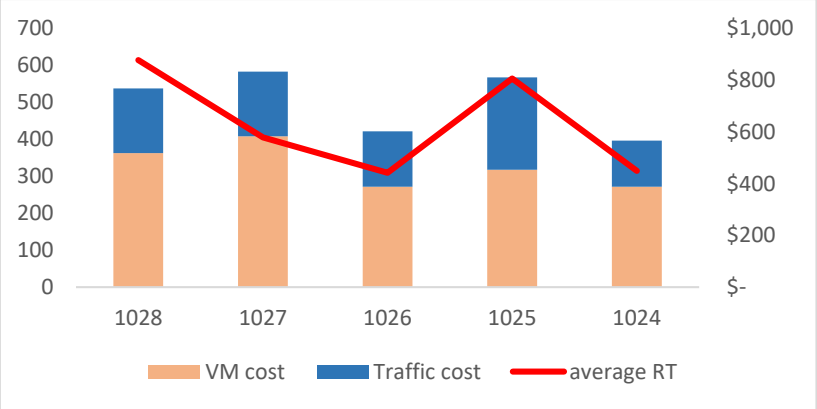


Figure 5-19: Comparison of the results of first scenario

5.2.2 Two Small Size Data Centers

The second scenario is deployment of VMs on two small-sized DC (A3 class or tier 3). This makes the combinations of C(5,2) or $\binom{5}{2}$, i.e. 10 experiments. The results of the 10 experiments are illustrated in Table 5-19 and compared in Figure 5-20.

Table 5-19: Detail results of the second scenario

code	average RT (ms)	VM cost ($)	Traffic cost ($)	Total ($)
1059	200.25	61.21	249.90	311.11
1058	259.28	50.41	229.52	279.93
1057	203.96	54.01	229.27	283.28
1056	541.48	54.01	340.20	394.21
1055	137.7	57.61	298.16	355.77
1054	203.1	46.80	275.51	322.31
1053	310.3	50.41	180.61	231.02
1052	124.63	54.01	208.63	262.64
1051	194.22	43.20	198.87	242.07
1050	286.23	46.80	212.39	259.19

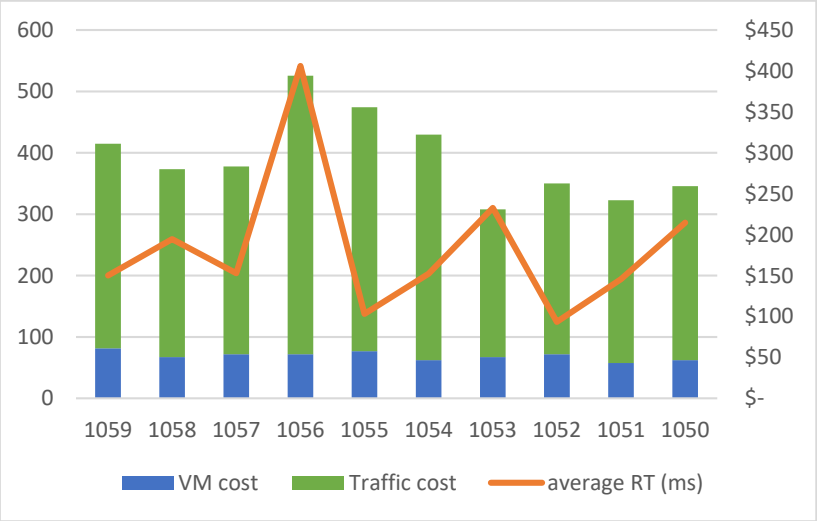


Figure 5-20: Comparison of the results of second scenario

5.2.3 Two Powerful Data Centers

As third scenario, the VMs are considered to be placed on two powerful DCs (A1 class or tier 1). The combinations of the scenario are the same as the previous scenario, i.e. 10 experiments. The results of the 10 experiments are illustrated in scenario Table 5-20 and compared in Figure 5-21.

Table 5-20: Detail results of the third scenario

code	average RT (ms)	VM cost ($)	Traffic cost ($)	Total ($)
1049	200.21	1,101.00	249.90	1,350.90
1048	259.28	907.30	229.52	1,136.82
1047	203.91	972.10	229.27	1,201.37
1046	541.48	972.10	340.20	1,312.30
1045	137.7	1,036.90	298.16	1,335.06
1044	203.1	842.49	275.50	1,117.99
1043	310.3	907.30	180.61	1,087.91
1042	124.58	972.10	208.63	1,180.73
1041	194.22	777.68	198.87	976.55
1040	286.23	842.49	212.39	1,054.88

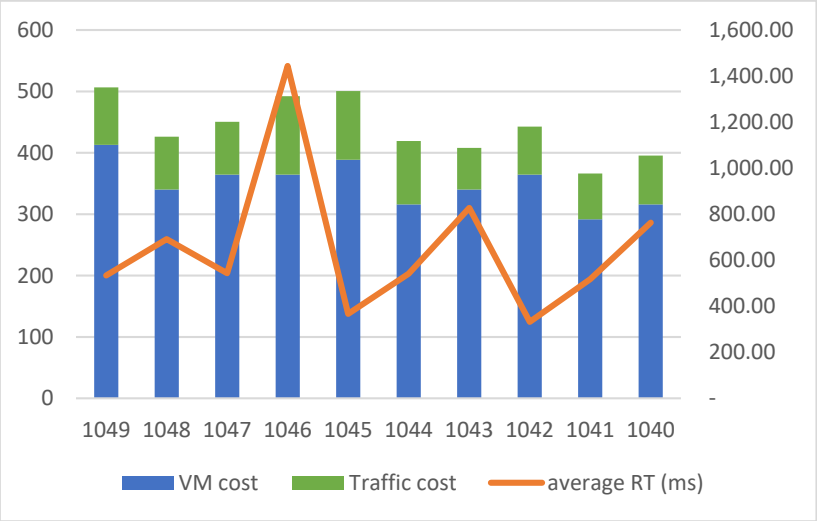


Figure 5-21: Comparison of the results of third scenario

5.2.4 Full Data Center Coverage

The last scenario is about deployment of five VMs on the five available DCs. This includes 4 experiments which are illustrated in Table 5-21 and compared in Figure 5-22.

Table 5-21: Results of the last scenario

No.	average RT (ms)	VM cost	Traffic cost	Total
1007	55.65	3,024.00	249.95	3,273.95
1008	55.67	2,333.00	249.95	2,582.95
1029	55.72	129.00	249.95	378.95
1030	55.66	561.00	249.95	810.95

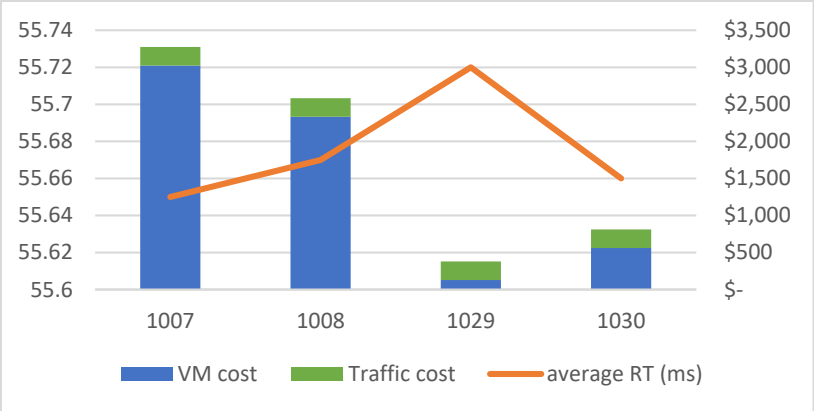


Figure 5-22: Comparison diagram of the results of the last scenario

As demonstrated in Figure 5-22, experiment # 1029 has the minimum overall cost, but maximum RT. Another point is the difference between RT of 1007 and 1008 that is only 0.02 *ms* and can be achieved spending $691. The cost-benefit analysis shows that the achievement does not worth at this cost.

5.3 Discussion

As explained in chapter 4 and 5, various simulation cases and scenarios are considered to evaluate the decision support tool proposed by this research. The first case was from the healthcare industry. Only two examples of it were considered in the simulation scenario, namely cardiac arrest and heart attack. Nevertheless, obviously, the

model can be applied for so many similar cases in which there are IoT sensors to detect a danger and there can be some IoT actors that can tackle the health danger.

The results of a variety of simulations prove that the tool can support the IoTSPs in two of its critical decisions, actor placement and VM placement. The first decision matters the places of the IoT actors while the second considers the infrastructure to support the proposed IoT system.

There are many real examples showing that the greater actor cost neither expands the total covered regions by the service, nor increases the population supported by the IoTSP. Hence, finding the best actor and VM placements requires complicated and time-consuming calculations. In the previous chapters, the proposed decision support tool was tested and evaluated with many IoT scenarios.

The result shows that the tool's solution outperforms other algorithms namely random selection and the greedy algorithm. Also in the second part, the tool finds out the optimum placement for the VMs. Finally, combining the result of the two parts of the tool, the IoTSP decides its actor placement and its VM placement. The near optimal solution provided by the tool helps the IoTSP utilize its resources in an optimum way.

The results of the experiments in multiple cases show that with fewer expenses,

IoTSP may enjoy higher service quality and consequently user satisfaction. This urges the IoT industry to utilize decision support tool to avoid budget and resource wastage.

The sensitivity analysis is performed on few simulation parameters to see the working of the tool in the variety of conditions. The distribution of the population and the distances among the regions are from the factors. The other parameter is the actor speed that may affect the results of the simulation. Performing the simulation experiments for two cases regarding the actor speed and four different situations considering the distribution types of population proved that in almost all cases, the tool provided better results than the other algorithms.

The experiments also shows that not necessarily increasing the cost of infrastructure and utilizing expensive data centers and VM classes may improve the user satisfaction. There are clear instances where the overall response time will not have big change by doing so. The results imply the fact that the geographical location of the users and their peak/off-peak times are effective factors that should be considered by IoTSP decision makers.

Another aspect to the problem is that utilizing the expensive DCs might affect the maximum response time. This might be interpreted that the expensive DCs are more reliable and will incur less risk for the service provider. Hence, the consequences show that

if the IoTSP intends to have a very reliable service for which the network delay cannot be accepted, they should consider high expenses of DCs. Otherwise, for the average purposes and normal users medium or even small size of DCs are seemed to be sufficient.

Chapter 6. Conclusion

The integration of IoT and Cloud computing is becoming a norm of future technology trends; hence, studying the issues of the integration is necessary for both industry and academia. In this work, we have addressed some of the economic issues for the IoT service providers that intend to offer particular services on Clouds.

The study shows that the emerging concept of IoT has a wide range of real applications to cover. This work tried to propose a solution for an abstract level of IoT services that many IoT Service Providers (IoTSPs) may offer. The industry of the application may vary from healthcare to the disaster recovery. In the general form, any application that involves a sudden occurrence requiring urgent reaction can be considered. To this end, smart sensors are required to detect the occurrence as well as smart devices (actors) to tackle the occurrence. There can be a wide range of occurrences, e.g. heart attacks or cardiac arrests, fires, fluids as well as some requests from the IoT customers through their wearable devices which should be addressed urgently

The main advantage of the proposed IoT system is the autonomy, because there is very limited time from the moment the occurrence arises until the reaction can address it. In other words, reaching too late to the spot of occurrence will not help the objective of the

system be achieved. For example, for the cases of heart attack, reaching later than a certain time cannot save the patient and all of the efforts will be useless. Alternatively, in some of the cases, reacting later than expected will bring fewer benefits; and this leads the service to be useless. Hence, if the autonomy of the IoT system let the reaction to be done without any human interference and the actor can reach in time to the spot, then this system is really worthy to be established.

This research is the first try to address the problems in this way. To make the system autonomous, a novel system architecture is designed. This architecture is an effort to eliminate the need for human decision-making. Therefore, the IoTSP can be sure that once any of the sensors report about an occurrence, the central system will realize and find the closest actor and command it to move immediately to reach to the danger spot.

Nevertheless, the main problem which this research put effort to address is that what are the optimum placements for the IoTSP actors and VMs? This is prerequisite for the above-mentioned advantage of the whole system. To address this, the two different placement problems, i.e. actor placement and VM placement are considered separately. For the first, the proposed heuristic algorithm, iCELF finds out a near optimal placement. The IoTSP may place the actors accordingly to ensure its optimum placement, in other words,

to achieve the highest possible population of the customers covered by the service and less possible time to react. The results of the simulation experiments showed that iCELF normally produces better placements in both senses.

From the other perspective, the second part of the tool can ensure the optimality of the VM placement. This may decrease the other great section of the IoTSP costs to the optimum level. The main advantage of the suggested VM placement is its high performance with less infrastructure expenditure. The other benefit of the proposed decision support tool is that IoTSP can assign importance to the cost parameters or performance parameters as input factors. In addition, the user can analyze the sensitivity of the results to the given weights and decide how to place its VMs.

In conclusion, this research has studied few possible methods to support the companies which intend to offer particular IoT-based services. Letting them utilize CloudIoT means to rely the IoT system on cloud computing to fulfill the shortcomings of IoT system. Then, to help the IoT service providers in their decisions, this research has reviewed some of the challenging points for them especially regarding economic and performance. Finding out the major decision-making points, two main questions, i.e. IoT actor placement optimization and VM placement optimization on multi Clouds were

considered and addressed. The developed decision support tool helps the IoTSP decision makers find out their optimum actor placement in a very short of time, similarly for the servers supporting the IoTSP in the form of Virtual Machines. Considering both of them, the IoTSP can ensure its highest efficiency with the minimum cost in both aspects.

The other conclusion is that the decision-making on this complicated and ever-changing environment is not a trivial problem to be solved easily. Therefor it definitely urges the IoTSP companies to utilize decision-making tools to choose their total system cost-optimally and efficiently. This is essential for the businesses who think of IoT services as their core. In other words, utilizing IoT technology is not enough to bring success to the business and they urge to consider these economic points well.

As a result, it can be said that the IoT industry should consider its business model and evaluates the economic perspectives that are involved in the success of the business. This requires comprehensive study to ensure that all of the success factors are counted and considered. Not only the technology perspective and factors, but also the economic and business related factors are to be considered well.

Here, another interesting filed is other application scenarios of the proposed model. One may suggest various applications in different industries to establish IoT-based services.

This is an emerging trend in different sections and many companies as well as research centers are working on it. They try to offer new solutions utilizing the IoT concepts and technology to serve users. The scale of the applications may vary from smart homes, to IoT-based security systems or smart cities. The application may cover the logistics and supply chain of a manufacturing company or a chain of retail shops. In addition, the service organizations may benefit from this. Basically, there is a huge potential to propose new systems based on IoT and all require this kind of decision support models and algorithms to help the IoT service providers to setup their system in an optimum way and enable them maintain their business profitable.

The future work of this research may also focus on other economic aspect of IoT and Cloud convergence more aiming the business to be more profitable for the IoT companies to leverage Clouds. These two components of the future internet (FI) may bring higher success if they merge and fulfill the shortcomings of each other.

Future researches may address the scenarios in the dynamic environment in which some of the factors may change over the time and IoTSP needs to reconsider and recheck the optimality of their current placement. The migration cost is another important point of view that should be considered in the dynamic environment where the change will make

the current placement not to be optimal anymore.

Moreover, it is suggested for the future researches to utilize the estimation methods to predict the upcoming trend and changes in the parameters. This will let the estimation to be more accurate and hence, the proposed placements to be more efficient.

Finally, it is predicted that the industry will emerge so many other application cases for IoT and the demand for further studies will arise more.

Chapter 7. Bibliography

(2015). Retrieved 2015 Dec, 2015, from https://azure.microsoft.com/en-us/pricing/.

(2015). Retrieved 2015 Dec, 2015, from https://aws.amazon.com/ec2/pricing/.

A Vouk, M. (2008). "Cloud computing–issues, research and implementations." CIT. Journal of Computing and Information Technology **16**(4): 235-246.

Ai, L., M. Tang and C. Fidge (2011). Resource Allocation and Scheduling of Multiple Composite Web Services in Cloud Computing Using Cooperative Coevolution Genetic Algorithm. Neural Information Processing. B.-L. Lu, L. Zhang and J. Kwok, Springer Berlin Heidelberg. **7063:** 258-267.

Alcaraz, C., P. Najera, J. Lopez and R. Roman (2010). Wireless sensor networks and the internet of things: Do we need a complete integration? 1st International Workshop on the Security of the Internet of Things (SecIoT'10).

Altmann, J. and M. M. Kashef (2014). "Cost model based service placement in federated hybrid clouds." Future Generation Computer Systems **41**: 79-90.

AO, N.-x., Y.-y. XU, D. HUANG, Y.-x. ZHAO and C.-j. CHEN (2013). "Cost-effective deployment of distributed cloud based on generic expense model." The Journal of China Universities of Posts and Telecommunications **20**(5): 9-29.

Armbrust, M., A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica and M. Zaharia (2009). Above the Clouds: A Berkeley View of Cloud Computing, University of California at Berkeley.

Asimakopoulou, E., S. Sotiriadis, N. Bessis, C. Dobre and V. Cristea (2013). "Centralized Micro-clouds: An Infrastructure for Service Distribution in

Collaborative Smart Devices." Procedia Computer Science **21**: 83-90.

Bianchini, R. and R. Rajamony (2004). "Power and energy management for server systems." IEEE Computer **37**(11): 68-74.

Botta, A., W. de Donato, V. Persico and A. Pescapé (2015). "Integration of Cloud computing and Internet of Things: A survey." Future Generation Computer Systems.

Boyd, S. and L. Vandenberghe (2004). Convex optimization, Cambridge university press.

Buyya, R. (2002). "Economic-based distributed resource management and scheduling for grid computing." arXiv preprint cs/0204048.

Buyya, R., D. Abramson and J. Giddy (2000). An Economy Driven Resource Management Architecture for Global Computational Power Grids. PDPTA.

Buyya, R., D. Abramson and S. Venugopal (2005). "The grid economy." Proceedings of the IEEE **93**(3): 698-714.

Buyya, R., C. S. Yeo and S. Venugopal (2008). Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on, Ieee.

Catrein, D. and C. QSC AG (2013). "Maintaining User Control While Storing and Processing Sensor Data in the Cloud." International Journal of Grid and High Performance Computing **5**(4): 97-112.

Chang, K.-D., C.-Y. Chen, J.-L. Chen and H.-C. Chao (2011). Internet of things and cloud computing for future internet. Security-Enriched Urban Computing and Smart Grid, Springer**:** 1-10.

Chang, R.-S., Y.-C. Chang and R.-C. Ye (2012). A Virtual Machine Scheduling Algorithm for Resource Cooperation in a Private Cloud. Computer Science and its Applications. S.-S. Yeo, Y. Pan, Y. S. Lee and H. B. Chang, Springer Netherlands. **203:** 207-215.

Chawla, C. and I. Chana (2015). "Day-ahead Pricing Model for Smart Cloud using Time Dependent Pricing."

Chawla, C. and I. Chana (2015). "Strategy-proof Pricing Approach for Cloud Market." arXiv preprint arXiv:1506.06648.

Chen, J., G. Soundararajan and C. Amza (2006). Autonomic provisioning of backend databases in dynamic content web servers. Autonomic Computing, 2006. ICAC'06. IEEE International Conference on, IEEE.

Chen, Y. and H. Hu (2013). "Internet of intelligent things and robot as a service." Simulation Modelling Practice and Theory **34**: 159-171.

Deelman, E., G. Singh, M. Livny, B. Berriman and J. Good (2008). The cost of doing science on the cloud: the montage example. Proceedings of the 2008 ACM/IEEE conference on Supercomputing, IEEE Press.

Diaz, C. O., H. Castro, M. Villamizar, J. E. Pecero and P. Bouvry (2013). Energy-aware vm allocation on an opportunistic cloud infrastructure. Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on, IEEE.

Distefano, S., G. Merlino and A. Puliafito (2012). Enabling the cloud of things. Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on, IEEE.

Dong, F. (2007). "A taxonomy of task scheduling algorithms in the Grid." Parallel Processing Letters **17**(04): 439-454.

Fox, G. C., S. Kamburugamuve and R. D. Hartman (2012). Architecture and measured characteristics of a cloud based internet of things. Collaboration Technologies and Systems (CTS), 2012 International Conference on, IEEE.

Goyal, R. (2015). An Efficient Auction Based Resource Allocation Algorithm in Cloud Computing, THAPAR UNIVERSITY PATIALA.

Grozev, N. and R. Buyya (2014). "Inter-Cloud architectures and application brokering: taxonomy and survey." Software: Practice and Experience **44**(3): 369-390.

Hao, W., I.-L. Yen and B. Thuraisingham (2009). Dynamic service and data migration in the clouds. Computer Software and Applications Conference, 2009. COMPSAC'09. 33rd Annual IEEE International, IEEE.

Hassanein, H. (2011). "Keynote I: Sensing and Identification in the Internet of Things Era." Procedia Computer Science **5**: 34-35.

Henze, M., S. Bereda, R. Hummen and K. Wehrle (2014). "SCSlib: Transparently Accessing Protected Sensor Data in the Cloud." Procedia Computer Science **37**: 370-375.

Henze, M., L. Hermerschmidt, D. Kerpen, R. Häußling, B. Rumpe and K. Wehrle (2015). "A comprehensive approach to privacy in the cloud-based Internet of Things." Future Generation Computer Systems.

Henze, M., R. Hummen, R. Matzutt and K. Wehrle (2014). A Trust Point-based Security Architecture for Sensor Data in the Cloud. Trusted Cloud Computing, Springer**:** 77-106.

Hilley, D. (2009). "Cloud computing: A taxonomy of platform and infrastructure-level offerings." Georgia Institute of Technology, Tech. Rep. GIT-CERCS-09-13.

Horri, A., M. Mozafari and G. Dastghaibyfard (2014). "Novel resource allocation algorithms to performance and energy efficiency in cloud computing." The Journal of Supercomputing **69**(3): 1445-1461.

Hummen, R., M. Henze, D. Catrein and K. Wehrle (2012). A Cloud design for user-controlled storage and processing of sensor data. Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on, IEEE.

Izakian, H., A. Abraham and B. T. Ladani (2010). "An auction method for resource allocation in computational grids." Future Generation Computer Systems **26**(2): 228-235.

Jiang, L., L. Da Xu, H. Cai, Z. Jiang, F. Bu and B. Xu (2014). "An IoT-oriented data storage framework in cloud computing platform." Industrial Informatics, IEEE Transactions on **10**(2): 1443-1451.

Jing Tai, P. and Y. Jun (2010). A Network-aware Virtual Machine Placement and Migration Approach in Cloud Computing. Grid and Cooperative Computing (GCC), 2010 9th International Conference on.

Kalra, S. and S. K. Sood (2015). "Secure authentication scheme for IoT and cloud servers." Pervasive and Mobile Computing **24**: 210-223.

Kessaci, Y., N. Melab and E.-G. Talbi (2014). "A multi-start local search heuristic for an energy efficient VMs assignment on top of the OpenNebula cloud manager." Future Generation Computer Systems **36**: 237-256.

Khuller, S., A. Moss and J. S. Naor (1999). "The budgeted maximum coverage problem." Information Processing Letters **70**(1): 39-45.

Kirkham, T., D. Armstrong, K. Djemame and M. Jiang (2014). "Risk driven Smart Home resource management using cloud services." Future Generation

Computer Systems **38**: 13-22.

Lazar, A. and N. Semret (1997). "Auctions for network resource sharing." Columbia Univ., TR: 468-497.

Le Vinh, T., S. Bouzefrane, J.-M. Farinone, A. Attar and B. P. Kennedy (2015). "Middleware to Integrate Mobile Devices, Sensors and Cloud Computing." Procedia Computer Science **52**: 234-243.

Lee, H., Y.-S. Jeong and H. Jang (2014). "Performance analysis based resource allocation for green cloud computing." The Journal of Supercomputing **69**(3): 1013-1026.

Leskovec, J., A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen and N. Glance (2007). Cost-effective outbreak detection in networks. Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM.

Li, A., X. Zong, S. Kandula, X. Yang and M. Zhang (2011). CloudProphet: towards application performance prediction in cloud. ACM SIGCOMM Computer Communication Review, ACM.

Li, H. and Z.-z. Xu (2013). Research on Business Model of Internet of Things Based on MOP. International Asia Conference on Industrial Engineering and Management Innovation (IEMI2012) Proceedings, Springer.

Li, M., S. Yu, Y. Zheng, K. Ren and W. Lou (2013). "Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption." Parallel and Distributed Systems, IEEE Transactions on **24**(1): 131-143.

Li, S., L. Da Xu and S. Zhao (2014). "The internet of things: a survey." Information Systems Frontiers **17**(2): 243-259.

Li, W., Y. Zhong, X. Wang and Y. Cao (2013). "Resource virtualization and service selection in cloud logistics." Journal of Network and Computer Applications **36**(6): 1696-1704.

Li, X., Z. Qian, R. Chi, B. Zhang and S. Lu (2012). Balancing resource utilization for continuous virtual machine requests in clouds. Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on, IEEE.

Li, Y., L. Zhuo and H. Shen (2013). An Efficient Resource Allocation Method for Multimedia Cloud Computing. Intelligence Science and Big Data Engineering. C. Sun, F. Fang, Z.-H. Zhou, W. Yang and Z.-Y. Liu, Springer Berlin Heidelberg. **8261:** 246-254.

Liu, J., X. Huang and J. K. Liu (2014). "Secure sharing of Personal Health Records in cloud computing: Ciphertext-Policy Attribute-Based Signcryption." Future Generation Computer Systems.

Lounis, A., A. Hadjidj, A. Bouabdallah and Y. Challal (2012). Secure and scalable cloud-based architecture for e-health wireless sensor networks. Computer communications and networks (ICCCN), 2012 21st international conference on, IEEE.

Lucas Simarro, J. L., R. Moreno-Vozmediano, R. S. Montero and I. M. Llorente (2011). Dynamic placement of virtual machines for cost optimization in multi-cloud environments. High Performance Computing and Simulation (HPCS), 2011 International Conference on, IEEE.

Lumpkins, W. (2013). "The Internet of Things Meets Cloud Computing [Standards Corner]." Consumer Electronics Magazine, IEEE **2**(2): 47-51.

Mahajan, K., A. Makroo and D. Dahiya (2013). "Round Robin with Server

Affinity: A VM Load Balancing Algorithm for Cloud Based Infrastructure." JIPS **9**(3): 379-394.

Malawski, M., G. Juve, E. Deelman and J. Nabrzyski (2012). Cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, IEEE Computer Society Press.

Mansouri, Y., A. N. Toosi and R. Buyya (2013). Brokering Algorithms for Optimizing the Availability and Cost of Cloud Storage Services. 2013 IEEE International Conference on Cloud Computing Technology and Science (CLOUDCOM '13), IEEE Computer Society.

Mathew, B. (2015). Introduction to HANA Smart Business and the Internet of Things. Beginning SAP Fiori, Springer**:** 281-322.

Mehdipour, F., H. Noori and B. Javadi Energy-Efficient Big Data Analytics in Datacenters. Advances in Computers, Elsevier.

Mell, P. and T. Grance (2009). "The NIST definition of cloud computing." National Institute of Standards and Technology **53**(6): 50.

Miorandi, D., S. Sicari, F. De Pellegrini and I. Chlamtac (2012). "Internet of things: Vision, applications and research challenges." Ad Hoc Networks **10**(7): 1497-1516.

Mohammad Mahdi Kashef, H. Y., Mehdi Keshavarz, Junseok Hwang (2016). Decision Support Tool for IoT Service Providers for Utilization of Multi Clouds. ICACT 2016. Phoenix Park, Korea.

Moon, S., J. Kim, T. Kim and J. Lee (2015). "Reverse Auction-based Resource Allocation Policy for Service Broker in Hybrid Cloud Environment." CLOUD COMPUTING 2015: 76.

Nan, X., H. Yifeng and G. Ling (2011). Optimal resource allocation for multimedia cloud based on queuing model. Multimedia Signal Processing (MMSP), 2011 IEEE 13th International Workshop on.

Olivier, F., G. Carlos and N. Florent (2015). "New Security Architecture for IoT Network." Procedia Computer Science **52**: 1028-1033.

Parwekar, P. (2011). From Internet of Things towards cloud of things. Computer and Communication Technology (ICCCT), 2011 2nd International Conference on, IEEE.

Patel, C. D. and A. J. Shah (2005). Cost model for planning, development and operation of a data center.

Persson, P. and O. Angelsmark (2015). "Calvin – Merging Cloud and IoT." Procedia Computer Science **52**: 210-217.

Poess, M. and R. O. Nambiar (2008). "Energy cost, the key challenge of today's data centers: a power consumption analysis of TPC-C results." Proceedings of the VLDB Endowment **1**(2): 1229-1240.

Pooja, B., M. M. Pai and M. P. Radhika (2014). A Dual Cloud Based Secure Environmental Parameter Monitoring System: A WSN Approach. Cloud Computing, Springer**:** 189-198.

Rao, B., P. Saluia, N. Sharma, A. Mittal and S. Sharma (2012). Cloud computing for Internet of Things & sensing based applications. Sensing Technology (ICST), 2012 Sixth International Conference on, IEEE.

Rochwerger, B., D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth and J. Cáceres (2009). "The reservoir model and architecture for open federated cloud computing." IBM Journal of Research and Development **53**(4): 4: 1-4: 11.

Singer, G., I. Livenson, M. Dumas, S. N. Srirama and U. Norbisrath (2010). Towards a Model for Cloud Computing Cost Estimation with Reserved Instances. Proc. of 2nd Int. ICST Conf. on Cloud Computing, CloudComp 2010.

Smith, J. E. and N. Ravi (2005). "The architecture of virtual machines." Computer **38**(5): 32-38.

Soldatos, J., M. Serrano and M. Hauswirth (2012). Convergence of utility computing with the internet-of-things. Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on, IEEE.

Song, F., D. Huang, H. Zhou, H. Zhang and I. You (2014). "An Optimization-Based Scheme for Efficient Virtual Machine Placement." International Journal of Parallel Programming **42**(5): 853-872.

Sotomayor, B., R. S. Montero, I. M. Llorente and I. Foster (2009). "Virtual infrastructure management in private and hybrid clouds." Internet Computing, IEEE **13**(5): 14-22.

Suciu, G., A. Vulpe, S. Halunga, O. Fratu, G. Todoran and V. Suciu (2013). Smart cities built on resilient cloud computing and secure internet of things. Control Systems and Computer Science (CSCS), 2013 19th International Conference on, IEEE.

Tan, L. and N. Wang (2010). Future internet: The internet of things. Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on, IEEE.

Tang, Q., S. K. Gupta and G. Varsamopoulos (2008). "Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach." Parallel and Distributed Systems, IEEE Transactions on **19**(11): 1458-1472.

Tian, W. D. and Y. D. Zhao (2014). Optimized Cloud Resource Management and Scheduling: Theories and Practices, Elsevier Science.

Toosi, A. N., K. Van Mechelen and R. Buyya (2014). "An Auction Mechanism for a Cloud Spot Market."

Van den Bossche, R., K. Vanmechelen and J. Broeckhove (2010). Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads. Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, IEEE.

Van den Bossche, R., K. Vanmechelen and J. Broeckhove (2011). Cost-Efficient Scheduling Heuristics for Deadline Constrained Workloads on Hybrid Clouds. Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on.

Van den Bossche, R., K. Vanmechelen and J. Broeckhove (2013). "Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds." Future Generation Computer Systems **29**(4): 973-985.

Verboven, S., P. Hellinckx, F. Arickx and J. Broeckhove (2008). Runtime prediction based grid scheduling of parameter sweep jobs. Asia-Pacific Services Computing Conference, 2008. APSCC'08. IEEE, IEEE.

Vermesan, O., P. Friess, P. Guillemin, S. Gusmeroli, H. Sundmaeker, A. Bassi, I. S. Jubert, M. Mazura, M. Harrison and M. Eisenhauer (2011). "Internet of things strategic research roadmap." O. Vermesan, P. Friess, P. Guillemin, S. Gusmeroli, H. Sundmaeker, A. Bassi, et al., Internet of Things: Global Technological and Societal Trends **1**: 9-52.

Vouk, M., S. Averitt, M. Bugaev, A. Kurth, A. Peeler, H. Shaffer, E. Sills, S. Stein and J. Thompson (2008). Powered by VCL-using virtual computing laboratory (VCL) technology to power cloud computing. Proceedings of the 2nd

International Conference on Virtual Computing (ICVCI).

Wang, L. and Y. Lu (2008). Efficient power management of heterogeneous soft real-time clusters. Real-Time Systems Symposium, 2008, IEEE.

Wang, L., G. Von Laszewski, J. Dayal, X. He, A. J. Younge and T. R. Furlani (2009). Towards thermal aware workload scheduling in a data center. Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on, IEEE.

Wang, X., Y. Wang and Y. Cui (2014). "A new multi-objective bi-level programming model for energy and locality aware multi-job scheduling in cloud computing." Future Generation Computer Systems **36**: 91-101.

Wei-Yu, L., L. Guan-Yu and W. Hung-Yu (2010). Dynamic Auction Mechanism for Cloud Resource Allocation. Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on.

Wenhong, T. (2008). Three Ways to Improve the Efficiency of Virtual/Cloud Computing Lab. Apperceiving Computing and Intelligence Analysis, 2008. ICACIA 2008. International Conference on.

Wolski, R., J. S. Plank, J. Brevik and T. Bryan (2001). "Analyzing market-based resource allocation strategies for the computational grid." International Journal of High Performance Computing Applications **15**(3): 258-281.

Woo, S. S. and J. Mirkovic (2014). "Optimal application allocation on multiple public clouds." Computer Networks.

Yang, D., X. Fang and G. Xue (2011). ESPN: Efficient server placement in probabilistic networks with budget constraint. INFOCOM, 2011 Proceedings IEEE, IEEE.

Zamanifar, K., N. Nasri and M. Nadimi-Shahraki (2012). Data-Aware

Virtual Machine Placement and Rate Allocation in Cloud Environment. Advanced Computing & Communication Technologies (ACCT), 2012 Second International Conference on, IEEE.

Zhao, L. and K. Sakurai (2013). Improving Cost-Efficiency through Failure-Aware Server Management and Scheduling in Cloud. Cloud Computing and Services Science, Springer**:** 23-38.

Zhou, J., T. Leppanen, E. Harjula, M. Ylianttila, T. Ojala, C. Yu, H. Jin and L. T. Yang (2013). Cloudthings: A common architecture for integrating the internet of things with cloud computing. Computer Supported Cooperative Work in Design (CSCWD), 2013 IEEE 17th International Conference on, IEEE.

Appendixes

Appendix A **The IoT-based healthcare simulation code**

The simulation explained in 4.2.3 and 5.1.1 is detailed here. It is written using the syntax of MATLAB R2016a and shown in

	Code
1	clc
2	clear all
3	SimRes1 = [0,0,0,0,0];
4	SimRes2 = [0,0,0,0,0];
5	SimRes3 = [0,0,0,0,0];
6	SimRes4 = [0,0,0,0];
7	SimRes5 = [0,0,0,0,0];
8	PopulationFactor =500000;
9	%NODES
10	TotalNodes=100;
11	%ACTORS
12	Type1Actors=10;
13	Type2Actors=10;
14	Type3Actors=10;
15	TotalActors=Type1Actors+Type2Actors+Type3Actors;
16	%Will be 100 for the final experiments
17	SimExp=100;
18	%Generating area network having nodes equal to 'TotalNodes'

```
FromNode=floor(1+rand(1,TotalNodes)*TotalNodes);
ToNode=floor(1+rand(1,TotalNodes)*TotalNodes);
for i=1:TotalNodes
    if(ToNode(i)==FromNode(i))
        ToNode(i)=floor(1+rand(1)*TotalNodes);
    end
end
FromNode=FromNode';
ToNode=ToNode';
for s=1:SimExp
%Generating random population Range Crowded: 10,000-500,000 &
%                        Retired: 1000-6000
%-------------------------------------------------------------------------%
% For all random nodes                                                    %
%-------------------------------------------------------------------------%
  %     NodePopulation=randi([1000    PopulationFactor],1,TotalNodes);
%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%
% For crowded and retarded nodes% with given pertentage of crowded nodes%
%-------------------------------------------------------------------------%
    %Crowdedpercen=50; % crowded %age
    %CrowdedNodes=ceil(TotalNodes*(Crowdedpercen/100));
    %Retnodes=TotalNodes-CrowdedNodes;
    %CrowdedNodesPopulation=randi([10000 PopulationFactor],1,CrowdedNodes);
    %RetnodesNodePopulation=randi([100    6000],1,Retnodes);
```

```
%NodePopulation=[CrowdedNodesPopulation,RetnodesNodePopulation];
%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%
% Colonies of crowded and colonies of retarded                            %
%-------------------------------------------------------------------------%
% NodePopulation=randi([1000    PopulationFactor],1,TotalNodes);
%NodesInCrowdedRegion= 2;
%NodesInRetardeddRegion= 2;
%CwodedNodeList    = [1, 7];
%RetardedNodeList= [5, 9];
%for i=1:length(CwodedNodeList)
%      for j=1:NodesInCrowdedRegion
%NodePopulation((CwodedNodeList(i)-1)+j)=randi([10000    PopulationFactor],1,1);
% end
%end
%for i=1:length(RetardedNodeList)
%       for j=1:NodesInRetardeddRegion
%              NodePopulation((RetardedNodeList(i)-1)+j)=randi([100    6000],1,1);
%       end
%end
%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%
% One big city surrounded by small cities                                 %
%-------------------------------------------------------------------------%
NodePopulation=randi([1000    PopulationFactor],1,TotalNodes);
```

```
69          BigCityList    = [1, 7, 20, 35, 40, 45, 68, 85, 90];
70          NumberofSmallCitiesSurrounding = [4, 3, 5, 3, 5, 7, 10, 3, 6];
71      for i=1:length(BigCityList)
72          NodePopulation(BigCityList(i))=randi([10000    PopulationFactor],1,1);
73          for j=1:NumberofSmallCitiesSurrounding(1,i)
74              NodePopulation(BigCityList(i)+j)=randi([100    6000],1,1);
75          end
76      end
77      %-------------------------------------------------------------------------%
78
79      TotalPopulation=sum(NodePopulation);
80      %Population to probability conversion
81      AttackProb=NodePopulation/PopulationFactor;
82      AttackProb=AttackProb';
83      %Distance for each pair of node range 100-500
84      NodeDistance=ceil(10+rand(1,TotalNodes)*1000);
85      NodeDistance=NodeDistance';
86
87      DistnceMatrix =[%from node, to node, distance
88                              FromNode, ToNode,      NodeDistance ];
89
90      ActorSpeed=[50, 100, 150];
91      ActorCost=[500, 1000, 1500];
92
93      ServiceTime=[(DistnceMatrix(:,3)/ActorSpeed(1)),(DistnceMatrix(:,3)/ActorSpeed(2)
```

	), (DistnceMatrix(:,3)/ActorSpeed(3))]; % Possible service time for each actor
94	
95	TargetTime=10; %Acceptable service time
96	iCELFSelectedNodes=0;
97	GreedySelectedNodes=0;
98	RandomSelectedNodes=0;
99	iCELFUniqueNodes=0;
100	GreedyUniqueNodes=0;
101	RandomUniqueNodes=0;
102	iCELFNodeList=0;
103	GreedyNodeList=0;
104	RandomNodeList=0;
105	iCELFPopulationServed=0;
106	GreedyPopulationServed=0;
107	RandomPopulationServed=0;
108	
109	%Making network connection
110	for i=1:length(DistnceMatrix(:,1))
111	ConnectionMatrix(i,i)=10;
112	if ((ServiceTime(1) < TargetTime) || (ServiceTime(2) < TargetTime) || (ServiceTime(3) < TargetTime))
113	ConnectionMatrix(DistnceMatrix(i,1),DistnceMatrix(i,2)) = DistnceMatrix(i,3);
114	ConnectionMatrix(DistnceMatrix(i,2),DistnceMatrix(i,1)) = DistnceMatrix(i,3);

```
115             else
116             ConnectionMatrix(DistnceMatrix(i,1),DistnceMatrix(i,2))=0;
117             ConnectionMatrix(DistnceMatrix(i,2),DistnceMatrix(i,1))=0;
118             end
119         end
120
121         SelectedNode=0;
122         SelectedNodeArray=0;
123         GreedyNodeArray=0;
124         CoverageFlag=zeros(1,length(ConnectionMatrix(1,:)));
125
126         for act=1:TotalActors
127             CoverageNodes=zeros(1,length(ConnectionMatrix(1,:)));
128             %Assesing coverable nodes for each point
129             for j=1:length(ConnectionMatrix(:,1))
130                 for i=1:length(SelectedNodeArray)
131                     if(j==SelectedNodeArray(i))
132                         ActionFlag=0;
133                     else
134                         ActionFlag=1;
135                     end
136                 end
137                 if(ActionFlag==1)
138                     for k=1:length(ConnectionMatrix(1,:))
139                         if(ConnectionMatrix(j,k)> 0 && CoverageFlag(k)==0)
```

```
                    CoverageNodes(j)=CoverageNodes(j)+1; %Finding number of nodes served by each location
                        end
                    end
                end
            end
            %Considering both distance and Attack probability
            DistanceFactor=(CoverageNodes/max(CoverageNodes))';
            AttackFactor=AttackProb;
            % If attack probility is not considered, remove "+ AttackFactor" from following equation only
            FinalFactor=DistanceFactor+AttackFactor;
        % FOR SIMPLE GREEDY ALGORITHM
        if (act==1)
            Greedy=FinalFactor;
            GreedyArray=sort(Greedy,'descend');
            for g=1:TotalActors
                for j=1:length(Greedy)
                    if(GreedyArray(g)==(Greedy(j)))
                        GreedyNodeArray=[GreedyNodeArray,j];
                    end
                end
            end
        end
        % FOR SIMPLE RANDOM ALGORITHM
```

```
163         RandonNode=ceil(rand(TotalActors, 1)*TotalNodes)';
164
165
166         %iCELF: Find the node with max feasible connections
167           ComapreFactor=0;
168           for j=1:length(CoverageNodes)
169                 if FinalFactor(j) > ComapreFactor
170                       SelectedNode=j;
171                       ComapreFactor=FinalFactor(j);
172                 end
173           end
174                SelectedNodeArray=[SelectedNodeArray,SelectedNode];
175
176           for j=1:length(ConnectionMatrix(1,:))
177                 if (ConnectionMatrix(SelectedNode,j)>0)
178                       CoverageFlag(j)=1;
179                       CoverageFlag(SelectedNode)=1;
180                 end
181           end
182     end
183
184     %Final RESULTS
185     iCELFAlgoNodes=SelectedNodeArray(2:length(SelectedNodeArray));
186     GreedyAlgoNodes=GreedyNodeArray(2:length(GreedyNodeArray));
187     RandomAlgoNodes=RandonNode;
```

```
188
189     %finding unique number of nodes covered by each algo
190     for i=1:TotalActors
191         for j=1:TotalNodes
192             if(ConnectionMatrix(j,iCELFAlgoNodes(i))> 0)
193                 iCELFSelectedNodes=[iCELFSelectedNodes,j];
194             end
195             if(ConnectionMatrix(j,GreedyAlgoNodes(i))> 0)
196                 GreedySelectedNodes=[GreedySelectedNodes,j];
197             end
198             if(ConnectionMatrix(j,RandomAlgoNodes(i))> 0)
199                 RandomSelectedNodes=[RandomSelectedNodes,j];
200             end
201         end
202     end
203
204     iCELFNodes=sort(iCELFSelectedNodes);
205     GreedyNodes=sort(GreedySelectedNodes);
206     RandomNodes=sort(RandomSelectedNodes);
207
208     iCELFAllNodes=length(iCELFNodes);
209     GreedyAllNodes=length(GreedyNodes);
210     RandomAllNodes=length(RandomNodes);
211
212     for i=1:length(iCELFSelectedNodes)-1
```

```
213             if(iCELFNodes(i)~=iCELFNodes(i+1))
214                 iCELFUniqueNodes=iCELFUniqueNodes+1;
215                 iCELFNodeList=[iCELFNodeList,iCELFNodes(i+1)];
216             end
217         end
218         iCELFNodeList;
219         for i=1:length(GreedySelectedNodes)-1
220             if(GreedyNodes(i)~=GreedyNodes(i+1))
221                 GreedyUniqueNodes=GreedyUniqueNodes+1;
222                GreedyNodeList=[GreedyNodeList,GreedyNodes(i+1)];
223             end
224         end
225         GreedyNodeList;
226         for i=1:length(RandomSelectedNodes)-1
227             if(RandomNodes(i)~=RandomNodes(i+1))
228                 RandomUniqueNodes=RandomUniqueNodes+1;
229               RandomNodeList=[RandomNodeList,RandomNodes(i+1)];
230             end
231         end
232         iCELFUniqueNodes;
233         GreedyUniqueNodes;
234         RandomUniqueNodes;
235
236         iCELFRedNodes=iCELFAllNodes-iCELFUniqueNodes;
237         GreedyRedNodes=GreedyAllNodes-GreedyUniqueNodes;
```

```
238 RandomRedRedNodes=RandomAllNodes-RandomUniqueNodes;
239
240 for i=2:length(iCELFNodeList)
241     iCELFPopulationServed
    =iCELFPopulationServed+NodePopulation(iCELFNodeList(i));
242 end
243 for i=2:length(GreedyNodeList)
244     GreedyPopulationServed
    =GreedyPopulationServed+NodePopulation(GreedyNodeList(i));
245 end
246 for i=2:length(RandomNodeList)
247     RandomPopulationServed
    =RandomPopulationServed+NodePopulation(RandomNodeList(i));
248 end
249
250 %For average service time
251 for i=2:length(iCELFNodeList)
252     iCELFTime1(i) =NodeDistance(iCELFNodeList(i))/ActorSpeed(1);
253     iCELFTime2(i) =NodeDistance(iCELFNodeList(i))/ActorSpeed(2);
254     iCELFTime3(i) =NodeDistance(iCELFNodeList(i))/ActorSpeed(3);
255 end
256 iCELFTime1=sort(iCELFTime1);
257 iCELFTime2=sort(iCELFTime2);
258 iCELFTime3=sort(iCELFTime3);
259
```

```
260     for i=2:length(iCELFNodeList)
261         a=ceil(length(iCELFNodeList)/(TotalActors/Type1Actors));
262         b=ceil(length(iCELFNodeList)/(TotalActors/Type2Actors));
263         c=length(iCELFNodeList)-(a+b);
264
265         iCELFTime
    =sum(iCELFTime1(1:a))+sum(iCELFTime2(a+1:a+b))+sum(iCELFTime3(a+b+1:a+
    b+c));
266         iCELFAverageTime=iCELFTime/length(iCELFNodeList);
267     end
268
269     %.....................
270     for i=2:length(GreedyNodeList)
271         GreedyTime1(i) =NodeDistance(GreedyNodeList(i))/ActorSpeed(1);
272         GreedyTime2(i) =NodeDistance(GreedyNodeList(i))/ActorSpeed(2);
273         GreedyTime3(i) =NodeDistance(GreedyNodeList(i))/ActorSpeed(3);
274     end
275     GreedyTime1=sort(GreedyTime1);
276     GreedyTime2=sort(GreedyTime2);
277     GreedyTime3=sort(GreedyTime3);
278
279     for i=2:length(GreedyNodeList)
280         a=ceil(length(GreedyNodeList)/(TotalActors/Type1Actors));
281         b=ceil(length(GreedyNodeList)/(TotalActors/Type2Actors));
282         c=length(GreedyNodeList)-(a+b);
```

```
283
284         GreedyTime
    =sum(GreedyTime1(1:a))+sum(GreedyTime2(a+1:a+b))+sum(GreedyTime3(a+b+1:a
    +b+c));
285         GreedyAverageTime=iCELFTime/length(GreedyNodeList);
286     end
287
288
289     %.....
290     for i=2:length(RandomNodeList)
291         RandomTime1(i) =NodeDistance(RandomNodeList(i))/ActorSpeed(1);
292         RandomTime2(i) =NodeDistance(RandomNodeList(i))/ActorSpeed(2);
293         RandomTime3(i) =NodeDistance(RandomNodeList(i))/ActorSpeed(3);
294     end
295     RandomTime1=sort(RandomTime1);
296     RandomTime2=sort(RandomTime2);
297     RandomTime3=sort(RandomTime3);
298
299     for i=2:length(RandomNodeList)
300         a=ceil(length(RandomNodeList) /(TotalActors/Type1Actors));
301         b=ceil(length(RandomNodeList) /(TotalActors/Type2Actors));
302         c=length(RandomNodeList)-(a+b);
303
304         RandomTime
    =sum(RandomTime1(1:a))+sum(RandomTime2(a+1:a+b))+sum(RandomTime3(a+b+
```

	1:a+b+c));
305	RandomAverageTime =iCELFTime/length(RandomNodeList);
306	end
307	
308	MaxReward = TotalNodes/((Type1Actors*ActorCost(1))+(Type2Actors*ActorCost(2))+(Type3Actors*ActorCost(3)));
309	iCELFReward = (iCELFUniqueNodes /((Type1Actors*ActorCost(1))+(Type2Actors*ActorCost(2))+(Type3Actors*ActorCost(3))))/MaxReward;
310	GreedyReward = (GreedyUniqueNodes /((Type1Actors*ActorCost(1))+(Type2Actors*ActorCost(2))+(Type3Actors*ActorCost(3))))/MaxReward;
311	RandomReward = (RandomUniqueNodes /((Type1Actors*ActorCost(1))+(Type2Actors*ActorCost(2))+(Type3Actors*ActorCost(3))))/MaxReward;
312	MaxReward = (TotalNodes/((Type1Actors*ActorCost(1))+(Type2Actors*ActorCost(2))+(Type3Actors*ActorCost(3))))/MaxReward;
313	
314	
315	Res1_UniqueNodes = [s,iCELFUniqueNodes, GreedyUniqueNodes, RandomUniqueNodes, TotalNodes];
316	Res2_RedNodes = [s,iCELFRedNodes, GreedyRedNodes, RandomRedRedNodes, TotalNodes];

```
317  Res3_PopServed      = [s,iCELFPopulationServed, GreedyPopulationServed,
     RandomPopulationServed, TotalPopulation];
318  Res4_AvgTime        = [s,iCELFAverageTime, GreedyAverageTime,
     RandomAverageTime];
319  Res5_Reward         = [s,iCELFReward, GreedyReward, RandomReward,
     MaxReward];
320
321  SimRes1 = [SimRes1; Res1_UniqueNodes];
322  SimRes2 = [SimRes2; Res2_RedNodes];
323  SimRes3 = [SimRes3; Res3_PopServed];
324  SimRes4 = [SimRes4; Res4_AvgTime];
325  SimRes5 = [SimRes5; Res5_Reward];
326  end
327  SimRes1
328  SimRes2
329  SimRes3
330  SimRes4
331  SimRes5
332  %iCELFPopulationServed
333  %GreedyPopulationServed
334  %RandomPopulationServed
```

Figure 8-1.

Code

```
1   clc
2   clear all
3   SimRes1 = [0,0,0,0,0];
4   SimRes2 = [0,0,0,0,0];
5   SimRes3 = [0,0,0,0,0];
6   SimRes4 = [0,0,0,0];
7   SimRes5 = [0,0,0,0,0];
8   PopulationFactor =500000;
9   %NODES
10  TotalNodes=100;
11  %ACTORS
12  Type1Actors=10;
13  Type2Actors=10;
14  Type3Actors=10;
15  TotalActors=Type1Actors+Type2Actors+Type3Actors;
16  %Will be 100 for the final experiments
17  SimExp=100;
18  %Generating area network having nodes equal to 'TotalNodes'
19  FromNode=floor(1+rand(1,TotalNodes)*TotalNodes);
20  ToNode=floor(1+rand(1,TotalNodes)*TotalNodes);
21  for i=1:TotalNodes
22      if(ToNode(i)==FromNode(i))
23          ToNode(i)=floor(1+rand(1)*TotalNodes);
24      end
25  end
```

```
FromNode=FromNode';
ToNode=ToNode';
for s=1:SimExp
%Generating random population Range Crowded: 10,000-500,000 &
%                          Retired: 1000-6000
%-------------------------------------------------------------------------%
% For all random nodes                                                    %
%-------------------------------------------------------------------------%
  %     NodePopulation=randi([1000    PopulationFactor],1,TotalNodes);
%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%
% For crowded and retarded nodes% with given pertentage of crowded nodes%
%-------------------------------------------------------------------------%
      %Crowdedpercen=50; % crowded %age
      %CrowdedNodes=ceil(TotalNodes*(Crowdedpercen/100));
      %Retnodes=TotalNodes-CrowdedNodes;
      %CrowdedNodesPopulation=randi([10000 PopulationFactor],1,CrowdedNodes);
      %RetnodesNodePopulation=randi([100    6000],1,Retnodes);
      %NodePopulation=[CrowdedNodesPopulation,RetnodesNodePopulation];
%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%
% Colonies of crowded and colonies of retarded                            %
%-------------------------------------------------------------------------%
  % NodePopulation=randi([1000    PopulationFactor],1,TotalNodes);
    %NodesInCrowdedRegion= 2;
```

```
51          %NodesInRetardeddRegion= 2;
52          %CwodedNodeList    = [1, 7];
53          %RetardedNodeList= [5, 9];
54      %for i=1:length(CwodedNodeList)
55      %      for j=1:NodesInCrowdedRegion
56      %NodePopulation((CwodedNodeList(i)-1)+j)=randi([10000    PopulationFactor],1,1);
57            % end
58      %end
59      %for i=1:length(RetardedNodeList)
60      %       for j=1:NodesInRetardeddRegion
61      %              NodePopulation((RetardedNodeList(i)-1)+j)=randi([100    6000],1,1);
62      %       end
63      %end
64      %-------------------------------------------------------------------------%
65      %-------------------------------------------------------------------------%
66      % One big city surrounded by small cities                                 %
67      %-------------------------------------------------------------------------%
68          NodePopulation=randi([1000    PopulationFactor],1,TotalNodes);
69          BigCityList    = [1, 7, 20, 35, 40, 45, 68, 85, 90];
70          NumberofSmallCitiesSurrounding = [4, 3, 5, 3, 5, 7, 10, 3, 6];
71      for i=1:length(BigCityList)
72              NodePopulation(BigCityList(i))=randi([10000    PopulationFactor],1,1);
73              for j=1:NumberofSmallCitiesSurrounding(1,i)
74                      NodePopulation(BigCityList(i)+j)=randi([100    6000],1,1);
75              end
```

```
76  end
77  %-------------------------------------------------------------------------%
78
79  TotalPopulation=sum(NodePopulation);
80  %Population to probability conversion
81  AttackProb=NodePopulation/PopulationFactor;
82  AttackProb=AttackProb';
83  %Distance for each pair of node range 100-500
84  NodeDistance=ceil(10+rand(1,TotalNodes)*1000);
85  NodeDistance=NodeDistance';
86
87  DistnceMatrix =[%from node, to node, distance
88                                  FromNode, ToNode,      NodeDistance ];
89
90  ActorSpeed=[50, 100, 150];
91  ActorCost=[500, 1000, 1500];
92
93  ServiceTime=[(DistnceMatrix(:,3)/ActorSpeed(1)),(DistnceMatrix(:,3)/ActorSpeed(2)
    ), (DistnceMatrix(:,3)/ActorSpeed(3))];    % Possible service time for each actor
94
95  TargetTime=10; %Acceptable service time
96  iCELFSelectedNodes=0;
97  GreedySelectedNodes=0;
98  RandomSelectedNodes=0;
99  iCELFUniqueNodes=0;
```

```
100     GreedyUniqueNodes=0;
101     RandomUniqueNodes=0;
102     iCELFNodeList=0;
103     GreedyNodeList=0;
104     RandomNodeList=0;
105     iCELFPopulationServed=0;
106     GreedyPopulationServed=0;
107     RandomPopulationServed=0;
108
109     %Making network connection
110     for i=1:length(DistnceMatrix(:,1))
111         ConnectionMatrix(i,i)=10;
112         if ((ServiceTime(1) < TargetTime) || (ServiceTime(2) < TargetTime) ||
        (ServiceTime(3) < TargetTime))
113             ConnectionMatrix(DistnceMatrix(i,1),DistnceMatrix(i,2)) =
        DistnceMatrix(i,3);
114             ConnectionMatrix(DistnceMatrix(i,2),DistnceMatrix(i,1)) =
        DistnceMatrix(i,3);
115         else
116         ConnectionMatrix(DistnceMatrix(i,1),DistnceMatrix(i,2))=0;
117         ConnectionMatrix(DistnceMatrix(i,2),DistnceMatrix(i,1))=0;
118         end
119     end
120
121     SelectedNode=0;
```

```
SelectedNodeArray=0;
GreedyNodeArray=0;
CoverageFlag=zeros(1,length(ConnectionMatrix(1,:)));

for act=1:TotalActors
    CoverageNodes=zeros(1,length(ConnectionMatrix(1,:)));
    %Assesing coverable nodes for each point
    for j=1:length(ConnectionMatrix(:,1))
        for i=1:length(SelectedNodeArray)
            if(j==SelectedNodeArray(i))
                ActionFlag=0;
            else
                ActionFlag=1;
            end
        end
        if(ActionFlag==1)
            for k=1:length(ConnectionMatrix(1,:))
                if(ConnectionMatrix(j,k)> 0 && CoverageFlag(k)==0)
                CoverageNodes(j)=CoverageNodes(j)+1; %Finding number of nodes served by each location
                end
            end
        end
    end
    %Considering both distance and Attack probability
```

```
146         DistanceFactor=(CoverageNodes/max(CoverageNodes))';
147         AttackFactor=AttackProb;
148         % If attack probility is not considered, remove "+ AttackFactor" from following
        equation only
149         FinalFactor=DistanceFactor+AttackFactor;
150       % FOR SIMPLE GREEDY ALGORITHM
151       if (act==1)
152             Greedy=FinalFactor;
153             GreedyArray=sort(Greedy,'descend');
154             for g=1:TotalActors
155                 for j=1:length(Greedy)
156                     if(GreedyArray(g)==(Greedy(j)))
157                         GreedyNodeArray=[GreedyNodeArray,j];
158                     end
159                 end
160             end
161       end
162       % FOR SIMPLE RANDOM ALGORITHM
163       RandonNode=ceil(rand(TotalActors, 1)*TotalNodes)';
164
165
166       %iCELF: Find the node with max feasible connections
167         ComapreFactor=0;
168         for j=1:length(CoverageNodes)
169             if FinalFactor(j) > ComapreFactor
```

```
170                         SelectedNode=j;
171                         ComapreFactor=FinalFactor(j);
172                     end
173                 end
174                 SelectedNodeArray=[SelectedNodeArray,SelectedNode];
175
176                 for j=1:length(ConnectionMatrix(1,:))
177                     if (ConnectionMatrix(SelectedNode,j)>0)
178                         CoverageFlag(j)=1;
179                         CoverageFlag(SelectedNode)=1;
180                     end
181                 end
182             end
183
184             %Final RESULTS
185             iCELFAlgoNodes=SelectedNodeArray(2:length(SelectedNodeArray));
186             GreedyAlgoNodes=GreedyNodeArray(2:length(GreedyNodeArray));
187             RandomAlgoNodes=RandonNode;
188
189             %finding unique number of nodes covered by each algo
190             for i=1:TotalActors
191                 for j=1:TotalNodes
192                     if(ConnectionMatrix(j,iCELFAlgoNodes(i))> 0)
193                         iCELFSelectedNodes=[iCELFSelectedNodes,j];
194                     end
```

```
195                     if(ConnectionMatrix(j,GreedyAlgoNodes(i))> 0)
196                         GreedySelectedNodes=[GreedySelectedNodes,j];
197                     end
198                     if(ConnectionMatrix(j,RandomAlgoNodes(i))> 0)
199                         RandomSelectedNodes=[RandomSelectedNodes,j];
200                     end
201                 end
202             end
203
204             iCELFNodes=sort(iCELFSelectedNodes);
205             GreedyNodes=sort(GreedySelectedNodes);
206             RandomNodes=sort(RandomSelectedNodes);
207
208             iCELFAllNodes=length(iCELFNodes);
209             GreedyAllNodes=length(GreedyNodes);
210             RandomAllNodes=length(RandomNodes);
211
212             for i=1:length(iCELFSelectedNodes)-1
213                 if(iCELFNodes(i)~=iCELFNodes(i+1))
214                     iCELFUniqueNodes=iCELFUniqueNodes+1;
215                     iCELFNodeList=[iCELFNodeList,iCELFNodes(i+1)];
216                 end
217             end
218             iCELFNodeList;
219             for i=1:length(GreedySelectedNodes)-1
```

```
220             if(GreedyNodes(i)~=GreedyNodes(i+1))
221                 GreedyUniqueNodes=GreedyUniqueNodes+1;
222                 GreedyNodeList=[GreedyNodeList,GreedyNodes(i+1)];
223             end
224     end
225     GreedyNodeList;
226     for i=1:length(RandomSelectedNodes)-1
227             if(RandomNodes(i)~=RandomNodes(i+1))
228                 RandomUniqueNodes=RandomUniqueNodes+1;
229               RandomNodeList=[RandomNodeList,RandomNodes(i+1)];
230             end
231     end
232     iCELFUniqueNodes;
233     GreedyUniqueNodes;
234     RandomUniqueNodes;
235
236     iCELFRedNodes=iCELFAllNodes-iCELFUniqueNodes;
237     GreedyRedNodes=GreedyAllNodes-GreedyUniqueNodes;
238     RandomRedRedNodes=RandomAllNodes-RandomUniqueNodes;
239
240     for i=2:length(iCELFNodeList)
241             iCELFPopulationServed
        =iCELFPopulationServed+NodePopulation(iCELFNodeList(i));
242     end
243     for i=2:length(GreedyNodeList)
```

```
244         GreedyPopulationServed
    =GreedyPopulationServed+NodePopulation(GreedyNodeList(i));
245     end
246     for i=2:length(RandomNodeList)
247         RandomPopulationServed
    =RandomPopulationServed+NodePopulation(RandomNodeList(i));
248     end
249
250     %For average service time
251     for i=2:length(iCELFNodeList)
252         iCELFTime1(i) =NodeDistance(iCELFNodeList(i))/ActorSpeed(1);
253         iCELFTime2(i) =NodeDistance(iCELFNodeList(i))/ActorSpeed(2);
254         iCELFTime3(i) =NodeDistance(iCELFNodeList(i))/ActorSpeed(3);
255     end
256     iCELFTime1=sort(iCELFTime1);
257     iCELFTime2=sort(iCELFTime2);
258     iCELFTime3=sort(iCELFTime3);
259
260     for i=2:length(iCELFNodeList)
261         a=ceil(length(iCELFNodeList)/(TotalActors/Type1Actors));
262         b=ceil(length(iCELFNodeList)/(TotalActors/Type2Actors));
263         c=length(iCELFNodeList)-(a+b);
264
265         iCELFTime
    =sum(iCELFTime1(1:a))+sum(iCELFTime2(a+1:a+b))+sum(iCELFTime3(a+b+1:a+
```

```
      b+c));
266         iCELFAverageTime=iCELFTime/length(iCELFNodeList);
267   end
268
269   %.....................
270   for i=2:length(GreedyNodeList)
271         GreedyTime1(i) =NodeDistance(GreedyNodeList(i))/ActorSpeed(1);
272         GreedyTime2(i) =NodeDistance(GreedyNodeList(i))/ActorSpeed(2);
273         GreedyTime3(i) =NodeDistance(GreedyNodeList(i))/ActorSpeed(3);
274   end
275   GreedyTime1=sort(GreedyTime1);
276   GreedyTime2=sort(GreedyTime2);
277   GreedyTime3=sort(GreedyTime3);
278
279   for i=2:length(GreedyNodeList)
280         a=ceil(length(GreedyNodeList)/(TotalActors/Type1Actors));
281         b=ceil(length(GreedyNodeList)/(TotalActors/Type2Actors));
282         c=length(GreedyNodeList)-(a+b);
283
284         GreedyTime
      =sum(GreedyTime1(1:a))+sum(GreedyTime2(a+1:a+b))+sum(GreedyTime3(a+b+1:a
      +b+c));
285         GreedyAverageTime=iCELFTime/length(GreedyNodeList);
286   end
287
```

```
288
289     %.....
290     for i=2:length(RandomNodeList)
291         RandomTime1(i) =NodeDistance(RandomNodeList(i))/ActorSpeed(1);
292         RandomTime2(i) =NodeDistance(RandomNodeList(i))/ActorSpeed(2);
293         RandomTime3(i) =NodeDistance(RandomNodeList(i))/ActorSpeed(3);
294     end
295     RandomTime1=sort(RandomTime1);
296     RandomTime2=sort(RandomTime2);
297     RandomTime3=sort(RandomTime3);
298
299     for i=2:length(RandomNodeList)
300         a=ceil(length(RandomNodeList) /(TotalActors/Type1Actors));
301         b=ceil(length(RandomNodeList) /(TotalActors/Type2Actors));
302         c=length(RandomNodeList)-(a+b);
303
304         RandomTime
    =sum(RandomTime1(1:a))+sum(RandomTime2(a+1:a+b))+sum(RandomTime3(a+b+
    1:a+b+c));
305         RandomAverageTime =iCELFTime/length(RandomNodeList);
306     end
307
308     MaxReward                                                    =
    TotalNodes/((Type1Actors*ActorCost(1))+(Type2Actors*ActorCost(2))+(Type3Actor
    s*ActorCost(3)));
```

```
309  iCELFReward    = (iCELFUniqueNodes
     /((Type1Actors*ActorCost(1))+(Type2Actors*ActorCost(2))+(Type3Actors*ActorCo
     st(3))))/MaxReward;
310  GreedyReward = (GreedyUniqueNodes
     /((Type1Actors*ActorCost(1))+(Type2Actors*ActorCost(2))+(Type3Actors*ActorCo
     st(3))))/MaxReward;
311  RandomReward = (RandomUniqueNodes
     /((Type1Actors*ActorCost(1))+(Type2Actors*ActorCost(2))+(Type3Actors*ActorCo
     st(3))))/MaxReward;
312  MaxReward       =
     (TotalNodes/((Type1Actors*ActorCost(1))+(Type2Actors*ActorCost(2))+(Type3Acto
     rs*ActorCost(3))))/MaxReward;
313
314
315  Res1_UniqueNodes = [s,iCELFUniqueNodes, GreedyUniqueNodes,
     RandomUniqueNodes, TotalNodes];
316  Res2_RedNodes       = [s,iCELFRedNodes,       GreedyRedNodes,
     RandomRedRedNodes, TotalNodes];
317  Res3_PopServed      = [s,iCELFPopulationServed, GreedyPopulationServed,
     RandomPopulationServed, TotalPopulation];
318  Res4_AvgTime        = [s,iCELFAverageTime, GreedyAverageTime,
     RandomAverageTime];
319  Res5_Reward         = [s,iCELFReward, GreedyReward, RandomReward,
     MaxReward];
320
```

```
SimRes1 = [SimRes1; Res1_UniqueNodes];
SimRes2 = [SimRes2; Res2_RedNodes];
SimRes3 = [SimRes3; Res3_PopServed];
SimRes4 = [SimRes4; Res4_AvgTime];
SimRes5 = [SimRes5; Res5_Reward];
end
SimRes1
SimRes2
SimRes3
SimRes4
SimRes5
%iCELFPopulationServed
%GreedyPopulationServed
%RandomPopulationServed
```

Figure 8-1: Simulation code for IoT-based healthcare in MATLAB application

Appendix B **Healthcare simulation results**

As multiple rounds of simulation for the “healthcare” case have been performed and each simulation experiment generates 100 independent set of values, the results of the simulation experiments are too lengthy to be copied here. However, to give a reference, one case of the simulation results is depicted in this section. First, some part of experiment results are shown in

Table 8-1 and then, a snapshot of MATLAB application environment after running the simulation experiments is depicted in Figure 8-2.

Table 8-1: Sample of redundant nodes results (selection)

# of experiment	iCELF	Greedy	Random	Total nodes
1	77	63	48	100
2	72	65	55	100
3	82	62	59	100
4	77	65	64	100
5	78	69	56	100
…….				
95	78	60	58	100
96	81	62	53	100
97	80	66	62	100

98	76	59	52	100
99	81	64	63	100
100	76	60	59	100

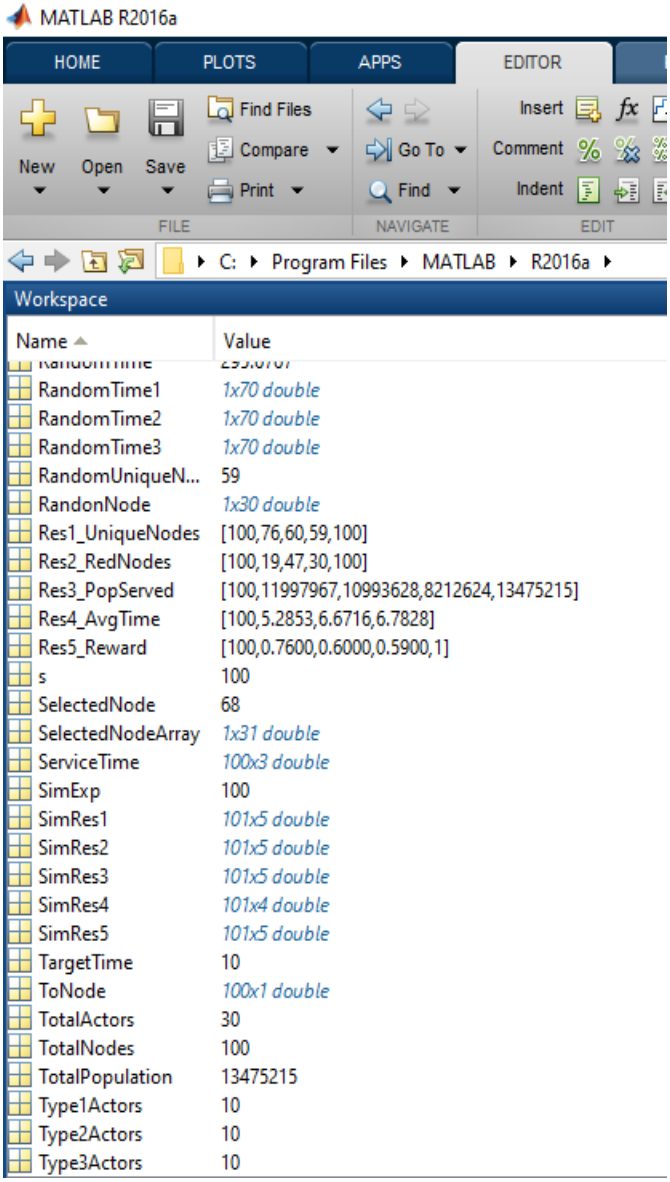


Figure 8-2 snapshot of workspace of Matlab application after running simulation

Appendix C **The IoT disaster recovery simulation code**

The simulation explained in 4.2.3, 4.2.4, 5.1.1, and 5.1.2 is copied here. It is written using the syntax of MATLAB R2016a and shown in

	Code
1	clc
2	clear all
3	SimRes1 = [0,0,0,0,0];
4	SimRes2 = [0,0,0,0,0];
5	SimRes3 = [0,0,0,0,0];
6	SimRes4 = [0,0,0,0];
7	SimRes5 = [0,0,0,0,0];
8	PopulationFactor =500000;
9	%NODES
10	TotalNodes=100;
11	%ACTORS
12	Type1Actors=10;
13	Type2Actors=10;
14	Type3Actors=10;
15	TotalActors=Type1Actors+Type2Actors+Type3Actors;
16	%Will be 100 for the final experiments
17	SimExp=100;
18	%Generating area network having nodes equal to 'TotalNodes'
19	FromNode=floor(1+rand(1,TotalNodes)*TotalNodes);
20	ToNode=floor(1+rand(1,TotalNodes)*TotalNodes);

```
for i=1:TotalNodes
    if(ToNode(i)==FromNode(i))
        ToNode(i)=floor(1+rand(1)*TotalNodes);
    end
end
FromNode=FromNode';
ToNode=ToNode';
for s=1:SimExp
%Generating random population Range Crowded: 10,000-500,000 &
%                    Retired: 1000-6000
%-------------------------------------------------------------------------%
% For all random nodes                                                    %
%-------------------------------------------------------------------------%
  %   NodePopulation=randi([1000   PopulationFactor],1,TotalNodes);
%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%
% For crowded and retarded nodes% with given pertentage of crowded nodes%
%-------------------------------------------------------------------------%
    %Crowdedpercen=50; % crowded %age
    %CrowdedNodes=ceil(TotalNodes*(Crowdedpercen/100));
    %Retnodes=TotalNodes-CrowdedNodes;
    %CrowdedNodesPopulation=randi([10000 PopulationFactor],1,CrowdedNodes);
    %RetnodesNodePopulation=randi([100   6000],1,Retnodes);
    %NodePopulation=[CrowdedNodesPopulation,RetnodesNodePopulation];
%-------------------------------------------------------------------------%
```

```
%-------------------------------------------------------------------------%
% Colonies of crowded and colonies of retarded                            %
%-------------------------------------------------------------------------%
 % NodePopulation=randi([1000    PopulationFactor],1,TotalNodes);
   %NodesInCrowdedRegion= 2;
   %NodesInRetardeddRegion= 2;
   %CwodedNodeList    = [1, 7];
   %RetardedNodeList= [5, 9];
%for i=1:length(CwodedNodeList)
%     for j=1:NodesInCrowdedRegion
%NodePopulation((CwodedNodeList(i)-1)+j)=randi([10000    PopulationFactor],1,1);
    % end
%end
%for i=1:length(RetardedNodeList)
%       for j=1:NodesInRetardeddRegion
%             NodePopulation((RetardedNodeList(i)-1)+j)=randi([100    6000],1,1);
%       end
%end
%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%
% One big city surrounded by small cities                                 %
%-------------------------------------------------------------------------%
   NodePopulation=randi([1000    PopulationFactor],1,TotalNodes);
   BigCityList    = [1, 7, 20, 35, 40, 45, 68, 85, 90];
   NumberofSmallCitiesSurrounding = [4, 3, 5, 3, 5, 7, 10, 3, 6];
```

```
71      for i=1:length(BigCityList)
72          NodePopulation(BigCityList(i))=randi([10000    PopulationFactor],1,1);
73          for j=1:NumberofSmallCitiesSurrounding(1,i)
74              NodePopulation(BigCityList(i)+j)=randi([100    6000],1,1);
75          end
76      end
77      %-------------------------------------------------------------------------%
78
79      TotalPopulation=sum(NodePopulation);
80      %Population to probability conversion
81      AttackProb=NodePopulation/PopulationFactor;
82      AttackProb=AttackProb';
83      %Distance for each pair of node range 100-500
84      NodeDistance=ceil(10+rand(1,TotalNodes)*1000);
85      NodeDistance=NodeDistance';
86
87      DistnceMatrix =[%from node, to node, distance
88                              FromNode, ToNode,      NodeDistance ];
89
90      ActorSpeed=[50, 100, 150];
91      ActorCost=[500, 1000, 1500];
92
93      ServiceTime=[(DistnceMatrix(:,3)/ActorSpeed(1)),(DistnceMatrix(:,3)/ActorSpeed(2)
        ), (DistnceMatrix(:,3)/ActorSpeed(3))];    % Possible service time for each actor
94
```

```
95  TargetTime=10; %Acceptable service time
96  iCELFSelectedNodes=0;
97  GreedySelectedNodes=0;
98  RandomSelectedNodes=0;
99  iCELFUniqueNodes=0;
100 GreedyUniqueNodes=0;
101 RandomUniqueNodes=0;
102 iCELFNodeList=0;
103 GreedyNodeList=0;
104 RandomNodeList=0;
105 iCELFPopulationServed=0;
106 GreedyPopulationServed=0;
107 RandomPopulationServed=0;
108
109 %Making network connection
110 for i=1:length(DistnceMatrix(:,1))
111     ConnectionMatrix(i,i)=10;
112     if ((ServiceTime(1) < TargetTime) || (ServiceTime(2) < TargetTime) ||
    (ServiceTime(3) < TargetTime))
113         ConnectionMatrix(DistnceMatrix(i,1),DistnceMatrix(i,2)) =
    DistnceMatrix(i,3);
114         ConnectionMatrix(DistnceMatrix(i,2),DistnceMatrix(i,1)) =
    DistnceMatrix(i,3);
115     else
116     ConnectionMatrix(DistnceMatrix(i,1),DistnceMatrix(i,2))=0;
```

```
117             ConnectionMatrix(DistnceMatrix(i,2),DistnceMatrix(i,1))=0;
118             end
119         end
120
121         SelectedNode=0;
122         SelectedNodeArray=0;
123         GreedyNodeArray=0;
124         CoverageFlag=zeros(1,length(ConnectionMatrix(1,:)));
125
126         for act=1:TotalActors
127             CoverageNodes=zeros(1,length(ConnectionMatrix(1,:)));
128             %Assesing coverable nodes for each point
129             for j=1:length(ConnectionMatrix(:,1))
130                 for i=1:length(SelectedNodeArray)
131                     if(j==SelectedNodeArray(i))
132                         ActionFlag=0;
133                     else
134                         ActionFlag=1;
135                     end
136                 end
137                 if(ActionFlag==1)
138                     for k=1:length(ConnectionMatrix(1,:))
139                         if(ConnectionMatrix(j,k)> 0 && CoverageFlag(k)==0)
140                     CoverageNodes(j)=CoverageNodes(j)+1; %Finding number of nodes
        served by each location
```

```
                                end
                            end
                        end
                    end
                    %Considering both distance and Attack probability
                    DistanceFactor=(CoverageNodes/max(CoverageNodes))';
                    AttackFactor=AttackProb;
                    % If attack probility is not considered, remove "+ AttackFactor" from following equation only
                    FinalFactor=DistanceFactor+AttackFactor;
                % FOR SIMPLE GREEDY ALGORITHM
                if (act==1)
                        Greedy=FinalFactor;
                        GreedyArray=sort(Greedy,'descend');
                        for g=1:TotalActors
                            for j=1:length(Greedy)
                                if(GreedyArray(g)==(Greedy(j)))
                                    GreedyNodeArray=[GreedyNodeArray,j];
                                end
                            end
                        end
                    end
                end
                % FOR SIMPLE RANDOM ALGORITHM
                RandonNode=ceil(rand(TotalActors, 1)*TotalNodes)';

```

```
165
166             %iCELF: Find the node with max feasible connections
167               ComapreFactor=0;
168               for j=1:length(CoverageNodes)
169                     if FinalFactor(j) > ComapreFactor
170                           SelectedNode=j;
171                           ComapreFactor=FinalFactor(j);
172                     end
173               end
174                   SelectedNodeArray=[SelectedNodeArray,SelectedNode];
175
176               for j=1:length(ConnectionMatrix(1,:))
177                     if (ConnectionMatrix(SelectedNode,j)>0)
178                           CoverageFlag(j)=1;
179                           CoverageFlag(SelectedNode)=1;
180                     end
181               end
182         end
183
184         %Final RESULTS
185         iCELFAlgoNodes=SelectedNodeArray(2:length(SelectedNodeArray));
186         GreedyAlgoNodes=GreedyNodeArray(2:length(GreedyNodeArray));
187         RandomAlgoNodes=RandonNode;
188
189         %finding unique number of nodes covered by each algo
```

```
190     for i=1:TotalActors
191         for j=1:TotalNodes
192             if(ConnectionMatrix(j,iCELFAlgoNodes(i))> 0)
193                 iCELFSelectedNodes=[iCELFSelectedNodes,j];
194             end
195             if(ConnectionMatrix(j,GreedyAlgoNodes(i))> 0)
196                 GreedySelectedNodes=[GreedySelectedNodes,j];
197             end
198             if(ConnectionMatrix(j,RandomAlgoNodes(i))> 0)
199                 RandomSelectedNodes=[RandomSelectedNodes,j];
200             end
201         end
202     end
203
204     iCELFNodes=sort(iCELFSelectedNodes);
205     GreedyNodes=sort(GreedySelectedNodes);
206     RandomNodes=sort(RandomSelectedNodes);
207
208     iCELFAllNodes=length(iCELFNodes);
209     GreedyAllNodes=length(GreedyNodes);
210     RandomAllNodes=length(RandomNodes);
211
212     for i=1:length(iCELFSelectedNodes)-1
213         if(iCELFNodes(i)~=iCELFNodes(i+1))
214             iCELFUniqueNodes=iCELFUniqueNodes+1;
```

```
215                     iCELFNodeList=[iCELFNodeList,iCELFNodes(i+1)];
216             end
217     end
218     iCELFNodeList;
219     for i=1:length(GreedySelectedNodes)-1
220             if(GreedyNodes(i)~=GreedyNodes(i+1))
221                     GreedyUniqueNodes=GreedyUniqueNodes+1;
222                 GreedyNodeList=[GreedyNodeList,GreedyNodes(i+1)];
223             end
224     end
225     GreedyNodeList;
226     for i=1:length(RandomSelectedNodes)-1
227             if(RandomNodes(i)~=RandomNodes(i+1))
228                     RandomUniqueNodes=RandomUniqueNodes+1;
229               RandomNodeList=[RandomNodeList,RandomNodes(i+1)];
230             end
231     end
232     iCELFUniqueNodes;
233     GreedyUniqueNodes;
234     RandomUniqueNodes;
235
236     iCELFRedNodes=iCELFAllNodes-iCELFUniqueNodes;
237     GreedyRedNodes=GreedyAllNodes-GreedyUniqueNodes;
238     RandomRedRedNodes=RandomAllNodes-RandomUniqueNodes;
239
```

```
240  for i=2:length(iCELFNodeList)
241      iCELFPopulationServed
     =iCELFPopulationServed+NodePopulation(iCELFNodeList(i));
242  end
243  for i=2:length(GreedyNodeList)
244      GreedyPopulationServed
     =GreedyPopulationServed+NodePopulation(GreedyNodeList(i));
245  end
246  for i=2:length(RandomNodeList)
247      RandomPopulationServed
     =RandomPopulationServed+NodePopulation(RandomNodeList(i));
248  end
249
250  %For average service time
251  for i=2:length(iCELFNodeList)
252      iCELFTime1(i) =NodeDistance(iCELFNodeList(i))/ActorSpeed(1);
253      iCELFTime2(i) =NodeDistance(iCELFNodeList(i))/ActorSpeed(2);
254      iCELFTime3(i) =NodeDistance(iCELFNodeList(i))/ActorSpeed(3);
255  end
256  iCELFTime1=sort(iCELFTime1);
257  iCELFTime2=sort(iCELFTime2);
258  iCELFTime3=sort(iCELFTime3);
259
260  for i=2:length(iCELFNodeList)
261      a=ceil(length(iCELFNodeList)/(TotalActors/Type1Actors));
```

```
262         b=ceil(length(iCELFNodeList)/(TotalActors/Type2Actors));
263         c=length(iCELFNodeList)-(a+b);
264
265         iCELFTime
    =sum(iCELFTime1(1:a))+sum(iCELFTime2(a+1:a+b))+sum(iCELFTime3(a+b+1:a+
    b+c));
266         iCELFAverageTime=iCELFTime/length(iCELFNodeList);
267     end
268
269     %.................
270     for i=2:length(GreedyNodeList)
271         GreedyTime1(i) =NodeDistance(GreedyNodeList(i))/ActorSpeed(1);
272         GreedyTime2(i) =NodeDistance(GreedyNodeList(i))/ActorSpeed(2);
273         GreedyTime3(i) =NodeDistance(GreedyNodeList(i))/ActorSpeed(3);
274     end
275     GreedyTime1=sort(GreedyTime1);
276     GreedyTime2=sort(GreedyTime2);
277     GreedyTime3=sort(GreedyTime3);
278
279     for i=2:length(GreedyNodeList)
280         a=ceil(length(GreedyNodeList)/(TotalActors/Type1Actors));
281         b=ceil(length(GreedyNodeList)/(TotalActors/Type2Actors));
282         c=length(GreedyNodeList)-(a+b);
283
284         GreedyTime
```

```
      =sum(GreedyTime1(1:a))+sum(GreedyTime2(a+1:a+b))+sum(GreedyTime3(a+b+1:a
      +b+c));
285       GreedyAverageTime=iCELFTime/length(GreedyNodeList);
286   end
287
288
289   %.....
290   for i=2:length(RandomNodeList)
291       RandomTime1(i) =NodeDistance(RandomNodeList(i))/ActorSpeed(1);
292       RandomTime2(i) =NodeDistance(RandomNodeList(i))/ActorSpeed(2);
293       RandomTime3(i) =NodeDistance(RandomNodeList(i))/ActorSpeed(3);
294   end
295   RandomTime1=sort(RandomTime1);
296   RandomTime2=sort(RandomTime2);
297   RandomTime3=sort(RandomTime3);
298
299   for i=2:length(RandomNodeList)
300       a=ceil(length(RandomNodeList) /(TotalActors/Type1Actors));
301       b=ceil(length(RandomNodeList) /(TotalActors/Type2Actors));
302       c=length(RandomNodeList)-(a+b);
303
304       RandomTime
      =sum(RandomTime1(1:a))+sum(RandomTime2(a+1:a+b))+sum(RandomTime3(a+b+
      1:a+b+c));
305       RandomAverageTime =iCELFTime/length(RandomNodeList);
```

```
306 end
307
308 MaxReward = TotalNodes/((Type1Actors*ActorCost(1))+(Type2Actors*ActorCost(2))+(Type3Actors*ActorCost(3)));
309 iCELFReward = (iCELFUniqueNodes /((Type1Actors*ActorCost(1))+(Type2Actors*ActorCost(2))+(Type3Actors*ActorCost(3))))/MaxReward;
310 GreedyReward = (GreedyUniqueNodes /((Type1Actors*ActorCost(1))+(Type2Actors*ActorCost(2))+(Type3Actors*ActorCost(3))))/MaxReward;
311 RandomReward = (RandomUniqueNodes /((Type1Actors*ActorCost(1))+(Type2Actors*ActorCost(2))+(Type3Actors*ActorCost(3))))/MaxReward;
312 MaxReward = (TotalNodes/((Type1Actors*ActorCost(1))+(Type2Actors*ActorCost(2))+(Type3Actors*ActorCost(3))))/MaxReward;
313
314
315 Res1_UniqueNodes = [s,iCELFUniqueNodes, GreedyUniqueNodes, RandomUniqueNodes, TotalNodes];
316 Res2_RedNodes = [s,iCELFRedNodes, GreedyRedNodes, RandomRedRedNodes, TotalNodes];
317 Res3_PopServed = [s,iCELFPopulationServed, GreedyPopulationServed, RandomPopulationServed, TotalPopulation];
```

```
318  Res4_AvgTime          = [s,iCELFAverageTime, GreedyAverageTime,
     RandomAverageTime];
319  Res5_Reward           = [s,iCELFReward, GreedyReward, RandomReward,
     MaxReward];
320
321  SimRes1 = [SimRes1; Res1_UniqueNodes];
322  SimRes2 = [SimRes2; Res2_RedNodes];
323  SimRes3 = [SimRes3; Res3_PopServed];
324  SimRes4 = [SimRes4; Res4_AvgTime];
325  SimRes5 = [SimRes5; Res5_Reward];
326  end
327  SimRes1
328  SimRes2
329  SimRes3
330  SimRes4
331  SimRes5
332  %iCELFPopulationServed
333  %GreedyPopulationServed
334  %RandomPopulationServed
```

Figure 8-1: Simulation code for IoT-based healthcare in MATLAB application

	Code
1	clc
2	clear all

```
3   SimRes1 = [0,0,0,0,0];
4   SimRes2 = [0,0,0,0,0];
5   SimRes3 = [0,0,0,0];
6   SimRes4 = [0,0,0,0,0];
7
8   PopulationFactor =500000;
9   %NODES
1   TotalNodes=100;
11  %ACTORS
12
13  Type1Actors=10;
14  Type2Actors=10;
15  Type3Actors=10;
16
17  TotalActors=Type1Actors+Type2Actors+Type3Actors;
18  %Will be 100 for the final experiments
19  SimExp=100;
20  %Generating area network having nodes equal to 'TotalNodes'
21  FromNode=floor(1+rand(1,TotalNodes)*TotalNodes);
22  ToNode=floor(1+rand(1,TotalNodes)*TotalNodes);
23  for i=1:TotalNodes
24      if(ToNode(i)==FromNode(i))
25          ToNode(i)=floor(1+rand(1)*TotalNodes);
26      end
27  end
```

```
FromNode=FromNode';
ToNode=ToNode';
for s=1:SimExp
        NodePopulation=randi([1000    PopulationFactor],1,TotalNodes);

%Generating random population Range Crowded: 10,000-500,000 &
%                       Retired: 1000-6000

%-------------------------------------------------------------------------%
% Both RISK and LOSS are
random                                                                    %
%-------------------------------------------------------------------------%
%       NodeRisk=rand(1,TotalNodes);
%       NodeLoss=rand(1,TotalNodes);
%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%
% Some high risk regions                                                  %
%-------------------------------------------------------------------------%
  %    HighRiskNodePrecent=15;
    % NodeLoss=rand(1,TotalNodes);

     %HighRiskNodes=ceil(TotalNodes*(HighRiskNodePrecent/100));
     %RemainingNodes=TotalNodes-HighRiskNodes;

%      HighRiskNodeRisk= (randi([500 1000],1,HighRiskNodes))/1000;
```

```
52      %    RemainingNodeRisk=(rand(1,RemainingNodes));
53        % NodeRisk=[HighRiskNodeRisk,RemainingNodeRisk];
54    %-------------------------------------------------------------------------%
55    %-------------------------------------------------------------------------%
56    % Some high loss regions                                                  %
57    %-------------------------------------------------------------------------%
58    %     HighLossNodePrecent=85;
59    %     NodeRisk=rand(1,TotalNodes);
60
61    %HighLossNodes =ceil(TotalNodes*(HighLossNodePrecent/100));
62      %    RemainingNodes=TotalNodes-HighLossNodes;
63
64      %HighLossNodeRisk = (randi([500 1000],1,HighLossNodes))/1000;
65      %    RemainingNodeRisk=(rand(1,RemainingNodes));
66      %    NodeLoss=[HighLossNodeRisk,RemainingNodeRisk];
67    %-------------------------------------------------------------------------%
68
69    %-------------------------------------------------------------------------%
70    % Some high loss and high risk regions                                    %
71    %-------------------------------------------------------------------------%
72        HighLossNodePrecent=15;
73        HighRiskNodePrecent=25;
74
75        HighLossNodes =ceil(TotalNodes*(HighLossNodePrecent/100));
76        RemainingNodes=TotalNodes-HighLossNodes;
```

```
77          HighLossNodeRisk= (randi([500 1000],1,HighLossNodes))/1000;
78          RemainingNodeRisk=(rand(1,RemainingNodes));
79          NodeLoss=[HighLossNodeRisk,RemainingNodeRisk];
80
81
82      HighRiskNodes =ceil(TotalNodes*(HighRiskNodePrecent/100));
83      RemainingNodes=TotalNodes-HighRiskNodes;
84      HighRiskNodeRisk= (randi([500 1000],1,HighRiskNodes))/1000;
85      RemainingNodeRisk=(rand(1,RemainingNodes));
86      NodeRisk=[HighRiskNodeRisk,RemainingNodeRisk];
87  %-------------------------------------------------------------------------%
88
89  TotalPopulation=sum(NodePopulation);
90
91  NodeRisk=NodeRisk';
92  NodeLoss=NodeLoss';
93  %Distance for each pair of node range 100-500
94  NodeDistance=ceil(10+rand(1,TotalNodes)*1000);
95  NodeDistance=NodeDistance';
96
97  DistnceMatrix =[%from node, to node, distance
98                              FromNode, ToNode,      NodeDistance ];
99  ActorSpeed=[50, 100, 150];
100 ActorCost=[500, 1000, 1500];
101
```

```
102  TargetTime=10; %Acceptable service time
103  iCELFSelectedNodes=0;
104  GreedySelectedNodes=0;
105  RandomSelectedNodes=0;
106  iCELFUniqueNodes=0;
107  GreedyUniqueNodes=0;
108  RandomUniqueNodes=0;
109  iCELFNodeList=0;
110  GreedyNodeList=0;
111  RandomNodeList=0;
112  iCELFPopulationServed=0;
113  GreedyPopulationServed=0;
114  RandomPopulationServed=0;
115
116  %Making network connection
117  for i=1:length(DistnceMatrix(:,1))
118      ConnectionMatrix(i,i)=10;
119      ConnectionMatrix(DistnceMatrix(i,1),DistnceMatrix(i,2)) = DistnceMatrix(i,3);
120      ConnectionMatrix(DistnceMatrix(i,2),DistnceMatrix(i,1)) = DistnceMatrix(i,3);
121  end
122
123  SelectedNode=0;
124  SelectedNodeArray=0;
125  GreedyNodeArray=0;
126  CoverageFlag=zeros(1,length(ConnectionMatrix(1,:)));
```

```
127
128  for act=1:TotalActors
129      CoverageNodes=zeros(1,length(ConnectionMatrix(1,:)));
130      %Assesing coverable nodes for each point
131      for j=1:length(ConnectionMatrix(:,1))
132          for i=1:length(SelectedNodeArray)
133              if(j==SelectedNodeArray(i))
134                  ActionFlag=0;
135              else
136                   ActionFlag=1;
137              end
138          end
139          if(ActionFlag==1)
140              for k=1:length(ConnectionMatrix(1,:))
141                  if(ConnectionMatrix(j,k)> 0 && CoverageFlag(k)==0)
142                      CoverageNodes(j) =CoverageNodes(j)+1; %Finding number of
     nodes served by each location
143                  end
144              end
145          end
146      end
147      %Considering both distance and Attack probability
148      DistanceFactor=(CoverageNodes/max(CoverageNodes))';
149      % If attack probility is not considered, remove "+ AttackFactor" from following
     equation only
```

```
150         FinalFactor=DistanceFactor+NodeLoss*0.5+NodeRisk*0.5;
151       % FOR SIMPLE GREEDY ALGORITHM
152       if (act==1)
153             Greedy=FinalFactor;
154             GreedyArray=sort(Greedy,'descend');
155             for g=1:TotalActors
156                   for j=1:length(Greedy)
157                         if(GreedyArray(g)==(Greedy(j)))
158                               GreedyNodeArray=[GreedyNodeArray,j];
159                         end
160                   end
161             end
162       end
163       % FOR SIMPLE RANDOM ALGORITHM
164       RandonNode=ceil(rand(TotalActors, 1)*TotalNodes)';
165
166       %iCELF: Find the node with max feasible connections
167         ComapreFactor=0;
168         for j=1:length(CoverageNodes)
169               if FinalFactor(j) > ComapreFactor
170                     SelectedNode=j;
171                     ComapreFactor=FinalFactor(j);
172               end
173         end
174             SelectedNodeArray=[SelectedNodeArray,SelectedNode];
```

```
175
176         for j=1:length(ConnectionMatrix(1,:))
177             if (ConnectionMatrix(SelectedNode,j)>0)
178                 CoverageFlag(j)=1;
179                 CoverageFlag(SelectedNode)=1;
180             end
181         end
182     end
183
184     %Final RESULTS
185     iCELFAlgoNodes=SelectedNodeArray(2:length(SelectedNodeArray));
186     GreedyAlgoNodes=GreedyNodeArray(2:length(GreedyNodeArray));
187     RandomAlgoNodes=RandonNode;
188
189     %finding total number of nodes covered by each algo
190     for i=1:TotalActors
191         for j=1:TotalNodes
192             if(ConnectionMatrix(j,iCELFAlgoNodes(i))> 0)
193                 iCELFSelectedNodes=[iCELFSelectedNodes,j];
194             end
195             if(ConnectionMatrix(j,GreedyAlgoNodes(i))> 0)
196                 GreedySelectedNodes=[GreedySelectedNodes,j];
197             end
198             if(ConnectionMatrix(j,RandomAlgoNodes(i))> 0)
199                 RandomSelectedNodes =[RandomSelectedNodes,j];
```

```
200                 end
201             end
202         end
203
204         iCELFNodes=sort(iCELFSelectedNodes);
205         GreedyNodes=sort(GreedySelectedNodes);
206         RandomNodes=sort(RandomSelectedNodes);
207
208         iCELFAllNodes=length(iCELFNodes);
209         GreedyAllNodes=length(GreedyNodes);
210         RandomAllNodes=length(RandomNodes);
211
212         for i=1:length(iCELFSelectedNodes)-1
213             if(iCELFNodes(i)~=iCELFNodes(i+1))
214                 iCELFUniqueNodes=iCELFUniqueNodes+1;
215                 iCELFNodeList=[iCELFNodeList,iCELFNodes(i+1)];
216             end
217         end
218         iCELFNodeList;
219         for i=1:length(GreedySelectedNodes)-1
220             if(GreedyNodes(i)~=GreedyNodes(i+1))
221                 GreedyUniqueNodes=GreedyUniqueNodes+1;
222                 GreedyNodeList =[GreedyNodeList,GreedyNodes(i+1)];
223             end
224         end
```

```
GreedyNodeList;
for i=1:length(RandomSelectedNodes)-1
    if(RandomNodes(i)~=RandomNodes(i+1))
        RandomUniqueNodes=RandomUniqueNodes+1;
        RandomNodeList =[RandomNodeList,RandomNodes(i+1)];
    end
end
iCELFUniqueNodes;
GreedyUniqueNodes;
RandomUniqueNodes;

iCELFRedNodes=iCELFAllNodes-iCELFUniqueNodes;
GreedyRedNodes=GreedyAllNodes-GreedyUniqueNodes;
RandomRedRedNodes=RandomAllNodes-RandomUniqueNodes;

% For average service time
for i=2:length(iCELFNodeList)
    iCELFTime1(i) =NodeDistance(iCELFNodeList(i))/ActorSpeed(1);
    iCELFTime2(i) =NodeDistance(iCELFNodeList(i))/ActorSpeed(2);
    iCELFTime3(i) =NodeDistance(iCELFNodeList(i))/ActorSpeed(3);
end
iCELFTime1=sort(iCELFTime1);
iCELFTime2=sort(iCELFTime2);
iCELFTime3=sort(iCELFTime3);

```

```
250  for i=2:length(iCELFNodeList)
251      a=ceil(length(iCELFNodeList)/(TotalActors/Type1Actors));
252      b=ceil(length(iCELFNodeList)/(TotalActors/Type2Actors));
253      c=length(iCELFNodeList)-(a+b);
254
255      iCELFTime
     =sum(iCELFTime1(1:a))+sum(iCELFTime2(a+1:a+b))+sum(iCELFTime3(a+b+1:a+b+
     c));
256      iCELFAverageTime=iCELFTime/length(iCELFNodeList);
257  end
258  %.....................
259  for i=2:length(GreedyNodeList)
260      GreedyTime1(i) =NodeDistance(GreedyNodeList(i))/ActorSpeed(1);
261      GreedyTime2(i) =NodeDistance(GreedyNodeList(i))/ActorSpeed(2);
262      GreedyTime3(i) =NodeDistance(GreedyNodeList(i))/ActorSpeed(3);
263  end
264  GreedyTime1=sort(GreedyTime1);
265  GreedyTime2=sort(GreedyTime2);
266  GreedyTime3=sort(GreedyTime3);
267
268  for i=2:length(GreedyNodeList)
269      a=ceil(length(GreedyNodeList)/(TotalActors/Type1Actors));
270      b=ceil(length(GreedyNodeList)/(TotalActors/Type2Actors));
271      c=length(GreedyNodeList)-(a+b);
272
```

```
273         GreedyTime
=sum(GreedyTime1(1:a))+sum(GreedyTime2(a+1:a+b))+sum(GreedyTime3(a+b+1:a+b
+c));
274         GreedyAverageTime=iCELFTime/length(GreedyNodeList);
275     end
276     %.....
277     for i=2:length(RandomNodeList)
278         RandomTime1(i) =NodeDistance(RandomNodeList(i))/ActorSpeed(1);
279         RandomTime2(i) =NodeDistance(RandomNodeList(i))/ActorSpeed(2);
280         RandomTime3(i) =NodeDistance(RandomNodeList(i))/ActorSpeed(3);
281     end
282     RandomTime1=sort(RandomTime1);
283     RandomTime2=sort(RandomTime2);
284     RandomTime3=sort(RandomTime3);
285
286     for i=2:length(RandomNodeList)
287         a=ceil(length(RandomNodeList) /(TotalActors/Type1Actors));
288         b=ceil(length(RandomNodeList) /(TotalActors/Type2Actors));
289         c=length(RandomNodeList)-(a+b);
290
291         RandomTime=sum(RandomTime1(1:a))
+sum(RandomTime2(a+1:a+b))+sum(RandomTime3(a+b+1:a+b+c));
292         RandomAverageTime=iCELFTime /length(RandomNodeList);
293     end
294
```

```
295  MaxReward = TotalNodes/((Type1Actors*ActorCost(1))+(Type2Actors*ActorCost(2))+(Type3Actors*ActorCost(3)));
296  iCELFReward = (iCELFUniqueNodes/((Type1Actors*ActorCost(1))+(Type2Actors*ActorCost(2))+(Type3Actors*ActorCost(3))))/MaxReward;
297  GreedyReward = (GreedyUniqueNodes/((Type1Actors*ActorCost(1))+(Type2Actors*ActorCost(2))+(Type3Actors*ActorCost(3))))/MaxReward;
298  RandomReward = (RandomUniqueNodes/((Type1Actors*ActorCost(1))+(Type2Actors*ActorCost(2))+(Type3Actors*ActorCost(3))))/MaxReward;
299  MaxReward = (TotalNodes/((Type1Actors*ActorCost(1))+(Type2Actors*ActorCost(2))+(Type3Actors*ActorCost(3))))/MaxReward;
300
301  Res1_UniqueNodes = [s,iCELFUniqueNodes, GreedyUniqueNodes, RandomUniqueNodes, TotalNodes];
302  Res2_RedNodes = [s,iCELFRedNodes, GreedyRedNodes, RandomRedRedNodes, TotalNodes];
303  Res3_AvgTime = [s,iCELFAverageTime, GreedyAverageTime, RandomAverageTime];
304  Res4_Reward = [s,iCELFReward, GreedyReward, RandomReward, MaxReward];
305
```

```
306  SimRes1 = [SimRes1; Res1_UniqueNodes];
307  SimRes2 = [SimRes2; Res2_RedNodes];
308  SimRes3 = [SimRes3; Res3_AvgTime];
309  SimRes4 = [SimRes4; Res4_Reward];
310  end
311  SimRes1
312  SimRes2
313  SimRes3
314  SimRes4
315
316  %iCELFPopulationServed
317  %GreedyPopulationServed
318  %RandomPopulationServed
```

Figure 8-3: Simulation code for IoT based disaster recovery in MATLAB application

Appendix D **Disaster recovery simulation results**

As multiple rounds of simulation for the “healthcare” case have been performed and each simulation experiment generates 100 independent set of values, the results of the simulation experiments are too lengthy to be copied here. However, to give a reference, one case of the simulation results is depicted in this section.

Table 8-2: Sample of the average service time results (selection)

# of experiment	iCELF	Greedy	Random
1	4.21	4.95	6.01
2	4.55	5.32	6.74
3	5.03	5.82	7.14
4	5.14	6.02	7.14
5	5.17	5.98	7.34
95	4.06	4.69	5.94
96	5.30	6.30	8.21
97	4.98	6.09	6.18
98	3.85	4.65	5.03
99	4.42	5.18	6.47
100	5.05	5.86	6.97

Appendix E **Sample of experiment results of CloudAnalyst application**

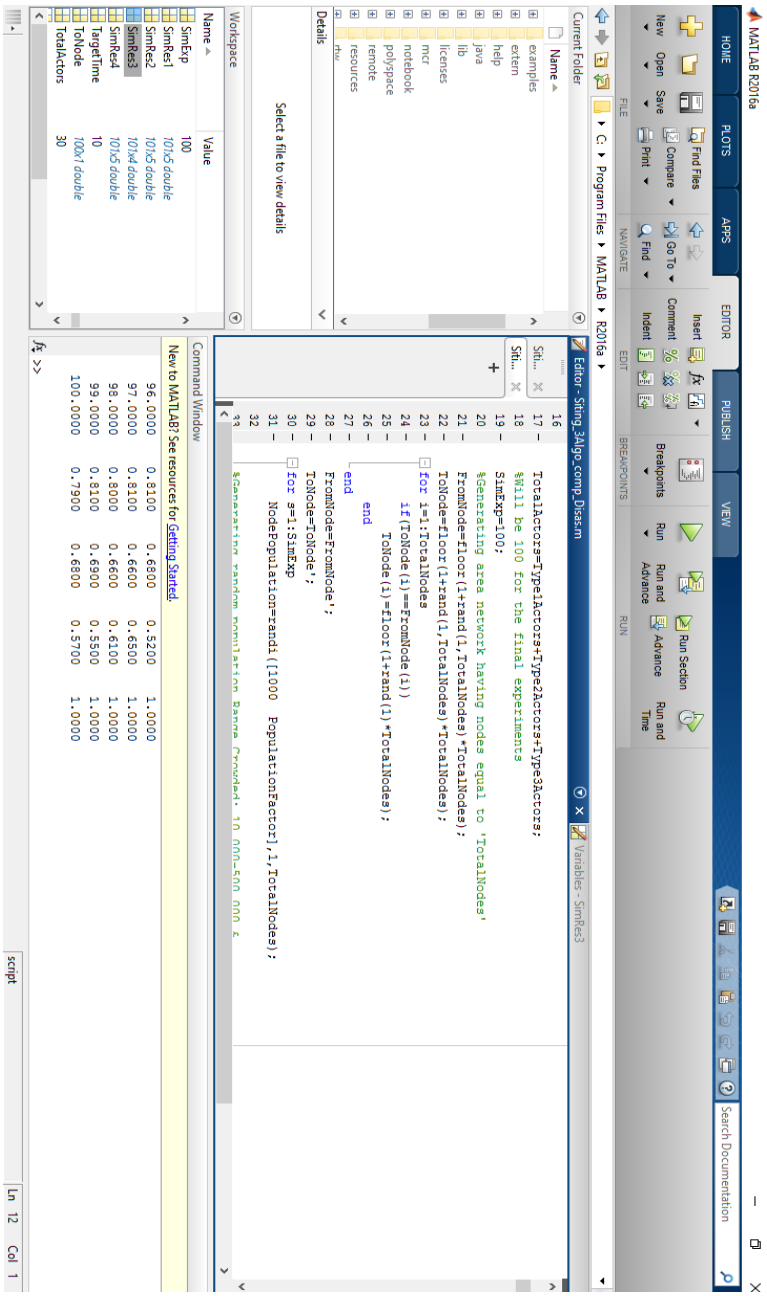


Figure 8-4 snapshot of the Matlab application after running simulation

국문 초록

IoT 서비스 제공자를 위한 의사 결정 지원 도구

IoT 기반 센서 액터 시스템을 위한 비용 성능 최적화

Mohammad Mahdi Kashef
협동과정 기술경영경제정책전공
공과대학
서울대학교

IoT 이란 고유하게 식별 가능한 객체 (것들) 이면 이것을 시각적으로 표현 가능한 것이 바로 인터넷과 같은 구조이다. IoT는 인터넷을 통해 물리적인 객체가 타 객체와 통신이 가능 하고 협조하여 사전에 정의 된 목표를 달성이 가능한 단계에서 출시되었다. 그러므로 IoT는 미래 인터넷의 (FI) 필수 구성 요소로 예측하였다.따라서 IoT는 다른 FI 통합 서비스와 원활하게 통합되어야 한다. 그럼에도 불구하고 IoT 장치는 위치에 따라 다르며, 더욱 중요한 것은 개발 및 배포 비용이 비싸다는 것이다. 그 이유는 IoT를 인프라를 지지 하는 즉 컴퓨팅 파워, 스토리지 및 네트워크가 리소스 제약을 받기 때문이다.

그러므로 이러한 단점을 해결하기 위해 최근에 클라우드 컴퓨팅이라는 또 다른 현상이 가장 유망하고 비용 효율적인 솔루션이 될 수 있어보인다. 실제로 클라우드는 인프라 비해 비교적 저렴하고 어디에서나 흔히 볼 수 있으며, 지원 무한하고 탄력적 인 솔루션이다. 따라서 IoT 기반 장치를 연결, 관리 및 추적하고 적절한 접속을 다수의 컴퓨팅에 제공하기 위해 많은 IoT 서비스 공급자 (IoTSP) 가 클라우드를 활용하여 사용자에게 특정 서비스를 제공 할 수 있다. 전 세계에 분산 되어있는 사용자에게 서비스를 제공하기 위해 IoTSP는 멀티 클라우드에 Virtual Machines (VM)을 배치하며, Cloud Service Providers (CSP)는 사용자에게 더욱 만족스러운 적용 범위와 성능을 제공한다.

하지만 완성된 IoT와 멀티 클라우드의 통합은 새로운 도전을 제기한다 하지만 이것이 IoTSP 사업 성공의 가장 중요한 요소 중 하나이다. 이 맥락에서 가장 큰 문제점 중 하나는 시스템을 운영하면서 만족스러운 수준의 성과를 유지하여 비즈니스가 수익을 창출하도록 하는 동시에 IoT 시스템의 전체 비용을 최소화하는 것이다. 이를 위해, IoTSP는 IoT 디바이스를 비용 최적화 된 상태로 유지해야하며, 또한 비용이 최 적화 된 VM을 사용 가능한 CSP에 배치한다. 다시 말하자면, IoTSP에 대한 가장 문제는 IoT 장치 및 VM에 대한 비용을 최적 배치를 찾아야 하는 것이다.

이 논문은 이러한 문제를 IoTSP가 비용 최적화 장치 및 멀티 클라우드에서의 VM 배치를 찾는 데 필요한 지원 도구를 제안하여 해결하고자 한다. 사운드 시스템 아키텍처 도구는 작업을 수행하도록 설계되었다. 이 툴은 IoT 디바이스 배치를위한 휴리스틱 알고리즘 이며 비용 추정 모듈 및 VM 배치 최적화 알고리즘을 포함한다. 또한 비용 추정 모듈은 다중 클라우드 환경을 고려한 특정인 VM 배치에 대한 총 인프라 비용으로 예측하며, 제안 된 최적화 알고리즘을 예상 비용과 예상 성능으로 실행하면 비용 최적화 된 IoT 장치와 IoTSP의 VM 배치가 반환된다. 제안 된 의사 결정 지원 도구는 여러 시뮬레이션 시나리오에 의해 검사되며 그 결과는 도구의 작동을 보여준다.

주요어: 사물 인터넷, 멀티 클라우드, 의사결정지원도구, 비용 추정, 비용 최적화

학 번: 2010-30812