SCHEDULING IOT REQUESTS TO MINIMIZE LATENCY IN FOG COMPUTING

by

Mazin Abdelbadea Nasralla Alikarar

A Thesis presented to the Faculty of the
American University of Sharjah
College of Engineering
In Partial Fulfillment
of the Requirements
for the Degree of

Master of Science in
Computer Engineering

Sharjah, United Arab Emirates

June 2017

© 2017 Mazin Abdelbadea Nasralla Alikarar. All rights reserved.

Approval Signatures

We, the undersigned, approve the Master's Thesis of Mazin Abdelbadea Nasralla Alikarar.

Thesis Title: Scheduling IoT Requests to Minimize Latency in Fog Computing.

Signature	**Date of Signature** (dd/mm/yyyy)
___________________________	_______________

Dr. Raafat Aburukba
Assistant Professor, Department of Computer Science and Engineering
Thesis Advisor

___________________________ _______________

Dr. Taha Landolsi
Professor, Department of Computer Science and Engineering
Thesis Co-Advisor

___________________________ _______________

Dr. Assim Sagahyroon
Professor, Department of Computer Science and Engineering
Thesis Committee Member

___________________________ _______________

Dr. Malick Ndiaye
Associate Professor, Department of Industrial Engineering
Thesis Committee Member

___________________________ _______________

Dr. Fadi Ahmed Aloul
Head, Department of Computer Science and Engineering

___________________________ _______________

Dr. Ghaleb Husseini
Associate Dean for Graduate Affairs and Research
College of Engineering

___________________________ _______________

Dr. Richard Schoephoerster
Dean, College of Engineering

___________________________ _______________

Dr. Mohamed El-Tarhuni
Vice Provost for Graduate Studies

Acknowledgement

First of all, I am thankful and grateful to Allah for giving me patience, courage and strength to work towards this graduate degree.

I would like to thank the American University of Sharjah, Computer Engineering Department, for granting me the Graduate Teaching Assistantship Scholarship (GTA) to do my master’s degree.

I would also like to express my sincere gratitude and appreciation to my advisors; Dr. Raafat Aburukba and Dr. Taha Landolsi for their valuable feedback, effort, guidance, and the long hours of meetings they provided me with to achieve this work; it would have been almost impossible without their rich and valuable experience and knowledge.

Many thanks also go to the committee member Dr. Assim Sagahyroon for his valuable comments and feedback. Special thanks also to the committee member Dr. Malick Ndiaye who has been providing help and support in major parts of this research.

Finally, I would like to extend my special thanks to the department of Computer Engineering, in American University of Sharjah, represented in Dr. Fadi Aloul, the department head, for his limitless care and support he provided for me and all students in the department.

Dedication

To my beloved mother for her endless love, care, faith and support.

To my father.

To my brothers.

To my professors Dr. Raafat Aburukba and Dr. Taha Landolsi the most remarkable and knowledgeable professors.

To my true friends.

Abstract

Delivering services for Internet of Things (IoT) applications that demand real-time and predictable latency is challenge. Several IoT applications require stringent latency requirements due to the interaction between the IoT devices and the physical environment through sensing and actuation. The limited capabilities of IoT devices require applications to be integrated in cloud computing and fog computing paradigms. Fog computing significantly improves on the service latency as it brings resources closer to the edge. The characteristics of both fog and cloud computing will enable the integration and interoperation of a large number of IoT devices and services in different domains. This thesis models the scheduling of IoT service requests as an optimization problem using integer programming in order to minimize the overall service request latency. The scheduling problem by nature is NP-hard, and hence, exact optimization solutions are inadequate for large size problems. Hence, this work uses the genetic algorithm (GA) as a heuristic approach to schedule the IoT requests and achieve the objective of minimizing the overall latency. The GA is tested in a dynamic simulation environment. The performance of the GA is evaluated and compared to the performance of waited-fair queuing (WFQ), priority-strict queuing (PSQ), and round robin (RR) techniques. The results show that the overall latency for the proposed approach is 21.9% to 46.6% better than the other algorithms. The proposed approach also showed significant improvement in meeting the requests deadlines by up to 31%.

Search Terms: ***Internet of Things; cloud computing; fog computing; latency; scheduling; optimization; genetic algorithm***

Table of Contents

Abstract....6
Table of Contents....7
List of Figures....9
List of Tables....10
List of Abbreviations....11
Chapter 1. Introduction....12
1.1 Fog Computing Overview....12
1.2 Fog Computing System Architecture....14
1.3 Scheduling in Fog Computing....14
1.4 Research Objective and Contribution....17
1.5 Research Methodology....18
1.6 Thesis Organization....19
Chapter 2. Literature Review....20
2.1 Characteristics of Fog Computing versus Cloud Computing....20
2.2 Other Platforms Similar to Fog Computing:....21
2.3 Implementations within Fog Computing....22
2.4 Scheduling Techniques....24
2.5 Fog Computing and IoT Interconnection....26
2.6 Latency Optimization using Genetic Algorithms....26
Chapter 3. Modeling the Problem....28
3.1 Environment Analysis....28
3.2 The Edge-Fog-Cloud Environment Model....29
3.3 Model validation using Lingo....33
Chapter 4. Proposed Solution....37
4.1 Genetic Algorithms....37
4.2 The GA Implementation....38
4.2.1 Initial Population....41
4.2.2 Selection using Roulette Wheel....41
4.2.3 Crossover....43
4.2.4 Mutation....45
4.2.5 Feasibility Check....46

4.2.6 Fitness Calculation. 47
4.3 GA Experimentation 48
4.3.1 Population Size, *U*. 48
4.3.2 Termination Counter, *T*. 52
4.3.3 Exact and Heuristic Comparison. 54
Chapter 5. Simulation and Results 57
5.1 GA validation in SimEvents 58
5.2 Static Scheduling 60
5.3 Average Data Size Breaking Point 62
5.4 Dynamic Scheduling 64
5.5 Cloud versus Fog Computing Comparison 65
5.5.1 The Average Delay Ratio, , $\frac{\bar{\delta}_f}{\bar{\delta}_c}$. 67
5.5.2 The Processing Speed Ratio, $\frac{P_f}{P_c}$. 67
5.5.3 The Number of Servers Ratio, $\frac{N_f}{N_c}$.. 69
Chapter 6. Conclusion and Future Research 71
6.1 Conclusion 71
6.2 Future Research 72
References 73
Vita 76

List of Figures

Figure 1: Edge-Fog-Cloud Architecture 15
Figure 2: Fog and Cloud Computing Architecture of Service Scheduling 28
Figure 3: Gantt chart for Lingo Optimal Scheduling Solution 36
Figure 4: Chromosome Representation as 2-D Array 38
Figure 5: Chromosome Representation as 1-D Array 38
Figure 6: GA Implementation Flowchart 40
Figure 7: Crossover Operation 44
Figure 8: The Cross-Point within a Chromosome 45
Figure 9: Mutation Operation 45
Figure 10: Request Chunks for Feasibility Check 47
Figure 11: Overall Latency versus Population Size 51
Figure 12: Runtime versus Population Size 52
Figure 13: Overall Latency versus Termination Counter 53
Figure 14: Runtime versus Termination Counter 53
Figure 15: Overall Latency Convergence through the GA Iterations 54
Figure 16: Overall Latency Comparison between Heuristic and Exact Methods 55
Figure 17: Runtime Comparison between Heuristic and Exact Methods 56
Figure 18: Edge-Fog-Cloud 3-Layered Simulation Setup 57
Figure 19: Analyzing the Number of Requests Allocated in Each Resource 60
Figure 20: Overall Latency versus Data Size in Static Scheduling 62
Figure 21: Missed-Deadline Requests versus Data Size in Static Scheduling 63
Figure 22: The GA Breaking Point versus Processing Speed and Average Delay 64
Figure 23: Overall Latency versus Data Size in Dynamic Scheduling 66
Figure 24: Missed-Deadline Requests versus Data Size in Dynamic Scheduling 66
Figure 25: Latency of Fog Compared to Cloud by Varying Average Delay 68
Figure 26: Latency of Fog Compared to Cloud by Varying Processing Power 68
Figure 27: Latency of Fog Compared to Cloud by Varying Number of Resources 69
Figure 28: Break Points of Fog and Cloud Computing Latency 70

List of Tables

Table 1: Fog versus Cloud Characteristics 20
Table 2: 5 Requests with Their Associated Attributes 34
Table 3: 2 Resources With Their Associated Attributes 34
Table 4: Lingo Optimal Scheduling Solution – Part 1 36
Table 5: Lingo Optimal Scheduling Solution – Part 2 36
Table 6: GA Implementation Pseudocode 41
Table 7: Initial Population Algorithm 42
Table 8: Selection using Roulette Wheel 43
Table 9: Crossover 44
Table 10: Mutation 46
Table 11: Feasibility Check 47
Table 12: Fitness Calculation 49

List of Abbreviations

GA – Genetic Algorithm

SA – Simulated Annealing

B&B – Branch and Bound

WFQ – Waited Fair Queuing

PSQ – Priority Strict Queuing

RR – Round Robin

FIFO – First In First Out

IoT – Internet of Things

RTT – Round-Trip Time

AP – Access Point

IoE – Internet of Everything

3G – Third Generation

4G – Fourth Generation

LTE – Long Term Evolution

WiFi – Wireless Fidelity

MCC – Mobile Cloud Computing

MEC – Mobile Edge Computing

CPU – Central Processing Unit

IaaS – Infrastructure as a Service

PSO – Particle Swarm Optimization

AHP – Analytic Hierarchy Process

VM – Virtual Machine

EDF – Earliest Deadline First

DCP – Dynamic Critical Path

ACO – Ant Colony Optimization

QoS – Quality of Service

Chapter 1. Introduction

1.1 Fog Computing Overview

Cisco introduced the concept of fog computing paradigm in 2012 [1]. Fog computing is a vision in which the edge of the network is transformed into a distributed computing infrastructure by pushing the cloud resources towards the network edge. The term 'fog computing' is chosen as an analogy where a fog is a cloud close to the ground or the edge [2]. Some other sources refer to fog computing as an abbreviation for "From cOre to edGe" [3]. In fog computing, computation and storage capabilities empowers the networking devices at different layers in the network architecture. These fog devices are also equipped with schedulers and decision capabilities that make them able to decide whether to serve or allocate a request in fog computing devices or transfer it to the cloud computing data centers [3, 4]. Such a decision is typically based on many attributes that are related to the requests and the resources available at the fog and cloud layers. Fog computing benefits Internet of Things (IoT) applications as it has the resources with physical proximity to the edge devices which will allow the applications, services and computations to run as close as possible to the data generated from devices, things and people (end users) connected to the Internet.

The work in [2, 3, 5-13] shared the same view for fog computing that it is a new distributed computing paradigm that extends the traditional cloud computing resources towards, but not exclusively, the edge of the network. Similar to cloud computing, fog computing paradigm provides ubiquitous computation, storage, networking, and application services in a highly visualized platform at the edge between end devices and traditional cloud computing data centers. Virtualization is a fundamental technology for fog computing as it separates physical infrastructures to create various dedicated resources that can run multiple operating systems and multiple applications at the same time on the same resource.

In [14], Cisco introduced fog computing as an extension to the cloud to be closer to the devices that produce and act on the data. These extended resources are called fog nodes; they can be deployed anywhere with a network connection. Any device with computing, storage, and network connectivity can be a fog node [14]. In [11], Yi *et al.* viewed the fog computing nodes as facilities or infrastructures that have the ability to cater services using the resources at the edge of the network. These infrastructures exist

in many different devices or equipment's forms, as poor-resources devices such as access points, routers, switches, base stations, and end devices, or as resource-rich machines such as Cloudlet. Cloudlet is basically a powerful computer connected to the Internet with rich resources that are available to host and use by nearby edge devices [11, 15]. Although Cloudlet has been given a different terminology, it falls under the same umbrella of fog computing [10].

E. Baccarelli *et al.* [16] formally viewed fog computing as a model to complement the cloud computing through the distribution of the computing plus networking resources from remote data centers towards edge devices. The final goal is to save energy and bandwidth, while simultaneously increasing the QoS level provided to the users. As a consequence, they defined Fog Nodes as virtualized networked data centers, which run atop (typically, wireless) Access Points (APs) at the edge of the access network, in order to give rise to a three-tier IoE–Fog–Cloud hierarchical architecture where IoE stands for Internet of Everything. In [17], Vaquero *et al.* proposed a formal definition for fog computing: "fog computing is a scenario where a huge number of heterogeneous (wireless and sometimes autonomous) ubiquitous and decentralized devices communicate and potentially cooperate among them and with the network to perform storage and processing tasks without the intervention of third parties. These tasks can be for supporting basic network functions (routing and switching) or new services and applications that run in a sandboxed (isolated and restricted) environment. Users leasing part of their devices to host these services get incentives for doing so". This definition succeeded to point out the proximity, wireless, decentralized characteristics of fog computing. The definition also pointed out the potential cooperation between fog computing devices which is a significant property within fog computing paradigm for load balancing purposes. However, in this definition, the authors did not focus on addressing the interplay and interaction between the fog layer at the edge and the cloud computing as a centralized platform. Each one of these platforms has its own characteristics which are suitable for specific type of use cases. The intervention is potential to enable new spectrum of applications.

The fog-cloud intervention is stressed in [18] where the authors developed a definition that covers all the significant properties of fog computing. Their definition states: "fog computing is a geographically distributed computing architecture with a resource pool that consists of one or more ubiquitously connected heterogeneous

devices (including edge devices) at the edge of network and not exclusively, but seamlessly backed by cloud computing services, to collaboratively provide elastic computation, storage and communication (and many other new services and tasks) in isolated (sandboxed) environments to a large scale of clients in proximity". The definition succeeded to realize the strong impact of the collaboration between cloud computing, fog computing and edge devices consistently and intelligently in a very large scale system. This latter definition has been adopted for this work.

1.2 Fog Computing System Architecture

Extending the cloud resources to the edge results in a three-layer service model as shown in Figure 1. The three layers are:

1- ***Edge layer***: this is the lowest layer in the fog computing architecture. It consists of terminal nodes, embedded systems, and sensors with very limited computation, energy and bandwidth. Each edge device, with its limited networking capabilities, is connected to the fog layer.

2- ***Fog layer***: This is the fog computing layer which has dozens of thousands of intelligent intermediate networking devices such as routers, gateways, switches, and access points which work on different protocols like 3G, 4G, LTE, and WiFi. These devices are supported with computational and storage capabilities in this layer. Fog computing devices can interact with each other for load sharing and balance purposes, and each single one of them is connected to cloud layer.

3- ***Cloud layer***: This is the top most layer in the architecture. It consists of cloud data centers that have very rich virtual capabilities in terms of storage and processing power.

1.3 Scheduling in Fog Computing

Scheduling is generally defined as the allocation of tasks to capable resources at a specific time. Usually, a scheduling problem is subject to a number of constraints and objectives that must be fulfilled. Moreover, optimization concepts are typically involved when modeling scheduling problems. Scheduling objectives can be, for instance, system utilization maximization or completion time minimization. Scheduling problems are not restricted to computer field only; they also exist in other domains such as manufacturing and airline flights.

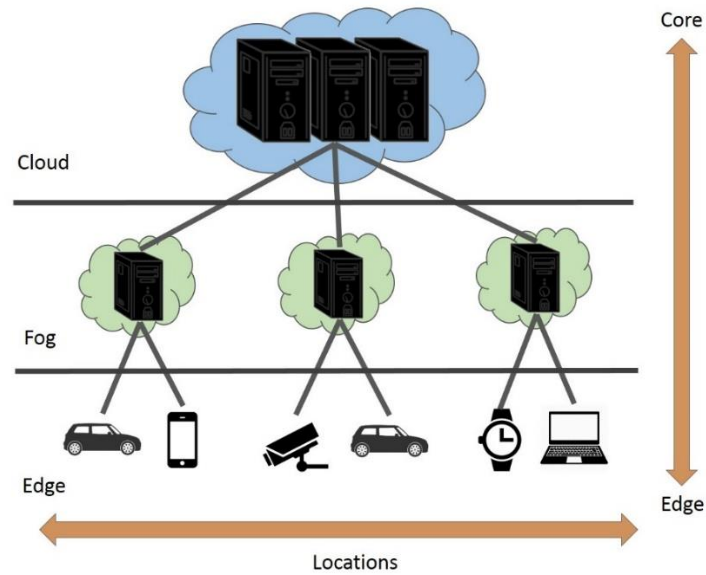

Figure 1: Edge-Fog-Cloud Architecture

According to [19], most of the scheduling problems consist of 4 basic elements:

1- Resources: physical/logical devices with the ability to execute or process tasks.
2- Tasks: the physical/logical operations that need to be executed by the resources.
3- Constraints: conditions must be regarded in scheduling the tasks into the resources. They may be operation-based, task-based, resource-based, or a combination of these. They could also be hard constraints, meaning constraints that must be full-filled or soft constraints can be relaxed.
4- Objectives: the evaluation criteria that need to be measured in order to assess the schedule performance.

To find an optimal solution to a scheduling problem, there are two broad categories of methods: Exact methods and heuristic methods. Exact methods find the absolute optimal solution to the scheduling problem. Examples of exact algorithms include Simplex and Branch-and-Bound. On the other hand, heuristic techniques do not guarantee finding the optimal solution. However, they are able to find a solution that has some degree of optimality in a reasonable computation time compared to the time required by an exact method to find the optimal solution. Examples of heuristic algorithms include: simulated annealing, ant colony algorithms and genetic algorithms.

1.4 Research Problem and Significance

The need for fog computing comes from the fact that cloud computing is not sufficient to satisfy requirements such as latency-sensitivity, mobility and location awareness [14, 20]. Cloud computing is facing many difficulties and challenges as mentioned in [3]. The first challenge is the massive growing number of IoT devices. This creates enormous traffic among cloud computing networks and consequently affects real-time or low-latency services. By adopting fog computing, less amount of data needs to be transmitted across the core of the networks which consequently leads to less bandwidth consumption. This helps in reducing congestion, traffic, cost and round-trip latency by eliminating the bottlenecks that exist in centralized platforms. "Milliseconds matter when you are trying to prevent manufacturing line shutdowns or restore electrical service. Analyzing data close to the device that collected the data can make the difference between averting disaster and a cascading system failure" [14].

Second, it is not practical to transport vast amounts of data from thousands or hundreds of thousands of edge devices to the cloud layer. It is also not necessary because many critical analyses do not require cloud-scale processing and storage. Integrating fog computing with cloud computing directs data to the optimum place for processing depending on the criticality of response and how fast the decision is needed. Time-sensitive decisions should be made closer to the things producing and acting on the data. In contrast, big data analytics on historical data needs the computing and storage resources of the cloud computing.

Third, the high operational cost of cloud computing data centers as they are confronted with service requests from IoT applications. Using fog computing, some of those requests can be served at the edge of the network. In this way, cloud computing centers get relieved significantly as they don't have to be running in full power all the time. Moreover, adopting the concept of edge computing within fog computing also provides high levels of scalability, reliability and fault tolerance [3].

In spite of all the challenges faced by cloud computing, fog computing is not a platform to compete with cloud. This is important as the goal is not to underestimate the power of cloud computing. Fog and cloud computing create a cooperative and comprehensive architecture in which each one completes what the other lacks. The interaction is expected to be a promising paradigm that will enable serving billions of IoT devices and applications with low latency. It will help significantly in monitoring

and managing such massive amounts of data generated from the IoT devices. Examples of these applications include but not limited to: industrial automation, transportation, live streaming, real-time and online gaming, augmented reality, connected vehicles, smart micro grid, and smart traffics [2, 3, 5].

1.5 Research Objective and Contribution

The objective of this research is to model the problem of scheduling IoT requests into resources available at both fog and cloud layers. The problem is modeled using integer programming in order to provide the minimum service time for IoT requests. The service latency is defined as the Round-Trip Time (RTT) for serving or processing an IoT request from the moment it gets initiated to the moment it gets completely processed and the results are returned back to the requesting device. This latency includes many delay components such as transmission delay, routing or queuing delay, propagation delay, and processing time, and waiting time as well.

The developed model is solved using Branch-and-Bound as an exact algorithm. However, Bitran *et al.* [21] proved that the scheduling problem is NP-Hard. NP-hardness (non-deterministic polynomial-time hard), in computational complexity theory, is a complexity class used to describe certain types of decision problems. Therefore, a heuristic will be developed to obtain a feasible solution with a good quality in a reasonable computational time. Genetic Algorithm (GA) is used as a heuristic approach for solving the integer programming model. The GA is studied using different problems with different sizes in order to evaluate the impact of changing the different model parameters and how they can be adjusted properly. After developing the GA, a comprehensive comparison is performed between the exact solution obtained for Branch-and-Bound algorithm and the heuristic solutions obtained from the GA.

The GA is then integrated within a real-time simulation environment to help in scheduling the requests as they arrive. The service latency provided by the hybrid fog-cloud architecture that implements GA is then compared to other systems with the same architecture but uses traditional scheduling algorithms, such as waited-fair queuing (WFQ), priority-strict queuing (PSQ), and round robin (RR).

The main contributions of this research include the following:

1. Reviewing the literature and the state-of-the-art research papers about the challenges of minimizing the service latency for real-time IoT applications and how fog and cloud computing are involved in solving this problem.
2. Developing an integer program that defines requests with their attributes and fog and cloud computing resources with their attributes. The model objective is to minimize the overall service latency. Then Branch-and-Bound algorithm is used as an exact approach for finding solutions of small size problems.
3. Developing a heuristic solver using Genetic Algorithm to obtain good quality solutions for large size problems within a reasonable computational time. The heuristic solutions are then compared with the exact solutions in terms of solution quality and computational time.
4. Comparing optimized service latency provided by the ILP model to following scheduling algorithms: WFQ, PSQ, and RR.

1.6 Research Methodology

The following steps were followed to achieve the outcomes of this research:

Step 1: The literature related to fog computing, cloud computing, scheduling, service latency optimization, and genetic algorithms is reviewed.

Step 2: An integer programming model is formulated for the scheduling problem that involves requests and resources with their characteristics called attributes. The model also includes assumptions, decision variables, objective function, and constraints.

Step 3: The formulated model is coded using Lingo optimization software for verification, validation and explanation purposes.

Step 4: The heuristic using Genetic Algorithms for solving large problems and for comparing the heuristic solutions to the exact solutions is developed.

Step 5: A simulation model is developed from the formulated model using the discrete event simulator SimEvent from Mathworks. The GA is integrated with the simulation so that it can be used as a solver to schedule the requests into the resources in real-time as they arrive.

Step 6: The GA service latency is compared to WFQ, PSQ, and RR algorithms.

1.7 Thesis Organization

In this chapter, an introduction has been given about fog computing, cloud computing, scheduling, the research significance, and the problem statement. Chapter 2 is dedicated to surveying relevant literature on fog computing, scheduling techniques, architecture, characteristics and similar concepts. Chapter 3 introduces the proposed mathematical model that represents the problem mentioned in this research with an illustrative numerical examples. Chapter 4 presents the Genetic Algorithm, its implementation, experimentation and comparison to the exact methods solutions. Chapter 5 contains the developed simulation and its experimentation. Finally, Chapter 6 gives the conclusion and prospect future work.

Chapter 2. Literature Review

The literature reviewed in this work will cover many areas that have interconnection with fog computing. This includes cloud computing, internet of things, mobile cloud computing and mobile edge computing. A major area that the reviewed literature will focus on is latency minimization using different types of algorithms.

2.1 Characteristics of Fog Computing versus Cloud Computing

The main factor that distinguishes fog computing from cloud computing is its closeness to end users. As in fog layer, the services can be hosted at edge devices such as access points, routers, switches, base stations, and even end devices. Fog computing, being at the edge of the network implies a list of characteristics mentioned in [2, 5, 16, 22, 23]. Table 1 recaps these characteristics and presents a cloud-vs-fog computing comparison.

Table 1: Fog versus Cloud Characteristics

Cloud computing characteristics	Fog computing characteristics
Vertical resource scaling	Vertical and horizontal resource scaling
Large-size and centralized	Small-size and spatially distributed
Multi-hop WAN-based access	Single-hop WLAN-based access
High communication latency and service deployment	Low communication latency and service deployment
Ubiquitous coverage and fault-resilient	Intermittent coverage and fault-sensitive
Context-unawareness	Context awareness
Limited support to device mobility	Full support to device mobility
Support to computing-intensive delay-tolerant analytics	Support to real-time streaming applications
Unlimited power supply (exploitation of electrical grids)	Limited power supply (exploitation of renewable energy)
Limited support to the device heterogeneity	Full support to the device heterogeneity
VM-based resource virtualization	Container-based resource virtualization
High inter-application isolation	Reduced inter-application isolation

Fog computing devices are provided with 'Intelligence' that makes them able to decide whether a request needs to be served in the fog layer or pushed up to the cloud layer. This is achieved using smart gateways as in [8]. The jobs that fog nodes are able to perform include, but not limited to, collecting data, processing, filtering data, monitoring status of end devices and uploading what needs to be uploaded to the cloud layer. The purpose of fog computing is delivering services for specific type of applications or requests that demand real-time and predictable latency (like industrial automation, transportation, live streaming, online gaming, connected vehicles, and smart traffics). In contrast, the requests that require cloud computing services rather than fog computing (like long term storage, analysis, and business intelligence) are transferred to the cloud layer. Fog computing devices act only as routers or gateways forwarding these requests.

2.2 Other Platforms Similar to Fog Computing:

There are many other similar concepts that overlap with fog computing, however, they are different. These concepts include:

1- ***Local cloud***: Local cloud is a complementary model for the public traditional cloud computing. Its main purpose is to run specific services in a local network to essentially strengthen the security of the computing environment. The local servers will be running cloud-enabling software and in most cases they support interplay with the public cloud layer.

2- ***Cloudlet***: Cloudlet is "a data center in a box" [18]. It is a secured resource-rich computer or cluster of computers that is well-connected to distant cloud on the Internet and these resources can be leveraged by usually-few nearby mobile devices [15]. The physical proximity of Cloudlet to users is very essential because it makes the end-to-end service time fast and predictable as it becomes only one-hop network latency. It also helps meeting the peak bandwidth service demand of real-time/interactive response for generating and receiving media such as high-definition video and high-resolution images.

3- ***Mobile Cloud Computing (MCC):*** MCC is defined as a model that combines mobile computing and cloud computing. In MCC the cloud is basically designed to remotely handle the large data storage and processing requirements for the mobile devices [11]. The remote cloud computing servers don't necessarily

have to have rich and powerful resources. They could at least cooperate with mobile devices for storage and processing [24].

4- ***Mobile Edge Computing (MEC).*** MEC, like fog computing, has the same concept of pushing "intelligence" to the edge of the network. It is also very similar to Cloudlet except that it is primarily located where the data originates within the range of Radio Access Network such as mobile base stations or access points [18]. The MEC servers are located at the ultimate edge of the network to perform specific (business oriented) tasks and jobs that are just not convenient to be executed on the traditional cloud computing [11]. In [25], the authors proposed this definition for MEC: "Mobile Edge Computing is a model for enabling business oriented, cloud computing platform within the radio access network at the close proximity of mobile subscribers to serve delay sensitive, context aware applications". According to [25], the prime objectives of MEC are: 1) Optimization of mobile resources by offloading the compute/storage-intensive tasks off the edge devices. 2) Optimization of the large data before sending to the cloud layer. 3) Reducing the latency by enabling cloud computing services within physical proximity to mobile subscribers. The MEC "sits" on the link the data pass through so that it can actively analyze and respond to user requests. The MEC servers are usual servers equipped with CPUs, memory, and communicating interfaces. Using MEC paradigm, it helps achieving less latency performance and less bandwidth consumption. Application of MEC include smart grids, smart transportation or smart traffic, video streaming, mobile big data analytics, mobile gaming, edge health care, and sensor networks application.

2.3 Implementations within Fog Computing

At present, there are only some existing studies in the literature in terms of design, algorithms or implementations within fog computing. In [26], Cisco has introduced Cisco IOx. Cisco IOx is a hosting environment for IoT applications. IOx brings the application execution capability down to the source of IoT data. IOx offers steady and consistent hosting across multiple network infrastructure devices such as Cisco routers, switches, and compute modules. In [10], Stojmenovic *et al.* addressed Cloudlet as a special case of fog computing. It is an intermediate layer located between

the cloud data centers and each mobile device. The edge devices connect wirelessly to the closest Cloudlet rather than accessing the far cloud data center. The authors also introduced fog computing within smart grids where each micro-grid can be depicted as a fog computing node. Customers communicate to nearby fog nodes rather than the remote cloud. Fog devices will coordinate with cloud data centers and customers in order to deliver power services. In [4], the authors presented a high level programming model for future internet applications that are characterized by being geospatially distributed, large-scale and latency-sensitive. Fog computing resources are allocated for serving the low-latency services while tolerant larger scope services that need aggregation are allocated in cloud layer. Reference [23] studied web optimization within fog computing context. Fog servers connect the end users to the internet so that all web requests have to pass through the fog servers on their way to the web servers of the cloud layer. There are two possible process flows for the web requests: (1) optimizing webpage for its initial request and (2) optimizing webpage for its subsequent request(s) where all the needed files and web objects are cached and stored locally within the fog server. In [27], Ottenwälder *et al.* introduced a placement and migration method for providers of infrastructures that involved the fog and the cloud computing resources. It works by planning the migration ahead of time so that it can meet the defined end-to-end service latency and at the same time it reduces the network bandwidth consumption. According to [28], mobile cloud is a very similar paradigm to fog computing in which the resources are shared not only from central data centers but also from pervasive mobile devices. Although these devices have heterogeneous resources (e. g. CPUs, bandwidth, content) and support services, they still have the ability to share these resources. Based on the key concept of service-oriented utility functions, the authors proposed an architecture and mathematical framework for heterogeneous resource sharing. The heterogeneous resources are most probably measured in dissimilar scales/units (e.g. power, bandwidth, latency). For this reason, the authors adopted a unified framework on their work where all quantities are mapped to time only. They also formulated their model for optimization and found the optimal solution using convex optimization approaches.

In [9, 11, 20], the security and privacy issues within fog computing were studied. The authors discussed security issues such as authentication, secure data storage, secure computation, access control, and network security. Fog computing,

being at the edge of the network, makes it surrounded by many security threats such as the Man-in-the-Middle Attack. These threats may not be easily avoided using the security solutions that exist in the context of cloud computing. They also highlighted privacy issues such as data privacy, usage privacy, and location privacy. The vicinity and the physical proximity of fog computing nodes to end users give it the ability to collect and leak sensitive information like data, location and usage. The same application code is applied on various devices for different large–scale applications in the fog and the cloud layers. In terms of real-world applications, Cao *et al.* in [29] designed fall detection algorithms and designed and employed a real-time fall detection system. Basically, they split the fall detection task between the edge devices and the server (e.g., servers in the cloud layer). The sensor data are transmitted to the fog server in real-time so that both edge device and fog server perform the computation for fall accurate detection. While in [30] and [31], the authors presented an intelligent mechanism that can dynamically choose whether it is beneficial to offload parts of the computation off the mobile devices or not and where to offload if so. That is achieved by monitoring all the available fog resources and their runtime configurations (e.g., the network latency and the bandwidth between the mobile device and the server, the size of the overhead data, etc.)

2.4 Scheduling Techniques

Simplex [32] as an exact solution technique, operates on what is called simplicial cones. These simplicial cones are the edge or the neighborhoods of the vertices of a geometric object called a polytope. The main idea of simplex algorithm is the replacement of the objectives and constraints set of a formulated problem by an alternative convex shape of feasible points and extreme rays. The algorithm begins at a starting vertex and moves along the edges of the polytope targeting the edge of the optimum solution within the feasible region. Branch-and-Bound (B&B) algorithm is an approach that works by means of enumeration of all possibilities which are finite in number [33, 34]. However, explicit enumeration is normally impossible due to the exponentially increasing number of potential solutions. Sometimes many possibilities can be implicitly eliminated by domination or feasibility arguments. The use of bounds for the function to be optimized combined with the value of the current best solution enables the algorithm to search parts of the solution space only implicitly. This is one

of the strong advantages of B&B that it throws out large parts of the search space by using previous estimates on the objective function under concern. This helps in eliminating large parts of the search space and reduces the runtime significantly.

On the heuristics side, one of the important algorithms is simulated annealing. Simulated annealing is based on ideas from statistical mechanics and is inspired by an analogy to the physical annealing of a solid [35, 36]. To set some material into a low-energy state it will be heated to a high temperature and then cooled very slowly. This process allows coming to thermal equilibrium at each temperature. The system is expected to acquire a low-energy state at freezing temperatures. The algorithm begins with a randomly generated solution at a high (artificial) temperature and then the temperature is reduced gradually until it reaches freezing point. The regions in the solution space are searched at each temperature by an algorithm called the Metropolis algorithm. An iteration of the Metropolis algorithm starts with implanting random perturbations to the candidate solutions and evaluating the impact on the solution quality [36]. Along with simulated annealing, genetic algorithm is also an important heuristic approach in the literature. The latter is founded based on the natural evolution mechanism [37-39]. The algorithm simulates the natural populations' reproduction operations in the process of achieving efficient optimized solutions. Through different generations, the algorithm searches for variations to evolve the solutions. The population consists of chromosomes, also called individuals, and all these chromosomes represent possible solutions to the problem regardless the feasibility. Each chromosome is associated with a fitness value to represent how 'good' it is compared to the others. This fitness function depends on the objective of the problem under concern.

Additionally, ant colony algorithm is also a heuristic algorithm that is taken from 'ants' and how they can manage to find shortest paths from their colony to feeding sources and backward [40]. These ants communicate using what are called pheromone trails. A moving ant lays some pheromone in different quantities in order to make the path it is following by a trail of this substance. Whereas the movement of an isolated ant is random, an ant encountering a previously laid trail can detect it and decide with high probability to follow it. The overall behavior that arises is a form of a behavior in which as more ants follow a trail, that trail becomes most probable of being followed. This process depicts a positive feedback loop, where the likelihood with which an ant

selects a specific path grows up with the number of ants that previously selected to take the same path.

2.5 Fog Computing and IoT Interconnection

Fog computing will play a crucial role in upcoming internet of things (IoT) applications [1, 6, 14]. The huge range of IoT applications that could be built on fog computing platform makes it a promising paradigm. In IoT, the end devices or "things" are provided with unique identifiers (for example IP address, RFID, NFC tag, bar code or QR code) and they are able to transfer data over the network without involving a human-being during the data transfer process [5]. These "things" are also equipped with embedded electronic components, software components, sensors, controllers and networking capabilities in a way that makes it able to communicate and exchange data with each other. Examples of future scenarios and applications of IoT that can be developed and built upon fog computing services include Smart Traffic Light System, Wind Farm, Connected Vehicles, Smart Grid, and Wireless Sensor Networks.

2.6 Latency Optimization using Genetic Algorithms

In [39], the authors proposed a budget constraint based scheduling model to minimize execution time while meeting a specified budget for delivering results. They modeled the workflow application as a Directed Acyclic Graph (DAG). The developed genetic algorithm is used to solve the scheduling optimization problem with a cost-fitness and time-fitness. The algorithm was tested in a simulated Grid environment. In [41], Buyya *et al.* proposed a genetic algorithm approach for scheduling workflow applications by either minimizing the monetary cost while meeting users' budget constraints, or minimizing the execution time while meeting users' deadline constraints. They evaluate the approach for balanced and unbalanced workflow structures. In [42], different existing approaches were comparatively examined for scheduling of scientific workflow applications in Grid environments. Three algorithms were evaluated; one of them is using Genetic algorithms (GA). The authors also studied the incremental workflow partitioning against the full-DAG-graph scheduling strategy. They demonstrated experiments using real-world scientific applications covering both balanced (symmetric) and unbalanced (asymmetric) workflows. In [43], Rodriguez *et al.* proposed a resource provisioning and scheduling strategy for scientific workflows

on Infrastructure as a Service (IaaS) cloud environments. The authors presented an algorithm based on the meta-heuristic optimization technique named particle swarm optimization (PSO). This algorithm aims to minimize the overall workflow execution cost while meeting deadline constraints. Their heuristic is evaluated using CloudSim simulation tool and various well-known scientific workflows of different sizes. In [44], the authors worked on deadline sensitive leases which can be scheduled using traditional backfilling algorithm. However, in the backfilling algorithm one of the leases is selected from the best effort queue which will provide the free resources to schedule the deadline sensitive lease. However, in some scenarios, backfilling algorithm does not provide better scheduling if there are similar types of leases and must be in conjugative in sequence. For this reason, the authors used an algorithm called AHP (Analytic Hierarchy Process) as a decision maker with the backfilling algorithm. AHP helps in choosing a possible best lease from a given best effort queue in order to schedule deadline sensitive leases. In [45], Mao *et al.* presented an auto-scaling mechanism for cloud computing environments. In their approach, the cloud resources are considered as virtual machines (VMs) of various sizes/costs. The jobs are specified as workflows where users specify performance requirements by assigning (soft) deadlines to jobs. The goal is to ensure all jobs are finished within their deadlines at minimum financial cost. They used the Earliest Deadline First (EDF) algorithm to schedule tasks on each VM type. After deadline assignment and instance consolidation, every task is scheduled to a VM type. They sort the tasks by their deadlines for each VM type, and schedule the task with the earliest deadline whenever an instance is available. In [46], the authors proposed a Dynamic Critical Path (DCP) based workflow scheduling algorithm that determines efficient mapping of tasks by calculating the critical path in the workflow task graph at every step. It assigns priority to a task in the critical path which is estimated to complete earlier. They compared the performance of their proposed approach with other existing heuristic and meta-heuristic based scheduling strategies for different types and sizes of workflows. In [47], Chen *et al.* aimed at proposing an ant colony optimization (ACO) algorithm to schedule large-scale work-flows with various QoS parameters. This algorithm enables users to specify their QoS preferences as well as define the minimum QoS thresholds for a certain application. The objective of this algorithm is to find a solution that meets all QoS constraints and optimizes the user-preferred QoS parameter.

Chapter 3. Modeling the Problem

3.1 Environment Analysis

As discussed in the literature review (Chapter 2, section 2.2), the integration between fog computing and cloud computing creates a 3-layered architecture. The edge devices reside at the edge of the network. Fog computing devices "sit" in the intermediate layer while cloud resources are in the upper most layer. The 3-layered architecture with the data flow direction is shown in Figure 2.

IoT devices, residing in the edge layer, require services from fog and cloud resources by sending requests that need to be processed. Based on [48], the IoT requests can be described or defined as data, with specific size, that need to be processed or analyzed. The analysis or processing stage refers to some computational processes that have to take place in fog or cloud layer.

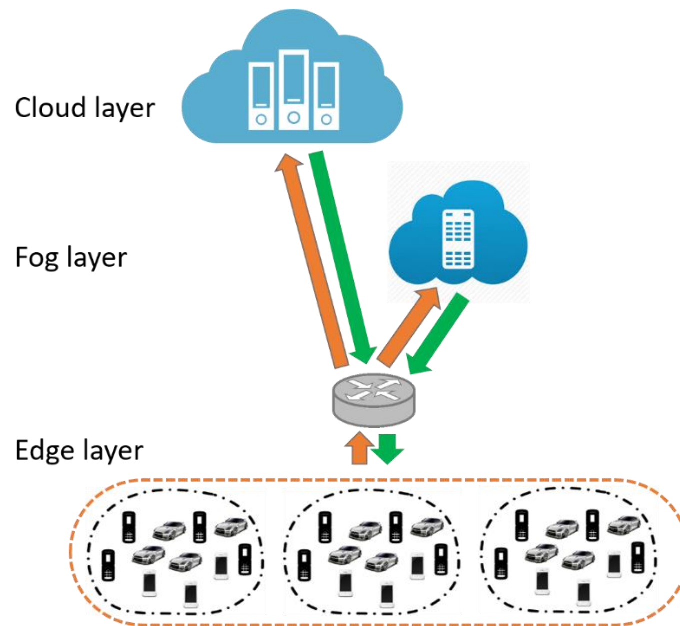

Figure 2: Fog and Cloud Computing Architecture of Service Scheduling

The main objective of the model is to minimize the latency as round-trip time (RTT) for serving or processing the requests coming from the edge layer. For this model, the latency of an IoT request is specifically defined as *the round trip time from the moment the request gets initiated at the edge layer to the moment it gets completely processed or served and the results are returned back to the requester*. This latency includes many delay components.

In networking, the latency is generally divided into several components:

1- ***Transmission delay***: this is the time it takes to push the packet's bits onto the networking or connection link.
2- ***Queuing or networking delay***: this is the time the data packets spend in passing through the network routers and switches.
3- ***Propagation delay***: this is the time for a signal to propagate or travel through the networking media and reach its destination. In this work, the propagation delay is completely neglected as it is insignificant compared to the other delay components.

Transmission delay is a function of the packet's length and has nothing to do with the distance between the two nodes. It is calculated from the transmission bit rate and the size of data to transmit and hence it is uniform and can be evaluated beforehand. On the contrary, the queuing time is stochastic with a certain average and it cannot be evaluated beforehand. For this reason, the total delay of summing the transmission and queuing time will also be stochastic with a certain average. This latter delay can be written as in equation (1). The Total delay is the summation of the two delay components, transmission and the queuing time.

$$\bar{\delta} = \delta_{transmission} + \bar{\delta}_{queuing} \quad (1)$$

Additionally, there is the actual processing time which is the time it takes to process the data of each request. Examples of IoT data processing are data storage, data aggregation and analysis, features extraction, images and video processing, etc. This processing time is defined by the data size each request has and the processing capability of the resource that will serve or process the request. The resource capability is defined in terms of processing speed. Moreover, a request might suffer some additional delay if the resource is busy serving other requests. The reason behind that is, it is assumed that a resource can process only one request at a time in the modeling.

3.2 The Edge-Fog-Cloud Environment Model

The scheduling problem model is built based on the following environment settings or assumptions:

- There is a set of fog and cloud computing resources (*m* resources) denoted as set by $S = \{\mathrm{S_1}, \mathrm{S_2}, \mathrm{S_3}, \dots, \mathrm{S_m}\}$. Each resource has its own attributes: processing power and average networking delay.

- The processing speed of each resource is measured as the number of packets that can be processed per unit time.
- No communication or cooperation exists between the processing nodes, at both fog and cloud layers.
- A resource, at both fog and cloud layers, can process only one request at a time.
- There is a set of n requests that need to be processed individually at fog and cloud layers. These requests are denoted as a set by $R = \{R_1, R_2, R_3, \dots, R_n\}$. Each request has its own attributes: initiation time, data size, priority and deadline.
- A request may get created (initiated) at any instant of time.
- The data size of each request is defined by the number of packets the request has, assuming that the size of one data packet in all requests is fixed.
- Each request priority is assigned as a normalized fractional weight to signify how important the request is compared to others. All requests weights are summed up to an exact 1 as shown in equation (2). The reason behind adopting this fractional weight is to define the most important requests that need to be served early even if some other requests with low priority take longer time.

$$\sum_{j=1}^{n} W_j = 1 \quad (2)$$

- The deadline requirement of each request is defined as a time duration that starts from the moment it gets created, not referenced to time 0.
- A request consists of only one operation or one task that needs to be performed by one resource.
- Each request experiences transmission and queuing delay in the forward journey moving from edge to fog or cloud layers to get processed. This transmission and queuing delay is a function of two parameters: the first one is the request data size in terms of packets, and the second one is the transmission and queuing delay per packet of the resource the request will be served at. This transmission and queuing time represents how far the resource is from the edge layer.
- The requests will also suffer the same transmission and queuing delay in the backward journey from fog or cloud layer to edge layer. This is because it is assumed that the size of the resulted data after processing is equal to the original data.

- The actual processing time of each request is equal to the request size (in packets) divided by the processing node speed (in packets per second).
- All requests in the model are independent.
- Preemption is not allowed. If a request starts processing, it must finish without interruption.
- All the needed scheduling problem parameters are known by the time they will be solved. The scheduling problem parameters are the number of requests and resources with their attributes.
- The objective in the model is achieving the least overall latency possible for serving all requests considering their different priorities or weights as explained in equation (2).

  Time in this model is continuous, not discrete-valued. This way, the latency may be any positive real-valued number and there is no limit to the time horizon that the schedule must be performed within. It also means that any request can be processed at any instant of time.

These environment characteristics can be translated into the following notations list:

$\boldsymbol{LT}$: Average weighted overall latency.

$\boldsymbol{LT_{ij}}$: Latency of request R_j that is served in resource S_i.

$\boldsymbol{n}$: Number of requests of a set of requests R numbered from R_1 to R_n.

$\boldsymbol{R_{size_j}}$: Data size of request R_j.

$\boldsymbol{R_{init_j}}$: Initiation time of request R_j.

$\boldsymbol{R_{deadline_j}}$: Deadline of processing request R_j.

$\boldsymbol{R_{pri_j}}$: Weighting factor of request R_j describing its priority.

$\boldsymbol{m}$: Number of resources exist in both fog and cloud with a set name S, numbered from S_1 to S_m.

$\boldsymbol{P_i}$: The processing power of resource S_i.

$\boldsymbol{\overline{\delta}_i}$: The average transmission and queuing delay per packet for reaching resource S_i.

$\boldsymbol{x_{ij}}$: A set of 0-1 variables such that x_{ij} equals 1 iff request R_j is allocated in a resource S_i.

Θ_{ijk}: A set of 0-1 variables such that Θ_{jk} equals 1 iff request R_j should be executed before R_k in the same resource, S_i.

$\boldsymbol{ST_{ij}}$: Start time of actual processing of request R_j within resource S_i.

$\boldsymbol{PT_{ij}}$: Processing time of requests R_j in resource S_i.

$\boldsymbol{TQT_{ij}}$: Transmission and queuing time of requests R_j to reach resource S_i.

The objective in the model is minimizing the sum of the RTT for serving all *n* requests using *m* resources available, considering the requests different priorities or latency costs. Mathematically, the latency, which is the objective function, is the difference between the initiation time and the end of service which includes the starting time, the processing time, and the transmission and queuing time. Then the latency of each request is weighted by its priority in order to evaluate the overall latency, as shown in equation (3).

$$\min\left\{LT = \sum_{i=1}^{m}\sum_{j=1}^{n}\left(LT_{ij} * R_{pri_j} * x_{ij}\right)\right\}$$
$$LT_{ij} = ST_{ij} + PT_{ij} + TQT_{ij} - R_{init_j} \tag{3}$$
$$\forall R_j \in R, \forall S_i \in S$$

Where:

$$PT_{ij} = {R_{size_j}}/{P_i} \qquad \forall R_j \in R, \forall S_i \in S \tag{4}$$

Equation (4) defines the processing time of request R_j in resource S_i. The processing time of each request is equal to the request data size in packets divided by the resource processing power in packets per second.

$$TQT_{ij} = \sum_{1}^{R_{size_j}} \bar{\delta}_i \qquad \forall R_j \in R, \forall S_i \in S \tag{5}$$

Equation (5) defines the transmission and queuing time of request R_j from the edge layer to fog or cloud resource S_i. A request delay is the summation of its packets delays that will suffer individual delays distributed around a specific mean. This means a request cannot start processing unless all its packets reach the resource.

Subject to:

$$\textstyle\sum_{i=1}^{m} x_{ij} = 1 \qquad \forall R_j \in R \tag{6}$$

Equation (6) means request R_j must be served and served only once by only one resource.

$$ST_{ij} \geq R_{init_j} + TQT_{ij} \qquad \forall R_j \in R \qquad (7)$$

Equation (7) means a request R_j cannot start processing before its initiation time plus the transmission and queuing time to where fog or cloud resources reside.

$$\begin{gathered} if \quad x_{ij} + x_{ik} = 2 \quad then \quad \Theta_{ijk} + \Theta_{ikj} = 1 \\ ST_{ik} \geq \Theta_{ijk} * (ST_{ij} + PT_{ij}) \\ \forall (R_j, R_k) \in R, R_j \neq R_k, \forall S_i \in S \end{gathered} \qquad (8)$$

Equation (8) means a resource S_i can process only one request at a time. If there are two requests arrive the resource at the same time, one of them should be shifted to start after the other.

$$ST_{ij} + PT_{ij} + TQT_{ij} - R_{init_j} \leq R_{deadline_j} \qquad \forall R_j \in R, \forall S_i \in S \qquad (9)$$

Equation (9) means each request R_j must be served within its deadline requirement $R_{deadline_j}$.

3.3 Model validation using Lingo

In this section, the model described in the previous section is validated using a software called Lingo. Lingo is an optimization modeling software for linear and nonlinear integer optimization models. It supports many exact algorithms such as simplex and Branch-and-Bound (B&B). We adopted B&B as an exact algorithm for solving the model. The reason behind that is, B&B throws out large parts of the search space by using previous estimates on the objective function under concern. This helps in eliminating large parts of the search space and reduces the runtime significantly.

To validate the model, a small scheduling problem that involves only 5 requests and 2 resources was built for demonstration purposes. The two resources represent 1 fog processing node and 1 cloud processing node. The attributes of these requests and resources are shown in Table 2 and Table 3. As seen from Table 2, all 5 requests are assumed to be initiated at the same time and they have the same data size for demonstration purposes. However, they have different priorities and deadline requirements. The 2 resources attributes are shown in Table 3. One of them represents a fog resource with lower delay and lower processing power compared to the other one with higher delay and higher processing power representing a cloud resource.

Table 2: 5 Requests with Their Associated Attributes

Requests, R_j	**Initiation time, R_{init_j}**	**Data size, R_{size_j}**	**Priority, R_{pri_j}**	**Deadline, $R_{deadline_j}$**
R_1	15	5000	0.2	1000
R_2	15	5000	0.38	1500
R_3	15	5000	0.01	2000
R_4	15	5000	0.4	600
R_5	15	5000	0.01	2000

Table 3: 2 Resources with Their Associated Attributes

Resource, S_i	***Processing power, P_i***	***Average delay, $\bar{\delta}_i$***
S_1 (Fog)	20	0.001
S_2 (Cloud)	100	0.1

Lingo software gives the absolute optimal solution for the scheduling problem using the objective function given in equation (3). Table 4 and Table 5 show this optimal solution with its associated parameters. The solution parameters are the allocation of the requests (x_{ij}) within the 2 resources with 1's and 0's. The starting time of each request (ST_j), the processing time (PT_{ij}), and the transmission and queuing delay (TQT_{ij}), the latency (LT_{ij}) of each request which must be less than deadline ($R_{deadline_j}$) and the overall weighted latency (LT). Table 5 shows the other part of the solution in terms of the order of execution (Θ_{ijk}) for each two requests allocated to the same resource as the resource can execute only one request at a time (refer to the constraint in equation (8)).

As the latency of each request, LT_{ij}, is given in Table 4, their summation can be simply obtained 4070 as shown in the table. However, that is not the actual objective function described by equation (3). Equation (3) defines the overall weighted latency considering requests' different weights. This latter one can be calculated as 627.5 as shown in the last column. The total non-weighted latency is just the summation of LT_{ij} while the weighted latency is the summation of multiplying LT_{ij} by R_{pri_j}. This way,

the model will give less delay to the requests that have higher priority in order to minimize the overall weighted latency. This difference between the latency and the weighted latency is a core concept in the formulated model.

In the allocation column in Table 4, it can be noticed that R1 (request 1) and R4 are allocated in S1 (resource 1) while the rest R2, R3, and R5 are allocated in S2. The order of processing requests that are allocated in the same resource is given in Table 5. Be reminded that in the model, it is assumed that each resource can process only one request at a time. For instance, it can be seen that $\Theta_{4,1}$is equal to 1 which means R_4 precedes R_1 in the execution phase. This means, the opposite parameter, $\Theta_{1,4}$ should equal to zero according to equation (8), as shown in Table 5. This also applies for the other 3 requests R_2, R_3 and R_5 allocated in S_2.

To make the scheduling clear, Table 4 and Table 5 are mapped into a Gantt chart shown in Figure 3. It can be seen from the graph that, while R_1 and R_4 are allocated in the same resource S_1, R_4 gets executed first, as Table 5 also stated. The moment R_4 finished processing, the resource is released and R_1 starts executing because it was kept waiting. On the other hand, R_2, R_3 and R_5 as they are allocated in S_2, the resource will process R_2 followed by R_3 followed by R_5, as Table 5 stated.

What can be concluded from the results shown in Table 4 and Table 5 is that, the allocation of the requests within the resources and their execution order gets affected directly by the requests priorities (weights). We can see that the highest prioritized request R_4 is scheduled in S_1 (which is a fog resource with less $\bar{\delta}$ compared to S_2) and it gets executed before R_1 (which is allocated in the same resource S_1). The main reason for that is having the weight factor in the overall weighted latency given in equation (3). As R_4 has more weight (priority), the optimizer will try to give the least latency possible for this specific request in order to minimize the overall latency.

Table 4: Lingo Optimal Scheduling Solution – Part 1

Requ-ests, R_i	**Allocation, x_{ij}**		**Start time, ST_{ij}**	**Proc. time, PT_{ij}**	**Trans. time, TQT_{ij}**	**Requests latency, $LT_{ij} < R_{deadline_j}$**	**Weighted overall latency, $LT_{ij} * R_{pri_j}$**
	S_1 (Fog)	S_2 (Cloud)					
R_1	1	0	270	250	5	510 < 1000	510 * 0.2
R_2	0	1	515	50	500	1050 < 1500	1050 * 0.38
R_3	0	1	615	50	500	1150 < 2000	1150 * 0.01
R_4	1	0	20	250	5	260 < 600	260 * 0.4
R_5	0	1	565	50	500	1100 < 2000	1100 * 0.01
						$\sum = 4070$	**$\sum = 627.5$**

Table 5: Lingo Optimal Scheduling Solution – Part 2

Resource, S_i	**Allocated requests, R_j**	θ_{ijk}	**Order of processing, $R_j \rightarrow R_k$**
S_1	R_1, R_4	$\theta_{141} = 1, \theta_{114} = 0$	$R_1 \rightarrow R_4$
S_2	R_2, R_3, R_5	$\theta_{223} = 1, \theta_{232} = 0$ $\theta_{225} = 1, \theta_{252} = 0$ $\theta_{235} = 1, \theta_{253} = 0$	$R_2 \rightarrow R_3 \rightarrow R_5$

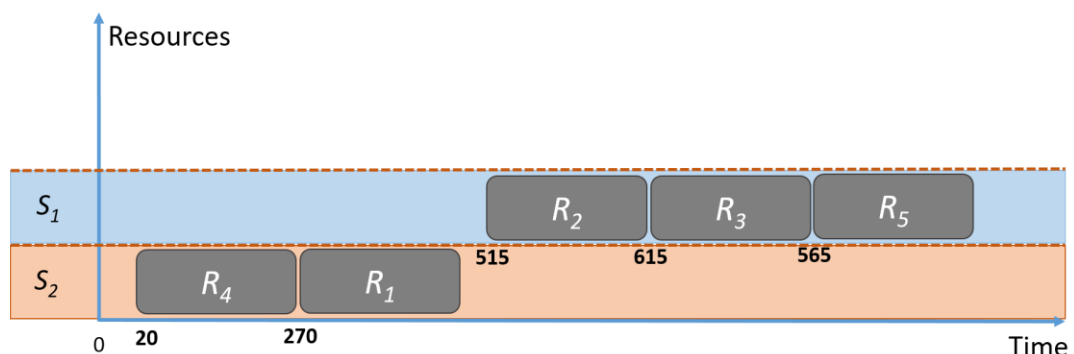

Figure 3: Gantt chart for Lingo Optimal Scheduling Solution

Chapter 4. Proposed Solution

4.1 Genetic Algorithms

In this thesis, the GA is adopted for solving the large size scheduling problems. The main reason behind using the GA as a heuristic approach is the complexity of the scheduling problem in general. It is not efficient to use an exact method for solving large scheduling problems that have hundreds or thousands of requests and resources as the time it takes to find the optimal solution grows significantly. In this section, before looking into solving the model, the GA and its implementation will be introduced.

GA is a meta-heuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms. In a GA, a population of candidate chromosomes to an optimization problem is evolved toward better ones. The evolution usually starts from a population of randomly generated chromosomes, and is an iterative process, with the population in each iteration called a generation. The GA iterates towards getting better chromosomes in terms of the objective. In each generation, the fitness of every chromosome in the population is evaluated. The GA fitness is the value of the objective function in the optimization problem being solved. The chromosomes also get evaluated for any feasibility conditions that exist in the optimization problem. The fitter chromosomes get stochastically selected from the current population, and each chromosome is modified (using the so called GA operators) to form a new generation. The new generation of chromosomes is then used in the next iteration of the algorithm.

The GA algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached, or there is no improvement in the fitness for a certain number of generations. A typical genetic algorithm requires a full representation or description to the optimization problem under concern. It requires a definition of the fitness function and all the constraints exist in the problem in order to evaluate the solution domain.

For a scheduling problem, the chromosome actually represents a candidate solution to the problem whether it is feasible or not. A candidate solution to the scheduling problem can be the allocation array of requests within resources, x_{ij}, as shown in Figure 4. What is shown in Figure 4 is just an example of allocating N requests

within M resources. The 0 means not allocated and the 1 means allocated. The 0's and 1's are "Genes". For instance, it is clear that R_1 (request 1) is allocated in S_3 (resource 3) and R_2 is allocated in S_2. This 2- dimensional array can be stretched vertically in order to get a 1-dimensional solution array that represents the chromosome for the GA. The latter one is shown in Figure 5 and it represents the exact same 2 dimensional array in Figure 4.

It is important to note that there is also another dimension in the scheduling problem solution which is the execution order of requests that are allocated to the same resource (refer to equation (8) or Table 5). This is an important parameter because it gives the time dimension to the scheduling problem and distinguishes it for a traditional allocation problem. This parameter is not considered as part of the chromosome. However, it is considered in the latency evaluation of each chromosome.

Requests

Resources

	R1	R2	R3	...	RN
S1	0	0	0	...	0
S2	0	1	0	...	0
S3	1	0	0	...	1
...	...	...	...	...	0
SM	0	0	1		0

Genes

Figure 4: Chromosome Representation as 2-D Array

R1					R2					R3					...
S1	S2	S3	...	SM	S1	S2	S3	...	SM	S1	S2	S3	...	SM	...
0	0	1	...	0	0	1	0	...	0	0	0	0	...	1	...

Figure 5: Chromosome Representation as 1-D Array

4.2 The GA Implementation

The GA is developed as described in the implementation flowchart given in Figure 6. The GA starts by setting the values for major parameters such as the population size, the termination counter, the GA maximum number of iterations, and the number of requests and the number of resources. This is followed by assigning these requests and resources attributes from the user. The requests attributes are: data size,

priority, initiation time, and deadline. While the resources attributes are: processing power and average transmission and queuing delay.

A pseudocode is provided for the GA implementation in Table 6. The GA initially generates a certain number of candidate solutions (chromosomes) randomly to represent the initial population for the first generation. The number of chromosomes in the population is defined by the population size in the problem. The next step is to evaluate these chromosomes individually based on the feasibility constraints and the fitness function available in the model. Note that since the problem is a minimizing problem, the fitness is inversely proportionate to the latency in equation (3) and this is shown is equation (10). This means the less latency the better fitness value and the more latency the worse fitness value for each chromosome.

$$Fitness = 1/Latency \quad (10)$$

The following step is to sort the chromosomes according to their fitness. The algorithm scans all the chromosomes looking for the best chromosome that is better than the "best-so-far" one. If a better chromosome is found, the "best-so-far" will be replaced and a counter will be reset to zero. This counter is called the "trials counter" and it is used for holding the number of iterations the GA loops without finding a better solution. If this counter reaches the GA termination counter, the algorithm breaks.

After that, the GA implements what is known as the "GA operators". The GA operators are 3 operations; selection, crossover, and mutation. These operations are used to evolve the current population and produce a new population that can have fitter chromosomes. After implementing the GA operators on the current population, the chromosomes' fitness will be re-evaluated and they will be re-sorted to get a new generation. The GA keeps iterating, implementing the GA operators, generating newer generations, and sorting the chromosomes while looking for a better chromosome than the "best-so-far". At the end, it terminates either by consuming the termination counter (as explained in the previous paragraph) or by consuming the maximum number of iterations.

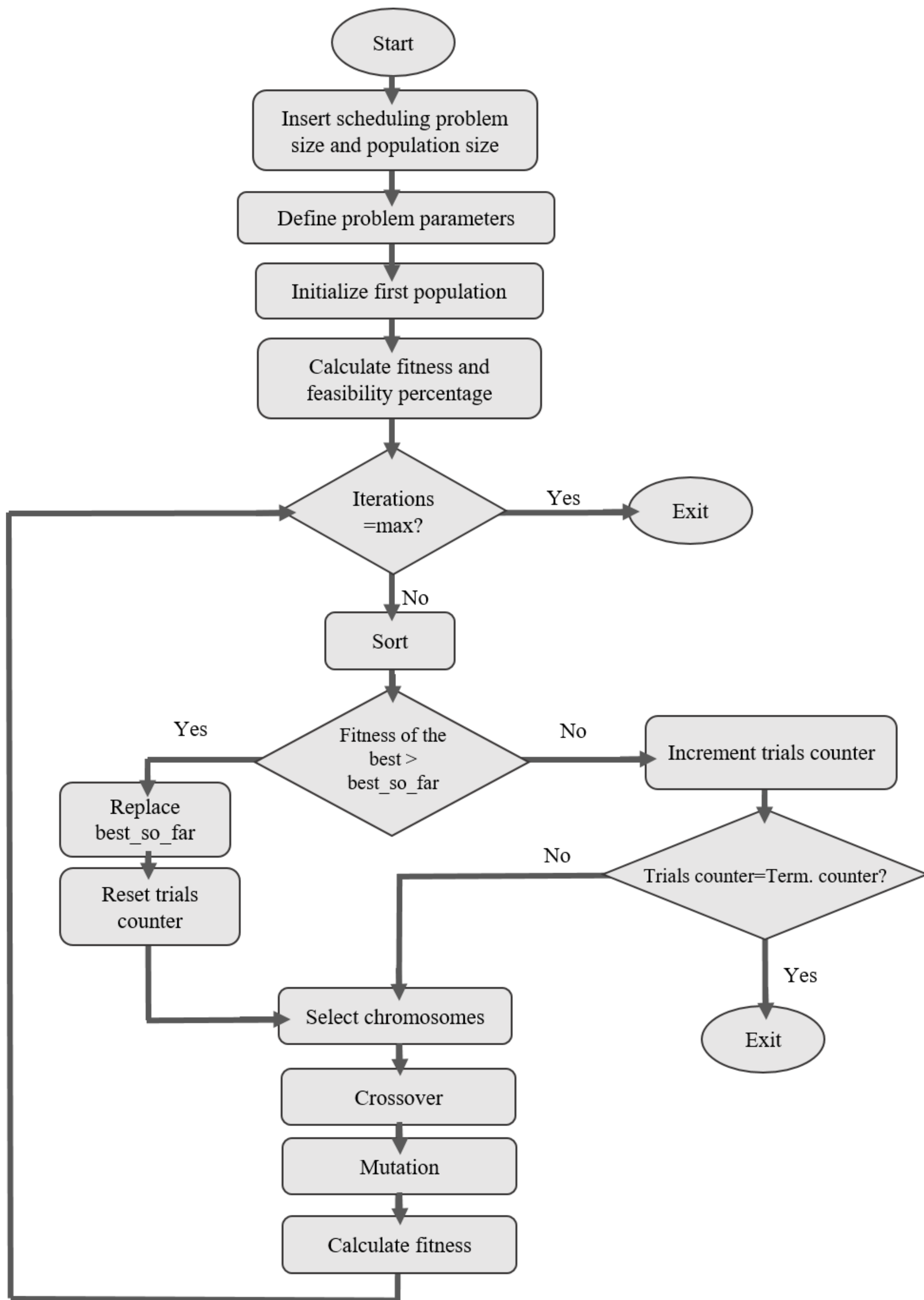

Figure 6: GA Implementation Flowchart

Table 6: GA Implementation Pseudocode

Algorithm: GA Implementation

```
Set population size, scheduling problem size, termination counter
Define requests and resources attributes
Create initial population
Evaluate fitness and check feasibility
repeat
        Sort chromosomes based on their fitness
        for (i=1 to population size) do
                if A better chromosome than best-so-far is found then
                        Replace best-so-far and reset trials counter
                endif
        endfor
        if A better chromosome not found then
                Increment trials counter
                if (trials counter = termination counter) then
                        exit
                endif
        endif
        Crossover
        Mutation
        Evaluate fitness and check feasibility
until (maximum number of iterations)
print best-so-far chromosome
```

4.2.1 Initial Population. As mentioned in section 4.2, the GA creates the first population or generation randomly. The function implementation is given in Table 7. The algorithm scans the chromosomes and for each chromosome it loops through the requests and the resources. For each request, the algorithm selects a resource randomly for execution.

4.2.2 Selection using Roulette Wheel. In order to implement the crossover and the mutation operators, the chromosomes that will be crossed over or mutated need to be selected. An algorithm called “roulette-wheel” has been adopted for selecting the chromosomes. Roulette-wheel is a selection algorithm based on the fitness value. It uses the fitness value as a probability to select the fitter chromosomes (the ones with better objective function or less latency). The probability of selecting any chromosome out of the whole population is calculated by normalizing the fitness as shown in

equation (11) and hence it is called the normalized fitness. It is calculated from the fitness of each chromosome divided by the summation of fitness for all the chromosomes. This way, the fitter chromosomes will have a high chance to be selected than the less fit ones. The summation of the normalized fitness of all the chromosomes in the population is equal to an exact one, hence it represents probability.

$$Probability\ of\ selection\ = \frac{F_i}{\sum F_i} \tag{11}$$

The roulette wheel algorithm is implemented as shown in Table 8. After normalizing the chromosomes fitness, the population is sorted by descending fitness values. Then accumulated normalized fitness values are calculated. The accumulated fitness value of a chromosome is the sum of its own fitness value plus the fitness values of all the previous chromosomes after sorting. This means, the accumulated normalized fitness of the last chromosome should be 1. Then, a random number between 0 and 1 is chosen. The selected chromosome is the last one whose accumulated normalized value is smaller than this random number.

Table 7: Initial Population Algorithm

Algorithm: Initial Population

Inputs: Population Size (pop_size), Number of Requests (num_requests), Number of Resources (num_resources)

Outputs: Initial Population (pop)

```
for (k = 1 to pop_size) do
        for (i = 1 to num_resources) do
                for (j = 1 to num_requests) do
                        For each request, select a resource randomly
                        Assign the request into the selected resource, x[i][j]
                endfor
        endfor
endfor
```

Table 8: Selection using Roulette Wheel

Algorithm: Selection using Roulette Wheel

Inputs: Population Size (pop_size), Population (pop)

Outputs: Index of a Selected Chromosome with the Population (index)

```
for (i=1 to pop_size) do
    Calculate normalized fitness
endfor
Sort chromosomes based on fitness
Choose a small random number, R
for (i=1 to pop_size) do
    Calculate accumulated normalized fitness
    if (accumulated normalized fitness > R) then
        Choose this chromosome and return its index within pop
    endif
endfor
```

4.2.3 Crossover. The crossover, also called recombination, is an operator that is used to generate new chromosomes from the current population. It works by selecting two chromosomes and recombining them as shown in Figure 7 to produce two new chromosomes. The first selected pair of chromosomes is called "Parent Chromosomes" while the second recombined one is called "Children Chromosomes". The idea behind the crossover operator is crucial for the GA evolvement. If the parent chromosomes are fit or have better objective values within the population, the children chromosomes are also more likely to have better level of fitness compared to other chromosomes.

In terms of implementation, the crossover is implemented as shown in Table 9. Two parent chromosomes get selected using roulette-wheel algorithm. Then, the crossover-point over the two chromosomes is selected randomly between 1 and 60% of the chromosome length represented by the number of requests. The two selected chromosomes get crossed over the selected cross point by swapping their opposite parts (as shown in Figure 7). The two generated children chromosomes are copied to a new population which is called the new offspring or the new generation. This process is repeated until the number of generated children chromosomes in the new generation is equal to the number of chromosomes in the current generation.

It is important to note that, although the crossover point is selected randomly, it is chosen delicately at a point (as shown in Figure 8) in order not to mix up the requests allocation. These points are referred to "requests chunks". It maps each request to an only one resource. If the cross point is not chosen as shown in Figure 8, a request might be allocated to more than one resource which validates the constraint in equation (6). This is the reason behind representing the chromosome length by the number of requests only, not by the actual chromosome length, when selecting the cross point. This way, it is easier to avoid allocating a request to more than one resource after the crossover.

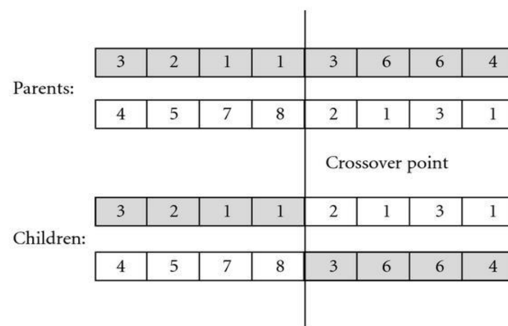

Figure 7: Crossover Operation

Table 9: Crossover

Algorithm: Crossover

Inputs: Population Size (pop_size), Population (pop), Number of Requests (num_requests), Number of Resources (num_resources)

Outputs: New Population (pop)

for (i=1 to pop_size, step 2) ***do***

    select first chromosome using roulette wheel

    select second chromosome using roulette wheel

    Choose a cross point from 1 to **60%***num_requests as an integer number

    for (j=1 to cross point) ***do***

        cross first chromosome genes with second chromosome genes

        Place children chromosome in the new pop

    endfor

endfor

R1					R2					R3					...
S1	S2	S3	...	SM	S1	S2	S3	...	SM	S1	S2	S3	...	SM	...
0	0	1	...	0	0	1	0	...	0	0	0	0	...	1	...

Cross-point

Figure 8: The Cross-Point within a Chromosome

4.2.4 Mutation. The mutation operator in the GA is used to implant small changes within the chromosomes after the crossover. This mutation operation is performed on the chromosomes by simply changing the allocation of a request from one resource to another one, as shown in Figure 9. In this case, the mutation shown in Figure 9 is changing the allocation of request 2 (R_2) from resource 2 (S_2) to resource 3 (S_3).

R1					R2					R3					...
S1	S2	S3	...	SM	S1	S2	S3	...	SM	S1	S2	S3	...	SM	...
0	0	1	...	0	0	1	0	...	0	0	0	0	...	1	...

R1					R2					R3					...
S1	S2	S3	...	SM	S1	S2	S3	...	SM	S1	S2	S3	...	SM	...
0	0	1	...	0	0	0	1	...	0	0	0	0	...	1	...

Figure 9: Mutation Operation

The mutation is the GA is implemented as shown in Table 10. It is important to note that the changes are not performed at the population level, i.e. the population is not mutated as a single 2-D array in different multiple locations. The mutation is performed at the chromosome level which means all chromosomes are scanned and each chromosome is selected and mutated individually and independently. Each chromosome is mutated based on a probability of 25% which means not every chromosome in the population will get mutated. If a chromosome is selected for mutation, the GA implants a number of changes defined by NUM_MUTATIONS. For each mutation operation, a random request is selected and re-allocated into another resource that is also selected randomly. The mutation algorithm repeats this process by the number of mutations defined by NUM_MUTATIONS.

Table 10: Mutation

Algorithm: Mutation

Inputs: Population Size (pop_size), Population (pop), Number of Requests (num_requests), Number of Resources (num_resources)

Outputs: Mutated Population (pop)

```
for (i = 1 to pop_size) do
        if ((rand() % 4)=0) then
                for (j = 1 to NUM_MUTATIONS ) do
                        Chose a request randomly from the pool
                        Deallocate the selected request from all resources
                        Chose a resource randomly from the pool
                        Re-allocate the selected request to the selected resource
                endfor
        endif
endfor
```

4.2.5 Feasibility Check. After each mutation and crossover, the GA needs to check the feasibility of all chromosomes in the population. To do that, the GA examine for two constraints. The first constraint is to make sure that each request must be allocated into one and only one resource and the second constraint is satisfying the deadline requirements modeled in equation (6) and equation (9). To test the first constraint, as show in Table 11, the GA sums each request chunk. The request chunks are divided as shown in Figure 10. A request chunk is a vector that maps each request to an only one resource as stated in the model formulation. This means a request chunk vector must be all zeros with a single one if the request is allocated in one resource and hence the summation should equal to 1. In case a request is allocated into more than 1 resource, the summation will not be equal to 1.

If a request is allocated into more than one resource, the chromosome will be flagged as infeasible and the algorithm continues checking for another chromosome. For testing the deadline requirement, the algorithm compares the service latency assigned to each request to its deadline. If the service latency is greater than the deadline requirement, the chromosome will be flagged as infeasible and the algorithm continues to check another chromosome. The latency and the fitness of each chromosome is evaluated in section 4.2.6.

Table 11: Feasibility Check

Algorithm: Feasibility Check

Inputs: Population Size (pop_size), Population (pop), Number of Requests (num_requests), Number of Resources (num_resources)

Outputs: Population with Feasibility Check (pop)

```
for (i=1 to pop_size) do
        for (j=1 to num_requests) do
                Calculate the request chunk sum
                if (request chunk sum =  1) then
                        Feasibility of the selected request is positive
                else
                        Feasibility of the selected request is negative and continue
                endif
        endfor
        for (j=1 to num_requests) do
                if (latency of request <= deadline of request) then
                        Feasibility of the selected request is positive
                else
                        Feasibility of the selected request is negative and continue
                endif
        endfor
endfor
```

	R1					R2					R3					...
	S1	S2	S3	...	SM	S1	S2	S3	...	SM	S1	S2	S3	...	SM	...
R1 chunck Sum = 1	0	0	1	...	0	0	1	0	...	0	0	0	0	...	1	...

Figure 10: Request Chunks for Feasibility Check

4.2.6 Fitness Calculation. In this function, the GA considers calculating the fitness values for each chromosome. This part of the algorithm is implemented as shown in Table 12. The GA starts by scanning all chromosomes within the population. For each chromosome, it loops through the requests and the resources while it calculates the processing time, the transmission and queuing time, and the starting time for each request (refer to equation (4), equation (5), and equation (7)). Then, the GA looks into

the resources one by one and it extracts the requests indices that are allocated in each resource and their count. This is followed by the execution phase where the algorithm executes the requests in sequence in a random order. This is to consider the assumption in the model that states each resource can process only one request at a time (refer to equation (6)). After execution, each request latency is calculated individually and used to calculate the overall weighted latency by multiplying each request latency by its priority (refer to equation (3)).

4.3 GA Experimentation

Before using the genetic algorithm in a real-time simulation environment, different GA parameters within the implementation were studied. The most important two parameters within the GA implementation are: population size (*U*) and termination counter (*T*). These two parameters have a direct impact on the solution quality and runtime. The runtime is the time the algorithm keeps running to come up with a solution. The solutions quality of the GA is also studied in this section by comparing it to the exact optimal solution obtained from Lingo.

4.3.1 Population Size, *U*. The GA population size, *U*, is one of the most important parameters of almost every GA implementation. Optimizing the population size is crucial because increasing it has a direct impact on the algorithm solution quality and runtime as well.

In this experiment, the objective is to study the population size versus solution quality which is the overall latency, *LT*, and runtime, *RT*. Different scheduling problem sizes were experimented to insure that the population size that will be selected is suitable to solve different size problems. The scheduling problem size, referred to as *N*, is defined as the number of requests and resources, regardless their parameters. The problems' sizes experimented namely are: 40 requests and 10 resources (40/10), 60 requests and 20 resources (60/20), 80 requests and 30 resources (80/30), 100 requests and 50 resources (100/50), as shown in Figure 11 and Figure 12. It is very important to note that these scheduling problems are totally independent and they are not related to each other. The parameters of these scheduling problems are set as follows: requests generation is in Poisson distribution with an inter-arrival rate mean of 2 seconds, data size is normally distributed with a mean of 1000 and a variance of 100, deadline is relaxed, and priority is uniformly distributed from 1 to 10. On the other hand, the

Table 12: Fitness Calculation

Algorithm: Fitness Calculation

Inputs: Population Size (pop_size), Population (pop), Number of Requests (num_requests), Number of Resources (num_resources)

Outputs: Population with Fitness Values (pop)

```
for (k=1 to pop_size) do
    for (j=1 to num_requests) do
        for (i=1 to num_resources) do
                if (x[i][j]=1) then
                    request[j].PT  = request[j].size / resource[i].speed;
                    request[j].TQT = resource[i].delta * request[j].size;
                    request[j].ST  = request[j].init_time + request[j].TQT;
                    break;
                endif
        endfor
    endfor
    for (i=1 to num_resources) do
        Collect the count of requests assigned to each resource and their indices to req_cnt
        & req_array respectively
        for (j=1 to req_cnt) do
            Select a request from req_array randomly
            Eliminate the selected request from req_array
            if (request[j].ST < sTime_next) then
                request[j].ST = sTime_next;
            endif
            sTime_next = request[j].ST + request[j].PT;
            request[j].LT=request[j].ST+request[j].PT+request[j].TT-
             request[j].init_time;
        endfor
    endfor
    for (j = 1 to num_requests) do
        LT += (request[j].LT * request[j]. priority);
    endfor
    Each chromosome fitness = 1/LT;
endfor
```

resources processing power is normally distributed with a mean of 200 and a variance of 25. Their average delays are also normally distributed with a mean of 50 milliseconds and a variance of 10 milliseconds. The deadline requirements are relaxed because the objective of this experiment is to study the objective value only. The number of missed deadline requests is not a concern in this experiment.

In this experiment, the objective is to study the population size versus solution quality which is the overall latency, *LT*, and runtime, *RT*. Different scheduling problem sizes were experimented to insure that the population size that will be selected is suitable to solve different size problems. The scheduling problem size, referred to as *N*, is defined as the number of requests and resources, regardless their parameters. The problems' sizes experimented namely are: 40 requests and 10 resources (40/10), 60 requests and 20 resources (60/20), 80 requests and 30 resources (80/30), 100 requests and 50 resources (100/50), as shown in Figure 11 and Figure 12. It is very important to note that these scheduling problems are totally independent and they are not related to each other. The parameters of these scheduling problems are set as follows: requests generation is in Poisson distribution with an inter-arrival rate mean of 2 seconds, data size is normally distributed with a mean of 1000 and a variance of 100, deadline is relaxed, and priority is uniformly distributed from 1 to 10. On the other hand, the resources processing power is normally distributed with a mean of 200 and a variance of 25. Their average delays are also normally distributed with a mean of 50 milliseconds and a variance of 10 milliseconds. The deadline requirements are relaxed because the objective of this experiment is to study the objective value only. The number of missed deadline requests is not a concern in this experiment.

The experimented population size starts from 1 up to 10, then it jumps to 15 and steps by 5 up to 100. We tried to be accurate within the range from 1 to 10 because the change is dramatic in this range. Since GA has a stochastic nature for finding the heuristic solutions, each experiment is repeated 5 times and the average is taken. The termination counter for this experiment is set to 50 (the termination counter is an important parameter and it will be studied in more details in section 4.2.2).

Figure 11 shows the fitness or latency versus the population size while Figure 12 shows the runtime versus the population size. Be reminded that, the objective is minimizing the latency. It can be noticed in Figure 11 that as the population size is increased, better minimized latency solutions are obtained, for all different scheduling

problem sizes. The drop can be seen very clearly in the curve starting, between population size 1 and 10. It can also be observed that all solutions stabilize (to a great extent) in around a population size of 60. It is also essential to note that the objective solutions or the objective latencies for these different sizes problems are not related to each other because they are different in size and they have different or independent attributes. In other words, the larger scheduling problem does not mean larger latency value. As can be seen in Figure 11, the problem with N=100/50 has less latency than the one with N=60/20.

Figure 12 shows the runtime versus the population size. It can be noticed that, as the population size increases, the runtime also increases. At this point, it can be understood that increasing the population size gives better and fitter solutions, however, at the expense of runtime. This indicates the importance of choosing a population size that gives fitter solutions and at the same time doesn't take long runtime. The second observation in Figure 12 is, as the problem size increases, the times it takes to find the best solution increases. As can be seen, the problem with N=100/50 has the largest runtime compared to the others, while the one with N=40/10 has the smallest. Increasing the problem size directly increases the number of loops and iterations within the GA implementation.

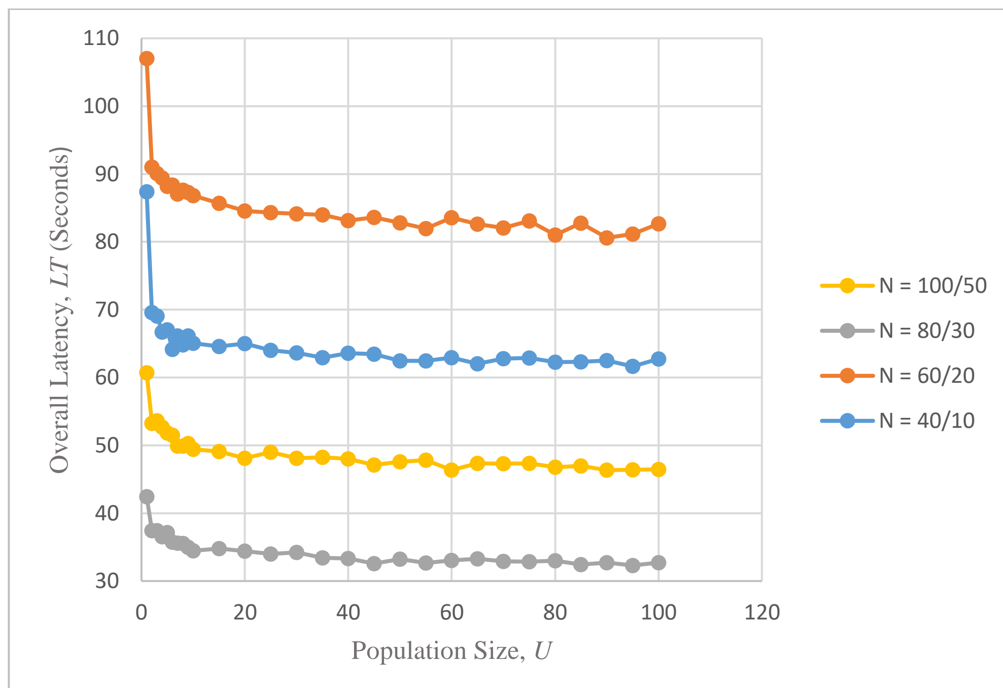

Figure 11: Overall Latency versus Population Size

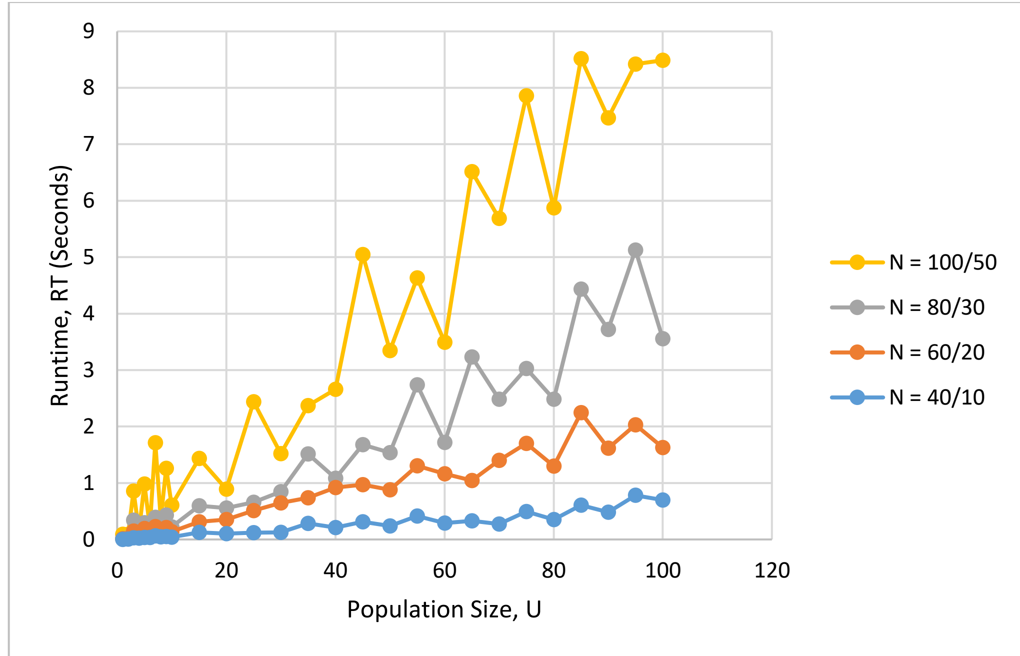

Figure 12: Runtime versus Population Size

4.3.2 Termination Counter, *T*. The termination counter, referred to as *T*, is the number of iterations the GA performs looking for better solutions before it terminates. If the GA consumes the number of iterations without finding a better solution than the best so far, it terminates. If it finds a better solution, it resets the counter to zero. This parameter has a direct impact on the algorithm solution refinement and runtime. This means, increasing the termination counter refines and improves the solution as it gives the algorithm more space and number of trials to find better solutions. However, it increases the runtime as it iterates more.

In this experiment, the objective is study the termination counter versus the solution quality and runtime as well. The same 4 different sizes scheduling problems used in section 4.2.1 are again used for this experiment. The experimented termination counter starts from 1 up to 40. Since the GA has a stochastic nature for finding the heuristic solutions, each experiment is repeated 5 times and the average is taken. The population size is set to 60 for this experiment as concluded in the last experiment.

Figure 13 shows the fitness function or latency versus the termination counter while Figure 14 shows the runtime versus the termination counter. From Figure 13, it can be noticed that, increasing the termination counter refines the solution for all different scheduling problems. It can also be seen that after 25 or 30 iterations, the solution stabilizes to a great extent. This means the objective value is not improving

anymore. Figure 14 shows that increasing the termination counter directly increases the runtime of the algorithm. This is intuitive as increasing the termination counter gives the algorithm extra more iterations and hence more runtime before termination.

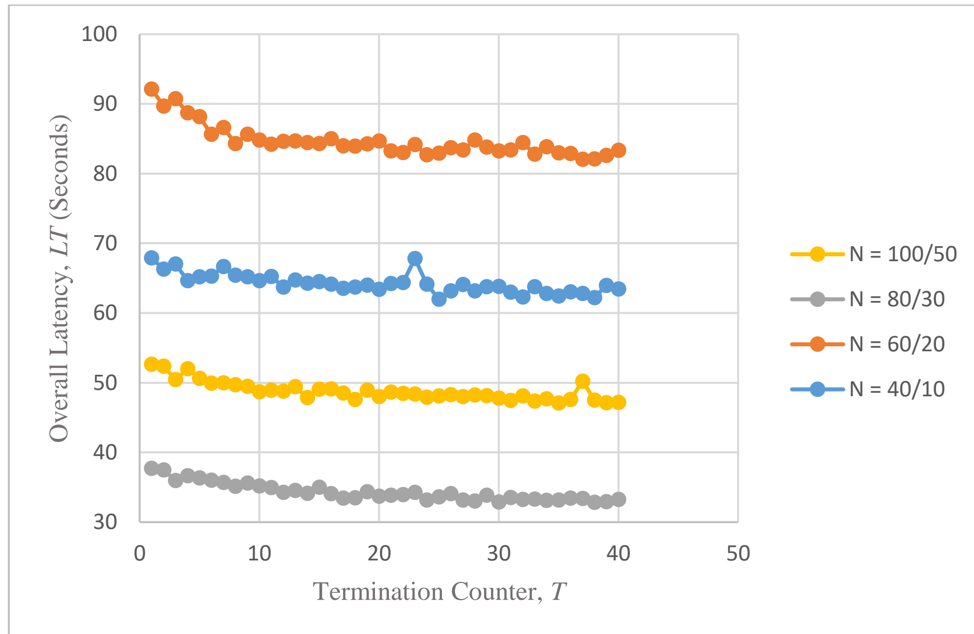

Figure 13: Overall Latency versus Termination Counter

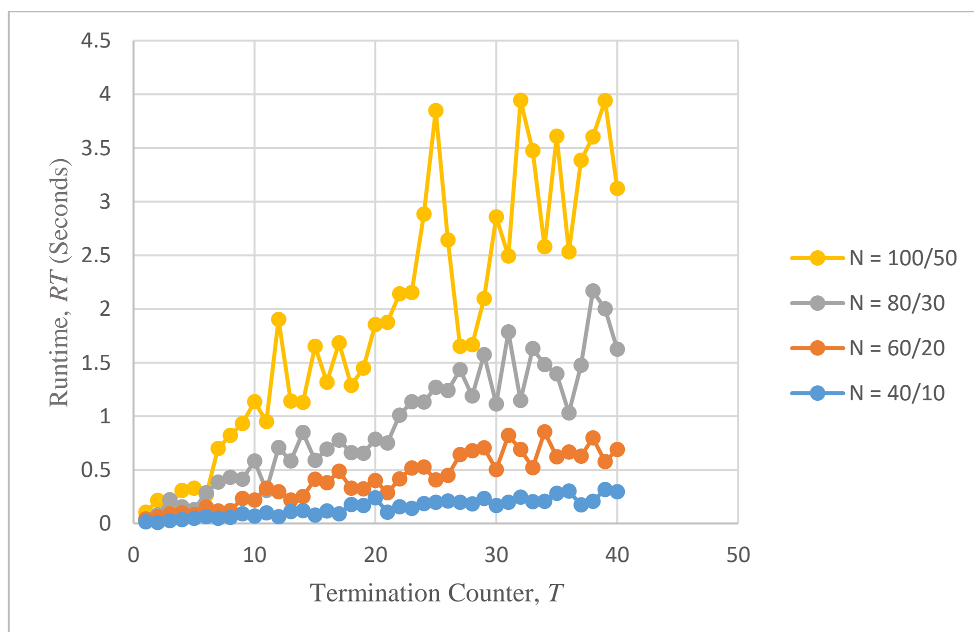

Figure 14: Runtime versus Termination Counter

After studying the population size and the termination counter of the GA algorithm, the population size is set to 60 and the termination counter is set to 20 as seen in Figure 11, Figure 12, Figure 13, and Figure 14. After these chosen values, the objectives stop improving and any extra iterations are not useful. Then the actual solutions convergence are plotted for the same 4 scheduling problems as shown in Figure 15.

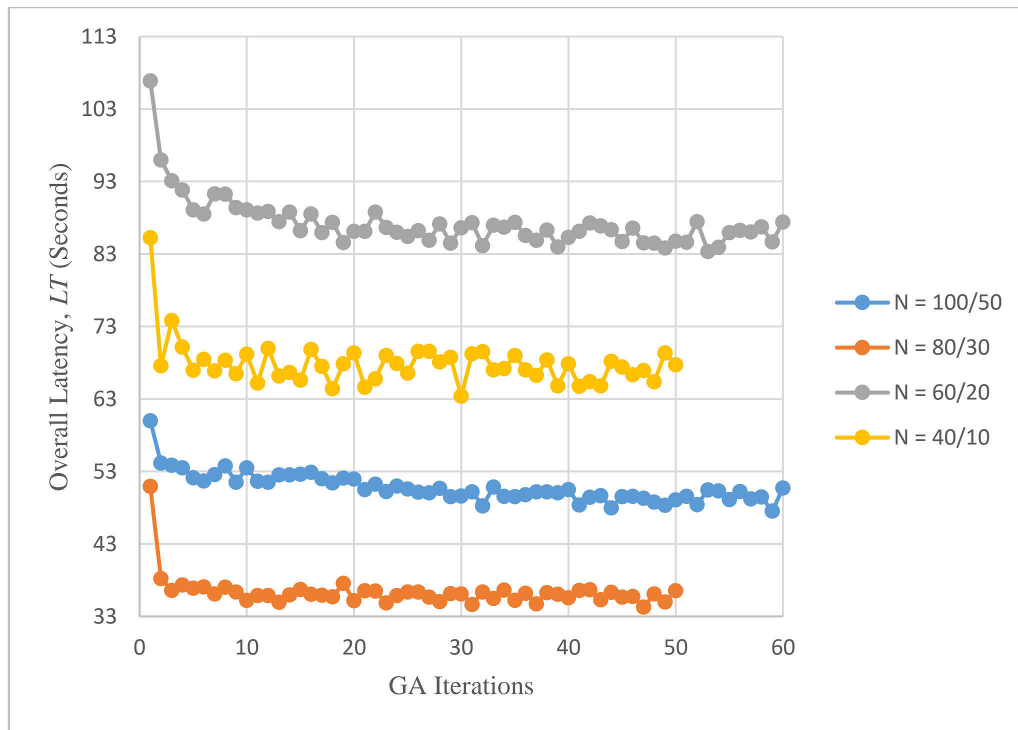

Figure 15: Overall Latency Convergence through the GA Iterations

4.3.3 Exact and Heuristic Comparison. In this section, the objective is to compare between the exact solutions and the heuristic solutions using two criteria. The first one is the solution quality (minimized latency) which represents how good the heuristic solution compared to the absolute optimal solution. The second criterion is the runtime which will show how much time the exact solution takes to come up with the optimal solution compared to the GA. Branch-and-bound algorithm (B&B) has been adopted on Lingo software as an exact solution algorithm and Generic Algorithm as a heuristic algorithm.

To do the comparison, Different scheduling problems with different sizes were experimented. However, small size scheduling problems were chosen for the experiment. Lingo, being an exact solution tool, takes very long time to come up with

an optimal solution. Moreover, the problem is modeled to be non-linear in the integer programming modeling. The objective functions and some constraints are non-linear. The model being non-linear means that there are variables that are multiplied with each other in the ILP model. Lindo, Lingo software developer company, is stating *"In general, Integer Nonlinear models are very difficult to solve for all but the smallest cases"* [49].

The sizes of the experimented scheduling problems in terms of the number of requests and resources (N) are: 4/2, 6/2, 8/2, 12/2, 8/3, and 10/3, as shown in Figure 16 and Figure 17. The parameters of these problems are chosen randomly and not based on any distribution or pattern. Figure 16 shows the solution quality (latency in milliseconds) for both B&B and GA. It can be noticed that the exact solution and heuristic solution are very close to each other. Figure 17 shows the runtime for the experimented problems.

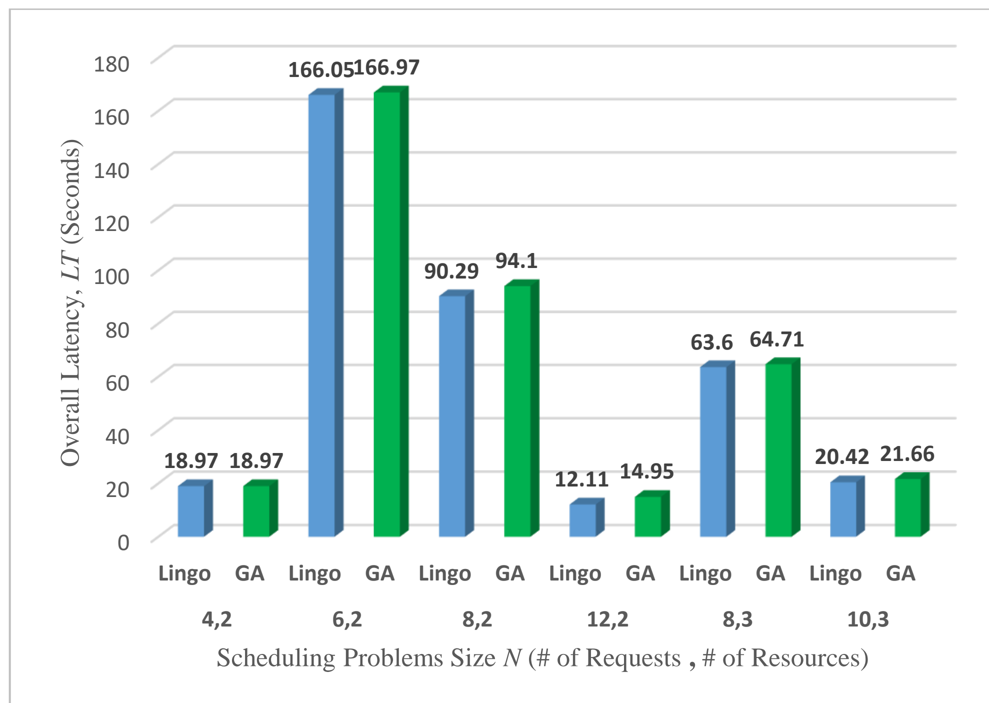

Figure 16: Overall Latency Comparison between Heuristic and Exact Methods

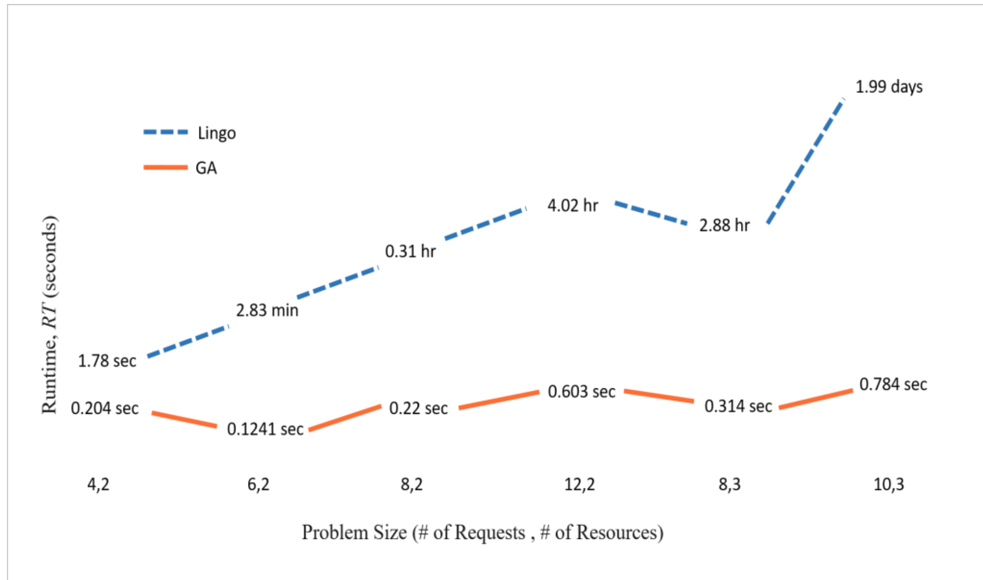

Figure 17: Runtime Comparison between Heuristic and Exact Methods

Chapter 5. Simulation and Results

A simulation model from the formulated model is developed using the discrete event simulator SimEvent. SimEvents provides a discrete-event simulation engine and components library for analyzing event-driven system models.

The simulation is built based on edge-fog-cloud 3-layered architecture as can be seen in Figure 18. At the edge layer, requests get generated in a specific distribution and inter-arrival time. Each generated request is associated with its attributes defined in the model. Then, the requests move from edge layer heading towards the upper two layers, fog and cloud, where they experience some small delay. This small delay represents the network gap between the edge layer and the upper two layers.

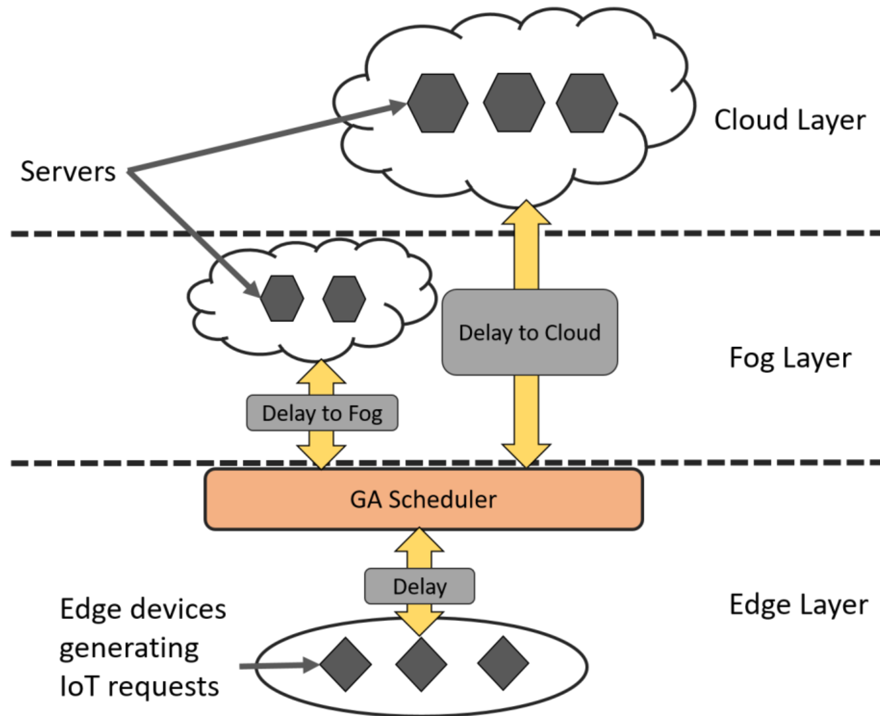

Figure 18: Edge-Fog-Cloud 3-Layered Simulation Setup

The upper two layers contain the cloud and fog resources or servers. All requests, on their way to these servers, get received in the GA scheduling algorithm. The GA scheduling algorithm developed in the previous chapter is rewritten in C language and integrated with SimEvents environment to help scheduling requests as they arrive. The GA scheduler receives requests within specific defined time frames and it runs the GA to find the scheduling solutions and dispatches the requests

accordingly. Within any time frame the GA scheduler can receive any number of requests as long as it does not exceed the maximum number the GA can handle. Requests arrival distribution and rate define the number of requests that get received in a time frame.

Initially, the resources in the simulation start as free resources, but as time proceeds and requests get processed or served, the resources are not assumed to be free anymore. This means, the GA has to be provided with the number of resources available and their attributes defined in the model. The GA runs and finds scheduling solutions and it keeps information and predictions about the resources status. The status is used to describe the time in which the resource will be busy processing other requests. For instance, if a resource is busy executing a request for 1 hour long, it will be more beneficial to take that into consideration and utilize other free resources that can provide less latency. However, the predicted status is not guaranteed to be true. The resources status is very important, because it is taken as an input to the GA algorithm alongside with the resources and requests attributes. It makes sense to not allocate requests within a resource that will be busy processing some other requests. Be reminded that preemption is not allowed in the model.

The resources attributes defined in the model are the processing power or speed and the average delay. The GA solver assumes all requests packets will be delayed by only the average value as it evaluates the delay and hence the latency beforehand within the algorithm. However, in the simulation environment, the actual delay per packet can be the same as the average value or different based on the distribution used. The transmission and queuing delay distribution is set to be Gaussian with specific mean and variance. When a request is sent to a specific resource to get served and it reaches the resource, if the resource is free and the waiting queue is empty, the request gets served or processed right away, otherwise it gets pushed into a waiting queue and served as soon as the resource becomes free.

5.1 GA validation in SimEvents

In this experiment, the GA scheduling algorithm is validated or verified to make sure that it works properly in optimizing the latency. To do that, the GA optimizer is observed for a simple and easy-to-follow resources setup. In other words, the GA is tested for a resources setup that have attributes (processing speed and average delay)

going in ascending or descending order. In this way, the GA behavior can be observed and judged by monitoring the number of requests allocated and served in each resource.

The validation involves a system that has 16 servers numbered from 1 to 16. Server number 1 is set to be the slowest server and server 16 is set to be the fastest one. The processing speed is started by 100 packets per second for server 1 and gets increased by 100 for the rest of the servers. This means server 2 speed is 200, server 3 speed is 300, up to server 16 which has a speed of 1600 packets per second. The processing speed setup is fixed throughout this experiment.

On the other hand, the average delay (δ_i) for each server at the beginning is set to be equal for all the 16 servers as 1 millisecond, for the first experiment. The first experiment is followed by other 2 experiments in which the average delays are changed. In experiment 2 and 3, the average delays are changed with a "common difference" of 10% and 80% respectively in an arithmetic series fashion. The common difference percentage is referred to by α. This means, for experiment 2, as an example, server 1 average delay is 1 millisecond, server 2 average delay is 1.1 milliseconds, and server 3 average delay is 1.2 milliseconds, up to server 16 with an average delay of 2.5 milliseconds. Server 1 will be the closest while server 16 will be the furthest. In experiment 3, the average delays are changed in the same manner but using a common difference of 80%.

The purpose of having such setup is to observe the number of requests allocated in each one of the 16 servers, in each experiment. As the server's attributes are set in ascending order, the optimizer behavior can be judged if it works properly as the powerful and closest servers are known. A total of 500 requests is used in the 3 experiments with an average size of 3000 packets with the same priority level. The deadline requirements are relaxed as the objective is not to evaluate the latency. The requests arrival rate is 1 request per second in a Poisson distribution. The GA optimizer receives requests and schedules them within a time frame of 5 seconds.

Figure 19 shows the number of requests allocated or served in each of the 16 servers as percentages. As can be seen, in experiment 1 where the average delay common difference is 0%, the optimizer looks into accommodating more requests in the higher 8 resources since they are more powerful. In experiment 2, as the average delays for servers are increased in an ascending order by a common difference of 10%, the higher 8 resources become a little far, despite their high processing speed. In this

case, the optimizer tends to accommodate more requests in the middle region where there are servers with moderate processing speed and average delays. In the third case, the higher 8 servers become too far and it becomes not feasible anymore to use them for serving requests. In this case, the optimizer tends to allocate more requests in the closer lower resources as they have less delays. It deserves mentioning in this last case, experiment 3, the average delays are set deliberately using a high common difference of 80% so that the high average delays for the higher 8 servers overcome their high processing speed.

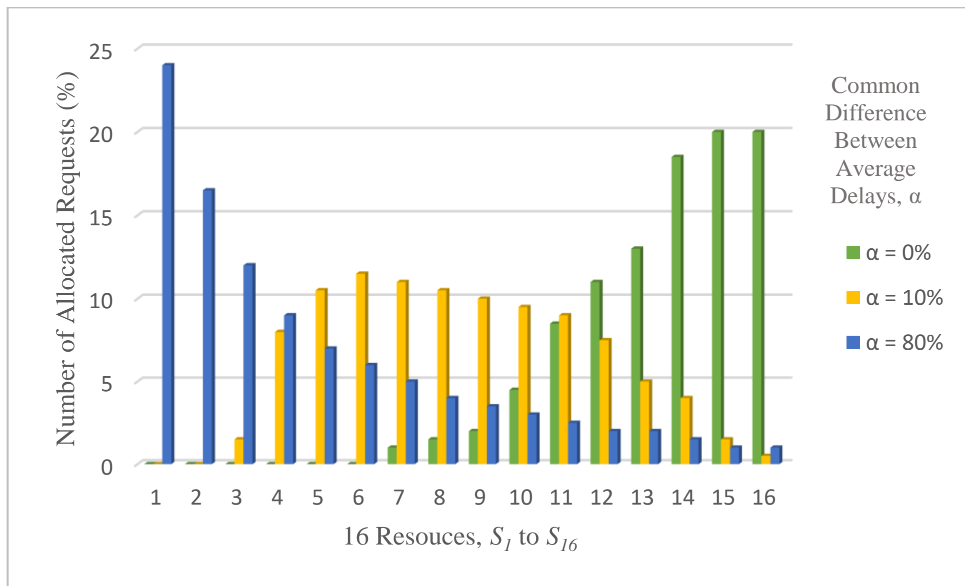

Figure 19: Analyzing the Number of Requests Allocated in Each Resource

5.2 Static Scheduling

In this experiment, the performance of the GA is evaluated in static scheduling mode in terms of two metrics: The overall average service latency and the number of missed-deadline requests. In static mode, the inter-arrival time between requests is removed and all requests are assumed to be generated at time 0 as one batch. In this experiment, the GA scheduler performance is compared to other traditional scheduling algorithms. These algorithms namely are WFQ, PSQ, and RR.

Waited-fair queuing and priority-strict queuing algorithms classify requests into priority classes at the output queue. Each priority class has its own queue. PSQ chooses requests from the highest priority class that has a nonempty queue. The choice among requests in the same priority class is typically done in a First-In-First-Out (FIFO) manner. On the other hand, WFQ uses a round robin scheduler to alternate selection

among the classes using a defined weight for each class. Since both WFQ and PSQ are priority based algorithm, the allocation in both of them is carried out based on priority, with high requests going to more powerful servers. The implementation for Round Robin algorithm is just allocating requests within resources in a Round Robin fashion.

This experiment involves a set of 16 servers with an average processing speed of 500 packets per second, but very widely distributed from 50 to 1000. The servers' average delays are set to an average of 5 milliseconds per packet. It is also very widely distributed from 1 millisecond up to 9.7 millisecond.

A total of 100 requests is used in the experiment. These requests are generated at time 0 with no inter-arrival time as mentioned earlier. The priorities are set to be uniformly distributed from 1 to 16. The deadline requirements are set to be 400 seconds on average with a variance of 50.

The objective of this experiment is to study the overall average service latency and the number of missed-deadline requests using different scheduling algorithms, GA, WFQ, PSQ, and RR. These two aspects are studied versus requests average data size while other attributes are not changed. Initially, the average data size starts from a small value in a way that makes requests deadline requirements very loose and zero requests miss their deadlines. Then, the average data size is increased to observe the impact on the overall latency and number of missed-deadline requests.

Figure 20 and Figure 21 show the average overall latency and the number of missed-deadline requests versus requests average size, respectively. As can be seen in Figure 20, the GA achieved better overall latency than the rest of the algorithms. WFQ and PSQ results are very close to each other since the allocation of requests within resources in both of these algorithms is achieved based on priority, however the dispatching is different. RR achieved the highest latency time and the reason behind that is the way requests get allocated in an RR fashion as without considering requests priorities. In Figure 21, it can be seen, the GA keeps the record clean of missed-deadline requests for longer time than the other algorithms. However, at an average size of 6500 the amount of data within each request becomes so heavy and the GA cannot guarantee meeting all requests deadlines as their service latency increases and their deadlines become very critical.

As shown in Figure 20 and Figure 21, it is important to mention that, within all the experimented data sizes using the GA, the latter was able to come up with a feasible

schedule solution in which all requests deadlines should be met. However, the simulation results show that it is not guaranteed that all requests will be met even if the evaluated GA scheduling solution is feasible. This can be seen in Figure 21 between 6000 and 8000 average data size. After a data size of 8000, the problem becomes infeasible and hence the GA cannot find a feasible schedule. This is true in the simulation because the actual delay per packet can be different from the average delay the algorithm assumes.

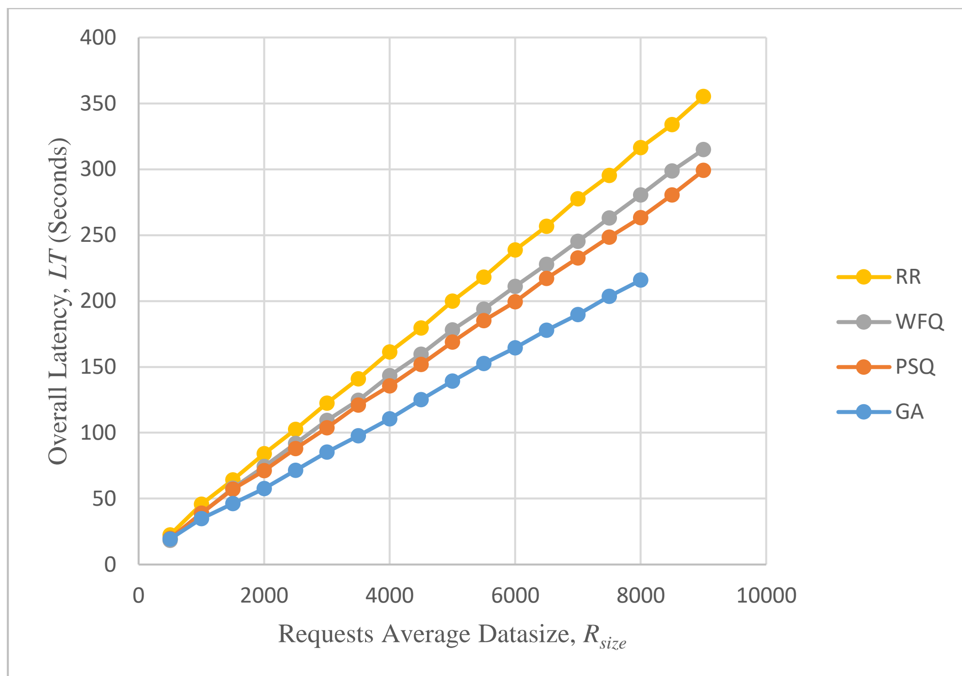

Figure 20: Overall Latency versus Data Size in Static Scheduling

5.3 Average Data Size Breaking Point

The objective of this experiment is to study requests average data size in which the GA will not be able to meet all requests deadlines using a specific setup of resources. As seen in the previous experiment, the resources available in the experiment could not meet all requests deadline requirements after an average size of almost 6000 packets. This data size breaking point is defined as the average data size after which the GA scheduler starts losing the ability to meet all requests deadlines.

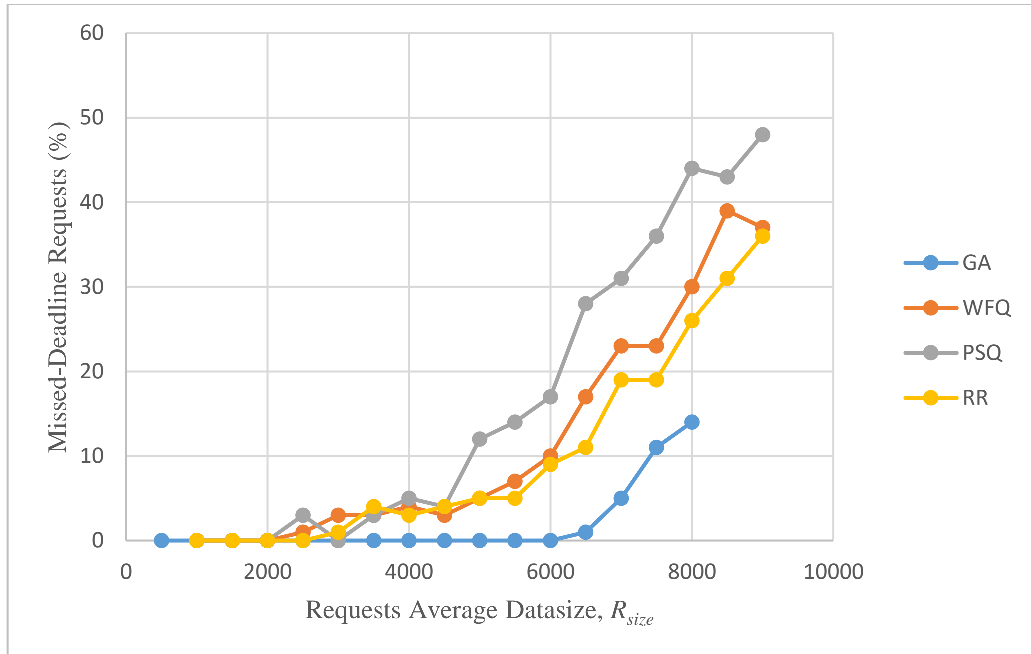

Figure 21: Missed-Deadline Requests versus Data Size in Static Scheduling

The experiment involves the same requests and resources setup used in the previous experiment. The breaking point is studied versus the resource attributes in terms of processing speed and average delay. P0 is set to be 250 while $\overline{\delta_0}$ is set to be 0.005. These two attributes will be increased and decreased in ratios in order to see their impact on the breaking point.

Figure 22 shows the average data size breaking point versus the processing speed and the average delay. As can be seen, increasing the processing speed allows processing more data as they will experience less latency. It can also be seen that decreasing the average delay will rise the breaking point higher because less average delays provide less latency. However, from the figure, it can be noticed that the algorithm is more sensitive to the average delay than the processing capacity. This means for a specific average delay, at some point increasing the processing speed will not be beneficial as much as decreasing the average delay. This is true because the average delay is a bottleneck in the model.

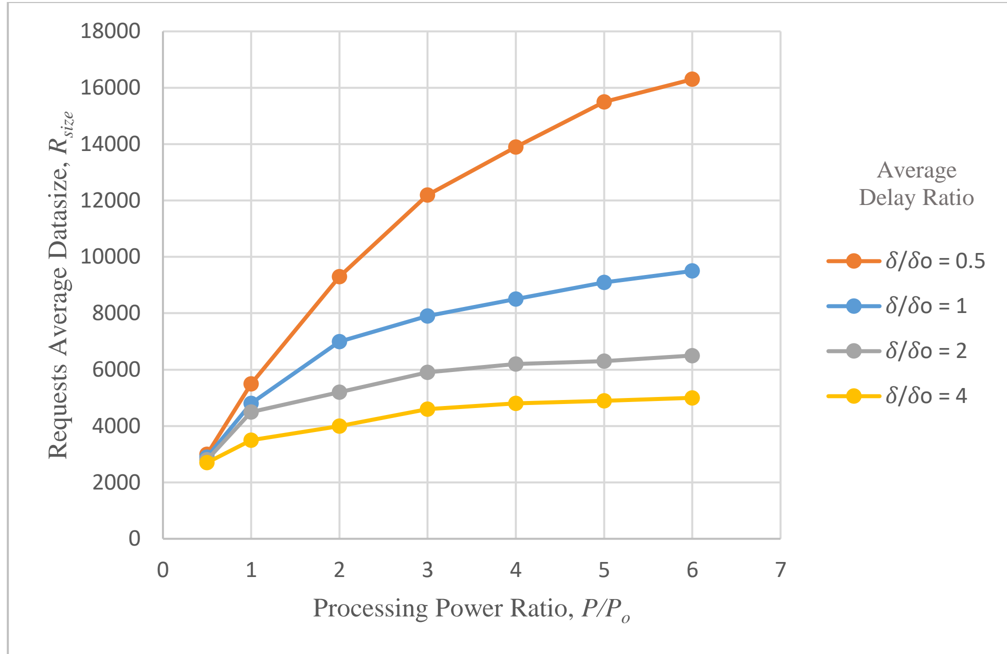

Figure 22: The GA Breaking Point versus Processing Speed and Average Delay

5.4 Dynamic Scheduling

In this experiment, the performance of the GA is evaluated in dynamic scheduling for the same two metrics studied in the static; the overall average service latency and the number of missed-deadline requests. In dynamic scheduling, requests are generated using a specific distribution and inter-arrival rate. On top of that, rescheduling feature has been also added into this experiment setup. As requests get received in time frames to get scheduled, any requests that are not dispatched yet will be rescheduled or reconsidered in the new schedule. The GA performance is compared to the other networking scheduling algorithms, WFQ, PSQ, and RR.

This experiment involves the same resources setup from the previous experiment. However, the number of requests is increased to 500 requests since the scheduling is dynamic and the objective is to evaluate the latency in a real-time manner. The requests are generated in a Poisson distribution with an inter-arrival mean of 1 second. The priorities are set to be between 1 and 16 in a uniform distribution. The deadline requirements on average are set to 200 seconds with a variance of 50. The time frame in which the resources get scheduled in is 10 seconds. The average request data size will be changed in the experiment from 1000 up to 10000 packets.

Figure 23 and Figure 24 show the overall average latency and the number of missed-deadline requests versus the average data size, respectively. As can be seen in

Figure 23, the GA achieved the best overall latency compared to the other algorithms. WFQ and PSQ results are very close to each other, however, the difference increases by increasing the average data size. On the other hand, Figure 24 shows that the GA achieved 0 missed-deadline requests if requests data size is less than 5000 packets on average. At an average size of 5000, while GA achieved 0 missed requests, WFQ and PSQ lost almost 10% and RR lost 6% missed requests. After this point, most of the requests deadline requirements become very critical and some of them even infeasible. For this reason, the GA also starts missing requests.

5.5 Cloud versus Fog Computing Comparison

In this experiment, the objective is to evaluate the service latency provided by resources setup that have cloud characteristics and fog computing characteristics. In general, cloud resources are powerful with high processing capabilities, but at the same time they have large average transmission and networking delay. Conversely, fog resources do not have rich processing power but they provide smaller average delay since they exist closer to the edge. This experiment gives a clear notion about cloud and fog resources from a design perspective, whether it is more beneficial to put very powerful resources at cloud layer or to put much less powerful resources closer to requests sources in fog layer.

In order to design a resource set that is able to serve a set of requests with a minimized latency, there are three parameters on the formulated model that need to be taken into consideration:

1- The average delay, $\bar{\delta}$.
2- The processing speed, P.
3- The number of resources, m.

The impact of each one of these 3 parameters on the service latency is studied independently. In other words, they are studied separately by fixing two of them and varying only one. For instance, to study the average delay impact, the processing power and number of resources are fixed.

The latency that can be achieved by varying these parameters is evaluated against a system that has a cloud characteristic. This cloud setup has a set of 4 super cloud servers with a very high processing capability of 5000 packets per second. However,

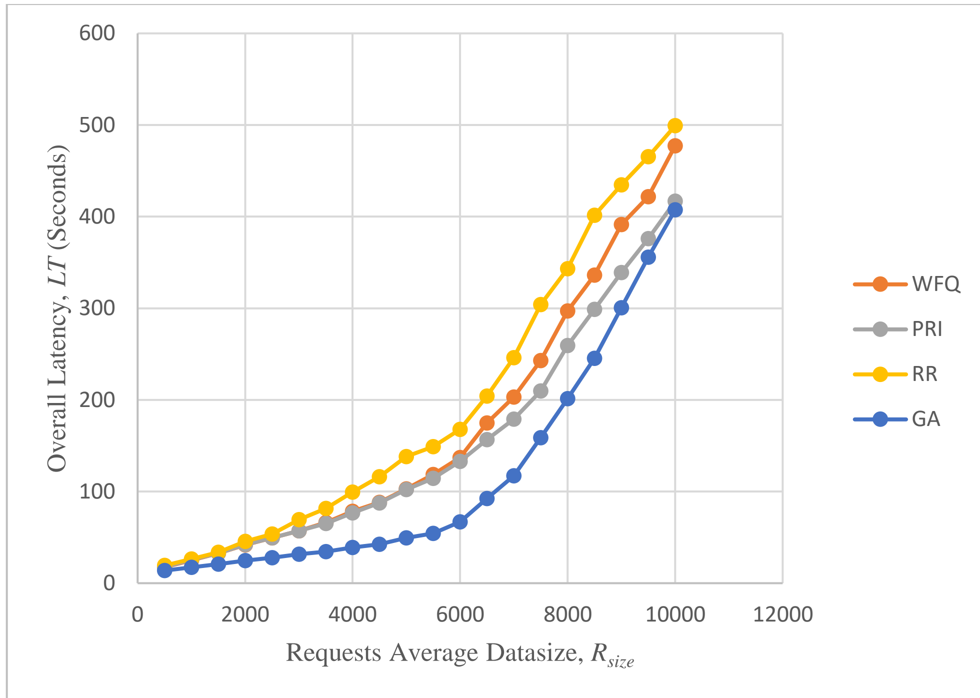

Figure 23: Overall Latency versus Data Size in Dynamic Scheduling

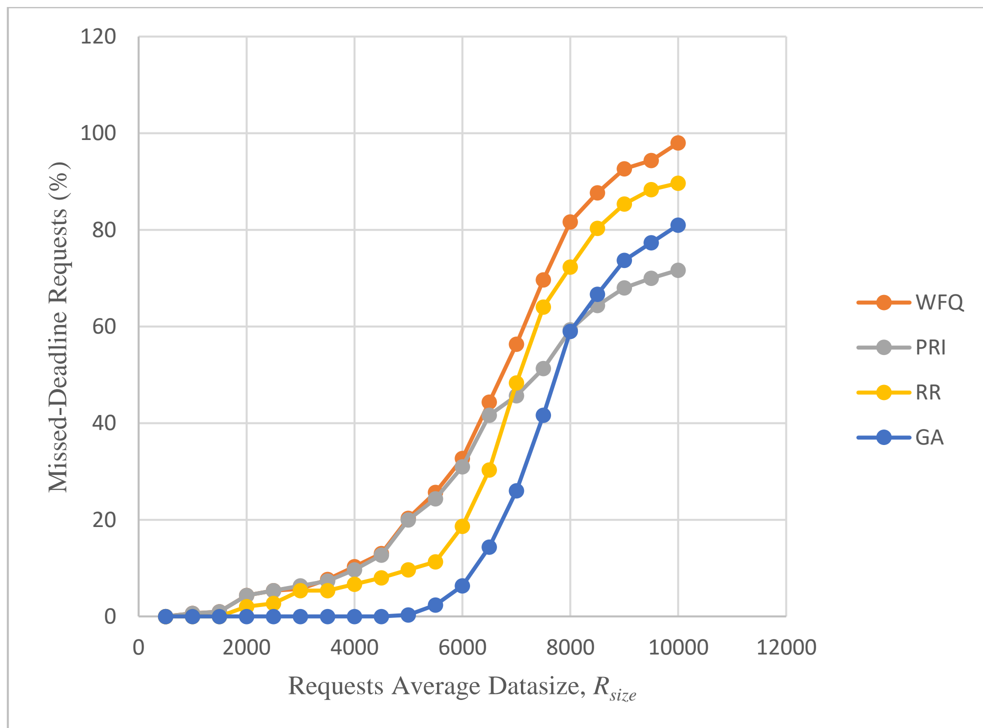

Figure 24: Missed-Deadline Requests versus Data Size in Dynamic Scheduling

the average delays for these servers set to be relatively high as 10 milliseconds per packet for each one.

To evaluate the latency, a total of 500 requests is used in this set of experiments. Their arrival is in Poisson distribution with an inter-arrival mean of 1 second. Requests priorities are set to be equal. The deadline requirements are relaxed in this experiment since the objective is to evaluate the service latency, not the number of missed requests. The latency is studied versus the average data size which will be changed from 1000 packets to 10000 packets.

5.5.1 The Average Delay Ratio, $\frac{\bar{\delta}_f}{\bar{\delta}_c}$. In this experiment, the latency of a set of fog computing servers is studied and compared to the latency provided by the Cloud setup described in this section. The fog servers' average delay is varied while their number and processing power are fixed. The number of fog servers is set to be 4 times the number of cloud server, $\frac{N_f}{N_c} = 4$. Their processing power is only 10% or their cloud peers, $\frac{P_f}{P_c} = 10\%$. The average delay, $\frac{\bar{\delta}_f}{\bar{\delta}_c}$, will be changed to 1%, 10%, 20%, 50% and 85%. Figure 25 shows the latency results of fog computing compared to cloud computing for different average delays. As can be seen, reducing the average delay has a significant impact on the service latency even if the resources processing capability is poor. It can be noticed also that increasing the average delay of fog computing at some point breaks the cloud computing latency since fog computing has lower processing speed. For instance, for an average data size of 5000, fog computing latency crosses cloud computing latency when , $\frac{\bar{\delta}_f}{\bar{\delta}_c} = 85\%$ for the specified $\frac{N_f}{N_c}$ and $\frac{P_f}{P_c}$. This breaking point will be studied in more details in the next experiment.

5.5.2 The Processing Speed Ratio, $\frac{P_f}{P_c}$. The objective of this experiment is to study the impact of the processing capability of fog computing on the service latency. To do that, fog servers processing power is varied while their number and average delay are fixed. The number of fog servers is set to be 4 times the number of cloud servers, $\frac{N_f}{N_c} = 4$. Their average delay is 10%, $\frac{\bar{\delta}_f}{\bar{\delta}_c} = 10\%$. The processing power, $\frac{P_f}{P_c}$, will be changed to 3%, 5%, 7%, 10% and 20%. Figure 26 shows the latency results of fog computing compared to cloud computing for different processing power ratios. The figure shows that if fog computing has 4 times the number of resources of cloud computing and their average delay is reduced by 10 times, the processing capability can

be reduced and slowed down up to 5% and it will still provide better latency than cloud computing does.

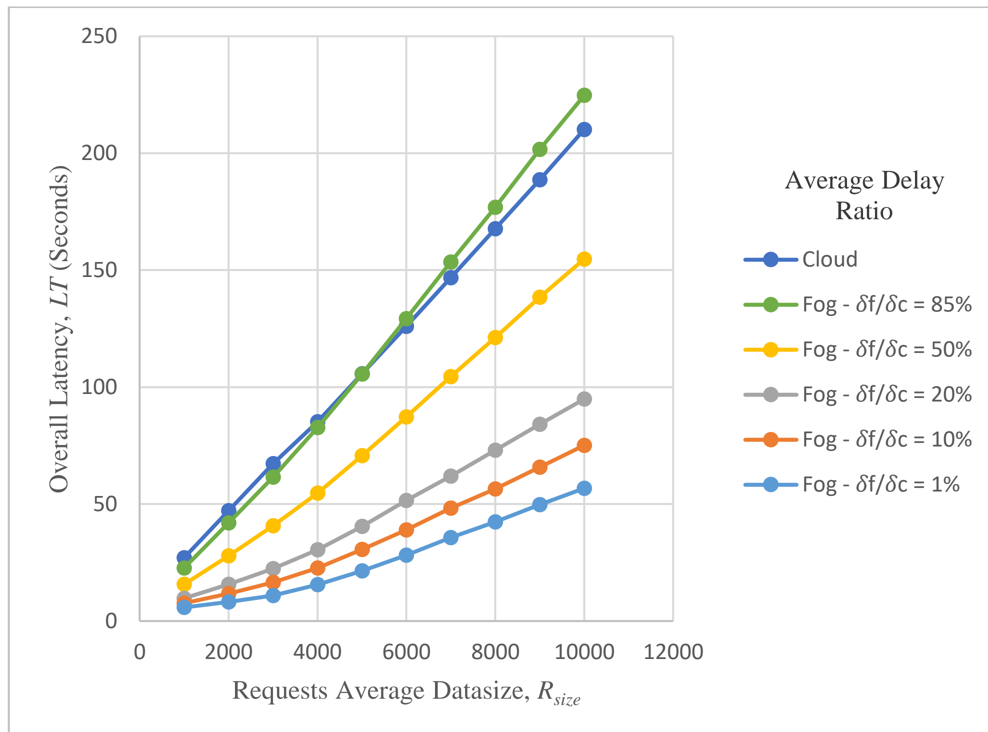

Figure 25: Latency of Fog Compared to Cloud by Varying Average Delay

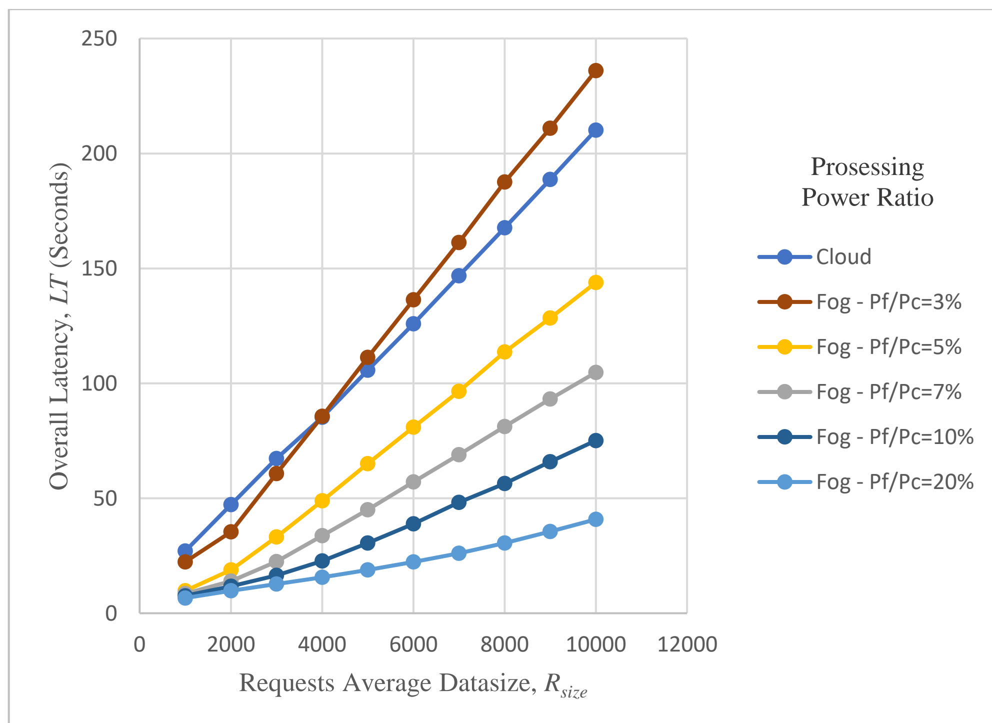

Figure 26: Latency of Fog Compared to Cloud by Varying Processing Power

5.5.3 The Number of Servers Ratio, $\frac{N_f}{N_c}$. The objective of this experiment is to study the impact of the number of fog servers on the service latency. To do that, the number of fog servers is varied while their processing power and average delay are fixed. Their average delay is set to 10% of cloud computing average delay, $\frac{\bar{\delta}_f}{\bar{\delta}_c} = 10\%$. Their processing power, $\frac{P_f}{P_c}$, is set to 10% as well. Their number, $\frac{N_f}{N_c}$, is varied to 1, 1.5, 2, 3, 4, 6, and 8. Figure 27 shows the latency results for this experiment. It is evident that the minimum $\frac{N_f}{N_c}$ ratio can achieve better latency than cloud computing is 1.5.

From the last three experiments, it can be seen that the latency performance of fog computing at some point crosses the cloud computing latency line. This happens in all the three cases, where fog computing average delay is increased much or processing power is decreased much or the number of servers is reduced significantly. In this experiment, the objective is to find these breaking points in terms of the 3 ratio parameters $\frac{\bar{\delta}_f}{\bar{\delta}_c}$, $\frac{P_f}{P_c}$, and $\frac{N_f}{N_c}$ for a specific average size of 5000 packets. Figure 28 shows the results of the experiment.

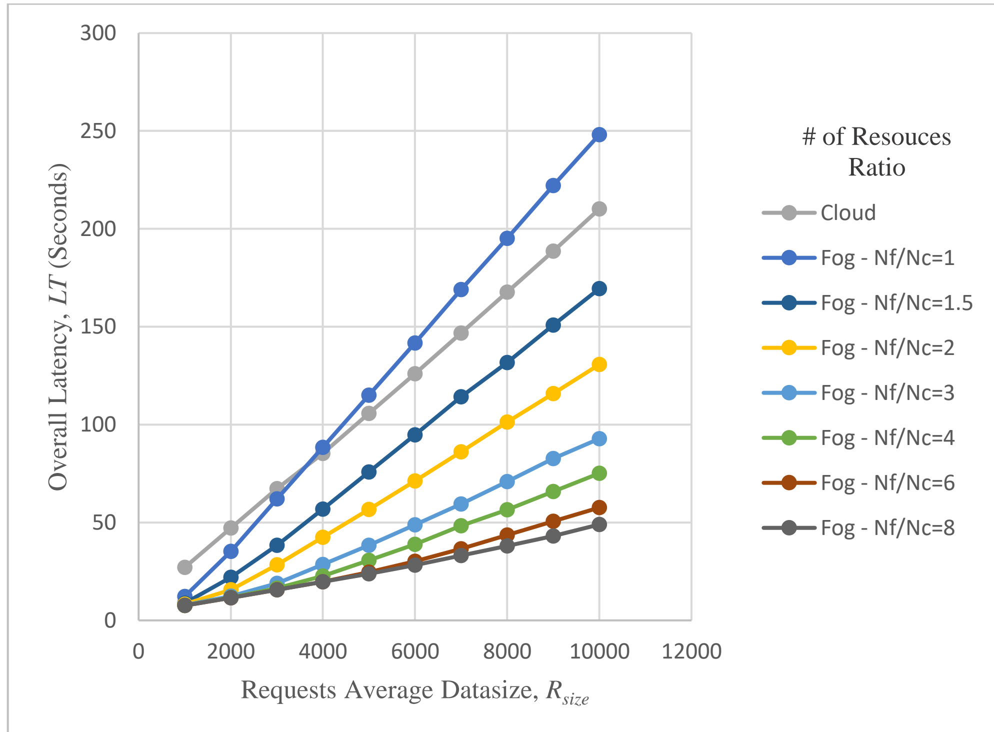

Figure 27: Latency of Fog Compared to Cloud by Varying Number of Resources

In conclusion, this work focused on modeling the edge-fog-cloud 3-layered architecture using ILP concepts. ILP has been involved in order to deliver optimal solutions and not just random solutions for scheduling the IoT requests within fog and cloud computing resources. The optimized service latency obtained from the GA is compared to non-optimized scheduling algorithms (WFQ, PRI, and RR) and the results showed the improvement in different scheduling scenarios with respect to service latency and satisfying deadline requirements. The experiments in section 5.5 show the significance of integrating fog computing with cloud computing. Fog computing is generally characterized by having small communication delay and wide spatial coverage. This allows using small-size low-power fog computing resources and it provides even better service latency than using cloud computing only. The experiments results give to what limit exactly fog computing with such characteristics can provide the better latency when it crosses the service latency of cloud computing.

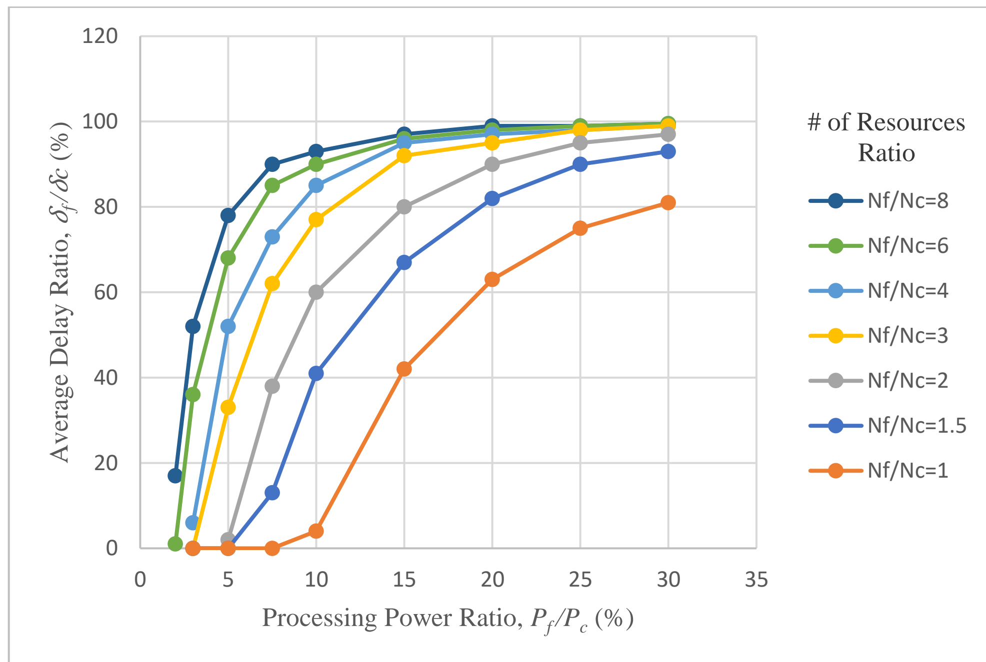

Figure 28: Break Points of Fog and Cloud Computing Latency

Chapter 6. Conclusion and Future Research

6.1 Conclusion

This research addressed the scheduling of Internet of Things (IoT) requests into resources available at both fog and cloud computing layers. The problem is modeled using integer programming where the objective is to optimize the service latency and provide minimum service time for the IoT requests. The service latency is defined as the Round-Trip Time (RTT) for serving or processing an IoT request from the moment it gets initiated to the moment it gets completely processed and the results are returned back to the requesting device. This latency includes many delay components such as transmission delay, routing or queuing delay, propagation delay, processing time, and waiting time in case the resources are busy. The IoT requests are characterized by having attributes such as creation time, data size, priority, deadline, and dependency constraints. On the other hand, the resources are defined in terms of the processing capability and the average delay per packet to reach the resource. The objective function in the scheduling problem is to find the schedule that can give the least weighted overall latency. The weighted latency is calculated by the summation of multiplying each request weight by its latency component. This way, the model will give the least latency possible for requests with high weights represented in their priorities.

The model is solved and validated using Lingo software to illustrate its solution and behavior. Lingo is set to use Branch-and-Bound as an exact algorithm for solving the model. All scheduling problems that are solved using Lingo are small-sized problems since the scheduling problem is proved to be an NP-hard problem [49]. For this reason, using exact methods is not efficient to solve large size problems. Therefore, Genetic Algorithms (GA) is developed as a heuristic approach to find feasible solutions with a good quality in a reasonable computational time. The GA is studied and evaluated on different problems with different sizes in order to estimate the effects of the model's different parameters and how they can be tuned properly. After developing the GA, a comprehensive comparison is performed between the exact solutions obtained from Lingo and the heuristic solutions obtained from the GA.

As a methodology to prove the efficiency of the GA algorithm in a real-time environment, the Edge-Fog-Cloud 3-layered architecture in a Matlab tool named Simevents is developed. The GA scheduler is integrated with SimEvents environment

to help scheduling requests as they arrive. The service latency provided by the GA is then compared to other traditional scheduling algorithms in computer networking. These algorithms namely are waited-fair queuing (WFQ), priority-strict queuing (PSQ), and round robin (RR).

6.2 Future Research

This work can be improved by extending the model in order to consider other attributes and characteristics in edge, fog and cloud computing environments. For instance, the resources attributes can be extended to cover attributes such as resource speed and architecture, storage capacity, storage speed, memory capacity, and types of operating systems. The resources might be modified to have the ability to process more than one request at a time. Preemption might be allowed also in order to be able to modify schedule solutions with more flexibility. Requests can also be extended to have more than one operation or task per request. As it is in this model, each request has only one task to perform. Requests can also have location constraints that a request must be served in a specific resource. The objective function can also be extended to include objectives like resource utilization, network utilization, and energy consumption.

Generally, fog computing paradigm and its relevance to IoT and cloud computing is a promising technology in the future. Its paradigms integration can foster a number of computing and network-intensive pervasive applications under the incoming realm of the future internet [16].

References

[1] S. Antonio, "Cisco Delivers Vision of Fog Computing to Accelerate Value from Billions of Connected Devices," Internet: https://newsroom.cisco.com/press-release-content?type=webcontent&articleId=1334100, Jan. 29, 2014 [Jan. 24, 2017].

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proc. of 1st Edition of MCC Workshop on Mobile Cloud Comput.*, 2012, pp. 13-16.

[3] S. Sarkar, S. Chatterjee, and S. Misra, "Assessment of the Suitability of Fog Computing in the Context of Internet of Things," in *IEEE Trans. on Cloud Comput.*, vol. 99, Oct. 2015, pp. 1-1.

[4] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, and B. Koldehofe, "Mobile fog: A programming model for large-scale applications on the internet of things," in *Proc. of 2nd ACM SIGCOMM workshop on Mobile cloud comput.*, 2013, pp. 15-20.

[5] P. V. Patil, "Fog Computing," in *IJCA Proc. on Nat. Conf. on Recent Trends in Mobile and Cloud Comput.*, 2015, pp. 1-6.

[6] M. Abdelshkour, "IoT, from Cloud to Fog Computing," Internet: http://blogs.cisco.com/perspectives/iot-from-cloud-to-fog-computing, Mar. 25, 2015 [Feb. 18, 2017].

[7] S. K. Datta, C. Bonnet, and J. Haerri, "Fog Computing architecture to enable consumer centric Internet of Things services," in *Int. Symp. on Consum. Electron.*, 2015, pp. 1-2.

[8] M. Aazam and E. N. Huh, "Fog computing and smart gateway based communication for cloud of things," in *Int. Conf. on Future Internet of Things and Cloud*, 2014, pp. 464-470.

[9] S. Yi, Z. Qin, and Q. Li, "Security and privacy issues of fog computing: A survey," in *Int. Conf. on Wireless Algorithms, Syst., and Applicat.*, 2015, pp. 685-695.

[10] I. Stojmenovic, "Fog computing: A cloud to the ground support for smart things and machine-to-machine networks," in *Telecommun. Networks and Applicat. Conf.*, 2014, pp. 117-122.

[11] S. Yi, C. Li, and Q. Li, "A survey of fog computing: concepts, applications and issues," in *Proc. of 2015 Workshop on Mobile Big Data*, 2015, pp. 37-42.

[12] C. Dsouza, G. Ahn, and M. Taguinod, "Policy-driven security management for fog computing: Preliminary framework and a case study," in *IEEE 15th Int. Conf. on Inform. Reuse and Integr.*, 2014, pp. 16-23.

[13] M. Yannuzzi, R. Milito, R. Serral-Gracià, D. Montero, and M. Nemirovsky, "Key ingredients in an IoT recipe: Fog Computing, Cloud computing, and more Fog Computing," in *IEEE 19th Int. Workshop on Comput. Aided Modeling and Design of Commun. Links and Networks*, 2014, pp. 325-329.

[14] Cisco, "Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are," Internet: https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf, Jun. 11, 2015 [Mar. 17, 2017].

[15] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," in *IEEE pervasive Comput.*, vol. 8, Oct. 2009, pp. 14-23.

[16] E. Baccarelli, P. Naranjo, M. Scarpiniti, M. Shojafar, J. Abawajy, "Fog of Everything: energy-efficient networked computing architectures, research challenges, and a case study," in *IEEE Access*, May 2017, pp. 9882-9910.

[17] L. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," in *ACM SIGCOMM Computer Commun. Review*, 2014, pp. 27-32.

[18] S. Yi, Z. Hao, Z. Qin, and Q. Li, "Fog computing: Platform and applications," in *3rd IEEE Workshop on Hot Topics in Web Syst. and Technol.*, 2015, pp. 73-78.

[19] R. Aburukba, H. Ghenniwa, and W. Shen, "Agent-based approach for dynamic scheduling in content-based networks," in *IEEE Int. Conf. on e-Bus. Eng.*, 2006, pp. 425-432.

[20] I. Stojmenovic and S. Wen, "The fog computing paradigm: Scenarios and security issues," in *Federated Conference on Comput. Sci. and Informat. Syst.,* 2014, pp. 1-8.

[21] G. Bitran and H. Yanasse, "Computational complexity of the capacitated lot size problem," in *Manage. Sci.*, vol. 28, Oct. 1982, pp. 1174-1186.

[22] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for internet of things and analytics," in *Big Data and Internet of Things: A Roadmap for Smart Environments*, 2014, pp. 169-186.

[23] J. Zhu, D. Chan, M. Prabhu, P. Natarajan, H. Hu, and F. Bonomi, "Improving web sites performance using edge servers in fog computing architecture," in *IEEE 7th Int. Symp. on Service Oriented Syst. Eng.*, 2013, pp. 320-323.

[24] Y. Wang, R. Chen, and D. Wang, "A survey of mobile cloud computing applications: perspectives and challenges," in *Wireless Personal Commun.*, vol. 80, Feb. 2015, pp. 1607-1623.

[25] A. Ahmed and E. Ahmed, "A survey on mobile edge computing," in *10th Int. Conf. on Intell. Syst. and Control.*, 2016, pp. 1-8.

[26] Cisco, "Cisco IOx," Internet: http://www.cisco.com/c/en/us/pro ducts/cloud-systems-management/iox/index.html [Apr. 8, 2017].

[27] B. Ottenwälder, B. Koldehofe, K. Rothermel, and U. Ramachandran, "MigCEP: operator migration for mobility driven distributed complex event processing," in *Proc. of 7th ACM Int. Conf. on Distrib. Event-Based Syst.*, 2013, pp. 183-194.

[28] T. Nishio, R. Shinkuma, T. Takahashi, and N. Mandayam, "Service-oriented heterogeneous resource sharing for optimizing service latency in mobile cloud," in *Proc. of the 1st Int. Workshop on Mobile Cloud Comput. & Networking*, 2013, pp. 19-26.

[29] Y. Cao, P. Hou, D. Brown, J. Wang, and S. Chen, "Distributed analytics and edge intelligence: Pervasive health monitoring at the era of fog computing," in *Proc. of the 2015 Workshop on Mobile Big Data*, 2015, pp. 43-48.

[30] M. A. Hassan, M. Xiao, Q. Wei, and S. Chen, "Help your mobile applications with fog computing," in *12th Annu. IEEE Int. Conf. of Sensing, Commun., and Networking Workshops*, 2015, pp. 1-6.

[31] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, "Towards wearable cognitive assistance," in *Proc. of 12th Annu. Int. Conf. on Mobile Syst., Applicat., and Services*, 2014, pp. 68-81.

[32] I. Maros, "Computational Techniques of the Simplex Method," 1st ed, New York: Springer, vol. 61, Dec. 2002.

[33] J. Clausen, “Branch and bound algorithms-principles and examples,” in *Dept. of Comput. Sci. in Univ. of Copenhagen*, 1999, pp. 1-30.

[34] E. Lawler and D. Wood, “Branch-and-bound methods: A survey,” in *Operations research*, vol. 14, Aug. 1966, pp. 699-719.

[35] N. Mansour and K. El-Fakih, “Simulated annealing and genetic algorithms for optimal regression testing,” *Journal of Software Maintenance,* vol. 11, Jan. 1999, pp. 19-34.

[36] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by simulated annealing,” in *Science*, vol. 220, May 1983, pp. 671-680.

[37] D. E. Goldberg and J. H. Holland, “Genetic algorithms and machine learning,” in *Machine* learning, vol. 3, Oct. 1988, pp. 95-99.

[38] F. D. Croce, R. Tadei, and G. Volta, “A genetic algorithm for the job shop problem,” in *Computers & Operations Research*, vol. 22, Jan. 1995, pp. 15-24.

[39] J. Yu and R. Buyya, “A budget constrained scheduling of workflow applications on utility grids using genetic algorithms,” in *Workflows in Support of Large-Scale Sci. Workshop*, 2006, pp. 1-10.

[40] M. Dorigo, V. Maniezzo, and A. Colorni, “Positive feedback as a search strategy," in *Tech. Rep.*, Politecnico di Milano, 1991, pp. 91-016.

[41] J. Yu and R. Buyya, “Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms,” in *Scientific Programming*, vol. 14, 2006, pp. 217-230.

[42] M. Wieczorek, R. Prodan, and T. Fahringer, “Scheduling of scientific workflows in the ASKALON grid environment,” in *ACM SIGMOD Record*, vol. 34, 2005, pp. 56-62.

[43] M. A. Rodriguez and R. Buyya, “Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds,” in *IEEE Trans. on Cloud Comput.,* vol. 2, Apr. 2014, pp. 222-235.

[44] S. C. Nayak and C. Tripathy, “Deadline sensitive lease scheduling in cloud computing environment using AHP,” *Journal of King Saud University-Computer and Informat. Sci.*, 2016.

[45] M. Mao and M. Humphrey, “Auto-scaling to minimize cost and meet application deadlines in cloud workflows,” in *Int. Conf. of High Performance Comput., Networking, Storage and Anal., 2011*, pp. 1-12.

[46] M. Rahman, S. Venugopal, and R. Buyya, “A dynamic critical path algorithm for scheduling scientific workflow applications on global grids,” in *IEEE Int. Conf. of E-Sci. and Grid Comput.*, 2007, pp. 35-42.

[47] W. N. Chen and J. Zhang, “An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements,” in *IEEE Trans. on Syst., Man, and Cybern.*, vol. 39, Jan. 2009, pp. 29-43.

[48] S. Dey, A. Mukherjee, H. S. Paul, and A. Pal, “Challenges of Using Edge Devices in IoT Computation Grids,” in *Int. Conf. of Parallel and Distrib. Syst.*, 2013, pp. 564-569.

[49] Lingo, “Solver Status Window,” Internet: http://www.lindo.com/doc/online_help/lingo15_0/solver_status_window.htm, [Apr. 04, 2017].

Vita

Mazin Abdelbadea Nasralla Alikarar was born in 1991, in Tabuk, Saudi Arabia. He moved to Sudan in 1999 where he studied in public schools and graduated as the top student in Sudan high school exams from El-Siekh Yousif El-Degair high school in 2008.

In 2013, he graduated from University of Khartoum. His degree was a Bachelor of Science in Electronics and Computer Systems. After Graduation he worked for two years as a software developer in a private company in Khartoum. In 2015, he joined the Computer Engineering Program at the American University of Sharjah where he was a Graduate Teaching Assistant.

Engineer mazin participated in “IEEE International Conference On Communication, Control, Computing, and Electronic Engineering” (ICCCCEE 2017) in Khartoum, Sudan, where he presented a paper titled “DSP-Based Dispersion Compensation: Survey and simulation”. He also participated in “Embedded Security Challenge” organized by New York University, Abu Dhabi (NYUAD) in 2016 where he won the 2nd place.