Scheduling Internet of Things Requests to Minimize Latency in Hybrid Fog-Cloud Computing

Raafat O. Aburukba[a,*], Mazin AliKarrar[a], Taha Landolsi[a], Khaled El-Fakih[a]

[a]*Department of Computer Science and Engineering, American University of Sharjah, Sharjah, PO Box 26666, UAE*

Abstract

Delivering services for Internet of Things (IoT) applications that demand real-time and predictable latency is a 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 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 work 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. This work introduces a customized implementation of the genetic algorithm (GA) as a heuristic approach to schedule the IoT requests to achieve the objective of minimizing the overall latency. The GA is tested in a simulation environment that considers the dynamic nature of the 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%.

Keywords: Internet of Things; cloud computing; fog computing; latency; scheduling; optimization; genetic algorithm

1. Introduction

Cisco introduced the concept of Fog computing paradigm in [1]. The term Fog computing is chosen as an analogy where a Fog is a Cloud close to the ground or the edge [2]. Some work refers to Fog as an abbreviation for "From cOre to edGe" [3]. In Fog computing, the computation and storage capabilities are deployed in networking devices at different levels in the network architecture. These devices are equipped with schedulers that make decisions whether to serve a request at the Fog nodes or at the Cloud Datacenters (DCs). Such decision is typically based on multiple attributes related to the requests and the resources available at the Fog and Cloud. Fog benefits the Internet of Things (IoT) applications as it brings the computational and networking resources closer to the edge devices. Thus, allowing services and computations to run as close as possible to end users and IoT devices that require computational capabilities in real-time with low latency.

With the high increase of the internet-connected devices and the advent of IoT, the data that needs to be processed and served is becoming voluminous. Cisco estimated that in 2020 there will be 50 billion devices connected to the Internet [1]. These devices will be generating and exchanging data in large volume, at high velocity, and with noticeable heterogeneity, which is known as "Big Data". A large amount of the data generated by IoT applications can be classified as latency sensitive where applications require a processing response in real-time (seconds or microseconds). Moreover, the generated data from some IoT devices may not be valid after seconds (data expiry time) from the generation time. Given the nature of such

* Corresponding author. fax: +971 6 515 2979.
E-mail address: raburukba@aus.edu.

applications, the stringent delay requirement makes processing this data at the Cloud level a potential performance bottleneck. Fog and Cloud computing create a cooperative and comprehensive architecture in which each one completes what the other lacks. The interaction between Fog and Cloud is a promising paradigm that will enable serving billions of IoT devices and applications with low latency. It helps significantly in monitoring and managing massive amount of data generated from the IoT devices. Examples of these applications include industrial automation, transportation, live streaming, real-time and online gaming, augmented reality, connected vehicles, remote patient health monitoring, smart micro grid, and smart traffic [2][3][4].

The main factor that distinguishes Fog computing from Cloud computing is its closeness to end users. In the Fog, the services can be hosted at edge devices such as access points, routers, switches, base stations, and even end-users' devices. Fog computing, being at the edge of the network, may be used to address the latency concern of Cloud computing. Fog computing has a list of characteristics mentioned in [2][4][5]. Table 1 summarizes these characteristics and presents a Cloud-vs-Fog computing comparison.

Table 1: Fog versus Cloud Characteristics

Cloud computing characteristics	**Fog computing characteristics**
Large number of computational resources	Small-size computational resources
Multi-hop WAN-based access	Single-hop WLAN-based access
High communication latency deployment	Low communication latency deployment
Ubiquitous coverage and fault-resilient	Intermittent coverage and fault-sensitive
Context-unawareness	Context awareness
Unlimited power supply (exploits electrical grids)	Limited power supply (exploits renewable energy)

The objective of this research is to model the problem of scheduling IoT requests generated by edge devices, and assigning them to the adequate resources available at both the Fog and Cloud. We use the Integer Linear Programming (ILP) technique to model the minimum service time for IoT requests. A heuristic approach using genetic algorithm (GA) is developed to obtain a feasible solution with a good quality in a reasonable computational time. The GA is customized for the proposed model to minimize latency. The work presents an adequate representation of a possible solution (chromosome) and a well-designed crossover operator. In addition, the GA includes a procedure to penalize possible solutions that do not satisfy the problem constraints. This will make the infeasible solutions to become less likely to be selected for producing new offspring chromosomes. The GA solution is provided to a network simulator to assess the impact of this solution on the delay performance. The service latency provided by the hybrid Fog-Cloud architecture that implements GA is 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) scheduling.

This paper is organized as follows: section 2 presents a review of related work, section 3 presents the formulation of the ILP model, section 4 illustrates the proposed adaptive genetic algorithm, section 5 presents the various simulation results, and section 6 presents the conclusions and future work.

2. Related Work

2.1. Fog computing definition

The work in [2-4][6-13] shared the same view of Fog computing as a distributed computing paradigm that extends the traditional Cloud computing resources at the edge of the network. Similar to Cloud computing, Fog computing provides ubiquitous computation, storage, networking, and application services in a highly visualized platform at the edge between end devices and 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.

The work in [14] 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 points out the proximity, wireless, decentralized characteristics of Fog computing. The definition also points out the potential cooperation between Fog computing devices which is a significant property within Fog computing paradigm for load balancing purposes. In this definition, the authors did not focus on addressing the interdependency and interaction between the Fog at the edge and the Cloud datacenter. Each one of these platforms has its own characteristics which are suitable for specific type of use cases to enable new spectrum of applications.

In contrast, the Fog-Cloud interdependency is stressed in [15] where the authors developed a definition that covers 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 realized the impact of the collaboration between Cloud computing, Fog computing, and edge devices in very large-scale systems. This latter definition is the most comprehensive and is adopted in this work.

2.2. Scheduling Techniques

Generally, scheduling is defined as the allocation of tasks to capable resources at a specific time. Usually, a scheduling problem is subject to many constraints and objectives that must be fulfilled. The scheduling problem can be mapped to an optimization problem where the scheduling problem objectives can be, for instance, minimizing latency and maximizing resource utilization. The work in [16] specifies that most of scheduling problems consist of four basic elements:

1. **Resources**: are physical/logical devices with the ability to execute or process tasks.
2. **Tasks**: are physical/logical operations that need to be executed by the resources.
3. **Constraints**: are conditions that must be considered 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**: are evaluation criteria that need to be measured in order to assess the schedule performance.

To find a 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 [17] and Branch-and-Bound [18][19]. On the other hand, heuristic methods 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. Examples of heuristic algorithms include: simulated annealing [20][21], genetic algorithms [22][24][25], and ant colony algorithms [26].

2.3. Fog Computing Implementations

There are few existing studies in the literature in terms of design, algorithms or implementations within Fog computing. Cisco introduced the IOx [27] which 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], the authors 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 applied Fog computing in smart grids where each micro-grid can be depicted as a fog computing node. Customers communicate to nearby Fog nodes rather than the Cloud. Fog devices coordinate with the Cloud data centers and customers to deliver power services.

The work in [28] 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 the Cloud datacenter. Moreover, the work in [29] proposed a healthcare reference model that consists of four layers: sensing layer that integrates different physical sensors that collect data, networking layer that offers networking support and data transfer in the wired and wireless networks, service layer that creates and manages all types of services aiming to satisfy user requirements, and interface layer that enables the interaction between users and applications. The work also elaborated in splitting the networking layer into two sub-layers namely the Fog layer to handle the local buffering and different connectivity requirements to IoT devices, and the Cloud layer that handles the connectivity to the Fog, user/device/data management, and application services such as dashboard, rule engine, big data analytics, and integration framework.

The work in [30] studied web optimization within the context of Fog computing. Fog servers connect the end users to the Internet so that all web requests must pass through the Fog servers on their way to the web servers of the Cloud. 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.

The authors in [31] 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 [32], mobile Cloud is a very similar model to Fog computing in which the resources are shared not only from central resources but also from pervasive mobile devices. Although these devices have heterogeneous resources (e. g. CPUs, bandwidth, content) and support services, they still can 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 [33], the authors introduced a new computational framework that enables remote real-time sensing, monitoring, and scalable computing for diagnosis and prognosis in a Fog-based environment. The framework also utilizes wireless sensor networks, Cloud computing, Fog computing and machine learning. The Fog computing-based framework has been used for data-driven machine health and process monitoring in cyber-manufacturing. The objective of Fog cyber-manufacturing is also

to provide the foundation to smart manufacturing networks in which manufacturers will have access to on-demand computing infrastructures, mobile applications for cyber-manufacturing, and parallel machine learning tools.

The authors in [34] proposed an approach for distributing event-based across edge and Cloud resources to support IoT applications. They considered an IoT deployment scenario with multiple event streams generated at the edge at high frequency and a user-defined analytics dataflow composed of queries that needs to be executed on these streams, and several edge devices in a private network and public Cloud VMs available to perform the queries. Their goal was to determine a distributed placement of these queries onto the edge and Cloud resources such that the end-to-end latency for performing the event analytics is minimized to support timely decision making. This placement schedule needs to meet constraints such as the throughput capacity supported on edge and Cloud machines by the queries, bandwidth and latency limits of the network, and energy capacity of the edge devices. Another work in [35] proposed a workflow service request and a dynamic minimum response time that considers the same level algorithm to map requests in edge computing. The experimentation showed that the proposed algorithm performed very well in response time and blocking rate. The network traffic problem in Cloud and Edge Computing is a core challenge that is addressed in [36]. The work presented two optimization models that minimize the maximum percentage of the average packet time and the average arriving time for each node. While the second model deals with maximizing the local network link utilization. The work introduced multi- topology routing to ensure the performance of the assigned egress routers with a near-optimal solution based on the proposed heuristic approach. The results show that the proposed algorithm has a lower execution time and better quality of service than other approaches with the aim to satisfy the flexibility and efficiency demands of network traffic problems in cloud and edge computing.

Security is a major challenge in Fog computing given the nature of the environment. The work in [37] proposed a cybersecurity framework to identify malicious edge devices in the distributed fog computing environment. The framework uses the two-stage Markov model for early prediction of the malicious and legitimate edge devices. The results presented the effectiveness of the proposed framework.

2.4. Related work using Genetic Algorithms

The work in [38] proposed a budget constraint based scheduling model to minimize execution time while meeting a specified budget for delivering results. They modelled 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 computing environment.

In [39], different existing approaches were comparatively examined for scheduling of scientific workflow applications in Grid computing 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 [40], the authors proposed a resource provisioning and scheduling strategy for scientific workflows on Infrastructure as a Service (IaaS) Cloud environments. They 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.

The authors in [41] 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.

The authors in [42] presented an auto-scaling mechanism for Cloud computing environments. In the authors 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. The approach sorts the tasks by their deadlines for each VM type, and schedule the task with the earliest deadline whenever an instance is available.

In [43], 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.

The authors in [44] presented a collaborative architecture of thin clients and conventional desktop or laptop computers, known as thick clients, aiming to improve Cloud access. Such an architecture enables offloading computation-intensive, resource-consuming tasks up to a powerful computing platform which is Cloud, leaving only simple jobs to the capacity-limited thin client devices such as smartphones and tablets. Additionally, they introduced a genetic approach for task scheduling such that the processing time is minimized. The network needs to support high bandwidth and low latency connection between clients and the Cloud. They compared the performance of their approach with two algorithms: Contention aware Scheduling (CaS) and Greedy for Cost (GfC) algorithm.

In [45], the authors investigated the collision issues within the Internet of Vehicles (IoV) and proposed a collision prediction and avoidance technique. The proposed technique was based on deep learning that uses genetic algorithm back propagation neural network to forecast the rear-end collision probability in the IoV. The work considered influential factors such as the driver, road, vehicle, and the surrounding external environment. The genetic algorithm was used to demonstrate the accuracy between the predicted and actual values for the Internet of Vehicles. The work also showed that GA can be improved with the use of deep learning. Moreover, the work in [46] presented genetic algorithm in predictive modeling and forecasting the green economic performance for achieving supplier selection in cities.

In this work, we are approaching the latency sensitivity for IoT applications and requests as a scheduling problem that involves resources in both Fog and Cloud. Previous studies (such as in in [5] and [47]) tried to solve the latency problem by using Fog as a replacement for Cloud. In such environment, the interoperation between Fog and Cloud is very limited. It is also not a realistic solution as Fog devices are resource-limited by nature where the Cloud is resource-rich. In this work, the GA is customized for the proposed ILP problem; namely, an appropriate representation of a possible solution (chromosome) and a well-designed crossover operator are provided. In addition, the GA included a procedure to penalize possible solutions that do not satisfy the problem constraints so that they become less likely to be selected for producing new offspring chromosomes.

3. Problem Model

3.1. Environment Analysis

Figure 1 depicts the high-level conceptual view of Fog computing. The integration of Fog and Cloud computing creates a 3-layered architecture. The three layers are:

1. **Edge layer**: it consists of terminal nodes, embedded systems, sensors and actuators with very limited computation, energy and bandwidth.
2. **Fog layer**: it has intermediate networking devices such as routers, gateways, switches, and access points that work on different protocols such as 3G, 4G, LTE, and WiFi. These devices are supported with computational and storage capabilities.
3. **Cloud layer**: it consists of the Cloud data centers that have very rich virtual capabilities in terms of storage and processing power.

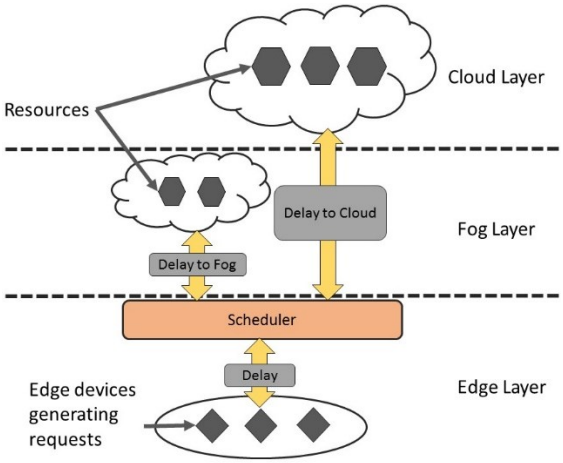

Figure 1. Fog and Cloud Computing High-level Architecture

In edge layer, there are a set of n requests denoted by $\mathcal{R}$ where $\mathcal{R} = \{r_j \mid j = 1, \dots, n\}$. Each request has the following characteristics:

- Q_j: size of request r_j in packets
- IT_j: Initiation time of request r_j
- D_j: Deadline for request r_j
- W_j: Priority weight factor for request r_j. Each request priority is assigned as a normalized fractional weight, W_j, to signify its importance with respect to the other requests. All weights should sum up to an exact 1, i.e. $\sum_{j=1}^{n} W_j = 1$.

Those requests require processing power P_i from the set of heterogeneous resources $\mathcal{S} = \{s_i \mid i = 1, \dots, m\}$. A resource is a physical device where a request is processed. Resource s_i might reside close to the edge at the Fog or further from the edge at the Cloud. Each request r_j experiences specific delays as follows:

- Transmission delay, which is the time it takes to push the request packets' bits onto the connection link.
- Queuing delay, which is the time the request data packets are spend in passing through the network routers and switches.
- Propagation delay, which is the time for a signal to propagate through the transmission media before it reaches its destination.

In this work, the propagation delay is neglected as it is insignificant compared to the other delays.

The transmission delay is a function of the packet's length and the link transmission rate. It has a deterministic value for fixed packet sizes and transmission rates, and hence, can be evaluated beforehand. On the contrary, the queuing time is stochastic in nature. For this reason, the total delay is modelled as a random variable with an average value given by:

$$\bar{\delta} = \delta_{tr} + \bar{\delta}_q \quad (1)$$

where δ_{tr} is the deterministic transmission delay and $\bar{\delta}_q$ is the average value of the queuing delay.

Additionally, the actual processing time is denoted as PT_{ij}, which is the time it takes to process each request r_j by resource s_i. 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 of the request and the processing capability of the resource that processes the request. The resource capability is defined in terms of processing speed P_i. Moreover, a request might experience additional delay if the resource is busy serving other requests.

The scheduler as shown in Fig. 1 is to allocate requests in $\mathcal{R}$ to the capable resources in $\mathcal{S}$ at a specific time with the objective of minimizing latency. The latency of request r_j is defined as the round-trip time (RTT) from the moment the request is initiated at the edge to the moment it is completely processed and the results are returned to the requester. The scheduler has the environment knowledge such as the number of resources, resources capabilities, resources availability, resource networking and transmission delays, number of requests, and the requirements of requests.

The scheduling problem model is built based on the following environment settings. Those settings are identified to present the scope of the problem model and its operating environment. The model assumptions are:

- A resource as a compute processor, at both the Fog and the Cloud, can process only one request at a time.
- A request can be initiated at any instant of time.
- The data size of each request is defined by the number of packets the request has. Each request may have multiple packets where each data packet is assumed to be of the same size.
- Pre-emption is not allowed. If a request starts processing, it must be finished without interruption.

3.2. Model Formulation:

Objective function:

Consider a set or n requests, $\mathcal{R} = \{r_j \mid j = 1, \dots, n\}$, that are generated from the edge-layer IoT devices and that need to be processed by a set of m resources $\mathcal{S} = \{s_i \mid i = 1, \dots, m\}$. The resources are distributed among the Fog and the Cloud. The objective is to minimize the overall latency defined as the sum of the RTT for serving all n requests using m available resources, while taking into account the various constraints of the problem. Mathematically, the latency is the difference between the initiation time and the end of service time as presented in equation (4). The service time includes the starting time, the processing time, and the transmission and queuing times. The overall latency to minimize as the problem objective function, is defined as the weighted sum of the individual requests' latencies as follows in equation (2):

$$\min \quad \sum_{i=1}^{m} \sum_{j=1}^{n} \left(LT_{ij} \, W_j \, X_{ij} \right) \quad (2)$$

where LT_{ij} as presented in equation (4) is the latency incurred when serving the j-th request, denoted r_j, by the i-th resource, denoted s_i. Here, W_j is the weighting factor of request r_j describing its priority, and X_{ij} is a decision variable as presented in equation (3), that indicates whether request r_j is allocated to resource s_i. The individual latency of a given request is:

$$X_{ij} \triangleq \begin{cases} 1 & if\ request\ r_j\ is\ allocated\ to\ resource\ s_i \\ 0 & otherwise \end{cases} \tag{3}$$

$$LT_{ij} = ST_{ij} + PT_{ij} + TQT_{ij} - IT_j \tag{4}$$

Here, ST_{ij} is the start time of the processing of request r_j by resource s_i, P_{ij} is the processing time of the request, TQT_{ij} is the transmission and queuing time before request r_j reaches the resource s_i, and IT_j is the initiation time of request r_j. 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). Let Q_j denote the data size of request r_j and P_i denote the processing power of resource s_i, then the processing time is given by $PT_{ij} = Q_j/P_i$. The transmission and queuing time of request r_j from the edge layer to resource s_i, located in the Fog or the Cloud, is given by equation (5).

$$TQT_{ij} = \sum_{i=1}^{Q_j} \bar{\delta}_i \tag{5}$$

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.

Model constraints:

The objective function defined in Equation (2) is to be minimized subject to sets of constraints. The constraint in equation 6 presents each request r_j must be served by only one resource.

$$\sum_{i=1}^{m} X_{ij} = 1, \forall\ r_j \in \mathcal{R} \tag{6}$$

In the constraint presented in equation 7, a request r_j cannot start processing before getting transmitted to the resource. This amount of time includes the initiation time plus the transmission and queuing time to where the Fog or the Cloud resources reside. In other words, in order to process a request, this request must be created and transmitted to the resource first.

$$ST_{ij} \geq IT_j + TQT_{ij}, \forall\ r_j \in \mathcal{R}, \forall\ s_i \in \mathcal{S} \tag{7}$$

The constraints presented in equations 8 and 9 present each resource s_i can process only one request at a time. If there are two requests that arrive at the resource at the same time for processing, one of them must start processing after the other one finishes. The order in which these two requests get executed is defined by the parameter θ_{ijk}. If $\theta_{ijk} = 1$, resource s_i executes request r_j then request r_k and vice versa. The decision variable θ_{ijk} is defined as presented in equations 8 and 9 for $\forall\ s_i \in \mathcal{S}$, $\forall\ r_j, r_k \in \mathcal{R}, r_j \neq r_k$

$$if\ \ X_{ij} + X_{ik} = 2 \quad then \quad \theta_{ijk} + \theta_{ikj} = 1 \tag{8}$$

$$ST_{ik} \geq \theta_{ijk}\left(ST_{ij} + PT_{ij}\right) \tag{9}$$

The constraint presented in equation 10 takes into account the precedence dependency among requests. A request is not able to start processing unless its precedents finish processing even if they are allocated in different processors. This constraint is presented in equation 10 for $\forall\, s_i \in \mathcal{S},\ \forall\, r_j, r_k \in \mathcal{R}, r_j \neq r_k$.

$$ST_{ik} \geq \beta_{jk} \times \left(ST_{ij} + \sum_{i=1}^{m} PT_{ij} \times X_{ij}\right) \tag{10}$$

The constraint presented in equation 11 takes into account that a request must be served before a specific deadline. This constraint is presented in equation 11 for $\forall\, s_i \in \mathcal{S},\ \forall\, r_j \in \mathcal{R}$.

$$LT_{ij} \leq D_j \tag{11}$$

Table 1 summarizes the nomenclature used in the problem formulation.

Table 1. Summary of Model Nomenclature

Symbol	**Description**
s_i	i-th resource
r_j	j-th request
$\mathcal{R}$	Set of n request i.e. $\mathcal{R} = \{r_j \mid j = 1, \dots, n\}$
$\mathcal{S}$	Set of m resources i.e. $\mathcal{S} = \{s_i \mid i = 1, \dots, m\}$
LT	Average overall latency
LT_{ij}	Latency of request r_j when served by resource s_i
Q_j	Size of request r_j in packets
IT_j	Initiation time of request r_j
D_j	Deadline for request r_j
W_j	Priority weight factor for request r_j
P_i	Processing power of resource s_i in packets/s
δ_i	Average transmission and queuing delay for one packet
X_{ij}	Decision variable set to 1 iff request r_j is assigned to resource s_i
θ_{ijk}	Decision variable set to 1 iff request r_j is executed before request r_k in resource s_i
β_{jk}	Decision variable set to 1 iff request r_k depends on request r_j
ST_{ij}	Start time of processing request r_j by resource s_i
PT_{ij}	Time duration of processing request r_j by resource s_i
TQT_{ij}	Transmission and queuing times for request r_j before reaching resource s_i

4. Proposed Genetic Algorithm

Given the complexity nature of the problem being NP-Hard, exact optimal solutions are not adequate. A full analysis and proof of the complexity of similar class of problem can be found in [47]. Hence, in this work we propose a customized implementation of the Genetics algorithm. Genetic algorithm [22] 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 solutions (chromosomes) to an optimization problem is evolved toward better or more fit one. For a scheduling problem, a population chromosome represents a candidate solution to the problem. A solution is feasible if it satisfies the problem constraints; otherwise, it is infeasible. We propose a GA that employs a chromosomal representation, a crossover and mutation operators appropriately designed for the considered problem. In addition, the GA includes a procedure for penalizing infeasible solutions so that they have less probability of selection (or survival).

The algorithm for our implementation is shown in Algorithm 1. The GA initially defines the problem size, i.e. the number of requests and the available resources, and then it generates a population of POP number of candidate solutions (chromosomes or individuals) to represent the initial population. The next step is to evaluate the fitness F of each chromosome, defined as the inverse of the total latency i.e. $F = 1/LT$. Afterwards, we select mates or parent individuals (chromosomes) that will undergo crossover and mutation to create a new offspring population from the current population [23].

Table 2. Proposed GA Implementation Pseudocode

Algorithm: Genetic Algorithm

```
Define number of requests and resources attributes (i.e., problem size);
Set population size POP, and MAX number of iterations;
Generate initial population of POP individuals (or chromosomes);
Evaluate fitness individuals ;
repeat

  for ( i =1 to POP /2) do
      Select 2 parent individuals  from the current population using the Roulette Wheel method [23];
      Apply crossover and mutation to the selected individuals producing two offspring individuals;
  endfor

  Evaluate fitness of offspring individuals;
  Determine  feasibility of individuals;
  Do penalization to 50% of infeasible individuals;
  Preserve the best-so-far individual (elitism);
until (termination criterion is satisfied)
Return best-so-far chromosome as a solution to the scheduling problem
```

After crossover and mutation, the fitness value of each created offspring chromosome is evaluated according to the fitness function F. Next, a procedure is applied to determine infeasible chromosomes and then another procedure is applied to penalize some of them. In addition, the best-so-far chromosome, i.e. the one with the least fitness value, of all generated populations is preserved. The GA repeats the above steps until some termination criterion is satisfied as given in more details later. Upon termination, the GA returns the best-so-far chromosome representing a solution to the considered problem.

In our proposed algorithm, the operators are implemented as follows:

- **Chromosomal Representation**: In our GA, a candidate solution (chromosome) to the scheduling problem is a 2-dimisional array of X_{ij} elements (or genes), of 0 or 1 integer values, representing a possible allocation of requests to resources as shown in Figure 2 considering an allocating of n requests within m resources. A gene X_{ij} has the value 1 (0) if request j is assigned (not assigned) to resource i. This 2-dimensional array is stretched vertically, as shown in Figure 3, in order to get a 1-dimensional solution array that represents a chromosome. 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 equations 8 and 9)). This is an important parameter since it gives the time dimension to the scheduling problem and distinguishes it from the traditional allocation problems. This parameter is not considered as part of the chromosome. However, it is considered in the latency evaluation of each chromosome.
- **Fitness Evaluation**: For each chromosome of the population, the fitness of the given chromosome is calculated as $F = 1/LT$ where LT is given by the objective function in Equation (2). In this function, the algorithm determines the values of the decision variable in the problem, X_{ij} and θ_{ijk}.
- **Crossover and mutation**: Our GA uses the traditional Roulette Wheel method [23] for selecting each two parents from the current population that will mate (crossover) to reproduce a new offspring population. Then, for each two selected chromosomes, a crossover-point (or position) k is selected as in integer value between 2 and the total number of requests j. Thus, the crossover-point k is chosen delicately at a position (as shown in Figure 3) where we not to mix up the requests allocation. Then, and all genes corresponding to requests k to j are swapped to create two new chromosomes. 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. Afterwards, standard mutation is applied, where for each randomly selected gene X_{ij}, its value flipped to the opposite value (i.e. from 1 to 0 or the converse).
- **Feasibility check and infeasibility penalization**: GA checks the feasibility of the new offspring chromosomes after crossover and mutation. More precisely, the GA determines which new chromosomes are feasible, i.e., satisfy constraints (6) and (11). In other words, for each chromosome, it determines that each of its requests is allocated to one and only one resource (equation 6) and if the chromosome satisfies the deadline requirements of equation (11). 50% of the infeasible chromosome that do not satisfy constraints (6) and (11) are penalized by increasing their fitness values so that they have less probability of survival to the next generation.
- **Elitism and termination criteria**: the GA saves the best-so-far chromosome (the one that is feasible and has the least fitness value) of all generated populations. The algorithm stops when the termination criterion is satisfied. According to our experiments, this is done when the fitness of the best-so-far did not change for 4 consecutive iterations or when the maximum number of iterations MAX = 20 (determined as shown in the following section 5) is reached.

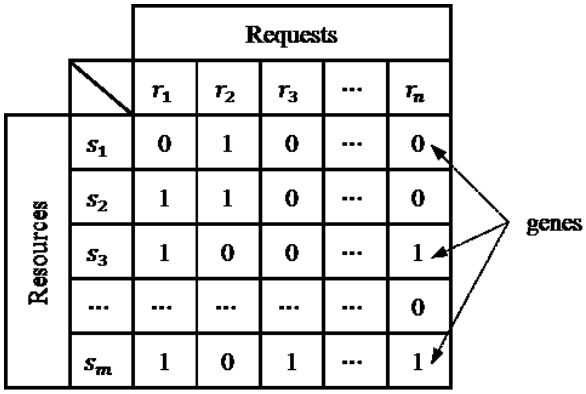

Figure 2. Chromosome representation as 2-d array.

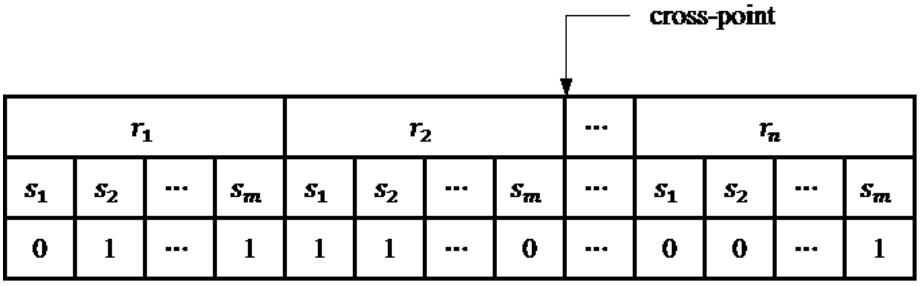

Figure 3. Chromosome representation as 1-d array showing a cross-point within a chromosome.

5. 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 (POP) and maximum number of iterations (MAX). These two parameters have a direct impact on the solution quality and runtime.

5.1. Population Size

Experiments are conducted to determine the population size (POP) considering both the quality of obtained solution as determined by the overall latency (LT) and the runtime (RT) of the GA. We consider varying size scheduling problems *N*, where each value of *N* is a tuple (*j*/*i*) of *j* requests and *i* resources. More precisely, we consider tuples of values (40/10), (6/20), (80/30), and (100/50) as shown in Figures 4 and 5, respectively. For each considered value of *N*, five different experiments are conducted and the averages of their corresponding LT and RT values are calculated as depicted in figures 4 and 5. For these experiments, the maximum number MAX of iterations of the GA is set to 50. Figures 4 and 5 depict the LT and RT versus POP, respectively. According to Figure 4, as POP increases, better solutions are usually obtained; yet, the gains in solution quality become marginal after POP = 40. However, according to Figure 5, RT continues to increase sharply as POP and problem size *N* increase. In this work, POP = 60 used as this number gives a good quality solution with a moderate runtime.

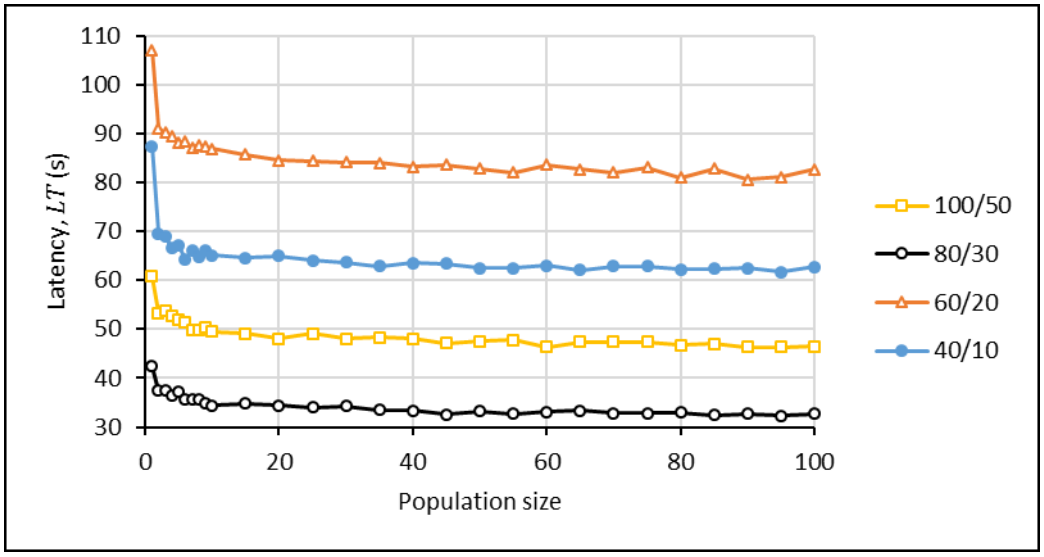

Figure 4. Overall latency vs. population size.

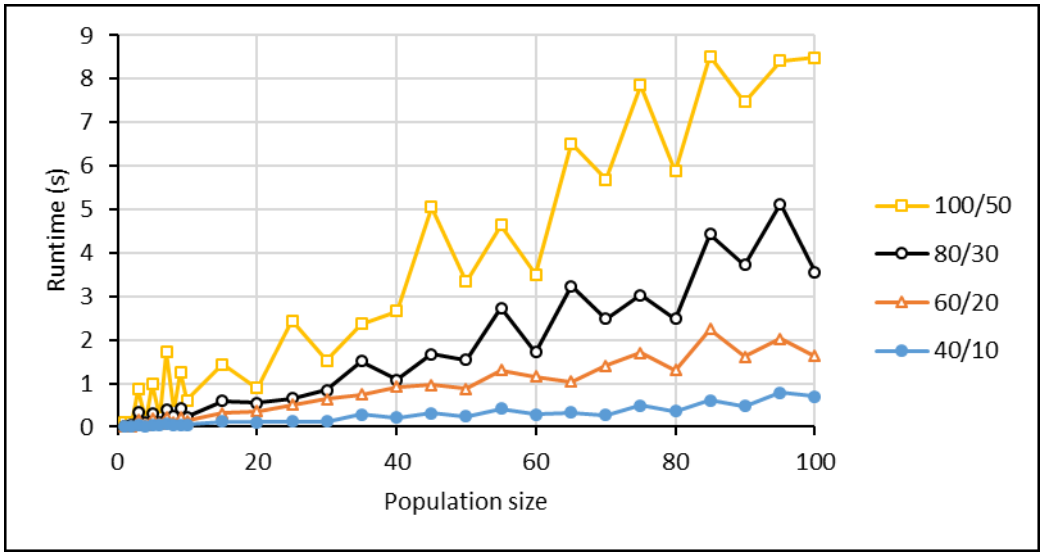

Figure 5. Runtime vs. population size.

5.2. Maximum Number of Iterations

In order to determine best value for the number of iteration MAX, experiments are conducted considering both the quality of obtained solution (as determined LT) and the runtime RT of the GA. In fact, we use the same values of N as given in the previous section, and vary the values of MAX as shown in Figures 6 and 7, respectively. For each combination of a considered value of N and MAX, five different experiments are executed and the average of the corresponding LT and RT values are depicted as shown in the figures. According to these experiments, an increase in MAX refines the solution quality (latency) up to a certain point; however, it usually sharply increases the runtime of the GA. In this work, MAX is set to 20 as it gives good quality solution with a moderate runtime.

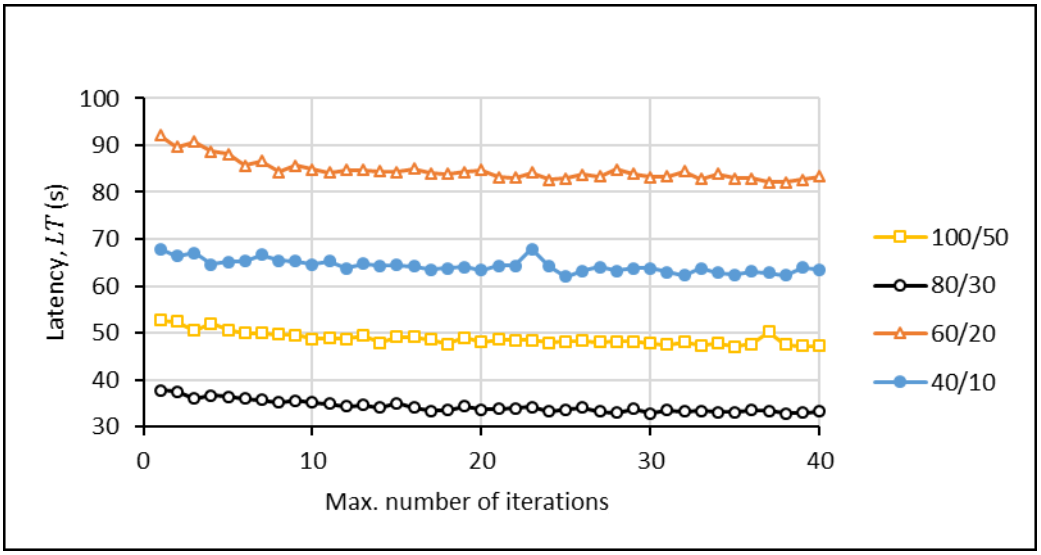

Figure 6. Overall latency vs. maximum number of iterations.

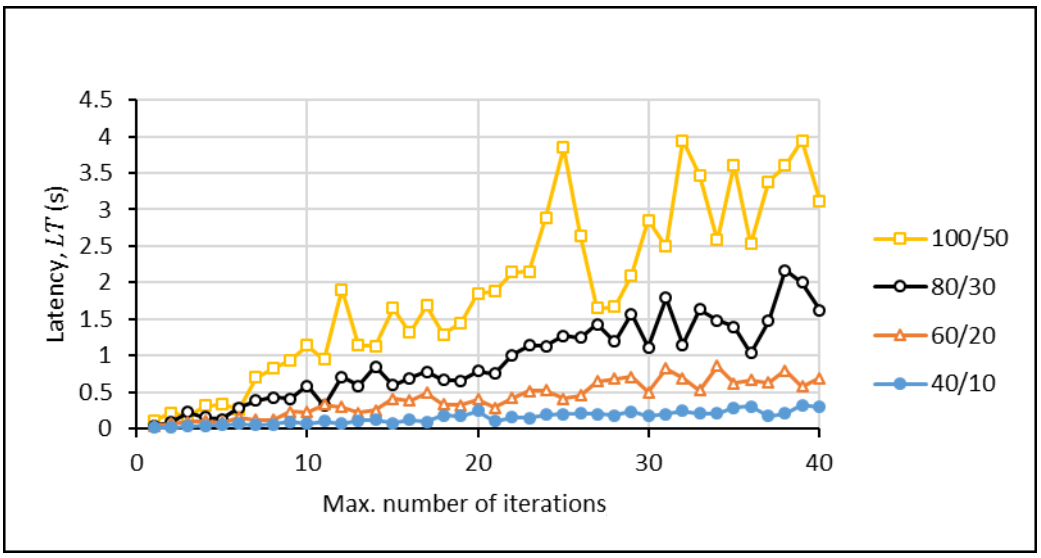

Figure 7. Runtime vs. maximum number of iterations.

5.3. Comparison of GA with Exact Algorithm

This section includes a comparison of the proposed GA with respect to an exact algorithm in terms of quality of solution (as determined by LT) and runtime (RT). Namely, we adopt the known Lingo software (Lingo, 2017) that uses a Branch-and-Bound algorithm (B&B) for providing the optimal quality solution of the considered problem. To this end, experiments are conducted using small size problems, as the exact algorithm did not converge to solutions for large size problems. In particular, we problems of the following combinations n requested and m resources are used, (4,2), (6,2), (8,2), (12,2), (8,3), and (10,3) as shown in Figure 8. Figures 8 and 9 depict the solution quality (latency in milliseconds) and runtime of both Lingo using B&B and GA, respectively, for the conduced experiments. According to these experiments, both the exact and the GA provide solutions of comparable quality; however, as shown in Figure 9, even for these small size problems, the runtime of the exact

is much more than that of the GA. In fact, the exact algorithm took almost two days to compute a solution for a problem with 10 requests and 3 resources.

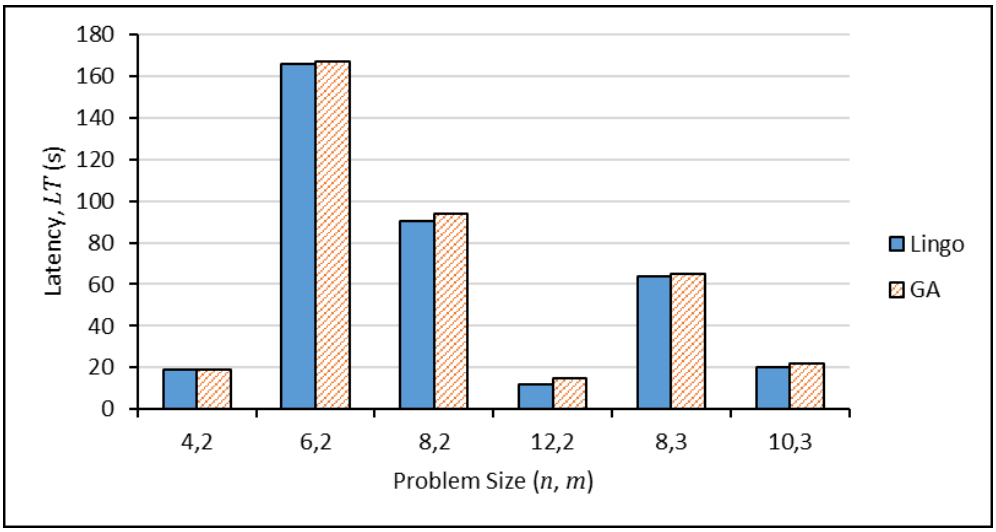

Figure 8. Latency comparison between heuristic (GA) and exact (Lingo) methods.

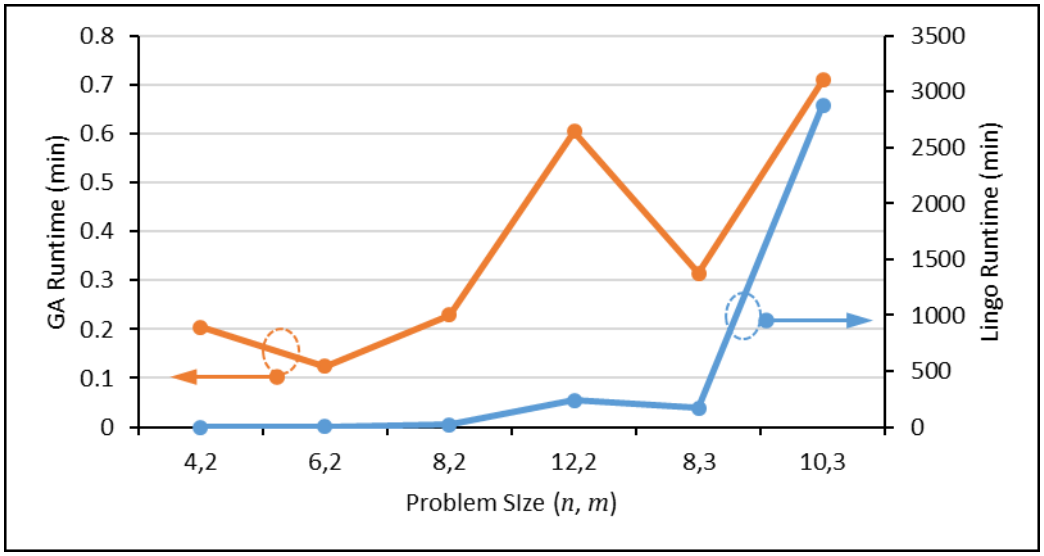

Figure 9. Runtime comparison between heuristic (GA) and exact (Lingo) methods.

6. Simulation and Results

A simulation model from the formulated model is developed using the discrete event simulator. The simulation is built based on Edge-Fog-Cloud 3-layered architecture as shown in Figure 1. At the edge layer, requests are generated with a specific distribution for the inter-arrival times. Each generated request is associated with its attributes defined in the model. The requests travel from the edge to the propose GA scheduler that decides on when and where to allocate the requests on the Fog or the Cloud resources. The GA scheduling algorithm is integrated with the discrete event simulator. The GA scheduler receives requests within specific defined time frames. Within any time frame, the GA scheduler can receive any number of requests. Initially, all the resources in the simulator are available. As time passes, and requests are initiated, those requests are

allocated to the resources. Hence, some of the resources will be busy processing those initiated requests. This means, the GA generates a scheduling solution and keeps the generated solution knowledge for future scheduling decisions as new requests arrive.

The main resources attributes defined in the model are the processing power and the average delay. The GA solver assumes all requests packets are delayed to be served based on the average historical delay of serving the previously processed requests. Hence, in the simulation environment, the actual delay per packet can be the same as the average value or different based on the distribution used. In the simulation environment for the produced results, the transmission and queuing delay distribution is set to Gaussian with specific mean and variance. When a request is sent to a specific resource to be served and it reaches the resource, if the resource is available and the waiting queue is empty, the request is served right away. Otherwise, the request is pushed into a waiting queue and served as soon as the resource become available.

In this experiment, the GA scheduler performance is compared to other traditional algorithms. These algorithms namely are Waited Fair Queuing, Priority Strict Queuing, and Round Robin [48]. They are compared upon two metrics: the overall average service latency and the number of missed-deadline requests. The population size and maximum number of iterations are set to 60 and 20 respectively as concluded in section 5.1 and 5.2. The comparison is performed based on two scheduling modes: static scheduling and dynamic scheduling. 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 dynamic scheduling, requests are generated using a normal distribution and inter-arrival rate.

6.1. Static Scheduling

This experiment involves a set of 16 servers with an average processing speed of 500 packets per second and widely distributed from 50 to 1000 packets per second. The servers' average delays are set to an average of 5 milliseconds per packet. It is also 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. 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. Initially, the average data size starts from a small value in a way that makes requests deadline requirements 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.

Figures 10 and 11 shows the average overall latency and the number of missed-deadline requests versus requests average size. 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 is the way requests are allocated in a RR fashion without considering requests priorities. It can also be observed that the GA keeps the record clean of missed-deadline requests for longer time than the other algorithms. However, at an average size of 6500 packets, the GA cannot guarantee meeting all requests deadlines as their service latency increases and their deadlines become very critical. 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 between 6000 and 8000 average data size in Figure 11. After a data size of 8000 packets, 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.

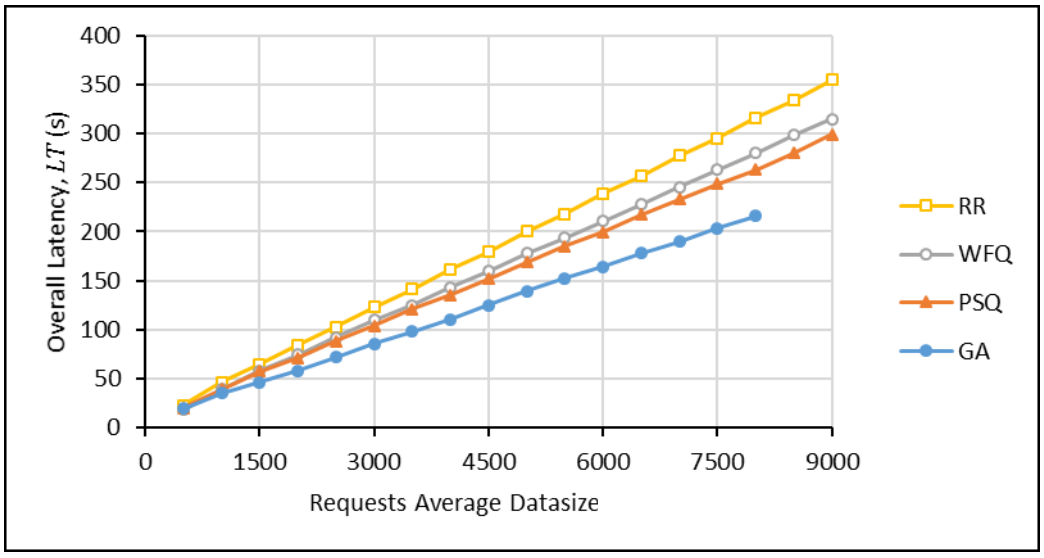

Figure 10. Overall latency vs. data size.

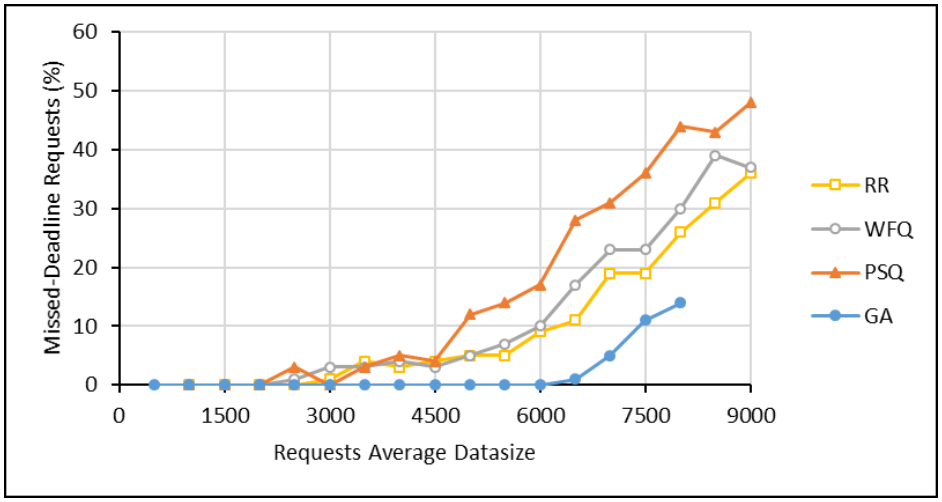

Figure 11. Missed deadline requests vs. data size in static scheduling.

6.2. Dynamic Scheduling

This experiment setup contains 16 servers and 500 requests with the objective of the experiment to evaluate the latency as requests arrive over time. 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 is 10 seconds. The average request data size is normally distributed from 1000 to 10000 packets.

Figures 12 and 13 show the overall average latency and the number of missed-deadline requests versus the average data size. 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 13 shows that the GA achieved 0 missed-deadline requests if the data size is less than 5000 packets on average. At an average data size of 6000 packets, the GA achieved 0 missed requests, WFQ and PSQ lost almost 10% of the requests and RR lost 6% of the

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 when the data size is increased above 6000 packets. However it still achieves better results than the other 3 algorithms.

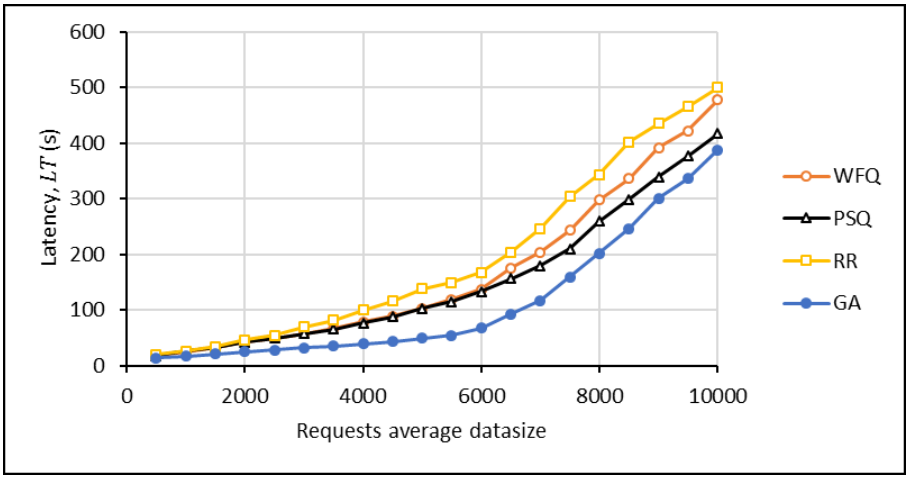

Figure 12. Dynamic scheduling latency versus data size.

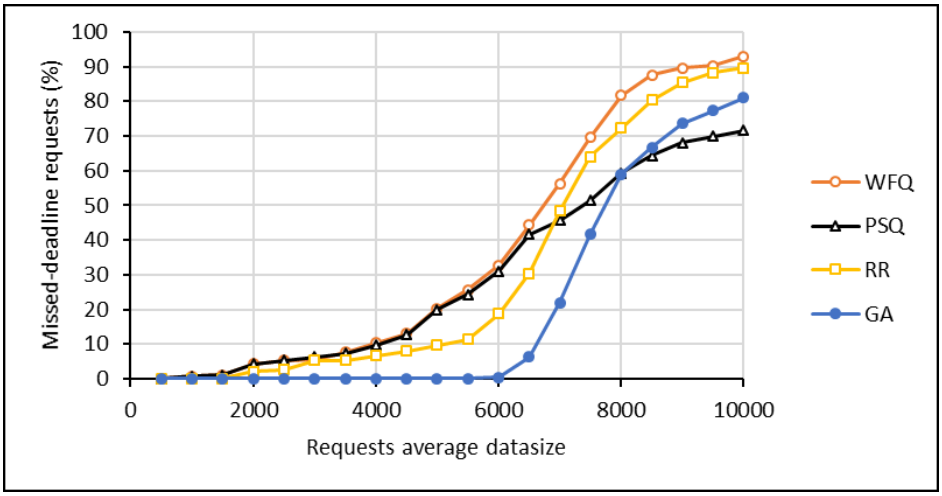

Figure 13. Missed-deadline requests versus data size in dynamic scheduling.

6.3. Comparison of Cloud-only and Hybrid Cloud-Fog Architectures

The objective of this section is to evaluate the service latency provided by resources that have cloud and fog computing characteristics. In general, cloud resources are classified as powerful with high processing capabilities, but at the same time they have large average transmission and networking delay. Conversely, fog resources have limited 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 the cloud or to put much less powerful resources closer to requests sources in the fog.

In order to be able to serve a set of requests with a minimized latency, three parameters are considered in the formulated model as follows:

- The ratio of average delays, $\bar{\delta}_f/\bar{\delta}_c$.
- The ratio of processing speeds, P_f/P_c
- The ratio of resource numbers, N_f/N_c

The impact of each parameter on the service latency is studied independently by fixing two of them and varying only one. The latency that is 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, the average delays for these servers set to be relatively high as 10 milliseconds per packet. To evaluate the latency, a total of 500 requests are used in this set of experiments. Their arrivals follow a Poisson distribution with a mean inter-arrival time of 1 second. The latency is studied versus the average data size which will be changed from 1000 packets to 10000 packets.

Impact of the Ratio of Average Delays:

In this experiment, the number of Fog servers is set to four times the number of Cloud server, $N_f\ /N_c = 4$. Their processing power is only 10% or their Cloud peers, $P_f\ /P_c = 10\%$. The average delay ratio, $\delta_f\ /\delta_c$, is changed to 1%, 10%, 20%, 50% and 85%. Figure 14 presents the latency results, and as observed, increasing the average delay ratio from 1% to 85% increases the latency of the Fog until it reaches a point where it crosses the Cloud latency. This is interpreted as the Fog servers are far from the edge devices and closer to the Cloud servers. In this case, the Fog servers provide as poor latency as the one provided by Cloud.

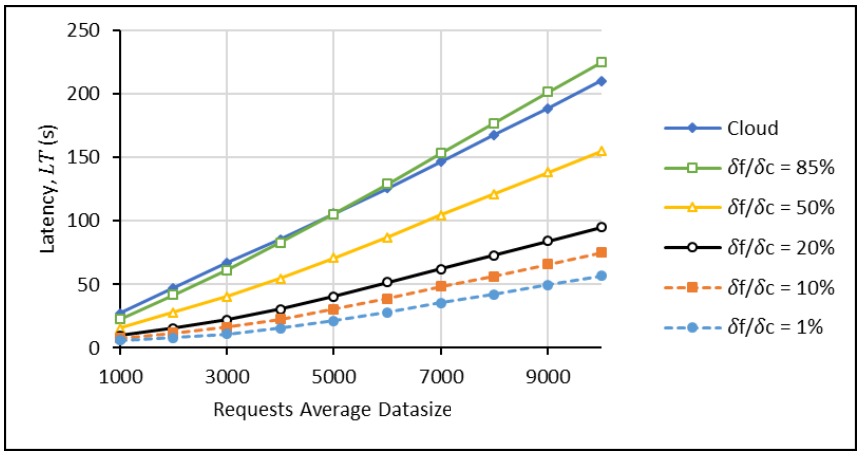

Figure 14. Latency of fog compared to cloud by varying average delay.

Impact of the Ratio of Processing Speed:

In this experiment, the number of Fog servers is set to four times the number of Cloud servers, $N_f\ /N_c = 4$. Their average delay, $\delta_f\ /\delta_c$ is 10%. The processing power, $P_f\ /P_c$, is changed to 3%, 5%, 7%, 10% and 20%. Figure 15 shows the latency results. As observed by the results presented in Figure 15, decreasing the processing speed ratio from 20% to 3% increases the latency of Fog until it reaches a point where it crosses the Cloud latency. This is interpreted as fog servers have slow processing capabilities. In this case, even if these resources are closer to edge than the Cloud resources, the Fog provide high latency due to the slow processing capabilities.

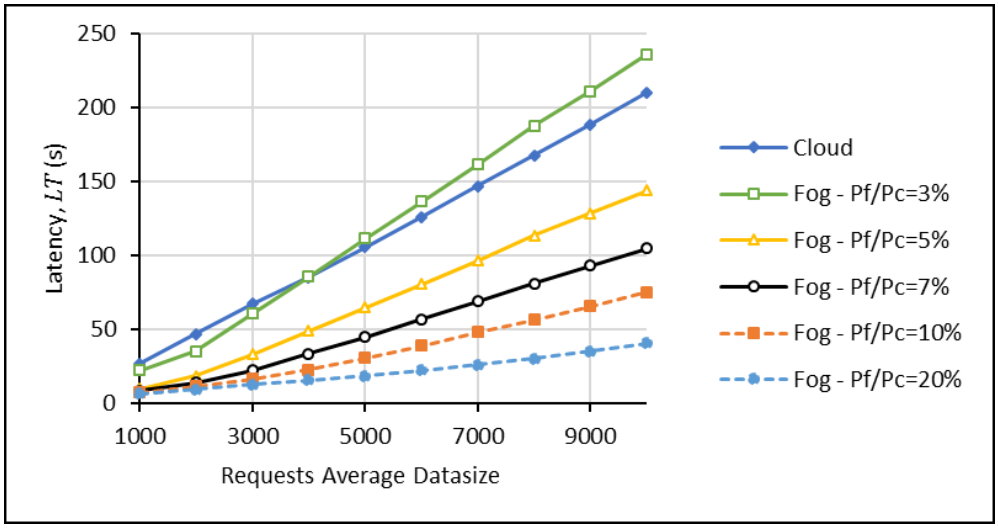

Figure 15. Latency of fog compared to cloud by varying processing speed in the fog layer.

Impact of the Ratio of Resource Numbers:

In the experiment presented in Figure 16, the Fog servers' average delay is set to 10% of the Cloud average delay. Also, the processing power capabilities of the Fog servers are set to 10% of the Cloud servers' capabilities. The number of Fog resources relative to Cloud is varied to 100%, 150%, 200%, 300%, 400%, 600%, and 800%. Figure 16 shows the latency results for this experiment. As observed from the experiment presented in Figure 16, decreasing the number of resources ratio from 800% to 100% increases the latency of Fog until it reaches a point where it crosses the Cloud latency. This can be interpreted as having Fog servers with very few resources. Even if these resources are closer to edge and have a good processing capability, having fewer resources affects the latency in a negative way.

Another experiment is conducted with the objective is to find the breaking points in terms of these three parameters, average delay, processing power and number of resources. The average data size is set to 5000 packets. Figure 17 shows the results of the experiment as crossing points between fog latency and cloud latency. For instance, for a specific average delay ratio and processing power ratio between Fog and Cloud, the graph tells the number of servers' ratio in which Fog outperforms Cloud or vice versa.

In the last three experiments, it was observed that the latency of Fog computing at some point crosses the Cloud computing latency. This happens in three cases where Fog computing has high average delay as presented in Figure 14, has resources with low processing power as presented in Figure 15, and has few number of servers presented in Figure 16.

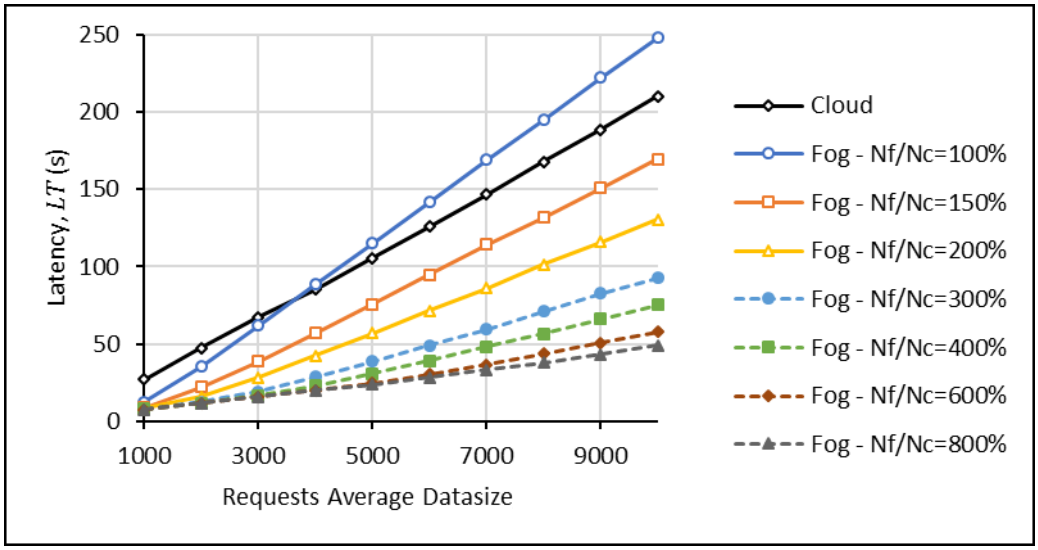

Figure 16. Latency of Fog compared to Cloud by varying the number of servers in the fog layer.

Another experiment is conducted with the objective is to find the breaking points for the three ratios with the average size of 5000 packets. Figure 17 shows the results of the experiment. As presented in Figure 17, the experiment shows where a set of Fog servers would provide better latency than a set of Cloud servers. For instance, for a specific average delay ratio and processing power ratio between Fog and Cloud, the graph shows the number of servers' ratio in which Fog outperforms Cloud or vice versa.

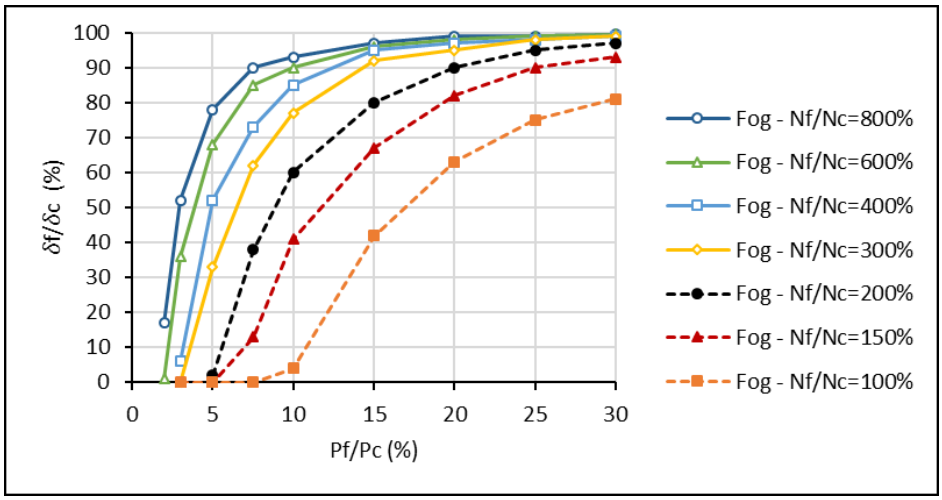

Figure 17. Break points of fog and cloud computing latency.

7. Conclusion and Future Research

This work modeled the scheduling problem for IoT requests to minimize latency in hybrid Fog-Cloud computing using integer linear program. The latency is addressed in this work as the Round-Trip Time (RTT) for processing an IoT request from the moment it is initiated to the moment it is completely processed and the results are returned back to the requester.

This latency includes many delay components such as transmission delay, routing or queuing delay, propagation delay, processing time, waiting time in case the resources are busy.

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 were solved using Lingo were small-sized problems since the problem by nature is an NP-hard problem. Hence, exact methods are not adequate to solve large size problems. In this work, we developed a customized implementation of the Genetic Algorithms (GA) 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 with different parameters such as population size and maximum number of iterations. After developing the GA, a comprehensive comparison is performed between the exact solutions obtained from Lingo and the heuristic solutions obtained from the GA.

The optimized service latency obtained from the GA is also compared to other non-optimized scheduling techniques (such as WFQ, PRI, and RR). The results of the proposed approach showed significant improvement in the overall latency from 21.9% and up to 46.6% better solutions than the other algorithms. The proposed approach also succeeded in meeting the requests deadlines by up to 31% with better performance than the other algorithms.

The last set of experiments in section 6.3 showed 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 better service latency than using only Cloud computing. The experiments showed typical parameters of where Fog outperforms Cloud in terms of resources average delay, processing power and number of resources.

Future work will extend the model to include critical request scheduling and to allow pre-emption. Other heuristic-based techniques will also be evaluated. Moreover, multiple objective functions can be included to maximize resource utilization and minimize latency.

Acknowledgements

The authors would like to acknowledge the Computer Science and Engineering department at the American University of Sharjah, UAE, for providing the support and resources. This research was partially funded by the American University of Sharjah Faculty Research Grant number FRG17-R-20.

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 Computing, vol. 99, Oct. 2015, pp. 1-1.

[4] P. V. Patil, “Fog Computing,” in IJCA Proc. on Nat. Conf. on Recent Trends in Mobile and Cloud Computing, 2015, pp. 1-6.

[5] 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 Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on, 2013, pp. 320-323.

[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. Symposium on Consumer Electronics, 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 Application, 2015, pp. 685-695.

[10] I. Stojmenovic, “Fog computing: A cloud to the ground support for smart things and machine-to-machine networks,” in Telecommunication Networks and Application 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 Information Reuse and Integrate, 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 Computer Aided Modeling and Design of Communication Links and Networks, 2014, pp. 325-329.

[14] L. Vaquero and L. Rodero-Merino, “Finding your way in the fog: Towards a comprehensive definition of fog computing,” in ACM SIGCOMM Computer Communication Review, 2014, pp. 27-32.

[15] 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.

[16] 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.

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

[18] J. Clausen, “Branch and bound algorithms-principles and examples,” in Department of Computer Science in University of Copenhagen, 1999, pp. 1-30.

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

[20] 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.

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

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

[23] Pencheva T , Atanassov K , Shannon A . Modelling of a roulette wheel selection operator in genetic algorithms using generalized nets. Int J Bioautom 2009, 13:257–64.

[24] 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.

[25] 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.

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

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

[28] 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 Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing, 2013, pp. 15-20.

[29] B. Farahani, F. Firouzi, V. Chang, M. Badaroglu, N. Constant, and K. Mankodiya, “Towards fog-driven IoT eHealth: Promises and challenges of IoT in medicine and healthcare,” Futur. Gener. Comput. Syst., vol. 78, pp. 659–676, Jan. 2018.

[30] 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. Symposium on Service Oriented System Engineering, 2013, pp. 320-323.

[31] 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 Distributed Event-Based Syst., 2013, pp. 183-194.

[32] 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 Computing & Networking, 2013, pp. 19-26.

[33] D. Wu, S. Liu., L. Zhang, J. Terpenny, R. Gao, T. Kurfess, and J. Guzzo, “A Fog Computing-Based Framework for Process Monitoring and Prognosis”, in Cyber-Manufacturing, Transactions of the SME, Journal of Manufacturing Systems, 43(1), 2017, pp. 25-34.

[34] R. Ghosh and Y. Simmhan, “Distributed scheduling of event analytics across edge and cloud.” CoRR, (1608.01537), 2016.

[35] G. Sun, Y. Li, Y. Li, D. Liao, and V. Chang, “Low-latency orchestration for workflow-oriented service function chain in edge computing,” Future. Generation. Computer System, vol. 85, pp. 116–128, Aug. 2018.

[36] J. Sun, S. Sun, K. Li, D. Liao, A. K. Sangaiah, and V. Chang, “Efficient algorithm for traffic engineering in Cloud-of-Things and edge computing,” *Computer and Electrical Engineering*, vol. 69, pp. 610–627, Jul. 2018.

[37] A. S. Sohal, R. Sandhu, S. K. Sood, and V. Chang, “A cybersecurity framework to identify malicious edge device in fog computing and cloud-of-things environments,” Computer and Security, vol. 74, pp. 340–354, May 2018.

[38] 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.

[39] 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.

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

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

[42] M. Mao and M. Humphrey, “Auto-scaling to minimize cost and meet application deadlines in cloud workflows,” in Int. Conference of High Performance Computing, Networking, Storage and Analysis, 2011, pp. 1-12.

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

[44] P. Hung and E. Huh, “An adaptive procedure for task scheduling optimization in mobile cloud computing.” Mathematical Problems in Engineering. 2015.

[45] C. Chen, H. Xiang, T. Qiu, C. Wang, Y. Zhou, V. Chang. A rear-end collision prediction scheme based on deep learning in the Internet of Vehicles. J. Parallel Distrib. Comput. 2017, 117, 192–204.

[46] M. Abdel-Baset, V. Chang, A. Gamal, and F. Smarandache. An integrated neutrosophic ANP and VIKOR method for achieving sustainable supplier selection: A case study in importing field. Computers in Industry, 106, 94-110, 2019.

[47] R.E. Burkard, E. C¸ ela, and L. Pitsoulis, The quadratic assignment problem, Handbook of Combinatorial Optimization (Dordrecht), Computer-aided chemical engineering, Kluwer Academic Publishers, 1998, pp. 241–339.

[48] J. F. Kurose, and W. R. Keith. Computer networking: a top-down approach. ed. 4. Addison Wesley, 2009.