People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
Mohamed Khider University of Biskra
Faculty of Exact Sciences, Natural Sciences and Life
Computer Science Department

Order number: SIOD1/M2/2018

Master Memory

Presented to obtain the diploma of academic master in

Computer Science

Path: **Information system, optimization and decision**

Fog computing task scheduling Based on simulated annealing algorithm

By:

LABED AYOUB

Examined on 26/06/2018, by the board of examiners, which is composed:

بِسْمِ اللّٰهِ الرَّحْمٰنِ الرَّحِيْمِ

Deduction

Before all, my sincere praise to Allah the almighty for giving me sufficient capacity and patience to accomplish this dissertation.

I dedicate this work to my dear parents who support me a lot in both sides physically and morally.

To my dear brothers and sisters: Hocine, Med Souhile, Siham, Roufida and wafa.

To all my friends: Abdel Jalil, Taher, Mohyeddine, Rezgallah, Fathi, Saleh, Nadir, Fares, Okba, Amine, Rabie and Aymen.

And to all I know without exceptions.

To all my teachers without exceptions

To all my extended family LABED

Finally, I offer my regards and blessings to all those who supported me to accomplish this work.

Acknowledgements

The first and the last thing is for Allah who providing me the sufficient capacity to finish this work.

I would like to thank my supervisor, Dr. Salim BITAM, for the patient guidance, encouragement and advice he has provided throughout my time as his student. I have been extremely lucky to have a supervisor who cared so much about my work, and who responded to my questions and queries so promptly.

I would not forget to thank the members of the jury, who have kindly accepted to read and examine our work.

I am indefinitely thankful for those teachers who were sincerely caring, giving, and understanding throughout my completely educational life.

Special thanks to all those who supported me to complete this work.

The main proposal of this study and the results were submitted to the 24th International European Conference on Parallel and Distributed Computing (Euro-Par 2018), Workshop on Fog-to-Cloud Distributed Processing (F2C-DP), 27-31 august 2018, Turin, Italy, (http://www.mf2c-project.eu/europar-2018-mf2c-workshop/).

الملخص

تهدف إنترنت الأشياء إلى جعل كل الأجهزة متصلة بالإنترنت (مثل أجهزة الاستشعار والكاميرات والمركبات) ، وبالتالي توليد كميات هائلة من البيانات والتي يمكن أن تشكل عبئا على أنظمة التخزين وتطبيقات تحليلات البيانات. تقدم الحوسبة السحابية خدمات على مستوى البنية التحتية قادرة على تلبية متطلبات التخزين والمعالجة لإنترنت الأشياء. ومع ذلك ، هناك تطبيقات مثل مراقبة الصحة والاستجابة لحالات الطوارئ التي تتطلب سرعة في الاستجابة ، والتأخير الناجم عن نقل البيانات إلى السحابة ومن ثم العودة إلى التطبيق يمكن أن يؤثر بشكل خطير على أدائها. للتغلب على هذا المشكل ، تم اقتراح نموذج الحوسبة الضبابي كمكمل قوي للسحابة ، حيث يتم توسيع الخدمات السحابية إلى حافة الشبكة لتقليل وقت استجابة الإرسال. لتحقيق الإمكانات الكاملة لنماذج الضباب وإنترنت الأشياء من أجل التحليلات في الوقت الحقيقي ، هناك العديد من التحديات التي يجب معالجتها. تتمثل المشكلة الأولى والأكثر أهمية في تصميم تقنيات إدارة الموارد التي تعمل على تحسين جودة الخدمة. يقترح هذا العمل خوارزمية جدولة المهام في طبقة الضباب تسمى خوارزمية التخمير المحاكى. وهي طريقة مستوحات من تقنية في علم المعادن ، وهذه التقنية تنطوي على التسخين والتبريد المضبوط للمادة لزيادة حجم بلوراتها وتقليل عيوبها. وتعتبر مشكلة إدارة الموارد من نوع NP-Complete وغرضها توزيع أعباء العمل بين موارد المعالجة بطريقة مثلى لتحقيق المفاضلة بين التكلفة الإجمالية ووقت تنفيذ المهام. من أجل تجسيد و تقييم فعالية الخوارزمية، تم إجراء مجموعة من الاختبارات التجريبية. بالمقارنة مع الخوارزمية الجينية و طريقة (FIFO) على محاكي ”iFogSim“ ، أظهرت النتائج التي تم التوصل إليها أن خوارزميتنا المقترحة حققت نتائج أفضل من الخوارزميات الأخرى من حيث التكلفة الإجمالية ووقت تنفيذ المهام .

كلمات البحث: الحوسبة الضبابية ، جدولة المهام ، خوارزمية التخمير المحاكي.

Résumé

L'Internet des objets « IoT » vise à mettre en ligne chaque objet (par exemple, des capteurs, des caméras et des véhicules), générant ainsi de grandes quantités de données susceptibles de surpasser les systèmes de stockage et les applications d'analyse de données. Le « Cloud Computing » offre des services au niveau de l'infrastructure qui peuvent évoluer vers les exigences de stockage et de traitement de l'Internet des objets. Cependant, il existe des applications telles que la surveillance de la santé et les interventions d'urgence qui nécessitent une faible latence, et les retards causés par le transfert des données vers le Cloud puis leur retour à l'application peuvent affecter sérieusement leurs performances. Pour surmonter cette limitation, le paradigme informatique « Fog » a été proposé comme un complément puissant au « Cloud », où les services « Cloud » sont étendus à la périphérie du réseau pour réduire la latence de transmission. Pour réaliser tout le potentiel des paradigmes « Fog » et « IoT » en matière d'analyse en temps réel, plusieurs défis doivent être relevés. Le premier défi et le plus important est de concevoir des techniques de gestion des ressources qui améliorent la qualité de service (QoS). Ce travail propose un algorithme d'ordonnancement des tâches pour une infrastructure du « Fog Computing » appelé « Fog Simulated Annealing algorithm » (FSA). FSA est basé sur le processus recuit-simulé en métallurgie, une technique impliquant l'échauffement et le refroidissement contrôlé d'un matériau pour augmenter la taille de ses cristaux et réduire leurs défauts. Pour l'ordonnancement de tâche, nous nous attaquons à l'objectif de parvenir à l'équilibre entre le temps d'exécution des tâches « makespan » et le coût monétaire requis en exploitant des ressources du Fog. Il est considéré comme un problème « NP-Complet » et il vise à répartir les charges des tâches entre les ressources de traitement de manière optimale pour obtenir un compromis entre le coût total et le temps d'exécution des tâches. Afin d'évaluer l'efficacité et la performance de cette proposition, un ensemble de tests expérimentaux a été réalisé. Après comparaisons avec l'algorithme génétique (GA) et l'algorithme FIFO sur le simulateur « iFogSim », les résultats obtenus ont montré que notre algorithme proposé atteint un meilleur résultat par rapport d'autres algorithmes avec moins de complexité.

Mots-clés: Fog Computing, gestion des tâches, Algorithme de recuit simulé.

Abstract

Abstract

Internet of Things (IoT) aims to bring online every object (e.g. sensors, cameras, and vehicles), therefore generating large amounts of data that can overpower storage systems and data analytics applications. Cloud computing offers services at the infrastructure level that can scale to IoT storage and processing requirements. However, there are applications such as health monitoring and emergency response that require low latency, and delay caused by transferring data to the cloud and then back to the application can seriously affect their performances. To overcome this limitation, Fog computing paradigm proposed as a powerful complement to the cloud, where cloud services are extended to the edge of the network to reduce transmission latency. To realize the full potential of Fog and IoT paradigms for real-time analytics, several challenges need to be addressed. The first and most critical problem is designing resource management techniques that improves the quality of service (QoS). This work proposes a task scheduling algorithm in the fog layer called Fog Simulated Annealing algorithm (FSA). FSA is based on the annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. We tackle the objective of achieving the balance between the makespan and the monetary cost of fog resources. It is considered as NP-Complete problem and it aims at spreading the workloads among the processing resources in an optimal way to achieve tradeoff between the total cost and execution time of tasks. In order to evaluate the effectiveness and the performance of this proposal, a set of experimental tests has been conducted. After comparison with genetic algorithm (GA) and FIFO on iFogSim Simulator, the reached results showed that our proposed algorithm achieves better tradeoff value than other algorithms with less complexity.

Keywords: Fog Computing, Task Scheduling, Simulated Annealing Algorithm

List of tables

List of graphs

Table of Contents

Table of contents

General introduction

Recently, academia and industry have been developing efficient architectures in order to deal with the demands of the ever-increasing number of IoT devices. Fog computing is one of promising solutions (Nisha, 2015). Fog computing is considered as an extension of the cloud computing paradigm from the core of the network to the edge of the network. It is a highly virtualized platform that provides computation, storage, and networking services between end devices and traditional cloud servers (Stojmenovic, et al., 2014), fog computing aims to process part of the service's workload locally on fog nodes, which are served as a near-end computing proxies between the front-end IoT devices and the back-end cloud servers (Bonomi, et al., 2012). Putting resources at the edge of the network only one or two hops from the data sources allows fog nodes to perform low latency processing while latency-tolerant and large-scale tasks can still be efficiently processed by the cloud. In addition, the cost and scale benefits of the cloud can help the fog to serve peak demands of IoT devices if the resources of fog nodes are not sufficient. Also, many applications require the interplay and cooperation between the edge (fog) and the core (cloud), particularly for the IoT and big data analysis (Pham, et al., 2016).

In this study, we consider task scheduling issue in a fog computing system, where a fog provider can exploit the collaboration between its fog nodes for efficiently executing users' large-scale offloading applications. The fog nodes are local resources, which can be any devices with computing, storage, and network connectivity such as set-top-boxes, access points, routers, switches, base stations, and end devices, etc. (Yi, et al., 2015). However, All distributed processing nodes, are managed by a resource broker (fog broker), which is a resource management component, and scheduler for the workflows submitted from users at the fog's side. In this case, a task schedule, which does not achieve good tradeoff between the completion time of the workflow and the monetary cost, is not an optimal solution.

Therefore, in this work we propose a new task scheduling algorithm called Fog Simulated Annealing (FSA) algorithm to achieve a good tradeoff between the workflow execution time and the cost for the use of fog resources. Our proposal is based on the annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. In order to evaluate the performance of our proposal, simulator named iFogSim choose to perform a set of simulations, and to compare it against other methods in terms of the workflow execution time and the cost for the use of fog resources.

The organization of this manuscript is as follows: the first chapter presents a general study on the concept of Fog Computing including definition, architecture and the different characteristics of the fog. In the second chapter, we consider the detailed definition of the problem of task scheduling, followed by state of the art in this problem. The third chapter explains the fog simulated annealing algorithm and its different stages such as the representation of an individual (a solution), the objective function, generating function, and accept function. In the fourth chapter, a simulation of the fog simulated annealing on a fog simulator called iFogSim is presented. It contains the description of the studied scenario, the results obtained, as well as a discussion based on a comparison with the conventional scheduling algorithm (First In First Out) and (Genetic Algorithm). Finally, we conclude our research activity with some perspectives.

Chapter 1

Fog computing:

Basic concepts

Chapter 1. Fog Computing

1.1. Introduction

In the past few years, the Cloud computing has provided many opportunities for enterprises by offering their customers a range of computing services. Current “pay-as you-go” Cloud computing model becomes an efficient alternative to owning and managing private data centers for customers facing Web applications and batch processing (Armbrust, et al., 2010). Cloud computing frees the enterprises and their end users from the specification of many details, such as storage resources, computation limitation and network communication cost. However, this bliss becomes a problem for latency-sensitive applications, which require nodes in the vicinity to meet their delay requirements (Bonomi, et al., 2012). When techniques and devices of IoT are getting more involved in people’s life, current Cloud computing paradigm can hardly satisfy their requirements of mobility support, location awareness and low latency. Fog computing is proposed to address the above problem. As Fog computing is implemented at the edge of the network, it provides low latency, location awareness, and improves quality-of-services (QoS) for streaming and real time applications. Typical examples include industrial automation, transportation, and networks of sensors and actuators. Moreover, this new infrastructure supports heterogeneity as Fog devices include end-user devices, access points, edge routers and switches. The Fog paradigm is well positioned for real time big data analytics, supports densely distributed data collection points, and provides advantages in entertainment, advertising, personal computing and other applications (Stojmenovic, et al., 2014).

1.1. Definition of fog computing

In the perspective of Cisco, fog computing is considered as an extension of the cloud computing paradigm from the core of network to the edge of the network. It is highly virtualized platform that provides computation, storage, and networking services between end devices and traditional cloud servers (Bonomi, et al., 2012).

While in flavor work, fog computing is defined as 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 or new services and applications that run in sandboxed environments. Users leasing part of their devices to host these services get incentives for doing so (Vaquero, et al., 2014).

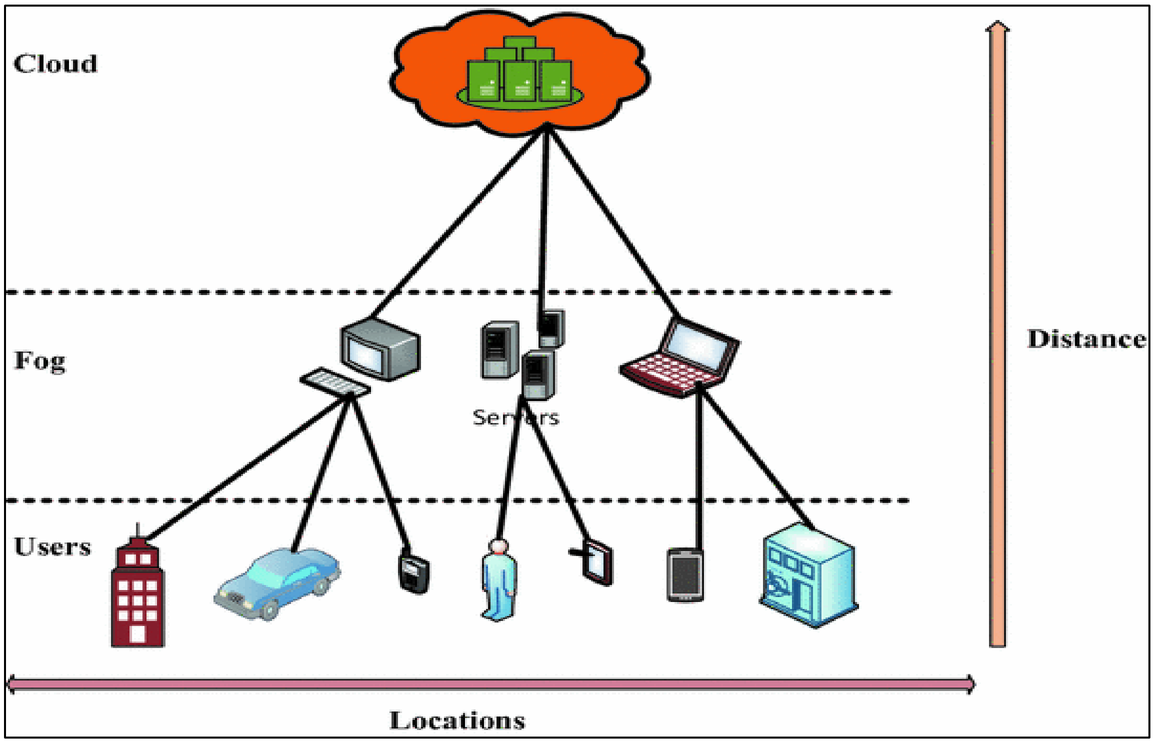


Figure 1: Fog computing architecture.

1.2. Fog computing architecture

The fog computing system architecture has three layers in a hierarchy network as represented in figure 1.

The bottommost layer consists of end user devices, which can be smartphones, tablets, wearable devices, thin-client, smart home appliances, wireless sensor nodes, and so on. They send requests to the upper layers for application execution.

The middle layer represents fog computing environment. The primary components of this layer are intelligent fog devices (e.g. routers, gateways, switches, access points) that have the capability of computing, networking, and storage. They are called fog nodes, which are deployed in the vicinity of end users to receive and process part of a workload of users' requests with the local short-distance high-rate connection. Also, they are connected to the cloud so as to benefit from a massive pool of redundant resources of the cloud on demand.

The uppermost layer represents cloud computing environment, which hosts a number of heterogeneous cloud nodes or VMs of different cloud service providers. The cloud nodes provide outsourced resources to execute the workload dispatched from the fog layer. (Pham, et al., 2017)

1.3. Essential characteristics

The different characteristics of fog computing are:

- Edge location, location awareness, and low latency: Fog computing support endpoints with finest services at the edge of the network.
- Geographical distribution **:** The services and application objective of the fog is widely distributed for example fog will play an important role in delivering high quality streaming to connected vehicles through proxies and access points positioned nearby.
- Support for mobility: Using LISP protocol fog devices provide mobility techniques like decouple host identity to location identity.
- Real time interactions: fog computing requires real time interactions for speedy service.
- Heterogeneity: Fog nodes can be deployed in a wide variety of environments.
- Interoperability: Fog components must be able to interoperate in order to give wide range of services like streaming.
- Support for on-line diagnostic and interplay with the Cloud: The fog sited to play a virtual role in the intake and processing of the data close to the source. (Nisha, 2015)

1.4. Application areas

We elaborate on the role of Fog computing in the following three motivating scenarios. The advantages of Fog computing satisfy the requirements of applications in these scenarios.

Smart Grid: Energy load balancing applications may run on network edge devices, such as smart meters and micro-grids. Based on energy demand, availability and the lowest price, these devices automatically switch to alternative energies like solar and wind. As shown in figure 2, Fog collectors at the edge process the data generated by grid sensors and devices, and issue control commands to the actuators. They also filter the data to be consumed locally, and send the rest to the higher tiers for visualization, real-time reports and transactional analytics. Fog supports ephemeral storage at the lowest tier to semi-permanent storage at the highest tier. Global coverage is provided by the Cloud with business intelligence analytics.

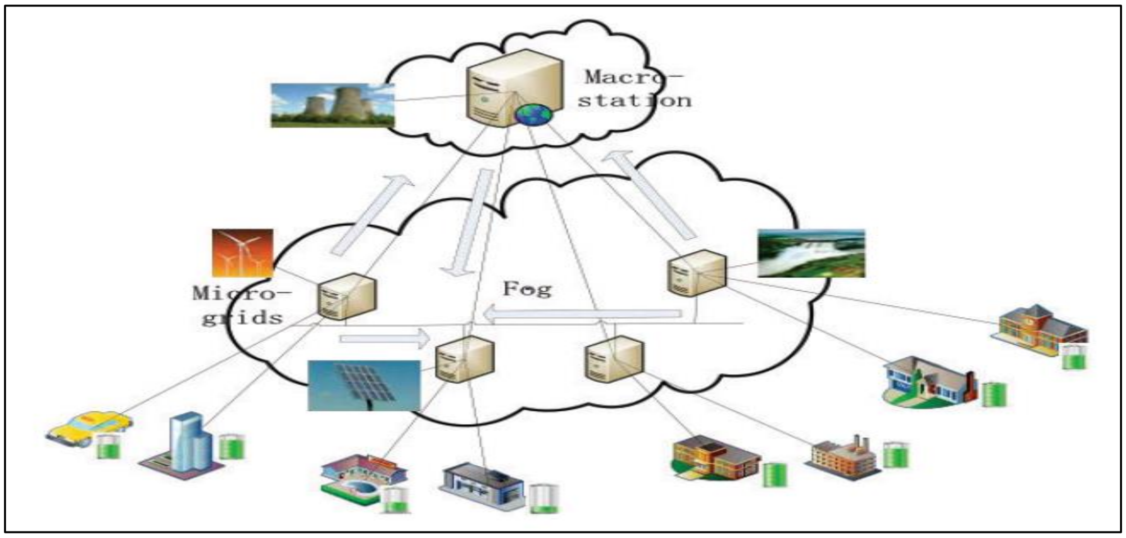


Figure 2: Fog computing in smart grid.

Smart Traffic Lights and Connected Vehicles: Video camera that senses an ambulance flashing lights can automatically change the streetlights to open lanes for the vehicle to pass through traffic. Smart streetlights interact locally with sensors and detect presence of pedestrian and bikers, and measure the distance and speed of approaching vehicles. As shown in figure 3, intelligent lighting turns on once a sensor identifies movement and switches off as traffic passes. Neighboring smart lights serving as Fog devices coordinate to create green traffic(efficient networking) wave and send warning signals to approaching vehicles. Wireless access points like Wi-Fi, 3G, roadside units and smart traffic lights are deployed along the roads. Vehicles-to-Vehicle, vehicle to access points, and access points to access points interactions enrich the application of this scenario.

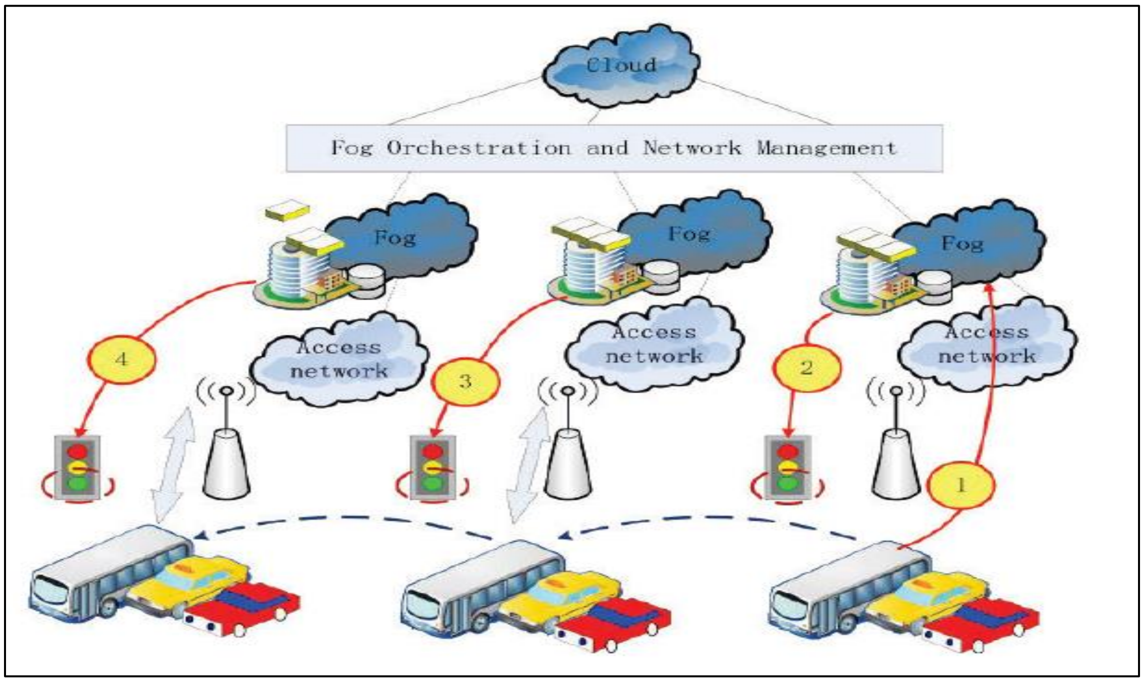


Figure 3: Fog computing in smart traffic lights and connected vehicles.

Wireless Sensor and Actuator Networks: Traditional wireless sensor networks fall short in applications that go beyond sensing and tracking, but require actuators to exert physical actions like opening, closing or even carrying sensors. In this scenario, actuators serving as Fog devices can control the measurement process itself, the stability and the oscillatory behaviors by creating a closed-loop system. For example, in the scenario of self-maintaining trains, sensor monitoring on a train’s ball bearing can detect heat levels, allowing applications to send an automatic alert to the train operator to stop the train at next station for emergency maintenance and avoid potential derailment. In lifesaving air vents scenario, sensors on vents monitor air conditions flowing in and out of mines and automatically change airflow if conditions become dangerous to miners. (Stojmenovic, et al., 2014)

1.5. Challenges of Fog computing and research directions

Many architectures and algorithms have been developed for the fog computing, but none of them meets all the identified criteria such as heterogeneity, scalability, mobility.... This section discusses the fog computing challenges.

1.5.1. Architectural Challenges and Research Directions

- Heterogeneity: the design of exhaustive and flexible semantic ontologies
- QoS Management: the design of appropriate SLA management techniques.
- Scalability: the design of mechanisms that in addition to scale the resources from cloud can support scaling the resources of involved domain in fog stratum and devices in IoT/end-user stratum.
- Mobility: the design of mechanisms that consider the mobility of IoT and fog nodes in addition to the cloud nodes.
- Federation: the design of appropriate composition mechanisms for the application components.
- Interoperability: the design of signaling control, and data interface between the serval domain parts of the fog system.

1.5.2. Algorithmic Challenges and Research Directions

- Heterogeneity: acquiring a clear vision on the degree of heterogeneity in term of computing and storage capabilities.
- QoS Management: considering various QoS metrics.
- Scalability: validating algorithms over large scale in real world environment.
- Mobility: ensuring the continuity of offered services despite the movement of IoT/end-user devices and/or fog nodes.
- Federation: designing algorithms for federation in fog systems. (Mouradian, et al., 2017)

1.6. Task scheduling problem in the fog computing

1.6.1. Definition

Task scheduling on a target system that has network topology is defined as the problem of allocating the tasks of an application to a set of processors with various processing capabilities so, as to minimize the makespan of the schedule. Thus, the input of task scheduling involves a task graph and a processor graph, and the output is a schedule representing the assignment of a processor to each task as shown in Figure 4. (Pham, et al., 2017)

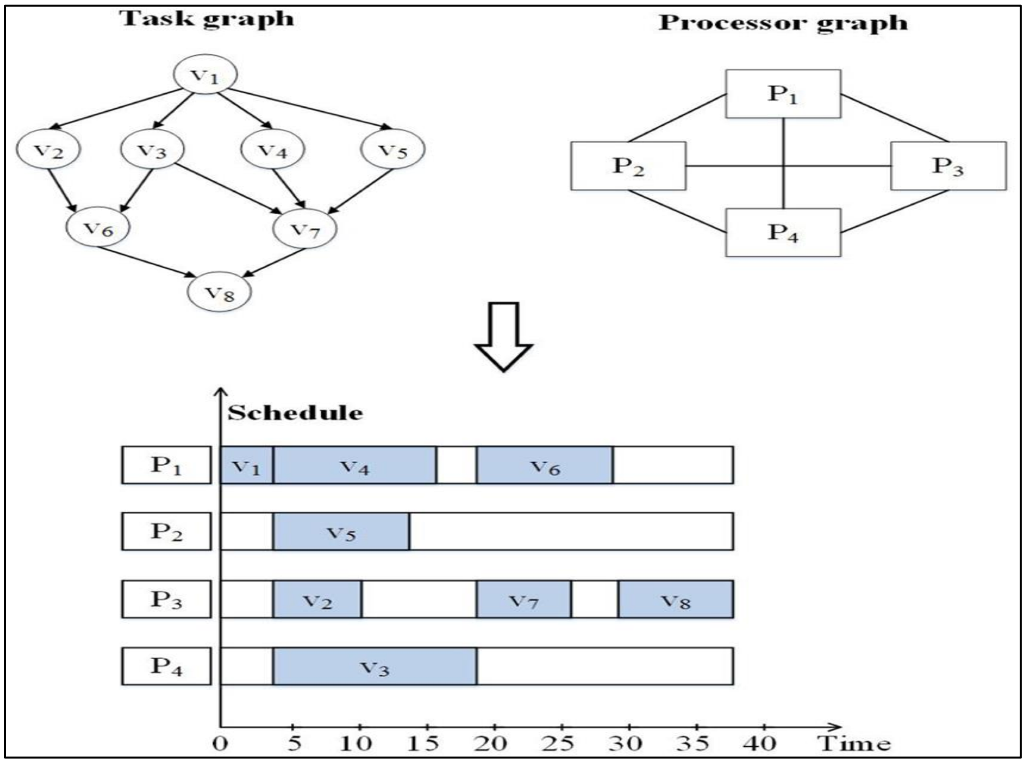


Figure 4: Illustration of task scheduling problem.

1.6.2. Motivation

The task scheduling problem for the fog computing environment deals with the problem of the increasing demand for computational resources requested by mobile users to perform a large number of tasks efficiently. The task scheduling problem determines an optimal assignment of various tasks submitted to be executed on the lowest number of fog computing resources (e.g. less memory) in the shortest CPU execution time. As a result, the mobile user achieves faster execution time of his/her tasks at the lowest cost. (Bitam, et al., 2017)

1.7. Conclusion

In this chapter, we have seen the different definitions of fog computing, its essential characteristics and some application areas. Next, we cited some challenges in this area as well as an introduction to the issue of task scheduling in the fog computing environment.

In the next chapter, we will see the optimization domain in general; later we will develop and deal with the problem of tasks scheduling as an optimization problem. In addition, we will see the different works proposed in the literature to solve this problem in the fog as a state of the art.

Chapter 2

Task scheduling

In fog computing

Chapter 2. Task scheduling in Fog computing: A state of the art

2.1. Introduction

The problem of task scheduling in fog computing environments considered as a very important research issue because of its impact on quality of service (QoS) such as end-user waiting time, CPU execution time and dedicated memory (Hao, et al., 2017).

In this chapter, we present the main research work and the methods of resolutions used to solve this problem. Task scheduling can be qualified as an NP-hard optimization problem (Bitam, et al., 2017). For this reason, all existed works used advanced optimization methods. By exposing the state of the art of this issue, we will detail principle of each proposed method.

2.2. Optimization

2.2.1. Definitions

- ➢ Optimization can be defined as finding solution of a problem where it is necessary to maximize or minimize a single or set of objective functions within a domain, which contains the acceptable values of variables while some restrictions (constraints) are to be satisfied.
- ➢ The solutions: there are two types of solutions
    a. Acceptable solutions: are solutions that maximize or minimize the objective function(s) while satisfying the described restrictions.
    b. An optimum solution: is the best solution.
- ➢ The objective function: expresses the main aim of the model that is either to be minimized or maximized.
- ➢ A set of variables: control the value of the objective function and these variables are essential for the optimization problems. We cannot define the objective function and the constraints without the variables.
- ➢ A set of constraints: are those, which allow the variables to take on certain values but they exclude others. The constraints depends on the requirements of the optimization problem (Rao, 2015).

2.2.2. Optimization problem types

In literature, there are two types of optimization problem, the former is called single objective optimization problem aiming at improving one objective, where the latter is known as multiple or multi-objective optimization problem in which more than one objective is tackled (Rao, 2015).

2.2.2.1. Optimization methods for single objective optimization problems

We can solve the single objective optimization problems by using traditional methods (such as simplex method, dynamic programming, separable programming, etc.) and advanced optimization methods (such as genetic algorithm (GA), simulated annealing (SA), particle swarm optimization (PSO), ant colony optimization (ACO), artificial bee colony algorithm (ABC), etc.

Then, we cannot know which method will give the best solution only after applying these methods to the same problem and whichever method gives the best solution is called the best optimization method for the given problem.

2.2.2.2. Optimization methods for multi-objective optimization problems

In the real world, there are many problems in which it is desirable to optimize two or more objectives at the same time, these problems are known as multi-objective optimization problems. To cope with this kind of issues, single or multiple objective problems), continuous research is being conducted in this field, and nature inspired heuristic optimization methods are proving to be better than the classical deterministic methods, and thus are widely used, we mention for example genetic algorithm (GA), ant colony optimization (ACO) algorithm, particle swarm optimization (PSO) algorithm; differential Evolution (DE) algorithm, artificial bee colony (ABC) algorithm, shuffled frog leaping (SFL) algorithm, harmony search (HS) algorithm, etc.

Generally, the multi-objective optimization problems have decision variable values which are determined in a continuous or integer domain with either an infinite or a large number of solutions, the best of which should satisfy the designer or decision-maker’s constraints and preference priorities (Rao, 2015).

2.2.3. Optimization approaches

Optimization approaches can be classified into two main categories (mansouri, 2013).

2.2.3.1. Exact approaches

An exact method can be defined as a method that provides an optimal solution for an optimization problem. The use of this type of method is particularly interesting in the case of small problems.

- Features
  - The solution found is accurate
  - This method is theoretical and less practical method
  - The processing complexity is exponential
- Examples
  - Branch and bound
  - Dynamic programming
  - A*
  - Etc.

2.2.3.2. Approximation strategies (Metaheuristic)

The word metaheuristic is derived from the composition of two Greek words:

- Heuristic which comes from the verb heuriskein and which means to find.

- Meta which is a suffix meaning beyond.

A metaheuristic method is a resolution algorithm that does not necessarily provide an optimal (global) solution for a given optimization problem. It offers a solution close to the so-called optimal solution (found).

- Features
- The solution found is not exact
- This realistic method
- The complexity of treatment is acceptable
- Representation of the research space as follows:

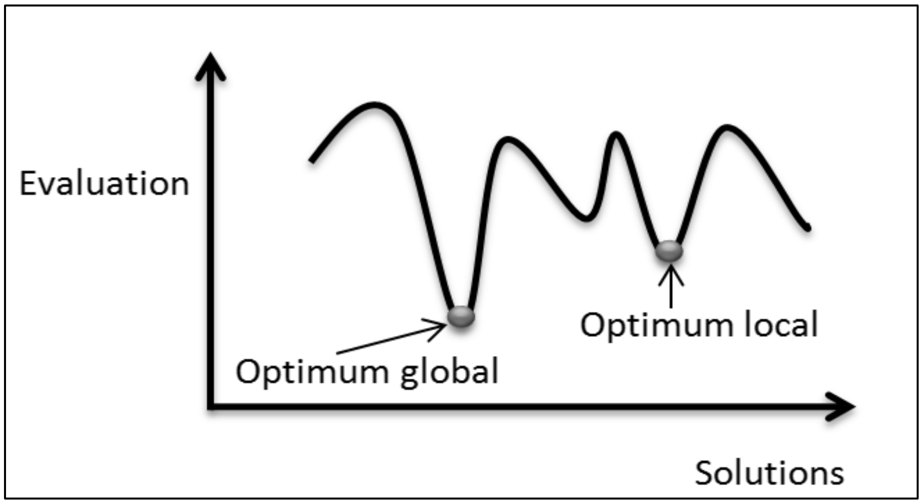


Figure 5: Representation of the research space

S-metaheuristic

- Principle: This is a method that searches around a chosen solution. It is easy to apply in its different stages: representation, evaluation function "fitness", and neighborhood.
- Limitations of S-metaheuristics:
  - Convergence towards local optima
  - The local optimum found depends on the initial solution
  - No general approach to limit the relative error (find the global optimum)

P-metaheuristic

- Generality:

  A P-metaheuristic construct a method inspired by a natural and biological phenomenon exercises a resolution based on a population where a population is a set of individuals (i.e. a set of solutions).
- Examples:
  - Genetic algorithms (GA): based on evolutionary operators such as selection, crossing and mutation.
  - Ant colony-based optimization (ACO): inspired by the search for food in ants by the use of a substance known as pheromone.

- Optimization based on bee colonies: inspired by the natural behavior of bees such as the search for food, communication, and reproduction.

2.3. Fog computing task scheduling problem

Fog computing has several advantages, including decrease latency time, decrease network traffic and energy efficiency, but due to the novelty, this concept also has challenges (KRAEMER, et al., 2017).One of these challenges is associated with the allocation of resources and scheduling (Puliafito, et al., 2013). As the IoT applications is growing rapidly and clients are demanding more services and better results, task scheduling for the Fog has become a very interesting and important research area (Al Nuaimi, et al., 2012).

2.3.1. Definitions

2.3.1.1. The tasks

A task is a unit of execution or a unit of work, with execution and competition time. It consists of a set of processes, which require for their implementation, certain resources, and that it is necessary for the program to achieve a specific objective.

- The Execution Time EET(j, r) is defined as the time the resource r will take to execute the job j from the time the job starts executing on the resource.
- The Completion Time ECT(j, r) is the time at which job j would complete execution at resource r:

  ECT(j, r) = EET(j, r) + max(EAT(j,r), FAT(j, r)) (Blythe, et al., 2005)

2.3.1.2. The task scheduling problem in fog computing

Task scheduling issue in a fog computing system, is where a fog provider can exploit the collaboration between its fog nodes for efficiently executing users’ large-scale offloading applications. Task scheduling aims to achieve a high performance computing and the best system throughput (Salot, 2013). In fog computing, task scheduling problem means assigning a set of tasks to fog nodes located at the edge of the network (Bitam, et al., 2017).

2.3.1.3. Task scheduling process

Scheduling process in fog can be generalized into three stages namely:

- Resource discovering and filtering: Fog Broker discovers the resources (fog devices) present in the network system and collects status information related to them.
- Resource selection: Target resource is selected based on certain parameters of task and resource. This is deciding stage.
- Task submission: Task is submitted to resource selected (Salot, 2013).

2.3.1.4. Task scheduling algorithms classification

According to a simple classification, task scheduling algorithms in fog computing can be categorized into two main groups; Batch mode heuristic scheduling algorithms (BMHA) and online mode heuristic algorithms (Salot, 2013)..

In BMHA, tasks are queued and collected into a set when they arrive in the system. The scheduling algorithm will start after a fixed period of time. The main examples of BMHA based algorithms are; First Come First Served scheduling algorithm (FCFS), Round Robin scheduling algorithm (RR), Min–Min algorithm and Max–Min algorithm.

By On-line mode heuristic scheduling algorithm, Jobs are scheduled when they arrive in the system. Since the cloud environment is a heterogeneous system and the speed of each processor varies quickly, the on-line mode heuristic scheduling algorithms are more appropriate for a cloud environment. Most fit task scheduling algorithm (MFTF) is suitable example of On-line mode heuristic scheduling algorithm.

a. **First Come First Serve Algorithm:** Job in the queue which come first is served. This algorithm is simple and fast.

b. **Round Robin algorithm:** In the round robin scheduling, processes are dispatched in a FIFO manner but are given a limited amount of CPU time called a time-slice or a quantum. If a process does not complete before its CPU-time expires, the CPU is preempted and given to the next process waiting in a queue. The preempted process is then placed at the back of the ready list.

c. **Min–Min algorithm:** This algorithm chooses small tasks to be executed firstly, which in turn large task delays for long time.

d. **Max – Min algorithm:** This algorithm chooses large tasks to be executed firstly, which in turn small task delays for long time.

e. **Most fit task scheduling algorithm:** In this algorithm task which fit best in queue are executed first. This algorithm has high failure ratio.

f. **Priority scheduling algorithm:** The basic idea is straightforward: each process is assigned a priority, and priority is allowed to run. Equal-Priority processes are scheduled in FCFS order. The shortest-Job-First (SJF) algorithm is a special case of general priority scheduling algorithm. An SJF algorithm is simply a priority algorithm where the priority is the inverse of the (predicted) next CPU burst. That is, the longer the CPU burst, the lower the priority and vice versa. Priority can be defined either internally or externally. Internally defined priorities use some measurable quantities or qualities to compute priority of a process (Salot, 2013).

2.3.2. Related works on fog comuting task scheduling

In the literature, many studies have been proposed to solve task scheduling problem in heterogeneous computing systems, where the sequence of tasks (workflow) is popularly presented by a Directed Acyclic Graph (DAG), nevertheless, there are few research activities aiming at scheduling tasks in fog computing domain. Because DAG task scheduling is a non-deterministic polynomial-time complete (NP-complete) problem, it is expected to be solved by heuristic algorithms for finding approximate optimal solutions. The heterogeneous earliest finish time (HEFT) algorithm is the most popular and widely used algorithm to solve task scheduling problem in heterogeneous computing systems such as cloud and fog computing systems. The HEFT includes two main phases: a task prioritizing phase for computing the priorities of all tasks based on upward rank value and a processor selection phase for selecting the task with the highest upward rank value at each step and assigning the selected task to the processor which minimizes the task's finish time (Pham, et al., 2017).

In the literature, we can find (Ningning, et al., 2016) in which the authors designed a fog computing load balancing mechanism of task allocation based graph partitioning, where fog computing tasks are assigned to a single or multiple virtual machines nodes according to the level of resources required by the task. The authors represent the physical nodes of the fog computing by a non-directional graph. These physical nodes come into a set of virtual machine's nodes according to the available fog computing resources, where virtual machine nodes provide services to the users by graph partitioning. To achieve this, a minimum spanning tree is constructed from the entire graph; edges that did not provide enough resources are removed. The resulting graph represents the load balancing partition that is handled by fog computing. This effectiveness of this proposed mechanism has been demonstrated in terms of tasks' run time. This study took into account only one criterion that is execution time, regardless of the existence of others important criterion such as the monetary cost.

In (Pham, et al., 2016), the authors proposed to use heuristic-based algorithm, whose major objective is achieving the balance between the makespan and the monetary cost of cloud resources. The authors consider task scheduling in a cloud-fog computing system, where a fog provider can exploit the collaboration between its fog nodes and the rented cloud nodes for efficiently executing users' large-scale offloading applications, and achieving a good tradeoff between the workflow execution time and the cost for the use of cloud resources. The results obtained during this study, could be more general, if the authors use other algorithms such Genetic Algorithm and Particle swarm optimization when comparing the proposed algorithm's results.

The purpose of (Pham, et al., 2017) is to study the tradeoff issue between the makespan and cloud cost when scheduling large-scale applications in such a platform. A scheduling algorithm called Cost-Makespan aware Scheduling heuristic was proposed where its major objective is to achieve the balance between the performance of application execution and the mandatory cost for the use of cloud resources. Additionally, an efficient task reassignment strategy based on the critical path of the directed acyclic graph modeling the applications is also proposed to refine the output schedules of the Cost-Makespan aware Scheduling algorithm to satisfy the user-defined deadline constraints or quality of service of the system. The results obtained of this study could be more convincing with a complexity analysis of the approach execution time.

The authors of (Bitam, et al., 2017) explored a novel approach called Bees Life Algorithm (BLA), whose major objective is to find an optimal tradeoff between CPU execution time and allocated memory required by fog computing services established by mobile users. The empirical performance evaluation results demonstrate that the proposal outperforms the traditional particle swarm optimization and genetic algorithm in terms of CPU execution time and allocated memory. This study could be generalized if BLA is tested on a large-scale fog computing platform.

2.4. Conclusion

In this chapter, we have exposed the scheduling problem, and then a state-of-the-art dealing with the treatment of this problem was presented. Following our comparison study, we choose Fog Simulated Annealing as a method to solve this problem. In the next chapter, we will see the detailed design of our system in order to use FSA to solve this problem.

Chapter 3

The general

Design

Chapter 3. The general design

3.1. Introduction

To deal with the task scheduling problem in fog computing, we propose a new optimization approach named Fog Simulated Annealing algorithm (FSA) (Labed, et al., 2018) to find an optimal allocation for tasks among the available fog resources (i.e. fog nodes) so that we can achieve a tradeoff between the cost and the time of executing the tasks. In this way, the response latency and bearable cost can satisfy mobile users' requests.

In this chapter, we will consider the design of our system, which is the use of a Simulated Annealing Algorithm to solve the problem of task scheduling in a fog computing environment. More specifically, we will explain the steps of the simulated algorithm to solve this optimization problem.

3.2. Contributions

The main contributions of this work are summarized as follows:

- ✓ To deal with the task scheduling problem in fog computing, we propose a new optimization approach named Fog Simulated Annealing algorithm (FSA) to find an optimal allocation for tasks among the available fog resources (i.e. fog nodes) so that we can achieve a tradeoff between the cost and the time of executing the tasks. In this way, the response latency and bearable cost can satisfy mobile users' requests.
- ✓ We evaluate the performance of the proposed novel approach FSA and demonstrate its efficiency by comparing its performances with other approaches using FIFO and the genetic algorithm (GA).

3.3. System model

The fog computing system has three layers in a hierarchy network, as represented in figure 6.

The front-end layer consists of IoT devices, which serve as user interfaces that send requests from users.

The fog layer, which is formed by a set of near-end fog nodes, receives and processes users' requests. Fog server or broker, is a centralized management component and task scheduler. The broker (1) receives all requests of users; (2) manages available resources on fog nodes (e.g. processing capacity, network bandwidth) as well as processing and communication costs together with results of data query returned from nodes; and (3) creates the most appropriate schedule for an input workflow.

The cloud layer, which hosts a number of computing machines or cloud nodes, provides outsourced resources to execute the workload dispatched from the fog layer.

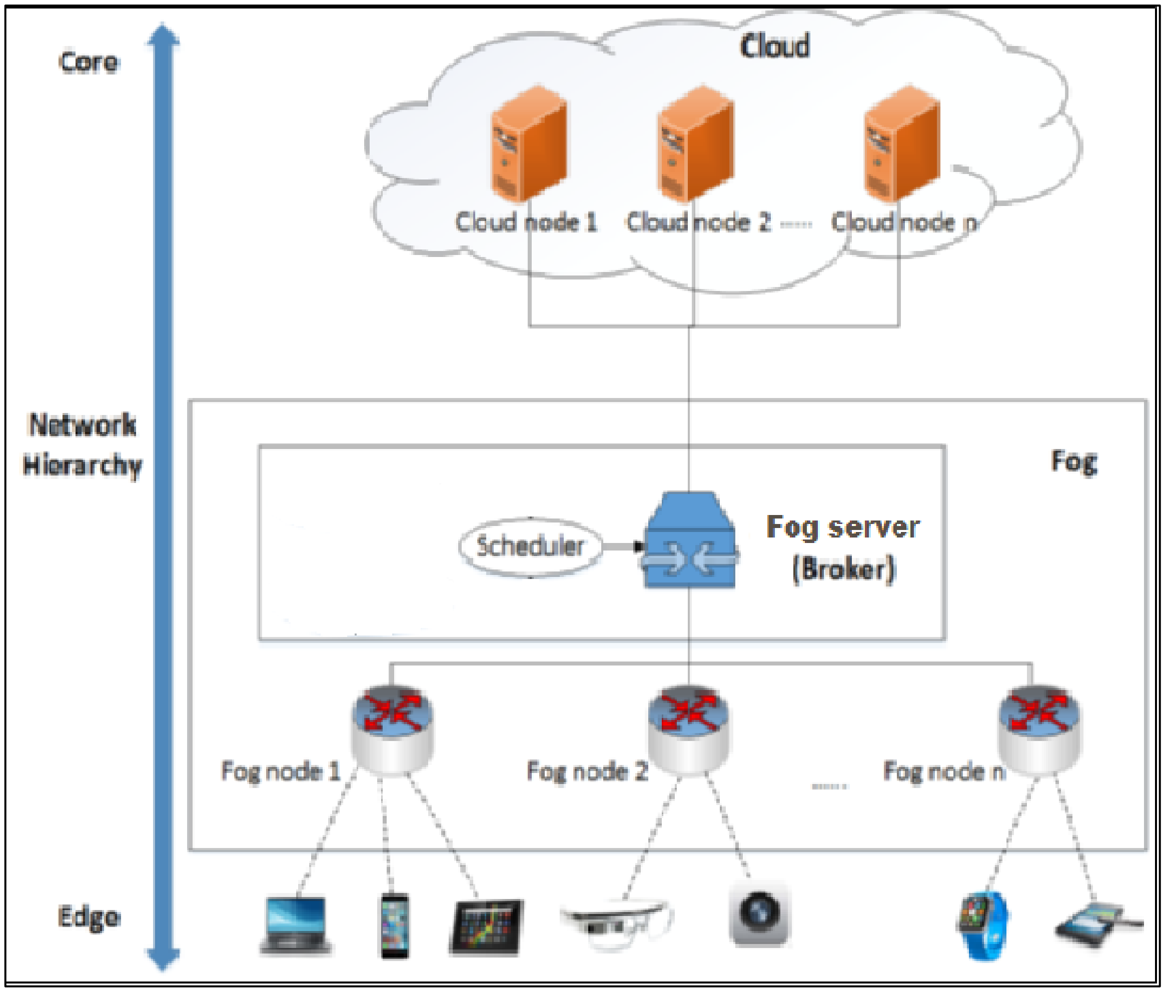


Figure 6: System architecture

To ensure task scheduling, we propose a new algorithm called Fog simulated annealing algorithm (FSA), performed by the administrator node in order to find the optimal order (scheduling), which is further executed by the fog nodes. Thus, the fog nodes can together guarantee cost and makespan performance of the scheduled services.

We describe the operation of our system model by summarizing the steps for running a scheduled service in figure 7.

First, a mobile user sends a service request to a fog node located at the edge of the network of this computing infrastructure (step 1). Next, the fog node sends data and parameters of this request to the administrator node often located far away from the user (step 2). In the following step 3, FSA is executed to find the best task scheduling. Next, each fog node receives its assigned tasks (step 4). These tasks are executed at the level of the fog nodes (step 5). Each fog node sends its results (step 6) to the administrator node. The final result is then sent to the mobile user as a service response (step 7).

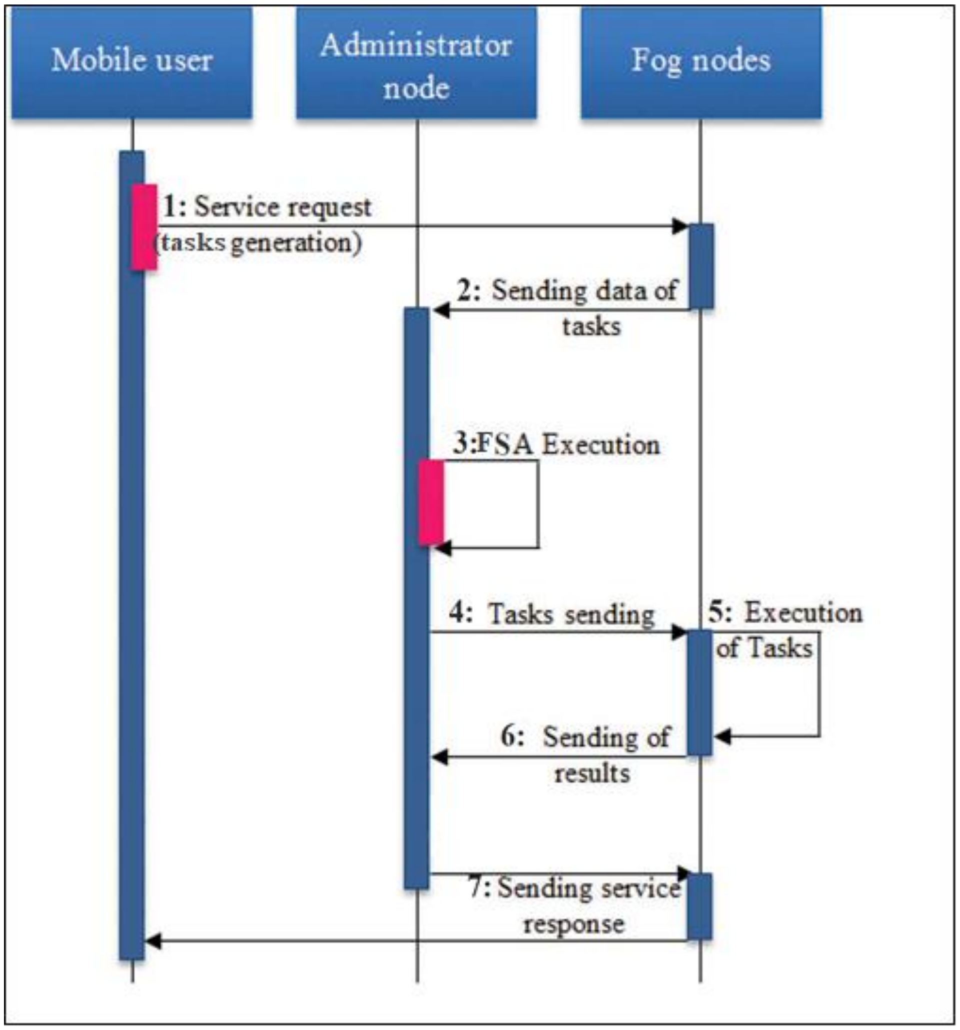


Figure 7: System model of fog tasks scheduling

3.4. Problem formulation

Task Scheduling aims at assigning tasks to fog nodes in the fog environment so that the cost and execution time (makespan) of the overall tasks minimized. This problem can be formulated as follows:

We denote by:

$$Tasks = \{T_1, T_2, \dots, T_i, \dots, T_n\}$$

A set of ‘n’ tasks. Disseminated among ‘m’ fog devices (FDs) in order to be executed. Consequently, each fog device can carry out tasks set, FD_j ensures the execution of the tasks as follows:

$$FD_jTasks = \{T_{1j}, T_{2j}, \dots, T_{kj}, \dots, T_{rj}\}$$

For example, FD_j carries out:

$$FD_jTasks = \{T_{2j}, T_{11j}, T_{13j}\}$$

Therefore, the total execution time of all tasks (‘r’ tasks) assigned to FD_j would be:

$$ExeTime(FD_jTasks) = \sum_{k=0}^{r} T_{kj}.ExeTime \qquad (1)$$

Where $T_i.ExeTime$ is the execution time of task ‘i’ at FD_j.

In addition, the cost of using a fog node FD_j would be as follows:

$$Cost(\ FD_j) = \ ExeTime(FD_jTasks) * usingCost(\ FD_j) \qquad (2)$$

Where, $usingCost(\ FD_j)$ is the cost of using a fog device during a specific time unit.

In order to evaluate the quality of the requested solution $FDTasks$, two fitness function is defined as follows:

The total execution time of all tasks (‘n’ task):

$$MakeSpane(FDTasks) = Max\left(ExeTime(FD_jTasks)\right) \qquad (3)$$

The total cost of executing all tasks (‘n’ task):

$TotalCost(FDTasks) = \sum_{j=0}^{m} Cost\big(FD_j\big)$ (4)

Thus, the tasks scheduling problem in the fog computing could be defined as searching of a set:

$$FDTasks = \{FD_1Tasks, FD_2Tasks, \dots, FD_mTasks\}$$

and:

$$FD_jTasks = \{T_{1j}, T_{2j}, \dots, T_{kj}, \dots, T_{rj}\}\ with\ 0 < r \leq n$$

which reduce the Makespan and the Total Cost.

3.5. Fog Simulated Annealing Algorithm

3.5.1. Origin of the algorithm

Simulated annealing algorithm (JOHNSON, et al., 1991) is an approach proposed to the approximate solution of difficult combinational optimization problems (problems that are central to the disciplines of computer science and engineering (Kirkpatrick, et al., 1983)).

Simulated annealing originates from the analogy between the physical annealing process and the problem of finding (near) minimal solutions for discrete minimization problems (Dekkers, et al., 1988).

The simulated annealing in metallurgy based on an analogy of thermodynamics with the way metals cool and anneal. The solid is heated to melt first, and then let it cool slowly solidified into a regular crystal. When heating the fixed, the inside of the solid particles can increase the internal energy with the increase of internal temperature. When internal energy achieves maximum, the arrangement state of the particle into a liquid disordered. This process is called smelting. When cooling, particle solidified into a solid crystalline state with the decrease of the temperature. The particle is orderly and solidified into a solid crystalline state. This process is called annealing (Liu, et al., 2010).

3.5.2. The basic properties of the simulated annealing algorithm

- Energy: Energy function is expressed as E， E=f(x)， f (x) is the general function.

- Generating function: Generating function g (x1) defines the probability density function of the difference between the current point and the next point to be accessed. $x1 = xnew - x$. According to x1, the algorithm will generate a new value.
- Accept function: After generating a new solution by generating functions, the algorithm based on accepted function P(E1, T) value to determine whether to accept or give up the new value.
- Probability accept function $P = \exp(-E1/Ct)$ .C is a constant system. T is the temperature. E1 is the energy difference between $f(xnew) - f(x)$. simulated annealing algorithms based on probability way to generate new value. When the E1 is negative, the algorithm tends to accept the solution. When E1 is positive, the algorithm tends to accept the solution with smaller probability.
- Simulated annealing schedule: Annealing schedule is a function of time or number of iterations that controls the temperature that drop from high to low. (Liu, et al., 2010).

3.5.3. Advantages of simulated annealing algorithm

SA has become one of the many heuristic approaches designed to give a good, not necessarily optimal solution. It is simple to formulate and it can handle mixed discrete and continuous problem with ease. It is also efficient and has low memory requirement. SA takes less CPU time than genetic algorithm (GA) when used to solve optimization problems, because it finds the optimal solution using point-by-point iteration rather than a search over a population of individuals. (Suman, et al., 2006).

It is a method to obtain an optimal solution of a single objective optimization problem and to obtain a Pareto set of solutions for a multi-objective optimization problem with a substantial reduction in computation time (Busetti, 2003).

It is quite powerful finding global minimum, and provides more features at the cost of an increase in execution time for a single run of the algorithm (Goffe, et al., 1994)

When adapted efficiently to optimization problems, SA is often characterized by fast convergence and ease of implementation. These characteristics motivate the choice of SA for NP-hard combinatorial optimization problems (Bouleimen K, 2003).

3.6. Fog Simulated annealing for task scheduling

In this section, we present our contribution, which is the proposal to solve the task scheduling problem in the fog computing by applying simulated annealing algorithm, we named our proposed algorithm Fog Simulated Annealing algorithm (FSA).

3.6.1. FSA Algorithm Illustration

Fog Simulated Annealing Algorithm (FSA) pseudo-code is as follows:

```
FSA_Begin
S0 = initialization; // generating an initial solution (a task scheduling) at random
S* = S0; // best task scheduling
f1(S0);  //exaction time off all tasks using scheduling S
f2(S0);  // total cost off executing all tasks using scheduling S
E(S0) =  w1 * f1(S0) + w2 * f2(S0); // evaluation of S0 by applying the objective
                                       function (Energy) according to the objective weights
T = Max_Temp; // temperature of system initialized with a high value
Tmin = Min_Temp; // the lower limit of temperature; When T achieves T_min, the
                              system stops its iterations
$while(T > Tmin)$
{
   $S = neighbor(S0);$   // Selecting a neighbor solution of S0
   E(S) =  w1 * f1(S) + w2 * f2(S);  // evaluation of S

        $if( E(S) < E(S0))\ then$
           {
```

```
$else$        // Accepting probabily a worest solution
          $S0 = S$;

          $S* = S0$; //set the new scheduling as the best solution

          }

  $if\ Probability\ (Exponential\left(-\frac{E(S0)-E(S)}{T}\right))$ > Random (0, 1) then

   {

              S0 = S;

        };

        $T = T * r$; // reducing the temperature and annealing, 0<r<1

   };

FSA_End.
```

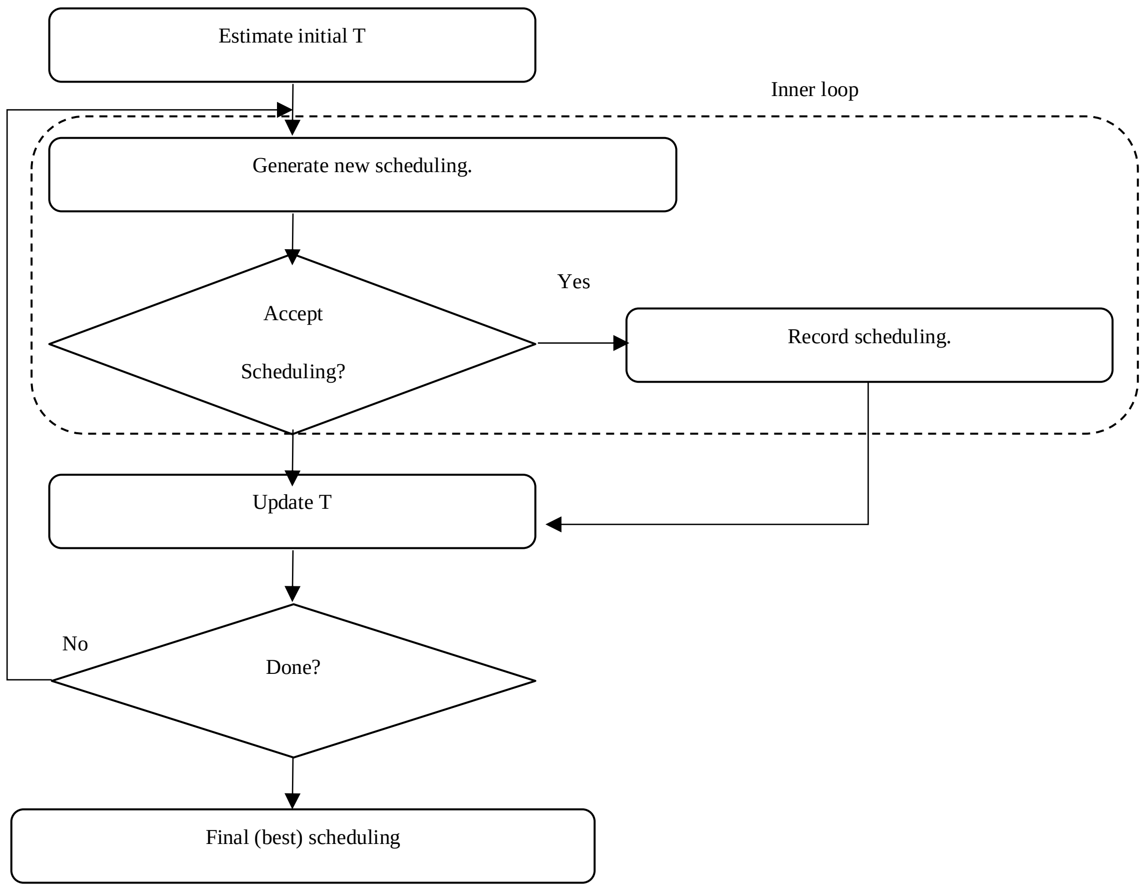


Figure 8: Pseudo code and Flowchart of FSA

Our Fog Simulated Annealing algorithm (FSA) for task scheduling in fog environment presented in figure 8 (FSA pseudo-code and flowchart) works as follows:

First, FSA starts with an initial task scheduling chosen at random (S0), then this found solution is temporary considered as the best one. For instance, for five tasks T_a, T_b, T_c, T_d, T_e, we select randomly five fog devices x, y, z, t, r. After that, this S0 solution is evaluation by applying a bi-objective function E(S0) based on both execution time and cost of the chosen task scheduling. This function is considered as bi-objectives (execution time and cost of the chosen task scheduling) biased by two weights w1 and w2 showing the importance of these two objectives. Next, temperature T is set with a high value Max_Temp and its minimum value is also set with value Min_Temp. Min_Temp is used a stopping criterion to stop FSA.

Now, the main loop of FSA is performed; we start by generating and selecting a neighbor solution (called S) of S0, which is evaluated.

If this neighbor solution is better that the last one, it will be considered temporary as the best solution, otherwise this newer solution (which is worse than S0) is accepted only with a probability: $Exponential(\frac{-(E(S0)-E(S))}{T})$. Next, temperature T is reduced.

This process is performed until satisfying the stopping criterion.

3.6.2. FSA Algorithm representation, evaluation and stopping criterion

The encoding system used in this work to represent the individuals is the list of strings as dynamic data structure. Each list (solution) contains 'N' fog device where each fog device carried out a set of tasks as follows:

$$FDTasks = \{FD_1Tasks, FD_2Tasks, \dots, FD_mTasks\}$$

Each fog device carried out a set of tasks as follows:

$$FD_jTasks = \{T_{1j}, T_{2j}, \dots, T_{kj}, \dots, T_{rj}\}$$

The FSA initialization aims at the generation of the first solution randomly:

$FDTasks = \{FD_1Tasks, FD_2Tasks, \dots, FD_mTasks =$
$\{<T_{11}, T_{21}, \dots, T_{r1}>, <\{T_{12}, T_{22}, \dots, T_{r2}>, \dots, <T_{1m}, T_{2m}, \dots, T_{rm}>\}$

To evaluate each solution, the fitness function (formula 4) is applied in order to find minimum $MakeSpane(FDTasks)$ and minimum $TotalCost(FDTasks)$.

As a stopping criterion, FSA iterations are carried out and stopped only when the temperature T became equal to the temperature 'Min_Temp'. The number 'Min_Temp' is a user parameter.

3.6.3. FSA Algorithm operators

3.6.3.1. Generating function (neighborhood generation):

There are many ways to generate a neighbor solution (similar solution). In our proposal, we create a neighbor using swapping by randomly choosing 'n' tasks, and swapping their assigned fog devices also randomly.

For example (for 5 tasks):

Solution = {<T_{11}, T_{21}, T_{31}>,<{T_{12}, T_{22}, T_{32}>,<T_{13}, T_{23}, T_{33}>}

Neighbor = {<T_{11}>,<.>,<$T_{13}, T_{23}, T_{33}, T_{43}, T_{53}, T_{63}, T_{73}, T_{83}$,>}

We can see that the 5th task has moved to the fog device FD3, two came from FD1 and the other three from FD2.

3.6.3.2. Accept function (neighborhood selection):

In our proposal, we select the first generated neighbor that reduces the execution time, this selection is named: the first improvement.

g. Acceptance probability:

The Simulated Annealing's major advantage over other optimization methods is its ability to avoid becoming trapped in local minima. The algorithm employs a random search which not only accepts changes that optimize the objective function f (assuming a minimization problem), but also some changes that decrease it; it is the case of a worse neighbor. The latter is accepted with a probability.

The applied probability in our FSA is as follows (as explained above):

$$Prob = \exp(-\frac{(E(S0) - E(S))}{T})$$

3.7. Conclusion

In this chapter, we presented the general design of our system, which based on a simulated annealing algorithm in order to solve the problem of task scheduling in fog computing environment.

In the next chapter, we will see the scenarios of testing simulation in addition to the simulation parameters and the obtained results.

Chapter 4

Experimental study

Chapter 4. Experimental study

4.1. Introduction

In the fourth chapter, we present the simulation of the fog simulated annealing algorithm as a task scheduler using the iFogSim simulator. We start with a detailed explanation of the scenario used and simulation settings, then we explain the way the tasks are scheduled on fog devices. Finally, extraction and analysis of the results are presented.

4.2. The iFogSim simulator

As the chosen Fog computing simulator, iFogSim is an open source framework for modeling fog computing environments. As well as a simulation tool allows investigation and comparison of resource management techniques based on QoS criteria such as latency, network congestion, energy consumption, and cost under different workloads (tuple size and transmit rate) (Gupta, et al., 2016). iFogSim was developed by Java language.

❖ Basics

- Tuple: Tuples form the fundamental unit of communication between entities in the Fog. Tuples are represented as instances of Tuple class in iFogSim, which is inherited from the Cloudlet class of CloudSim.
- Application: An application is modeled as a directed graph, the vertices of the DAG representing modules that perform processing on incoming data and edges denoting data dependencies between modules. These entities are realized using the following classes:

✓ AppModule: Instances of AppModule class represent processing elements of Fog applications. AppModule is implemented by extending the class PowerVm in CloudSim.

✓ AppEdge: An AppEdge instance denotes the data-dependency between a pair of application modules and represents a directed edge in the application model. Each edge is characterized by the type of tuple it carries, which is captured by the tupleType attribute of AppEdge class along with the processing requirements and length of data encapsulated in these tuples (Gupta, et al., 2016).

4.3. Experimental settings

In order to evaluate our proposal, FSA algorithm is simulated on iFogSim simulator. We precisely modified the Java class named FogBroker.java, which is responsible for the distribution of the tasks across the fog devices (scheduling) using initially First In First Out method (FIFO). In this work, we have used a new method named Fog Simulated Annealing for task scheduling in the fog computing environment and we have compared its results against FIFO method and Genetic Algorithm results.

In iFogsim as an extension of Cloudsim (Gupta, et al., 2016), the application modules are defined as nodes and the connection between them as the edge (AppEdge) in the fog network topology (Rahbari, et al.).

In this work, we define fog computing task scheduling problem, in iFogSim simulator as finding mapping between the Tuples and the Application modules, that achieves best tradeoff between cost and time of execution the tuples. By default, the iFogSim ensures the mapping in FIFO method, where first tuple goes to first application module and so on. To generate new mapping, our algorithm works on changing the attribute of AppEdge, which is named ***destination***, and which is referred to the application module responsible of processing tuple carried by this AppEdge.

The effectiveness of FSA is tested to schedule tasks between fog devices in Fog infrastructure composted of 20 fog devices. We assume that there are 100, 500 and 1000 tasks to execute by the fog devices after the scheduling process. Note that we have chosen w1 = w2 = 1 to express the importance of the execution time and the cost of the task scheduling.

The fog devices generated as follows:

```
Algorithm 1 Create fog device
Create processor list.
Create hosts (Input, OS, VMs, cost, cost per storage).
Create storage list.
Set latency, upper and lower bandwidth.
Mapping application to modules.
```

The generated fog devices as follows:

Table 1: Fog Devices capabilities

Fog device	FD1	FD2	FD3	…	FDn
CPU(MIPS)	500	500*2	500*3	…	500*n
Using Cost($)	10	10*2	10*3		10*n

The tasks generated as follows:

Algorithm 1 Create application

Add all modules to the application (module name, ram capacity).

Add all edges between modules for the application (source, destination module name, tuple CPU length, and direction).

Add tuple mapping (module name, input tuple type, output tuple).

Add Loops of modules.

The generated tasks as follows

Table 2: The sizes of the tasks

Task	T1	T2	T3	…	Tn
CPU LENGTH(MIPS)	100	100+(50)	100+2*(50)	…	100+n*(50)

The Makespan and The Cost calculated as follows:

The execution time for each task:

$$T_{kj}.execTime = \frac{FD_j.CPU}{T_k.CpuLenght}\ (second) \qquad (5)$$

The total execution time of all tasks ('r' tasks) assigned to FD_j:

$$ExeTime\left(FD_jTasks\right) = \sum_{k=0}^{r} T_{kj}.ExeTime \qquad (6)$$

The total execution time of all tasks (‘n’ task):

$$MakeSpane(FDTasks) = Max\left(ExeTime\left(FD_jTasks\right)\right) \quad (7)$$

The total execution time of all tasks (‘r’ tasks) assigned to FD_j:

$$Cost\left(FD_j\right) = \sum_{k=0}^{r}(T_{kj}.ExeTime * FD_j.UsingCost) \ (\$) \quad (8)$$

The total cost of executing all tasks (‘n’ task):

$$TotalCost(FDTasks) = \sum_{j=0}^{m} Cost\left(FD_j\right) \quad (9)$$

4.1. Application preview

iFogSim is a simulation tool that does not have a graphical interface; meaning that all parameters and data simulations are fixed in the source code. Therefore, by using NetBeans integrated development environment (IDE) for Java, we have added a graphical user interface that allows fixing the number of default tasks and devices as well as displaying the results in a window rather than displaying them in the device. The following figures illustrate the graphical interface before and after simulations.

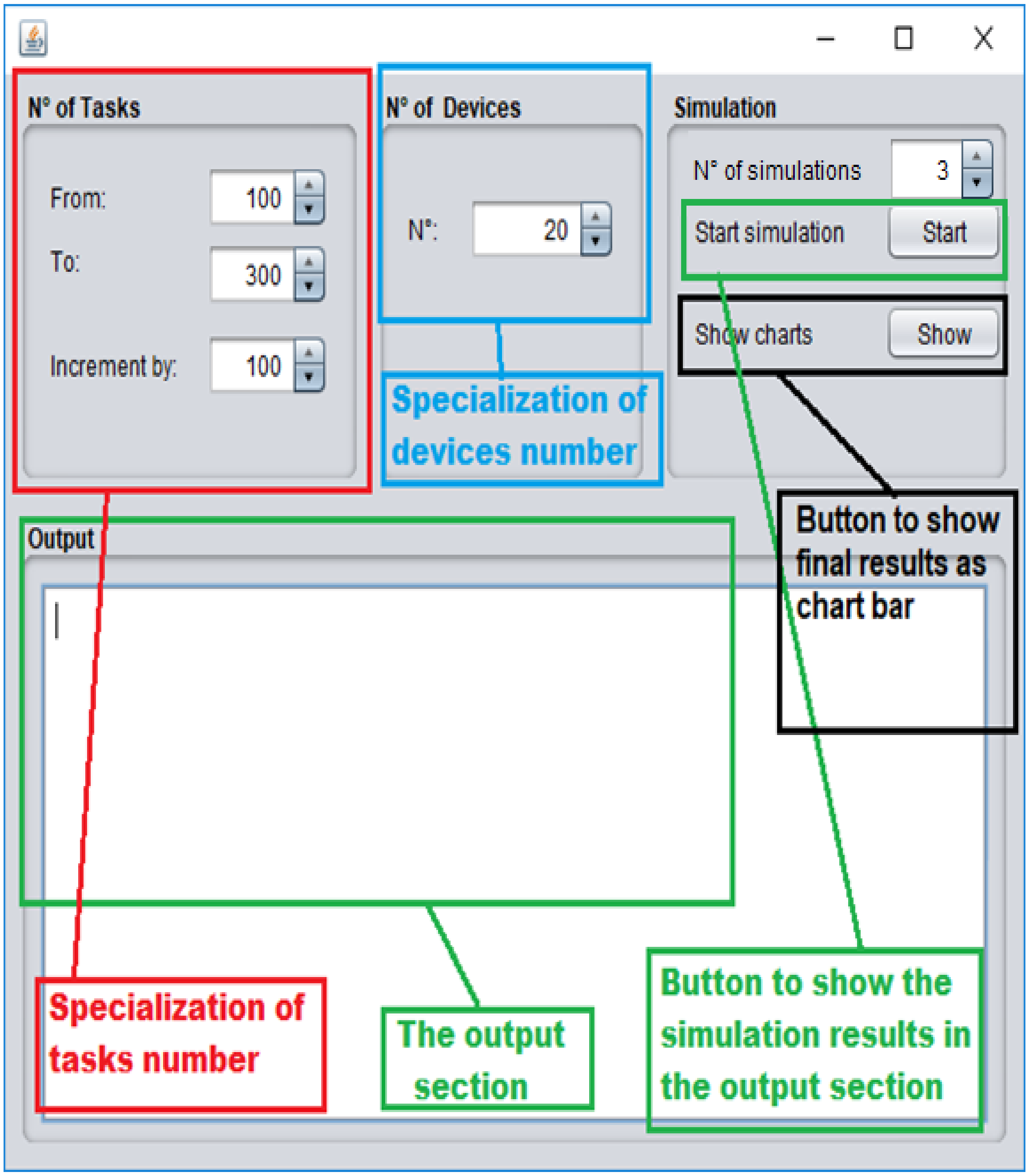
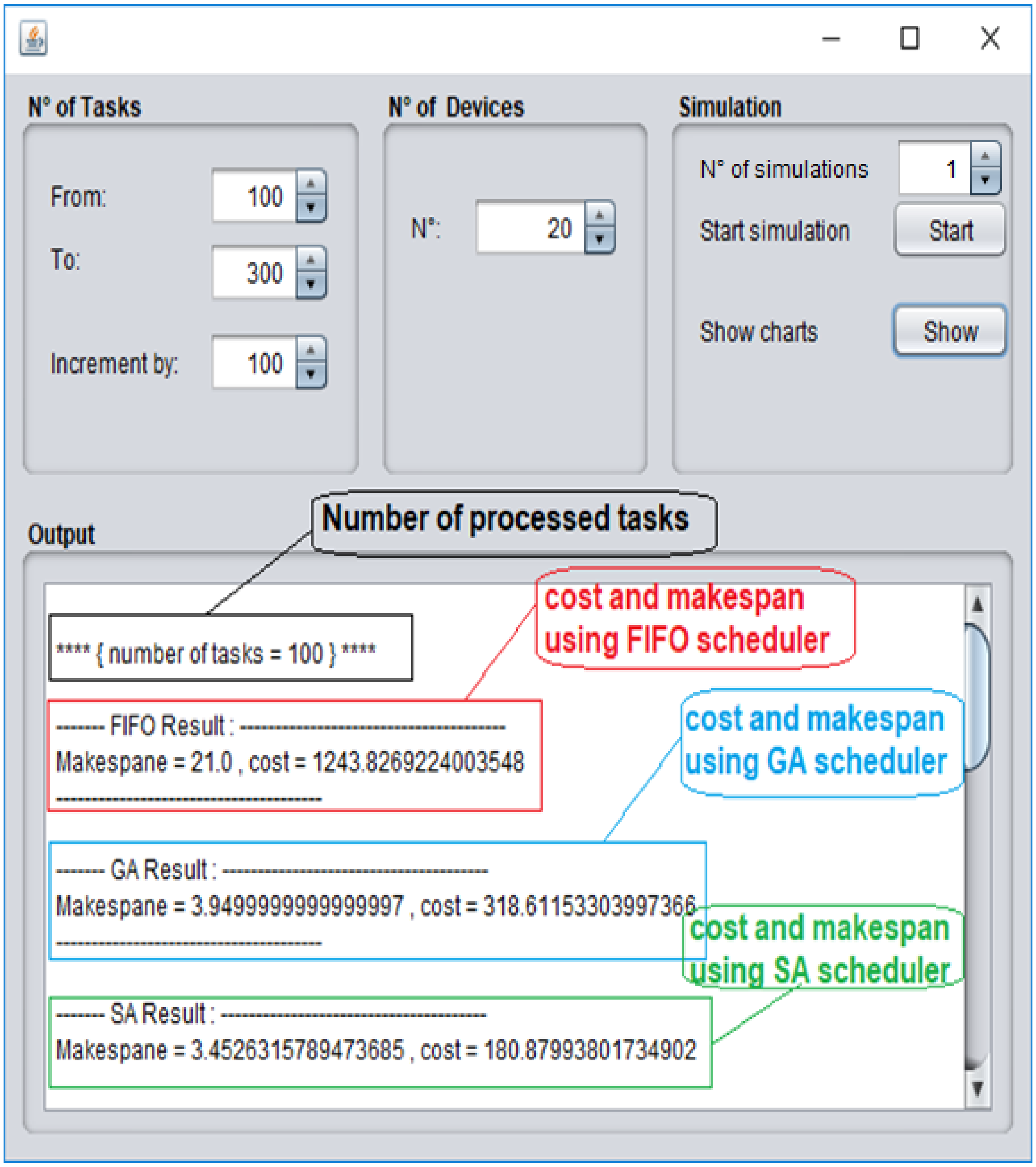
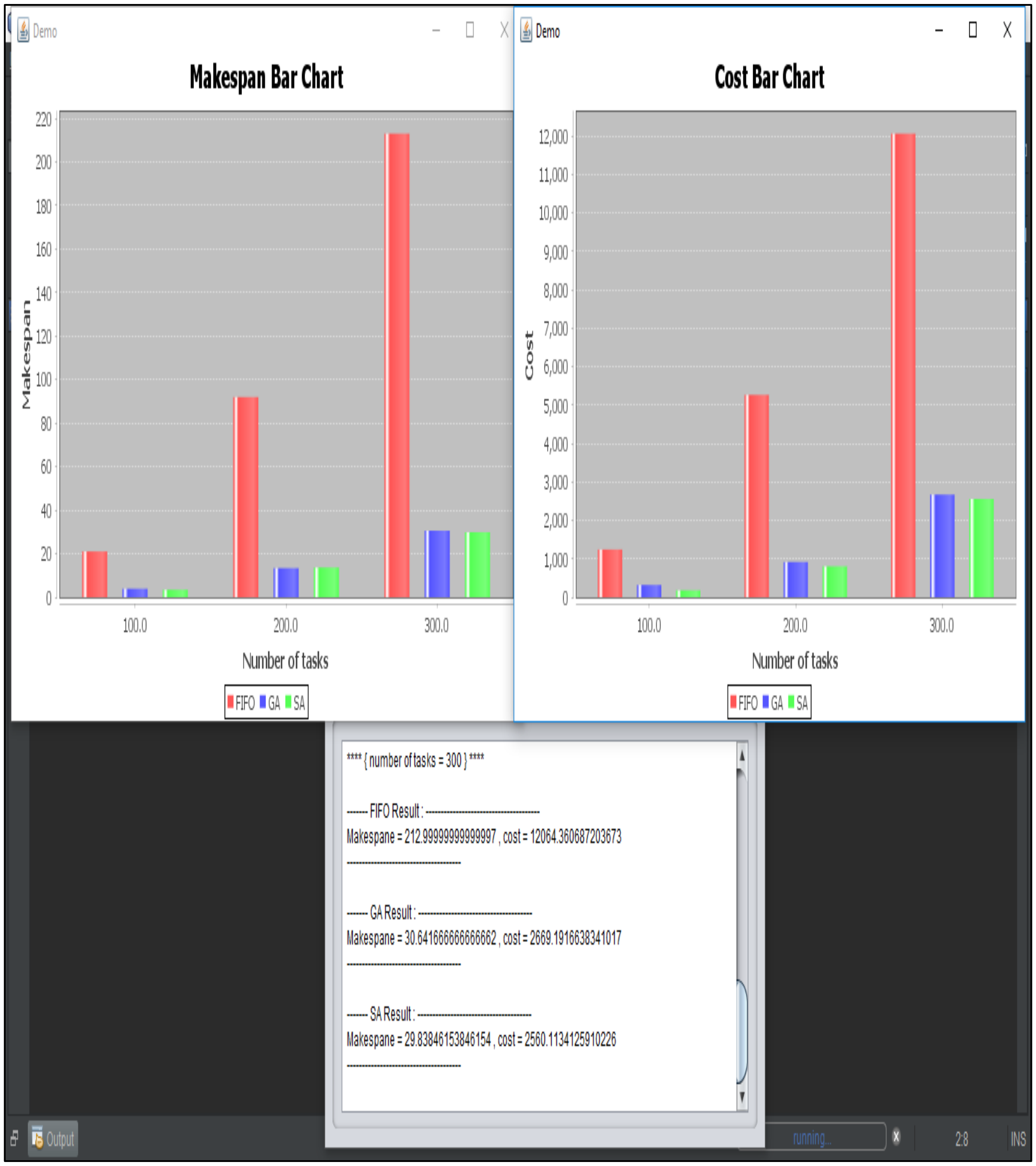

N° of Tasks
From: 100
To: 300
Increment by: 100
N° of Devices
N°: 20
Simulation
N° of simulations 3
Start simulation Start
Show charts Show
Output
Specialization of devices number
Button to show final results as chart bar
Specialization of tasks number
The output section
Button to show the simulation results in the output section

Figure 9: The principal interface


N° of Tasks
From: 100
To: 300
Increment by: 100
N° of Devices
N°: 20
Simulation
N° of simulations 1
Start simulation
Start
Show charts
Show
Output
Number of processed tasks
cost and makespan using FIFO scheduler
cost and makespan using GA scheduler
cost and makespan using SA scheduler

```
**** { number of tasks = 100 } ****
------- FIFO Result : -----------------------------------
Makespane = 21.0 , cost = 1243.8269224003548
-----------------------------------------
------- GA Result : -----------------------------------
Makespane = 3.9499999999999997 , cost = 318.61153303997366
-----------------------------------------
------- SA Result : -----------------------------------
Makespane = 3.4526315789473685 , cost = 180.87993801734902
```

Figure 10: The interface after clicking the start button



```
**** { number of tasks = 300 } ****

------- FIFO Result : -----------------------------------
Makespane = 212.99999999999997 , cost = 12064.360687203673
-----------------------------------


------- GA Result : -----------------------------------
Makespane = 30.641666666666662 , cost = 2669.1916638341017
-----------------------------------


------- SA Result : -----------------------------------
Makespane = 29.83846153846154 , cost = 2560.1134125910226
-----------------------------------
```

Figure 11: The interface after clicking the show button

Figures 12 and 13 show the results of scheduling as two bar charts, makespan and cost bar chart.

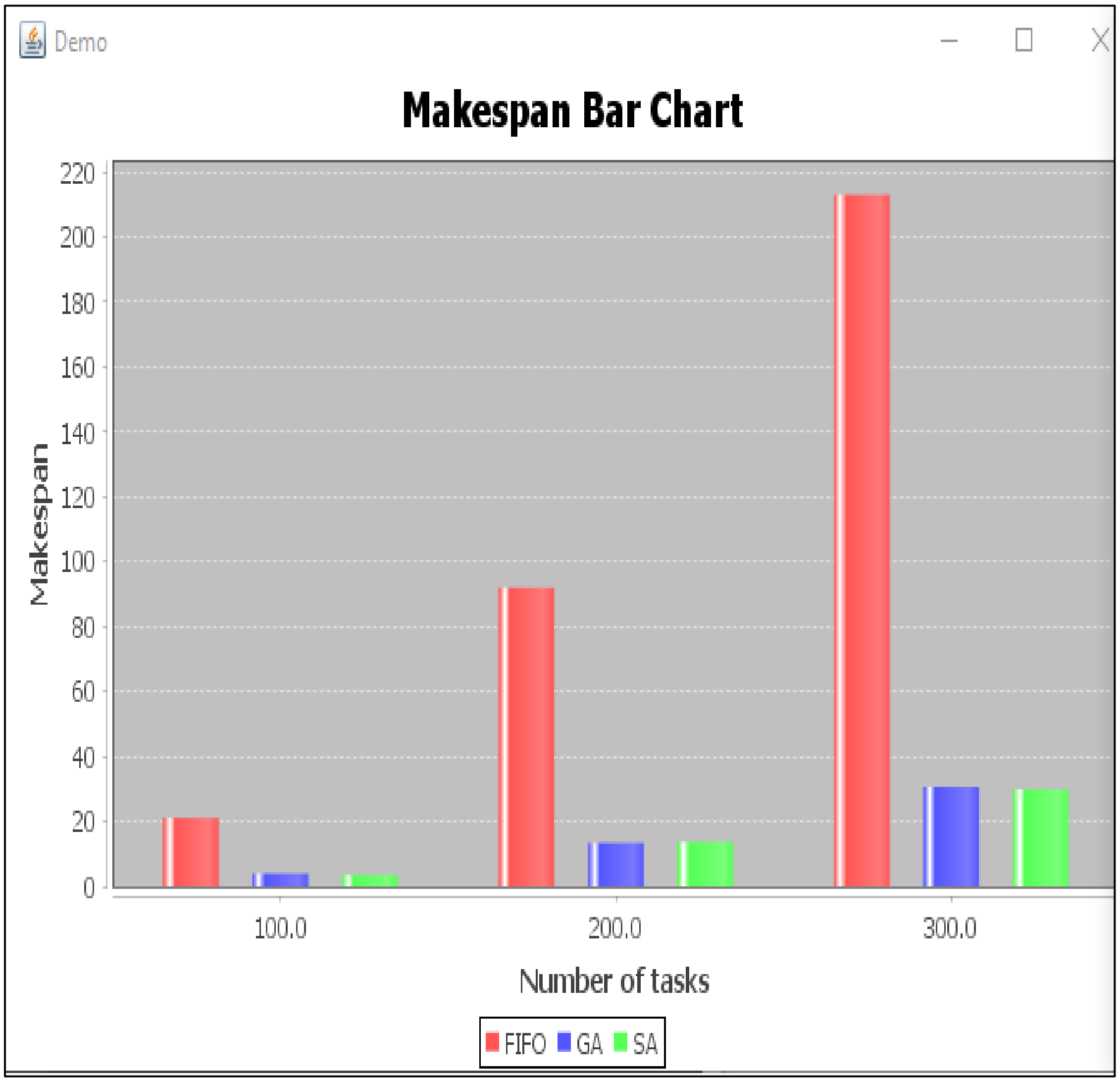


Figure 12: The makespan bar chart

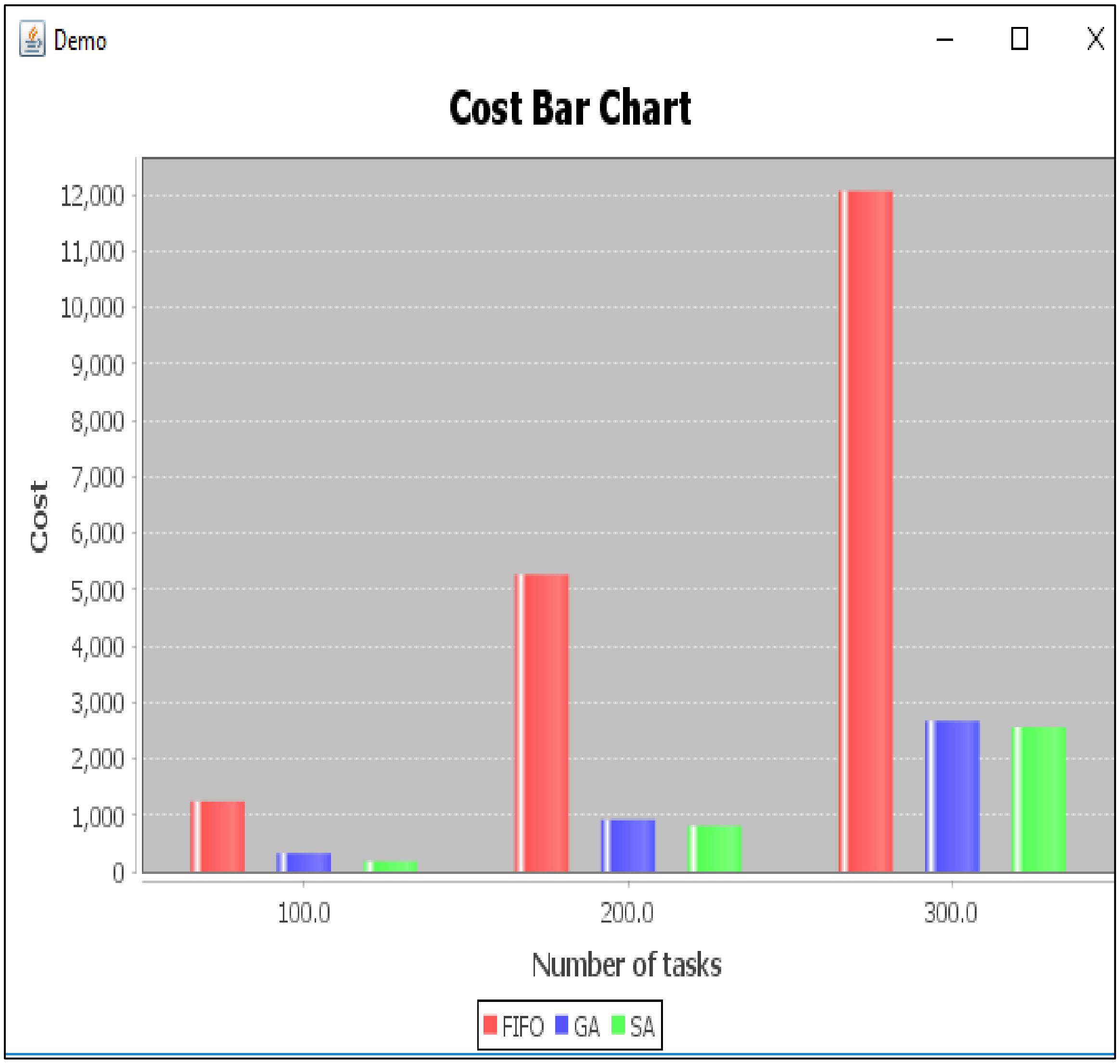
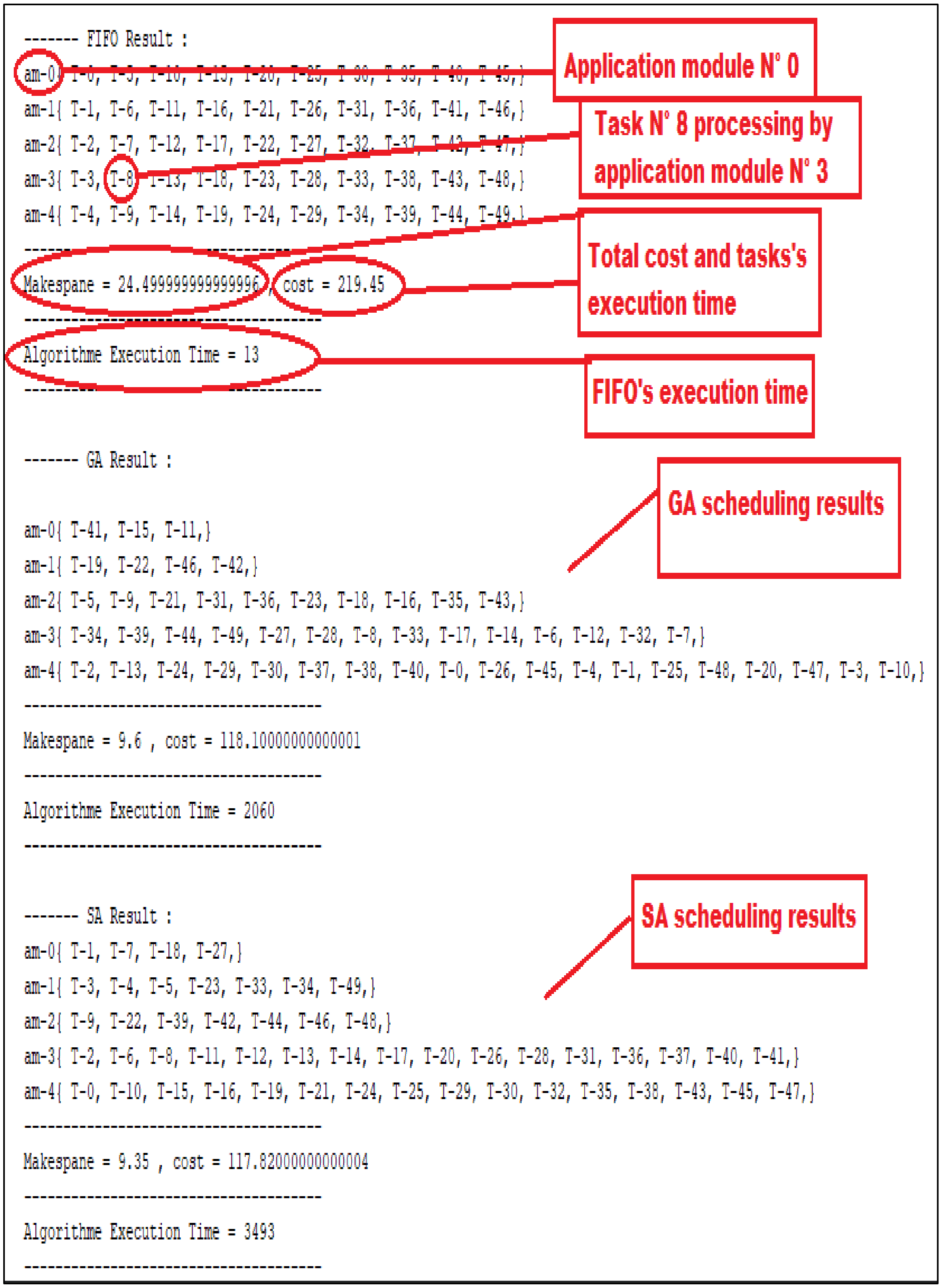


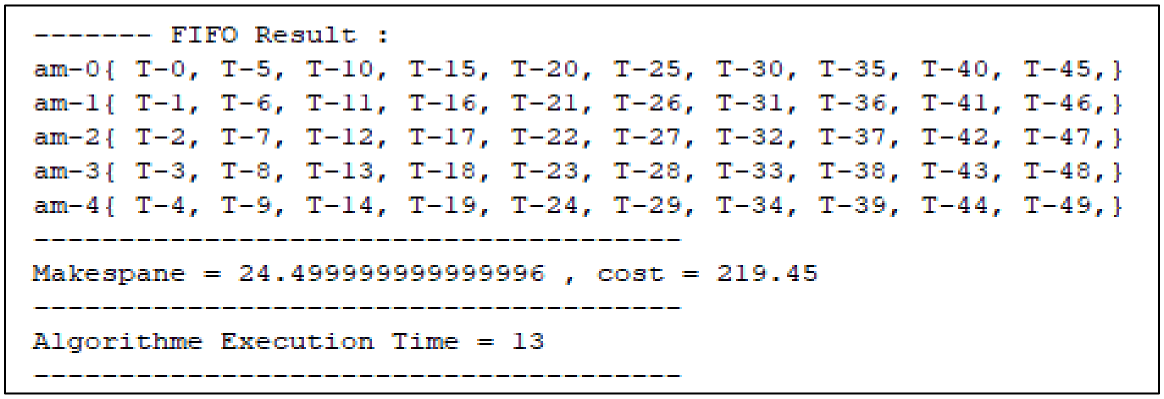
Figure 13: The cost bar chart

Figures 14. 15. 16 and 17 show the scheduling results as a list, where AM represents the application module, and T represents a task. Moreover, each application module is responsible of processing a set of tasks.



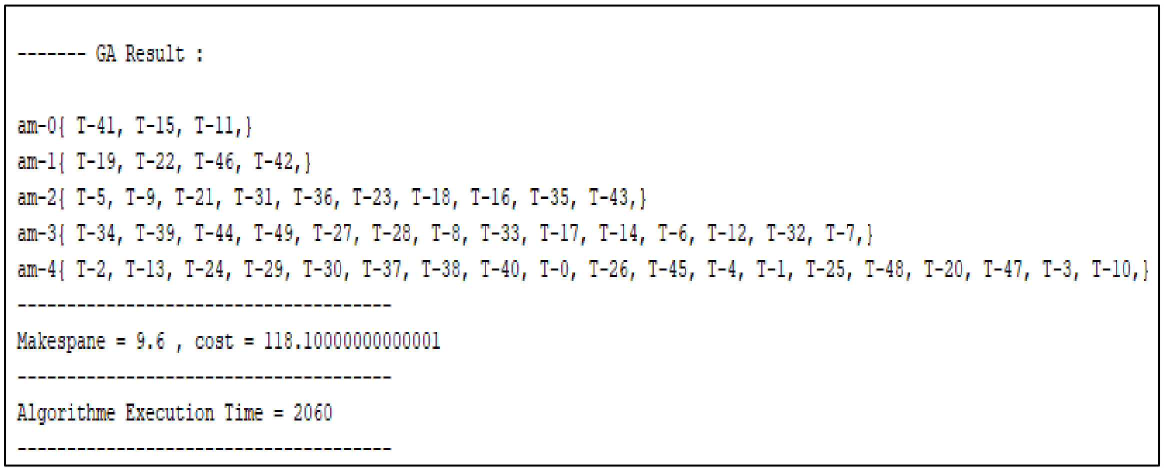
```
------- FIFO Result :
am-0{ T-0, T-5, T-10, T-15, T-20, T-25, T-30, T-35, T-40, T-45,}
am-1{ T-1, T-6, T-11, T-16, T-21, T-26, T-31, T-36, T-41, T-46,}
am-2{ T-2, T-7, T-12, T-17, T-22, T-27, T-32, T-37, T-42, T-47,}
am-3{ T-3, T-8, T-13, T-18, T-23, T-28, T-33, T-38, T-43, T-48,}
am-4{ T-4, T-9, T-14, T-19, T-24, T-29, T-34, T-39, T-44, T-49,}
--------------------------------------
Makespane = 24.499999999999996 , cost = 219.45
--------------------------------------
Algorithme Execution Time = 13
--------------------------------------


------- GA Result :


am-0{ T-41, T-15, T-11,}
am-1{ T-19, T-22, T-46, T-42,}
am-2{ T-5, T-9, T-21, T-31, T-36, T-23, T-18, T-16, T-35, T-43,}
am-3{ T-34, T-39, T-44, T-49, T-27, T-28, T-8, T-33, T-17, T-14, T-6, T-12, T-32, T-7,}
am-4{ T-2, T-13, T-24, T-29, T-30, T-37, T-38, T-40, T-0, T-26, T-45, T-4, T-1, T-25, T-48, T-20, T-47, T-3, T-10,}
--------------------------------------
Makespane = 9.6 , cost = 118.10000000000001
--------------------------------------
Algorithme Execution Time = 2060
--------------------------------------


------- SA Result :
am-0{ T-1, T-7, T-18, T-27,}
am-1{ T-3, T-4, T-5, T-23, T-33, T-34, T-49,}
am-2{ T-9, T-22, T-39, T-42, T-44, T-46, T-48,}
am-3{ T-2, T-6, T-8, T-11, T-12, T-13, T-14, T-17, T-20, T-26, T-28, T-31, T-36, T-37, T-40, T-41,}
am-4{ T-0, T-10, T-15, T-16, T-19, T-21, T-24, T-25, T-29, T-30, T-32, T-35, T-38, T-43, T-45, T-47,}
--------------------------------------
Makespane = 9.35 , cost = 117.82000000000004
--------------------------------------
Algorithme Execution Time = 3493
--------------------------------------
```

Figure 14: The results of scheduling using FIFO, GA and SA algorithms



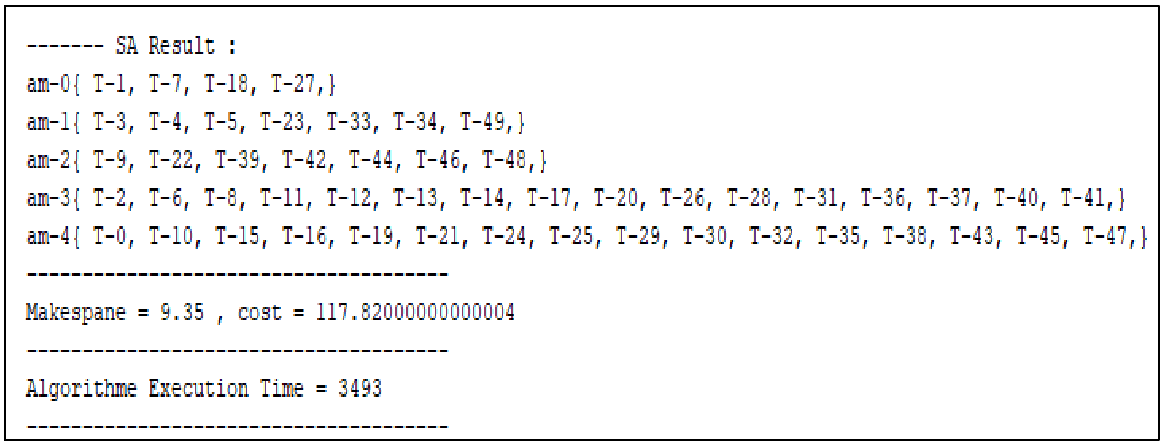
```
------- FIFO Result :
am-0{ T-0, T-5, T-10, T-15, T-20, T-25, T-30, T-35, T-40, T-45,}
am-1{ T-1, T-6, T-11, T-16, T-21, T-26, T-31, T-36, T-41, T-46,}
am-2{ T-2, T-7, T-12, T-17, T-22, T-27, T-32, T-37, T-42, T-47,}
am-3{ T-3, T-8, T-13, T-18, T-23, T-28, T-33, T-38, T-43, T-48,}
am-4{ T-4, T-9, T-14, T-19, T-24, T-29, T-34, T-39, T-44, T-49,}
--------------------------------------
Makespane = 24.499999999999996 , cost = 219.45
--------------------------------------
Algorithme Execution Time = 13
--------------------------------------
```

Figure 15: The results of scheduling 100 tasks using FIFO



```
------- GA Result :

am-0{ T-41, T-15, T-11,}
am-1{ T-19, T-22, T-46, T-42,}
am-2{ T-5, T-9, T-21, T-31, T-36, T-23, T-18, T-16, T-35, T-43,}
am-3{ T-34, T-39, T-44, T-49, T-27, T-28, T-8, T-33, T-17, T-14, T-6, T-12, T-32, T-7,}
am-4{ T-2, T-13, T-24, T-29, T-30, T-37, T-38, T-40, T-0, T-26, T-45, T-4, T-1, T-25, T-48, T-20, T-47, T-3, T-10,}
--------------------------------------
Makespane = 9.6 , cost = 118.10000000000001
--------------------------------------
Algorithme Execution Time = 2060
--------------------------------------
```

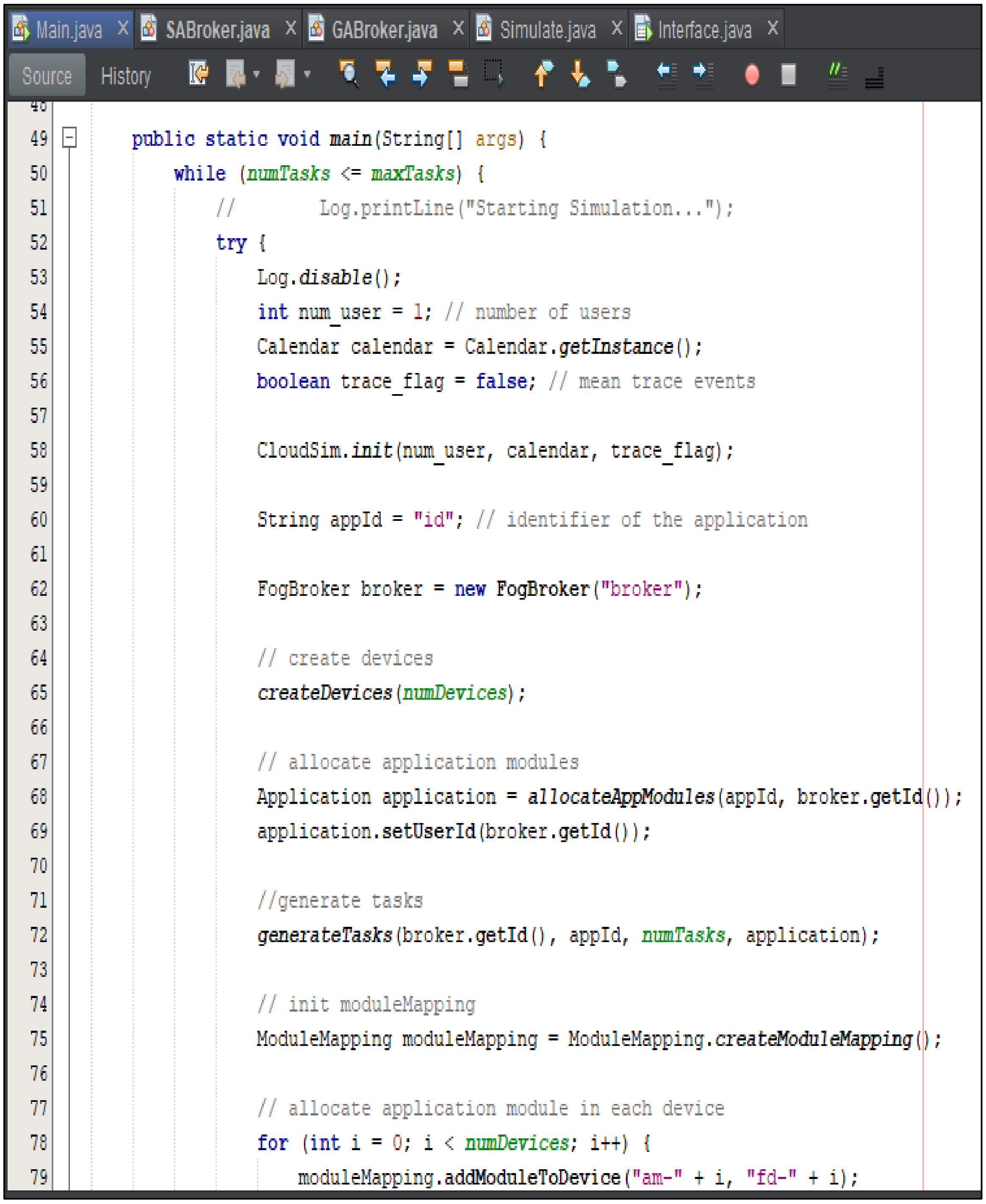
Figure 16: The results of scheduling 100 tasks using GA



```
------- SA Result :
am-0{ T-1, T-7, T-18, T-27,}
am-1{ T-3, T-4, T-5, T-23, T-33, T-34, T-49,}
am-2{ T-9, T-22, T-39, T-42, T-44, T-46, T-48,}
am-3{ T-2, T-6, T-8, T-11, T-12, T-13, T-14, T-17, T-20, T-26, T-28, T-31, T-36, T-37, T-40, T-41,}
am-4{ T-0, T-10, T-15, T-16, T-19, T-21, T-24, T-25, T-29, T-30, T-32, T-35, T-38, T-43, T-45, T-47,}
--------------------------------------
Makespane = 9.35 , cost = 117.82000000000004
--------------------------------------
Algorithme Execution Time = 3493
--------------------------------------
```

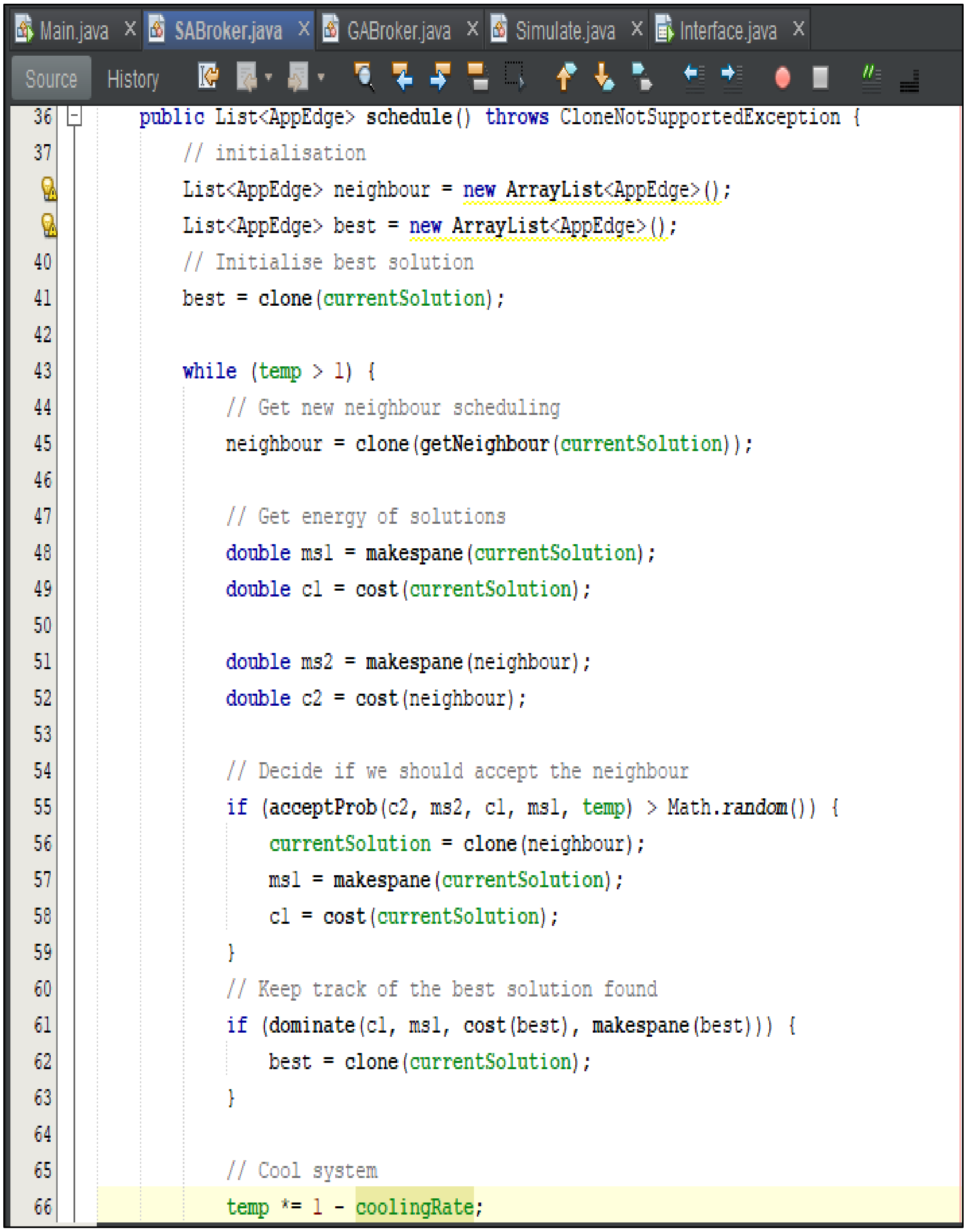
Figure 17: The results of scheduling 100 tasks using SA

The figure 18. 19 and 20 shows some essential parts of the code of our application.



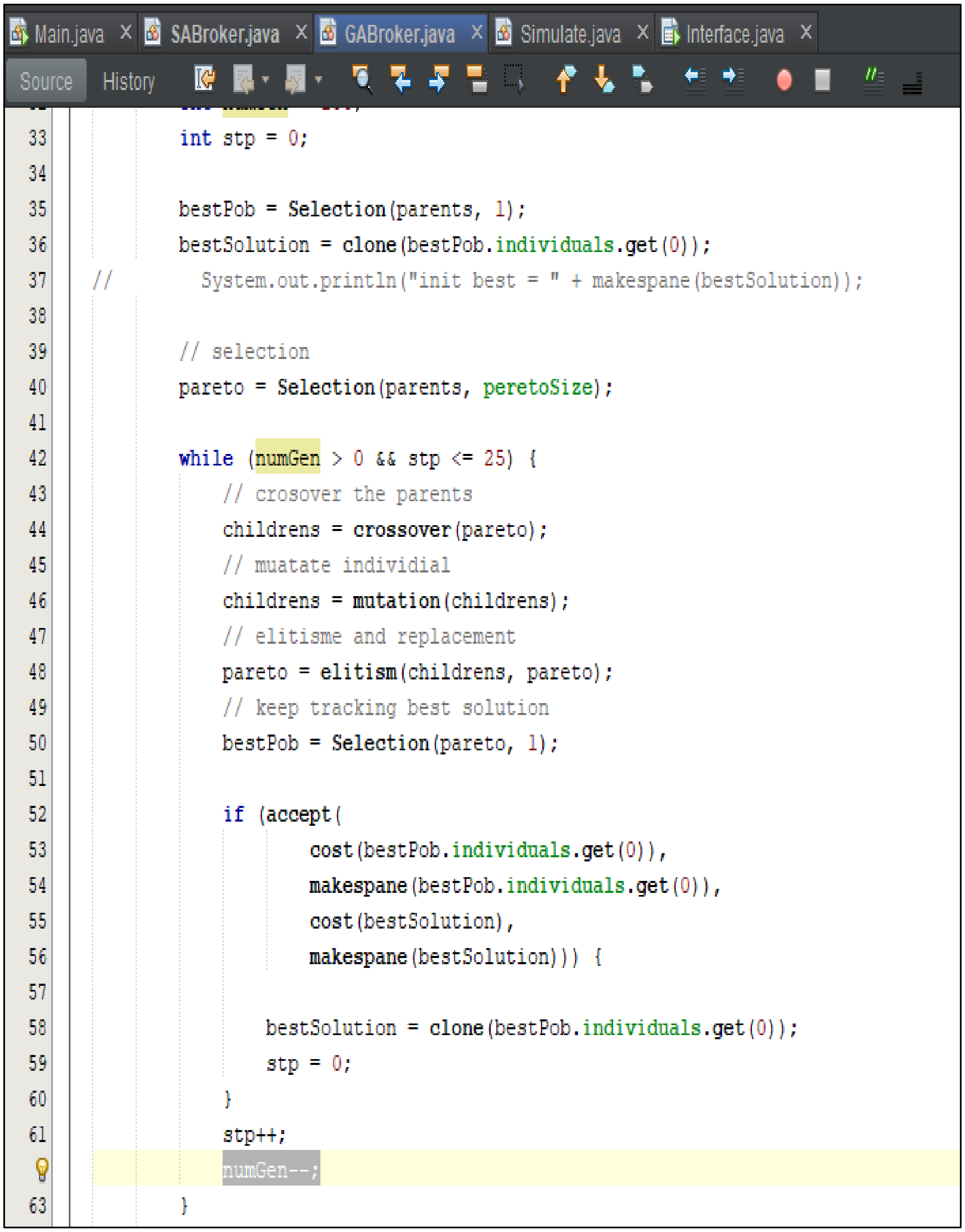
```java
    public static void main(String[] args) {
        while (numTasks <= maxTasks) {
//          Log.printLine("Starting Simulation...");
            try {
                Log.disable();
                int num_user = 1; // number of users
                Calendar calendar = Calendar.getInstance();
                boolean trace_flag = false; // mean trace events

                CloudSim.init(num_user, calendar, trace_flag);

                String appId = "id"; // identifier of the application

                FogBroker broker = new FogBroker("broker");

                // create devices
                createDevices(numDevices);

                // allocate application modules
                Application application = allocateAppModules(appId, broker.getId());
                application.setUserId(broker.getId());

                //generate tasks
                generateTasks(broker.getId(), appId, numTasks, application);

                // init moduleMapping
                ModuleMapping moduleMapping = ModuleMapping.createModuleMapping();

                // allocate application module in each device
                for (int i = 0; i < numDevices; i++) {
                    moduleMapping.addModuleToDevice("am-" + i, "fd-" + i);
```

Figure 18: Part of task and devices generating code.



```java
    public List<AppEdge> schedule() throws CloneNotSupportedException {
        // initialisation
        List<AppEdge> neighbour = new ArrayList<AppEdge>();
        List<AppEdge> best = new ArrayList<AppEdge>();
        // Initialise best solution
        best = clone(currentSolution);

        while (temp > 1) {
            // Get new neighbour scheduling
            neighbour = clone(getNeighbour(currentSolution));

            // Get energy of solutions
            double ms1 = makespane(currentSolution);
            double c1 = cost(currentSolution);

            double ms2 = makespane(neighbour);
            double c2 = cost(neighbour);

            // Decide if we should accept the neighbour
            if (acceptProb(c2, ms2, c1, ms1, temp) > Math.random()) {
                currentSolution = clone(neighbour);
                ms1 = makespane(currentSolution);
                c1 = cost(currentSolution);
            }
            // Keep track of the best solution found
            if (dominate(c1, ms1, cost(best), makespane(best))) {
                best = clone(currentSolution);
            }

            // Cool system
            temp *= 1 - coolingRate;
```

Figure 19: Part of SA Broker code.



```
        int stp = 0;

        bestPob = Selection(parents, 1);
        bestSolution = clone(bestPob.individuals.get(0));
//          System.out.println("init best = " + makespane(bestSolution));

        // selection
        pareto = Selection(parents, peretoSize);

        while (numGen > 0 && stp <= 25) {
            // crosover the parents
            childrens = crossover(pareto);
            // muatate individial
            childrens = mutation(childrens);
            // elitisme and replacement
            pareto = elitism(childrens, pareto);
            // keep tracking best solution
            bestPob = Selection(pareto, 1);

            if (accept(
                    cost(bestPob.individuals.get(0)),
                    makespane(bestPob.individuals.get(0)),
                    cost(bestSolution),
                    makespane(bestSolution))) {

                bestSolution = clone(bestPob.individuals.get(0));
                stp = 0;
            }
            stp++;
            numGen--;
        }
```

Figure 20: Part of GA Broker code.

4.2. Experimental results

After FSA, GA and FIFO test executions with the same parameter settings according to the fog, the best solutions found (task scheduling) in terms of makespan and cost are listed in table 3.4 and 5. The listed values choose by doing three simulations for both FSA and GA with same parameter settings, then to choose best cost and makespan from the obtained results of the three simulations, where:

- ❖ Case of 100 Tasks:

The values obtained by FIFO was

- ✓ Makespan = 21.0, cost = 1243.8269224003548

The results obtained by GA was as follows

- ✓ Makespan = 3.8384615384615386, cost = 314.04876487547534
- ✓ Makespan = 3.88, cost = 512.1730780957951
- ✓ Makespan = 4.271428571428572, cost = 324.5282098551603

The results obtained by SA was as follows

- ✓ Makespan = 3.5300000000000002, cost = 290.47534185628604
- ✓ Makespan = 3.5555555555555554, cost = 289.7361589068825
- ✓ Makespan = 3.6357142857142857, cost = 437.8327442230113

- ❖ Case of 500 Tasks:

The values obtained by FIFO was

- ✓ Makespan = 605.0, cost = 33995.4010120104

The results obtained by GA was as follows

- ✓ Makespan = 94.15, cost = 13505.873315011859
- ✓ Makespan = 124.76666666666667, cost = 16121.917173154507
- ✓ Makespan = 104.07499999999997, cost = 15024.74265722539

The results obtained by SA was as follows

- ✓ Makespan = 91.4625, cost = 13097.201396310082

- ✓ Makespan = 78.57, cost = 11661.966306912902
- ✓ Makespan = 87.86666666666669, cost = 10710.879149247292

❖ Case of 1000 Tasks:

The values obtained by FIFO was

- ✓ Makespan = 2460.0 , cost = 137431.4680240429

The results obtained by GA was as follows

- ✓ Makespan = 493.39999999999986, cost = 67926.77202872078
- ✓ Makespan = 642.7500000000001, cost = 75726.93659198248
- ✓ Makespan = 487.80000000000007, cost = 69082.59664295963

The results obtained by SA was as follows

- ✓ Makespan = 358.97999999999996 , cost = 52045.818634460215
- ✓ Makespan = 367.0 , cost = 53992.39871310453
- ✓ Makespan = 333.025 , cost = 56932.20563399832

Table 3: TASK SCHEDULING OF 100 TASKS USING FSA, GA AND FIFO

Method	Makespan (ms)	Cost ($)
FIFO	21.0	1243.82
Genetic Algorithm	3.83	314.05
Fog Simulated Annealing Algorithm	**3.53**	**289.74**

Table 4: TASK SCHEDULING OF 500 TASKS USING FSA, GA AND FIFO

Method	Makespan (ms)	Cost ($)
FIFO	605.0	33995.4
Genetic Algorithm	94.15	13505.87
Fog Simulated Annealing Algorithm	**78.57**	**10710.87**

Table 5: TASK SCHEDULING OF 1000 TASKS USING FSA, GA AND FIFO

Method	Makespan (ms)	Cost ($)
FIFO	2460.0	137431.46
Genetic Algorithm	487.8	67926.77
Fog Simulated Annealing Algorithm	**333.025**	**52045.81**

Tables 3-5 showed that the best fitness is the fitness obtained by FSA compared with the fitness given by the GA and FIFO. It represents the best execution time and the best cost of all the tasks). These results confirm the reliability and efficiency of FSA to solve task scheduling problem in the fog by an optimal workload balancing.

This superiority is due to the acceptance probability operator, which guarantees the diversity of the solution and ensures the local optima escape. It is also has low memory requirements. FSA takes less CPU time than the genetic algorithm (GA) when used to solve optimization problems, because it finds the optimal solution using point-by-point iteration rather than a search over a population of individuals.

4.3. Conclusion

In this chapter, we have presented all the steps in the design of our project with all the tools, programming languages and platforms used. Furthermore, the obtained results discussed, and proved the efficiency of our proposal FSA algorithm in scheduling tasks in fog computing environments, also showed its superiority against FIFO and GA in term of makespan and monetary cost.

General conclusion

In our Master memory, task scheduling problem in the fog computing is studied and solved with a novel metaheuristic called Fog Simulated Annealing algorithm (FSA). In order to prove the reliability and the efficiency of this proposal, an implementation and a set of tests of the FSA have been carried out and compared against (First In First Out) and (Genetic Algorithm) in iFogSim simulator. The obtained results of FSA proved the efficiency and the performance of the proposed algorithm against FIFO and GA in terms of cost and execution time. This superiority due to the advantage of the Simulated Annealing Algorithm escaping local optimal solution to find better solutions (global optimum).

As future research, there are a number of directions, which can enhance our algorithm performance in the context of improving the QoS on task scheduling, we highlight:

Failures of Fog devices management: Future research can focus on extracting the failed devices. The developed algorithm can used to evaluate and compare the fog devices and designing recovery and resuming policies for a wide range of applications.

Power-Aware resource management: One of the biggest challenges that most of Fog computing solutions face is how to get extra bit of battery life for Fog devices. Future studies can look into developing the algorithm to schedule tasks dynamically and based on the battery life of devices.

References

References

Al Nuaimi Klaithem [et al.] A Survey of Load Balancing in Cloud Computing: Challenges and Algorithms [Journal]. - [s.l.] : IEEE, 2012.

Armbrust Michael [et al.] A view of cloud computing [Journal]. - 2010.

Bitam Salim, Zeadally Sherali and Mellouk Abdelhamid Fog computing job scheduling optimization based on bees swarm [Journal] // IEEE. - 2017.

Blythe Jim [et al.] Task Scheduling Strategies for Workflow-based Applications in Grids [Journal]. - [s.l.] : IEEE, 2005.

Bonomi Flavio [et al.] Fog Computing and Its Role in the Internet of Things [Journal] // CISCO. - 2012.

Bonomi Flavio [et al.] Fog Computing and Its Role in the Internet of Things [Article]. - 2012.

Bouleimen K Lecocq H A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version [Journal]. - [s.l.] : Elsevier, 2003.

Busetti Franco Simulated annealing overview [Journal] // Citeseer. - [s.l.] : Citeseer, 2003.

Dekkers Anton and Aartsl Emile GLOBAL OPTIMIZATION AND SIMULATED ANNEALING [Journal]. - [s.l.] : Springer, 1988.

Goffe William L, D Ferrier Gary and Rogers John Global optimization of statistical functions with simulated annealing [Journal]. - [s.l.] : Elsevier, 1994.

Gupta Harshit [et al.] iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments [Journal]. - 2016.

Hao Zijiang [et al.] Challenges and Software Architecture for Fog Computing [Journal]. - [s.l.] : IEEE, 2017.

References

JOHNSON S DAVID [et al.] OPTIMIZATION BY SIMULATED ANNEALING: AN EXPERIMENTAL EVALUATION; PART II, GRAPH COLORING AND NUMBER PARTITIONING [Journal]. - [s.l.] : pubsonline, 1991.

Kirkpatrick S, Gelatt C D and Vecchi M P Optimization by Simulated Annealing [Journal]. - [s.l.] : sciencemag, 1983.

KRAEMER FRANK ALEXANDER [et al.] Fog Computing in Healthcare-A Review and Discussion [Journal]. - [s.l.] : IEEE, 2017.

Labed Ayoub, Bitam Salim and Mellouk Abdelhamid FSA: Simulated Annealing Algorithm for Task Scheduling in Fog Computing [Journal]. - [s.l.] : submitted to the 24th International European Conference on Parallel and Distributed Computing (Euro-Par 2018), Workshop on Fog-to-Cloud Distributed Processing (F2C-DP), 27-31 august 2018, Turin, Italy, (http://www.mf2c-project.eu/europar-2018-mf2c-worksho, 2018.

Liu Xi and Liu Jun A Task Scheduling Based on Simulated Annealing Algorithm in Cloud Computing [Journal]. - 2010.

mansouri khalil L’Ordonnancement des Tâches dans le Cloud Computing par une Approche d’Optimisation Parallèle [Report]. - 2013.

Mouradian Carla [et al.] A Comprehensive Survey on Fog Computing: State-of-the-art and Research Challenges [Journal]. - 2017.

Ningning SONG [et al.] “Fog Computing Dynamic Load Balancing Mechanism Based on Graph Repartitioning [Journal]. - 2016.

Nisha Peter FOG Computing and Its Real Time Applications [Journal] // IJETAE. - 2015.

Nisha Peter FOG Computing and Its Real Time Applications [Article]. - 2015.

Pham Xuan-Qui [et al.] A cost- and performance-effective approach for task scheduling based on collaboration between cloud and fog computing [Journal]. - 2017.

References

Pham Xuan-Qui and Huh Eui-Nam Towards task scheduling in a cloud-fog computing system [Journal]. - [s.l.] : IEEE, 2016.

Puliafito Carlo, Mingozzi Enzo and Anastasi Giuseppe Fog Computing for the Internet of Mobile Things: issues and challenges [Journal]. - [s.l.] : IEEE, 2013.

Rahbari Dadmehr and Nickray Mohsen Scheduling of Fog Networks with Optimized Knapsack by Symbiotic Organisms Search [Journal]. - [s.l.] : fruct.

Rao Venkata Teaching Learning Based Optimization Algorithm And Its Engineering Applications [Book]. - 2015.

Salot Pinal A SURVEY OF VARIOUS SCHEDULING ALGORITHM IN CLOUD COMPUTING ENVIRONMENT [Journal]. - [s.l.] : IJREI, 2013.

Stojmenovic Ivan and Wen Sheng The Fog Computing Paradigm: Scenarios and Security Issues [Journal] // ACSIS. - 2014.

Stojmenovic Ivan and Wen Sheng The Fog Computing Paradigm: Scenarios and Security Issues [Journal]. - 2014.

Suman B and Kumar P A survey of simulated annealing as a tool for single and multiobjective optimization [Journal]. - 2006.

Vaquero M Luis and Rodero-Merino Luis Finding your Way in the Fog: Towards a Comprehensive Definition of Fog computing [Article]. - 2014.

Verma Manisha, Bhardwaj Neelam and Yadav Arun Kumar Real Time Efficient Scheduling Algorithm for Load Balancing in Fog Computing Environment [Journal]. - [s.l.] : MECS, 2016.

Yi Shanhe, Li Cheng and Li Qun A Survey of Fog Computing: Concepts, Applications and Issues [Journal]. - 2015.