Artificial Intelligence Models for Scheduling Big Data Services on the Cloud

Gaith Rjoub

A Thesis

In

Concordia Institute for Information Systems Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy (Information & Systems Engineering) at

Concordia University

Montréal, Québec, Canada

October 2021

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Gaith Rjoub**

Entitled: **Artificial Intelligence Models for Scheduling Big Data Services on the Cloud**

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Information Systems Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Dr. Irfan-Ullah Awan

_____ Dr. Juergen Rilling

_____ Dr. Roch Glitho

_____ Dr. Farnoosh Naderkhani

_____ Dr. Jamal Bentahar

_____ Dr. Omar Abdul Wahab

Approved by _____
Dr. Mohammad Mannan Graduate Program Director

October 15, 2021
Date of Defence

_____
Dr. Mourad Debbabi, Gina Cody School of Engineering and Computer Science

ABSTRACT

Artificial Intelligence Models for Scheduling Big Data Services on the Cloud

Gaith Rjoub, Ph.D.

Concordia University, 2021

The widespread adoption of Internet of Things (IoT) applications in many critical sectors (e.g., healthcare, unmanned autonomous systems, etc.) and the huge volumes of data that are being generated from such applications have led to an unprecedented reliance on the cloud computing platform to store and process these data. Moreover, cloud providers tend to receive massive waves of demands on their storage and computing resources. To help providers deal with such demands without sacrificing performance, the concept of cloud automation had recently arisen to improve the performance and reduce the manual efforts related to the management of cloud computing workloads.

However, several challenges have to be taken into consideration in order to guarantee an optimal performance for big data storage and analytics in cloud computing environments. In this context, we propose in this thesis a smart scheduling model as an automated big data task scheduling approach in cloud computing environments. Our scheduling model combines Deep Reinforcement Learning (DRL), Federated Learning (FL), and Transfer Learning (TL) to automatically predict the IoT devices to which each incoming big data task should be scheduled to as to improve the performance and reduce the execution cost. Furthermore, we solve the long execution

time and data shortage problems by introducing a FL-based solution that also ensures privacy-preserving and reduces training and data complexity.

The motivation of this thesis stems from four main observations/research gaps that we have drawn through our literature reviews and/or experiments, which are: (1) most of the existing cloud-based scheduling solutions consider the scheduling problem only from the tasks priority viewpoint, which leads to increase the amounts of wasted resources in case of malicious or compromised IoT devices; (2) the existing scheduling solutions in the domain of cloud and edge computing are still ineffective in making real-time decisions concerning the resource allocation and management in cloud systems; (3) it is quite difficult to schedule tasks or learning models from servers in areas that are far from the objects and IoT devices, which entails significant delay and response time for the process of transmitting data; and (4) none of the existing scheduling solutions has yet addressed the issue of dynamic task scheduling automation in complex and large-scale edge computing settings.

In this thesis, we address the scheduling challenges related to the cloud and edge computing environment. To this end, we argue that trust should be an integral part of the decision-making process and therefore design a trust establishment mechanism between the edge server and IoT devices. The trust mechanism model aims to detect those IoT devices that over-utilize or under-utilize their resources. Thereafter, we design a smart scheduling algorithm to automate the process of scheduling large-scale workloads onto edge cloud computing resources while taking into account the trust scores, task waiting time, and energy levels of the IoT devices to make appropriate scheduling decisions. Finally, we apply our scheduling strategy in the healthcare domain to investigate its applicability in a real-world scenario (COVID-19).

ACKNOWLEDGEMENTS

First and foremost, praises and thanks to the God, the Almighty, for His showers of blessings throughout my research work and granting me the health, ability, and patience to complete this thesis.

I would like to express my deep and sincere gratitude to my Ph.D. supervisors, Dr. Jamal Bentahar and Dr. Omar Abdul Wahab for giving me the opportunity to do research and providing invaluable guidance throughout this research. Their dynamism, vision, sincerity and motivation have deeply inspired me. They have taught me the methodology to carry out the research and to present the research works as clearly as possible. It was a great privilege and honor to work and study under their guidance. I am extremely grateful for what they have offered me.

Moreover, I would like to thank my Ph.D. committee members: Dr. Irfan-Ullah Awan, Dr. Juergen Rilling, Dr. Farnoosh Naderkhani, and Dr. Roch Glitho for their valuable time and effort in reviewing my thesis and providing me with insightful comments and recommendations. Your deep knowledge and expertise have made every encounter a great opportunity to discuss new ideas and enhance the quality of the research outcomes.

Furthermore, I would like to thank all my colleagues in the research lab at Concordia University especially Ahmad Bataineh, Dr. Mona Taghavi, and Dr. Nagat Drawel for providing me with a warm and friendly atmosphere to work in. I would like also to thank my non-concordian friends with whom I shared precious and enjoyable moments. Your presence has allowed me to appreciate my stay in Montreal and added lots of fun to my Ph.D. journey.

This research would not have been possible without the financial assistance of the Natural Sciences and Engineering Research Council of Canada (NSERC), The

Department of National Defence (DND), Fonds de Recherche du Québec - Nature et Technologies (FRQNT) and Concordia University. This support was very important for me to alleviate the financial burdens and focus on my research duties.

Last but not least, I would like to thank my parents for their endless and unconditional love and support. Their guidance has been always important for me to overcome the difficulties of life. Without my mother, father, wife, and my sons, Adam and Tim, I could not have been able to succeed throughout my life and become the person I am today.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

In this chapter, we introduce the context of our research work, highlight the problems tackled in this thesis, pose the corresponding research questions, and identify the objectives and contributions of our research work.

1.1 Context of Research

The growing adoption of Internet of Things (IoT) in many applications (e.g., intelligent transportation systems, healthcare management, etc.) has led to the generation of unprecedented amounts of data on a daily basis. This raises serious challenges concerning the storage and processing of such huge amounts of data. To deal with these challenges, cloud computing has been the imminent choice for most IoT manufacturers and vendors to provide scalable storage and processing of their big data. Most companies prefer to migrate the big data storage and analytics responsibilities to the cloud instead of having to purchase and maintain expensive hardware equipment on their own. More specifically, the concept of virtualization, which allows the sharing of computing and storage resources (e.g., CPU, RAM, disk

storage, bandwidth) among several users enables cloud systems to support massive simultaneous storage and computing requests. This, on the other hand, entails a serious problem for cloud administrators, namely, how to efficiently schedule the big data tasks in the virtualized environments in such a way to guarantee optimal performance and minimal resource wastage. Cloud computing has been widely used for most of the businesses seeking to keep up with the big data evolution trends, thanks to the wide variety of advantages it offers such as multi-tenancy, elasticity, and virtualization.

In cloud computing, resources, either software or hardware, are virtualized and allocated as services from providers to users. The computing resources can be allocated dynamically upon the requirements and preferences of consumers, where the resources are located in different regions and have various processing abilities, characteristics (number of CPU cores, amount of main memory, etc.), and cost [110]. The process of allocating services to perform a set of tasks while satisfying constraints in terms of time, cost, Quality of Service (QoS), and service availability is known as task scheduling [65]. Task scheduling and resource allocation should be carefully arranged and optimized simultaneously in order to attain an overall cost and time-effective schedule. Specifically, the process of scheduling big data analytics tasks in cloud computing environments is quite challenging since it involves the optimization of several (sometimes conflicting) objectives. On the one hand, guaranteeing minimal timespan for big data tasks is crucial especially when it comes to delay-critical applications (e.g., healthcare management, intelligent transportation systems) wherein small delays might cause loss of life. Moreover, as the number of needed big tasks is quite large, minimizing the cost of each single task is a major concern for these data sources (e.g., IoT manufacturer). From the cloud providers' site, managing the available resources so as to increase the capacity of

simultaneously receiving the largest possible number of tasks is a major concern. That is, cloud providers should schedule the big data tasks in such a way to guarantee a minimal CPU, RAM, bandwidth, and disk storage consumption on each tasks, without sacrificing the overall performance. The concept of cloud automation, which takes advantage of the latest advancements in artificial intelligence, had recently arisen to improve the performance and reduce the manual efforts associated with the provisioning and management of cloud computing workloads [99]. It involves designing orchestration automation tools and algorithms that run on top of virtualized environment to make real-time decisions concerning the resource allocation and management in cloud systems. In this thesis, we provide a novel contribution toward the concept of cloud automation by proposing an automated big data task scheduling approach in cloud and edge computing environments using various approaches of machine learning, namely, Deep Learning (DL), Deep Reinforcement Learning (DRL), Federated Learning (FL), and Transfer Learning (TL).

1.2 Research Questions

Several approaches have been proposed to improve the task scheduling process in cloud computing environments. The main idea of these approaches is to reduce the makespan by trying to reduce the waiting time of the tasks in the queues and attempting to map tasks to the nearest VMs to reduce the transfer time (i.e., the data locality concept). However, this does not always guarantee a better performance in a dynamic and open environment like cloud computing, which leads us to our first research question (RQ 1):

- ***RQ 1: How to jointly minimize the makespan of big data tasks, while minimizing at the same time the monetary cost of their execution?***

With the large numbers of deployed VMs in cloud-based systems, the chances of encountering untrusted or poorly-performing VMs is fairly high. Such VMs would not only cause the makespan to be high, but also increase the amounts of wasted resources in case of malicious or compromised VMs. Most of the existing cloud-based scheduling solutions consider the scheduling problem only from the viewpoint of the tasks by setting a priority degree for each task to determine the order in which it will be scheduled and executed. Nevertheless, by focusing only on the task priorities and ignoring the reliability of the VMs, there would still be a large risk to the overall QoS in the presence of untrusted VMs. For example, a high-priority task might be assigned to the nearest VM to minimize the transfer time. However, such a VM might undergo some failure, thus causing a huge spike in both the total makespan and monetary cost. Therefore, a more intelligent approach which takes into account all these factors is needed. Thus, our second research question (RQ 2) is:

- ***RQ 2: How to insure the trustworthness of the VMs that will be executing the big data tasks?***

Growing adoption of Internet of Things (IoT) in many applications (e.g., intelligent transportation systems, healthcare management, etc.) has lead to the generation of unprecedented amounts of data on a daily basis. The concept of cloud automation, which takes advantage of the latest advancements in artificial intelligence, had recently arisen to improve the performance and reduce the manual efforts associated with the provisioning and management of cloud computing workloads [99]. More specifically, making real-time decisions concerning resources scheduling and management in cloud systems requires developing an automation tool and algorithms that run on top of the virtualized environment. Therefore, our third research question (RQ 3) is:

- ***RQ 3: How can we benefit from the cloud automation concept to automate the process of scheduling big data tasks in cloud-based environments, so as to improve the performance and reduce monetary and resource costs?***

The centralized architecture of cloud computing hinders its adoption in environments wherein the generated data needs to be analyzed in real-time in order to make prompt decisions. Specifically, the fact that cloud datacenters reside in areas that are far from the objects and devices (e.g., smart cars, radars, etc.) that generate the data entails significant delay and response time for the process of transmitting data to/from cloud datacenters. The concepts of edge computing and federated learning had recently been proposed to tackle this problem and complement cloud computing so as to improve the response time of the big data analytics process, tackle the issue of long training period, limited data accessibility, and privacy preservation. Therefore, our fourth research question (RQ 4) is:

- ***RQ 4: How to adapt the automated big data task scheduling approach to work in distributed environments using the concepts of edge computing and federated learning?***

1.3 Research Objectives and Contributions

The ultimate goal of our research work is to propose an automated big data task scheduling approach in cloud and edge computing environments that operates in a distributed fashion to reduce the makespan and monetary costs of the analytics process. The following objectives are identified to attain this goal:

- **Objective** 1**:** Propose a scheduling approach to optimize the performance of big data services execution. This approach should minimize the hardware and network resources cost while improving the response time delivered to customers.

- **Objective** 2**:** Integrate trust in the scheduling approach for big data tasks in cloud computing environments to avoid assigning these tasks to poorly performing VMs, thus degrading the overall performance of the analytics process. The trust-aware scheduling approach should consider the following stages: (1) VMs' trust level computation; (2) tasks priority levels determination; and (3) trust-aware scheduling.

- **Objective** 3**:** Integrate a Long Short-Term Memory (LSTM) layer into the design of the scheduling model to keep track of the historical long-term dependencies that exist between the resource requirements of the tasks and the resource specifications of the VMs and their impact on the resulting execution cost.

- **Objective** 4**:** Elaborate a distributed and automated big data scheduling model in the edge computing environments. The objective is to reduce the response time of the big data analytics process in delay-critical applications.

In order to attain these objectives, the following contributions are offered by this thesis:

- **Contribution** 1: We designed a hybrid approach, called Multi Label Classifier Chains Swarm Intelligence (MLCCSI) to guide the cloud choose the scheduling technique by using multi criteria decision to optimize the performance. This approach is based on two strategies. The first strategy is the swarm intelligence, which we applied on the Ant Colony Optimization (ACO) algorithm, Artificial

Bee Colony (ABC) algorithm and, Particle Swarm Optimization (PSO) algorithm to find the optimal resource allocation for each task in the dynamic cloud system. Then, the second strategy is the application of the machine learning algorithm (Classifier Chains) on the results from the three algorithms, which generates a new hybrid model considering the size of the tasks and the number of the virtual machines. This strategy not only minimizes the makespan of a given tasks set, but it also adapts to the dynamic cloud computing system and balances the entire system load. This contribution is discussed in Chapter 3.

- **Contribution** 2: We developed a trust-aware scheduling mechanism to increase the performance of big data services execution, which leads to minimize the makespan and the execution cost. This will be achieved by terminating the untrusted VMs and assigning high priority tasks to the most trusted VMs. The contribution is discussed in Chapter 4.

- **Contribution** 3: We introduced four deep and reinforcement learning-based scheduling approaches to automate the process of scheduling large-scale workloads. We then compared the performance of these approaches and identified the best one in terms of minimizing the task execution cost and waiting time. These approaches are: reinforcement learning (RL), deep Q networks (DQN), recurrent neural networks that use long short-term memory (RNN-LSTM), and deep reinforcement learning combined with LSTM (DRL-LSTM). This contribution is discussed in Chapter 5.

- **Contribution** 4: We introduced DDQN-Trust, a trust establishment technique for IoT devices. DDQN-Trust is an algorithm that enables the edge servers to find the optimal scheduling decisions in terms of energy efficiency and

trustworthiness. In particular, we first formulate a stochastic optimization problem that seeks to derive a set of IoT devices that, by sending the federated learning tasks to them, the edge server can maximize the trust and minimize the energy cost. The algorithm is designed to solve the optimization problem while modeling the uncertainty that the server faces regarding the resource and trust levels of the IoT devices. This contribution is discussed in Chapter 6.

- **Contribution** 5: In order to conduct a real-world application study of our scheduling approach, we applied our solution in the healthcare domain to investigate scalability and applicability in real-world scenarios. We introduced a comprehensive solution named COVID-FED. COVID-FED is a multi-faceted COVID-19 detection approach which incorporates federated learning, trust management, and deep reinforcement learning (DRL) in an edge computing setting that considers IoT devices and medical imaging. We capitalize on federated transfer learning over IoT and edge devices for dynamic detection of COVID-19 from X-ray images. Edge servers collaborate with IoT devices to train the COVID-19 detection model using federated learning without exchanging raw confidential data. Transfer learning is important to handle the scarcity of data in some regions and compensate potential lack of learning at some servers. DRL and trust management are combined to assign the COVID-19 detection tasks to the most trusted and resource-efficient IoT devices. The contribution is discussed in Chapter 7.

It is worth noting that the content of this thesis has been published in [86–90].

1.4 Thesis Organization

We present in Chapter 2 the background needed to understand the different concepts and techniques we are using in this thesis. In particular, we give an overview of the main techniques that contributed in task scheduling, namely deep learning, cloud computing, cloud federation, edge computing, trust, and big data. Then, we give a detailed tutorial on the concept of scheduling model and explain its main categories, mathematical foundations, and illustrative examples. Afterwards, we provide literature reviews on the scheduling models proposed for tasks over edge and cloud computing, the main approaches that used deep learning and federated learning to solve big data and large scale problems, and the major task allocation systems proposed in the domain of edge and cloud computing.

In Chapter 3, we advance a machine learning algorithm to guide the cloud choose the scheduling technique by using a multi criteria decision to optimize the performance. The algorithm contributes in minimizing the makespan of a given task set. The new strategy is simulated using the CloudSim toolkit package where the impact of the algorithm is checked with different numbers of VMs and various task sizes.

In Chapter 4, we tackle the problem of task scheduling process in cloud environments in the presence of untrusted or poorly-performing VMs. In particular, we put forward a comprehensive trust-aware scheduling solution called BigTrustScheduling that consists of three stages: VMs' trust level computation, tasks priority level determination, and trust-aware scheduling.

In Chapter 5, we discuss the cloud automation to reduce the manual intervention and improve the resource management in large-scale cloud computing workloads. We capitalize on this concept and propose four deep and reinforcement learning-based scheduling approaches to automate the process of scheduling large-scale cloud

computing workloads, while reducing both the resource consumption and task waiting time.

In Chapter 6, we discuss a comprehensive detection for those IoT devices that do not dedicate enough resources to serve the federated learning tasks as well as those that carry out additional computations to achieve some malicious objectives. In particular, we put forward DDQN-Trust, a federated learning approach that uses a Double Deep Q Learning-based selection algorithm. DDQN-Trust takes into account the trust scores and energy levels of the IoT devices to make appropriate scheduling decisions.

In Chapter 7, we conduct a real-world application study to explore practicality in the healthcare sector and see how our scheduling model works in a real world scenario. In particular, we discuss the multi-faceted model and highlight its healthcare impact on COVID-19 detection by means of a practical example that exploits emerging technologies of IoT and edge computing.

Finally, in Chapter 8, we summarize the thesis contributions and shed light on some research gaps that need further consideration by the research community.

Chapter 2

Research Background

In this chapter, we explain the main concepts that are needed to understand the core of the thesis and present a profound and systematic literature review on the different aspects that our thesis aims to address. In Section 2.1, we explain the main concepts related to the task scheduling, namely those of big data (Section 2.1.1), cloud computing (Section 2.1.2), edge computing (Section 2.1.3), and Internet of things (Section 2.1.4). We define as well the notion of federated learning (Section 2.1.5).

In Section 2.2, we review the relevant related work. Section 2.2.1 is dedicated to discussing the main traditional scheduling models proposed for cloud computing environments. Section 2.2.2 surveys the main task scheduling approaches in IoT and edge computing environments. We give in Section 2.2.3 an overview of the main trust models proposed in the domain of cloud computing and discuss their principal shortcomings. In Section 2.2.4, we give a systematic survey on the artificial intelligence-based scheduling models proposed in the cloud environment.

2.1 Background

In this section, we explain the main concepts related to the task scheduling, namely those of cloud computing, big data, and federated learning.

2.1.1 Big Data

The overwhelming flow of data in both structured and unstructured formats, and the continuous increase in the volume and detail of data captured by organizations in many sectors such as health-care, science, engineering, finance, and business have created a major challenge for manufacturers (e.g., IoT manufacturers). This is mainly due to the fact that the data growth rate exceeds the ability to design effective storage and analytics solutions. The big data technology is receiving unprecedented attention from both academia and industry due to its importance in enabling the extraction of valuable insights that can be turned into concrete business values. Big data are characterized by the following main features [44]: (a) big data are numerous; (b) big data cannot be categorized into regular relational databases; and finally (c) data are generated, captured, and processed rapidly. Big data can be defined with the following properties as shows in Fig. 2.1:

A Variety

The evolutions in technology allows firms to use and generate various types of structured, semi-structured, and unstructured data. Structured data represent the tabular data found in spreadsheets or relational databases refer to the structured data, which account only for 5% of all existing data [33]. Images, audio, and video are examples of unstructured data, which are generated by many organizations and devices (e.g., devices, unmanned vehicles, etc.). Extensible Markup Language (XML), a textual language for exchanging data

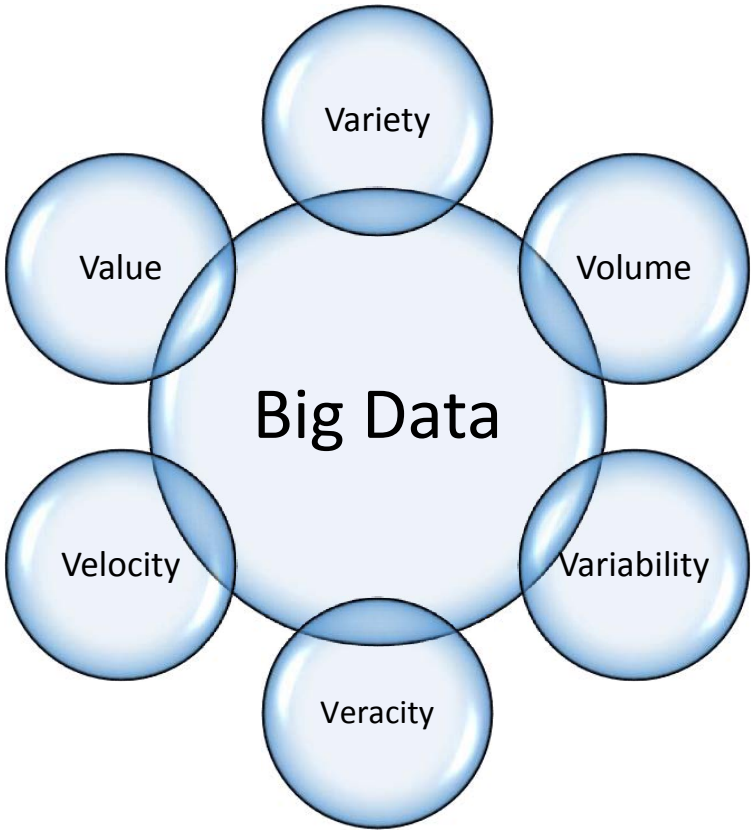


Figure 2.1: Big data V's properties

on the Web, is a typical example of semi-structured data.

B Volume

A survey conducted by IBM in mid-2012 revealed that just over half of the 1144 respondents considered datasets over one terabyte to be big data [94], where one terabyte fit on 1500 CDs or 220 DVDs, enough to store around 16 million Facebook photograph. The problem withs such volumes of data stems from the impossibility of relying on traditional storage systems to store such amounts of data. This necessitates thinking of modern distributed approaches that can host these data in a reliable fashion.

C Velocity

Velocity refers to the speed at which vast amounts of data are being generated, collected and analyzed. The proliferation of digital devices such as smart-phones and sensors has led to the increase in the amounts of data and the speed at which they need to be stored and analyzed [120]. For example, some applications (e.g., credit card fraud detection) demand real-time data analytics in order to prevent undesirable consequences.

D Variability

Variability points to the variation in the data flow rates and mostly focuses on properly understanding and interpreting the correct meanings of raw data which depends on its context.

E Veracity

Veracity refers to what extent can data be trusted and the reliability inherent in some sources of data. Therefore, the need to deal with imprecise and mysterious data is another side of big data, which is directed utilizing tools and analytics sophisticated for management and mining of uncertain data [34].

F Value

The organizations need to get some sort of value after the massive efforts and resources spent on the above properties.Thus, value refers to the process of discovering massive hidden values from large datasets with diverse types and prompt generation [44].

2.1.2 Cloud Computing

Cloud computing is facing abnormal growth in the incoming demands due to a new class of traffic from IoT devices such as smart wearables and autonomous

cars. Cloud computing resources are offered as services via the Internet as a layered architecture that consists of five main layers [61, 85] as shows in Fig. 2.2: Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS), Communications as a Service (CaaS), and Data Storage as a Service (DaaS). SaaS providers use the Internet to deliver software to the end user on-demand. PaaS provides a platform and environment to allow developers to build applications and services over the Internet such as operating systems, Web servers, databases, and programming languages' execution environments. IaaS provides virtualized computing resources over the Internet. CaaS enables users to directly communicate with each other on the cloud the cloud via E-mails, Texts, Voice and Video Calls, chat box etc. DaaS is a service model in which data is maintained, managed, backed up remotely and made available to users over a network.

2.1.3 Edge Computing

The adoption of edge computing is a recent idea in the computing world. A cloud-centric service connects end users to the benefits of cloud storage and technologies, with service and device processing responsiveness taking place on demand. These technologies, such as tracking, augmented reality and real-time traffic monitoring that are now built on the Internet, need rapid processing and quick response time. Typically, the applications are executed on a resource-challenged mobile devices while cloud services handle the core service and processing.

Using cloud computing tools for mobility may be problematic because of their large latency and mobility requirements. edge computing provides these criteria by removing computing from the central data center and location in the network, to locations near where the data is accessed and processed. In general, there are three forms of Edge computing models which address computing problems (i.e. Fog

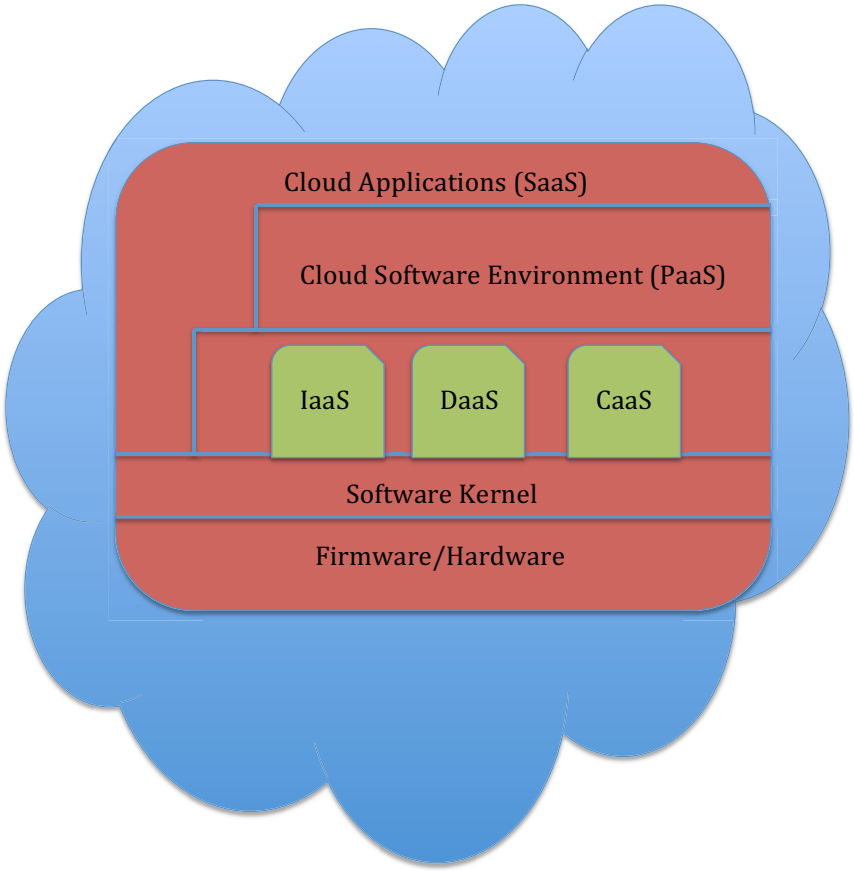


Figure 2.2: The five layers of cloud computing

computing [5], Cloudlets [98], and Mobile Edge computing [54]).

2.1.4 Internet of Thing

The Internet of Things (IoT) is now pervasive in our everyday lives, offering critical measurement and data gathering resources that help us make better decisions. There are literally millions of sensors and devices are generating and transmitting vast amounts of data as well as sending critical messages through intricate networks and tracking critical machines. As a countermeasure to increased resource use, edge

computing has taken on a new role as a solution for the Internet of Things and the premise of local computing.

The structure of the IoT consists of four layers: service, platform, network, and device layer. As the classification shown in Fig. 2.3, several research institutions choose to use a single system architecture for IoT production.

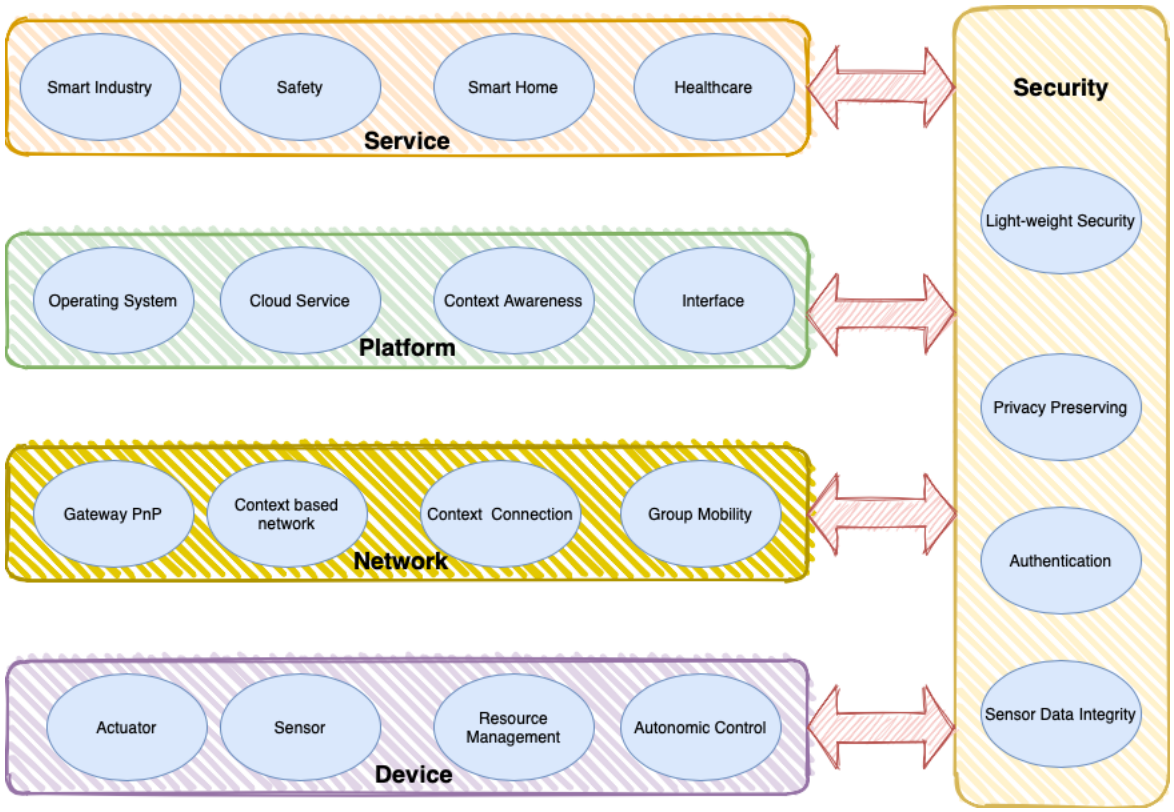


Figure 2.3: The four layers of IoT structure

The interface and communication with users are provided by the service layer. The service layer, for example, includes automated vehicles, health care, wearable technologies, and home-security systems [57]. These services are linked with a platform layer to provide customized services to the users. The platform layer is the next layer in the IoT framework. This layer is situated under the service layer

and serves IoT applications and utilities. There are several kinds of platforms, such as device platform, data analysis platform, service development platform, and service platform. For example, the device platform offers a services execution environment and development for users. Examples of data analysis platforms include context awareness and prediction, collaboration among things, and link between the service layer and other layers through natural language to machine language translation. It is essential to provide software toolkits for users to build IoT services and this provides by the service development platform. To sum up, the service's functional capabilities, it also facilitates generation and implementation of a wide range of applications.

In addition to service and platform layers, the network layer is a core layer in the IoT setting. It transmits data between devices, contents, services and users. The layer of the network should be able to store, monitor, and handle huge quantities of network traffic. The device layer acts as a sender, responsible for gathering and sending environmental data to the sink node, or gateway, and, as well, as a layer that receives that data. It must be self-adapting in order to autonomously implement this actuation [45].

2.1.5 Federated Learning

Federated Learning (FL) is one of the highly effective paradigm shifts in recent years in AI-based computing. It allows us to obtain better results by cooperatively training a single machine learning model instead of forcing each edge machine to share its actual input data. The FL architecture consists of two phases, namely, global computing and local training. In the local training phase, a parameter server, such as an edge server, initializes the machine learning algorithm and shares the initial parameters with the end/edge devices (e.g., IoT devices). The shared parameters are thereafter used by these devices to train the model on their own data. Then, the devices share

the modified parameters acquired from the training of the model on their data with the parameter server. Global computing allows the whole model to be reconstructed by aggregating all the received parameter updates in coordination with all the IoT devices. This method is repeated until a certain degree of accuracy is achieved. The applications of FL in medical big data are quite promising [18, 115]. We provide in Section 2.2.4 of this chapter a detailed literature review about the main approaches that contributed in solving problems related to FL.

2.2 Literature Review and Discussions

In this section, we classify the existing task scheduling approaches in the domain of cloud computing into five categories, namely task scheduling, trust-based scheduling, artificial intelligence-based scheduling, resource management using deep learning, and other approaches.

2.2.1 Traditional Task Scheduling

To take benefits of the cloud computing resources availability and abundance, several scheduling approaches have been proposed [95, 102, 108]. A scheduling algorithm called Linear Scheduling for Tasks and Resources (LSTR) is designed in [1]. The algorithm focuses on the scheduling of the cloud resources among the task requesters, with the aim of improving the system throughput and resource utilization. Babu et al. [55] proposed a novel strategy for load balancing of tasks in cloud computing environments inspired by the behavior of honey bees, which helps to achieve even load balancing across virtual machine to maximize throughput. It considers the priority of task waiting in queue for execution in virtual machines. After the work load on VM has been calculated it decides whether the system is overloaded, under loaded,

or balanced. Based on this VMs are grouped. While load balancing is an important element of scheduling, the strategy proposed is meant for scheduling independent tasks and not dependent tasks work flows. In [16], the authors propose a batch-based (DAG) scheduling algorithm, which combines centralized static scheduling and dynamic workflow to improve resource utilization and achieve quasi-optimal throughput on heterogeneous platforms.

In [122], the authors propose a task scheduling algorithm considering game theory designed for energy management in cloud computing, they proposed a task scheduling model for computing nodes by establishing mathematical model to deal with big data. Magulur et al. [71] propose several virtual machine configuration scheduling algorithms for cloud computing platforms that reach an arbitrary fraction of cloud capacity. To do so, they employ a stochastic model of a cloud computing cluster in which tasks are performed based on a stochastic process. They introduce as well a set of policies for configuring non-preemptive frame-based virtual machines. In [123], by leveraging a game-theoretical model, the authors propose a task scheduling algorithm designed for energy management in cloud computing. They initiate a balanced task scheduling model for computing nodes by establishing a mathematical model to deal with big data. In [106], the authors propose a conceptual, user-centric game theoretical frame work that includes a two-stage game. The first stage to capture the user demand preferences, a Stackelberg game is used where IaaS providers are leaders and IaaS users are followers. On the other hand, the authors propose a differential game as a second stage to enhance the service ratings given by users in order to improve the provider position in the market and increase the future users' demand.

In [129], Zhang et al. propose a two-stage scheduling strategy to maximize task scheduling performance and minimize unreasonable task allocation in cloud centres.

In the first phase, a Bayes classifier is employed to cluster tasks according to historical scheduling data and correspondingly create suitable VMs. In the second phase, dynamic scheduling algorithms are proposed to match tasks with the created VMs. Lu and Gu [68] proposed using an Ant Colony Optimization algorthim (ACO) to find the closest free cloud resource rapidly and to share some load of the overload cloud resources adaptively. In their study, the approach does not schedule a set of VMs on different cloud resources but just execute the ACO process when some VMs are overloaded. The ACO process just acts like the ant to search for food to find the optimal physical resource and to create new VMs to share the load of the overloaded VMs.

Conclusive Remarks Despite their importance, the above-mentioned approaches overlook the trust levels of the VMs when performing the scheduling process. This might lead to assigning tasks to some poor-performing VMs, thus increasing the makespan of the tasks and causing resource wastage from the cloud providers' site.

2.2.2 Task Scheduling in IoT and Edge Computing Environments

Since our solution includes a scheduling component over IoT and edge devices, it is important to survey the relevant literature. In [58], the authors propose a semi-distributed joint computation and multi-user scheduling algorithm in Narrow-Band IoT and edge computing systems. The objective is to minimize the long-term average weighted sum of delay and power consumption over all the IoT devices under stochastic traffic arrival. Technically speaking, the authors consider the stochastic arrival model, and formulate the dynamic optimization problem into an infinite-horizon average-reward continuous-time Markov decision process (CTMDP)

model. To deal with the curse-of-dimensionality problem, they use approximate dynamic programming techniques including linear function approximation and semi-gradient TD learning. In [48], the authors study the request scheduling problem in ultra-dense edge computing (UDEC) networks and provide a non-cooperative game model based on sub-gradients. The considered UDEC network consists of a macro base station, many micro stations, and a large number of mobile users under the 5G architecture.

In [127], the authors propose a deep reinforcement learning approach to deploy the services in 5G-based mobile edge computing networks. They consider the request patterns and resource constraints of users to improve the number of services executed on the ESs and also to reduce the total response time. The problem is modeled as a Markov decision process and solved using the Dueling-Deep Q-network algorithm to learn the access patterns of a large number of requests on ESs. In [101], the authors address the host load prediction problem to improve the management and consumption of resources in cloud-based systems. In the light of the significant load variance in cloud environments, the challenge is to establish accurate predictions. A long short-term memory model (LSTM) is designed to predict the mean host load in consecutive future intervals to meet this challenge.

In [70], the authors propose a multi-cloud to multi-fog scheduling architecture. The proposed scheduling solution is based on transmission energy consumption of terminal devices and makes use of a dynamic threshold strategy to schedule requests in real-time, thereby guaranteeing the energy balancing of terminal devices without increasing the delay. To reinforce the load balancing and throughput of cloud networks, the authors of [72] suggest FMPSO, a hybrid task scheduling algorithm focused on the Fuzzy method and the Modified Particle Swarm Optimization technique. To improve the global search capability, FMPSO first considers four

updated velocity updating methods and a selection technique. Then, to solve any of the PSO's drawbacks, it employs cross-over and mutation operators used in genetic algorithms. Finally, the fitness values are computed using fuzzy inference in the proposed process.

In [132], the authors suggest a new algorithm called MGGS (modified genetic algorithm (GA) combined with greedy strategy). To improve the task scheduling process, the proposed algorithm uses a modified GA algorithm in combination with a greedy strategy.

Conclusive Remarks These methods primarily rely on evaluating past resource data from IoTs in order to forecast future workload and, as a result, enhance IoT management and allocation processes. Despite this, none of them have yet tackled the topic of automating work scheduling in dynamic and large-scale edge computing systems.

2.2.3 Trust-based Scheduling

In [131], the authors propose a scheduling model based on three factors, namely cost, time, and trust. The main objective is to ameliorate users' satisfaction using trust feedback data and risk cost estimation. Wang et al. [116] propose a Bayesian approach for a cognitive trust model which relies on direct and indirect trust sources to derive trust values for cloud resources. Thereafter, a trust-based dynamic level scheduling algorithm called Cloud-DLS is advanced to minimize time costs and ensure a secure execution of the tasks.

The authors of [103] focus on the challenge of achieving secure scheduling of users' requests. They propose a trust model which includes both direct trust and indirect reputation metrics. The paper focuses on assuring a trustworthy execution environment in the cloud. Based on the designed trust model, users' requests are

scheduled onto the appropriate resources using a trust-based stochastic scheduling algorithm.

In [124], the authors propose to integrate trust into workflow execution in cloud environments. They first investigate the trust relationship between users and cloud resources and then derive a trust-based directed acyclic graph (DAG) scheduling algorithm. In [26], the authors propose a trust-aware adaptive DAG tasks scheduling algorithm using the reinforcement learning and Monte Carlo Tree Search (MCTS) method. By employing the scheduling state space, action space and reward function, the authors design a reinforcement learning model to train the policy gradient-based reinforcement agent. Moreover, the MCTS method can determine actual scheduling policies when DAG tasks are simultaneously executed on trusted and untrusted entities.

Conclusive Remarks In summary, the existing trust-based scheduling approaches focus on improving the security of the cloud environment through relying on feedback collected from users. On the other hand, the existing approaches ignore the performance of the VMs, which leads to decreasing the QoS delivered to users.

2.2.4 Artificial Intelligence-based Scheduling

In [100], the authors address the problem of achieving a tradeoff between minimizing energy consumption in cloud centers, while minimizing at the same time the makespan of tasks. To do so, they design a multi-objective optimization problem based on the Dynamic Voltage Frequency Scaling System and employ the Non-dominated Sorting Genetic Algorithm (NSGA-II) to obtain feasible solutions. Moreover, they advance an Artificial Neural Network (ANN) technique to predict the corresponding VMs based on the tasks' characteristics and resources' features.

In [38], the authors design a multi-objective optimization problem which

seeks to minimize three conflicting objectives, i.e, execution cost, makespan, and resource utilization. The Epsilon-fuzzy dominance-based composite discrete artificial bee colony (EDCABC) is then employed to derive Pareto optimal solutions for multi-objective task scheduling in cloud-based systems. In [56], the authors propose a novel strategy for load balancing of tasks in cloud computing environments inspired by the behavior of honey bees, which helps to achieve even load balancing across virtual machine to maximize throughput. It considers the priority of task waiting in queue for execution in virtual machines. After the work load on VM has been calculated it decides whether the system is overloaded, under loaded, or balanced. Based on this VMs are grouped. While load balancing is an important element of scheduling, the strategy proposed is meant for scheduling independent tasks and not dependent tasks work flows.

In [86], the authors propose a scheduling approach called Multi Label Classifier Chains Swarm Intelligence (MLCCSI) whose aim is to reduce the makespan of tasks' scheduling. The approach consists of two strategies. The first strategy applies the Ant Colony Optimization (ACO) algorithm, Artificial Bee Colony (ABC) algorithm and, Particle Swarm Optimization (PSO) algorithm to derive the optimal solution in terms of resource allocation for each corresponding task. In the second strategy, a machine learning technique is employed to predict the best algorithm (i.e, ACO, ABC, or PSO) that needs to be run to execute each corresponding task, based on several factors such as task size and number of VMs. The authors in [68], also schedule the cloud resource for an optimized load balance of various nodes, using an ACO-based algorithm. They debated that the ACO approach can first choose the node that has the greatest number of neighboring nodes. This way, the ant can travel in the maximum potential directions to find more nodes that are overloaded or underloaded. Furthermore, the ant will detect the heavy-load nodes, and reschedule some load to

the light-load nodes.

The authors of [40] propose a Multi-Agent-based Cloud Monitoring (MAS-CM) model to boost the performance and security of the tasks gathering, scheduling, and execution in large-scale cloud systems. The main objective of this work is to prevent unauthorized task injection and alteration, while optimizing the scheduling process and maximizing resources utilization. In [69], the authors propose an improved version of the Particle Swarm Optimization (PSO) approach to improve the efficiency of resource management and reduce task makespan in cloud-based task scheduling scenarios. The main idea is to change the weights of the particles as the number of iterations grows up and to introduce random weights in the final stages. This helps avoid the case where the PSO algorithm generates local optimum solutions in its final stages. The authors compare their approach experimentally with the traditional PSO, Short Job First (SJF), Round-Robin (RR), and First Come First Serve (FCFS) and show that their approach can reduce the makespan with the increase in the number of tasks.

In [128], the authors propose an improved particle swarm optimization which the outcome of the evaluation showed that the proposed technique can minimize the job average execution time, and increase the rate of availability of resources in the environment. Where the PSO does not solve large scale optimization, then simulated support algorithm is added into the PSO algorithm which increases the convergence speed of PSO. The authors in [21], propose a swarm intelligence based algorithm which reduced searching time. Traditional PSO algorithm is modulated with a random factor to handle with the precocious concourse problem. Results show that proposed system is more workable, solid, and scalable than previous methods. The proposed method gives better performance when compared with the traditional PSO-based algorithm. Total time taken to complete the task is shorter and stable than traditional PSO-based

method which results in low power consumption.

Conclusive Remarks In summary, these approaches employ AI-based techniques to improve the scheduling process. However, there is no work that provides a comprehensive view of the task scheduling problem and improves the performance of big data tasks scheduling in a wide variety of scenarios.

A. Resource Management using Deep Learning

We now move to explaining the literature on resource management in cloud computing using deep learning, since we intend to design a deep learning-based approach to automate the big data task scheduling process in cloud environments. In [6], a parallel Q-learning approach is advanced to reduce the time taken to determine optimal resource allocation policies in cloud computing. A state action space formalism is also proposed to guide Q-learning-based agents with no prior experience with finding appropriate VM allocation policies in IaaS clouds.

In [63], the authors discuss an algorithm for task scheduling that depends on genetic-ant colony algorithm. The improvement is having a strong optimistic feedback of ACO and taking into account the convergence rate of the algorithm. However, the convergence rate is strongly affected by choice of the initial pheromone. The global search capability of the GA is used to solve the optimal solution quickly and then converts it into the initial pheromone of ACO. Under the same conditions, the results shown by simulation experiments suggest that this algorithm exceeds the weights of GA and ACO.

In [101], Song et al. address the problem of host load prediction to enhance the resource management and consumption in cloud-based systems. The challenge here is to come up with accurate predictions in the light of the high load variance in cloud environments. To tackle this challenge, a Long Short- Term Memory (LSTM)-based

model is designed to predict the mean host load over consecutive future time intervals. In [30] the the authors aim to minimize the cost, especially for a large task scheduling problem. Therefore, they propose an algorithm based on a deep reinforcement learning architecture (RLTS) to dynamically schedule tasks with precedence relationship to cloud servers to minimize the task execution time. Moreover, the authors use Deep Q-Network, as a kind of deep reinforcement learning algorithms to consider the problem of complexity and high dimension.

In [41], the authors improve the ABC algorithm by adding one more step including a mutation operator after the process performed by the employed bees in ABC. The mutation operator is used after the employed bees have scouted the solution space. The selection of the food source is done by a random technique, and the mutation operator is performed if a mutation probability is satisfied. Through mutation, there is an opportunity of changing the local best position, and the algorithm may not be enclosed into local optima. When applying the mutation operator, new food sources are produced. Accordingly, the new generated food sources replace the older if their fitness value is better.

In [64], a two-phase framework is proposed to address both VM resource allocation on servers and power management on each server. In the first phase, the authors capitalize on deep reinforcement learning to achieve VM resource allocation on servers. In the second phase, LSTM and weight sharing have been employed for efficient loval power management on servers. In [81], the authors capitalize on deep learning for VM workload prediction. Specifically, a Deep Belief Network (DBN) is constructed using multi-layered restricted Boltzmann machines and regression layers. This DBN is then applied to extract high-level features from VMs' workload data and the regression layer is employed to predict the future VMs' workload.

The improved ant colony algorithm depends on partial swarm optimization

which is known as ACA-PSO is proposed by authors in [121]. Initially, the ants are in the line up with ant colony algorithm for the completion of the traverse, and re-arrangement of the solutions, while keeping in view the confined and universal solutions. While ACA-PSO controls the short comings of the algorithm, it easily gets into confined solutions in cloud computing resource scheduling. In [27], a routing load balancing policy for grid computing environments is presented. It uses routing concepts from computer networks to define a neighborhood and search an adequate computers to divide the applications workload. This algorithm is designed to equally distribute the workload of tasks of parallel applications over Grid computing environments. Route algorithms are indicated for environments where there are several heterogeneous computers and parallel applications that are composed of multiple tasks. When dealing with large scale systems, an absolute minimization of the total execution time is not the only objective of a load balancing strategy. In [82], the authors improve a complete multi objective model for task scheduling optimization. They considered the conflict between the tasks, and the authority of PSO algorithm regarding the accuracy, and the speed. In order to deliver an optimal solution for the presented model, a multi objective algorithm that is based on Multi Objective PSO (MOPSO) method was proposed. Jswarm package to multi objective Jswarm (MOJ) package has been used to calculate and implement the proposed model, with extending Cloudsim toolkit put on MOJ as its task scheduling algorithm. Optimal task organization among VMs is defined by MOJ in Cloudsim according to MOPSO algorithm.

Conclusive Remarks These approaches focus mainly on analyzing historical resource data from VMs to predict future workload and hence improve the VMs management and allocation processes. Nonetheless, none of them has yet addressed the problem of automating the process of scheduling big data tasks in cloud computing

systems using deep learning that would benefit from the large amount of available data.

B. Client Scheduling in Federated Learning Environments

In [42], the authors propose an approach based on genetic algorithms and evolutionary game theory in order to study the problem of forming highly profitable federated clouds, while maintaining stability among the members in the presence of dynamic strategies. In [47], the authors propose a decentralised FL at the segment level to enhance the efficiency of network resources usage among client devices. The authors explicitly recommend a segmented gossip strategy, which not only makes maximum use of node-to-node bandwidth, but also achieves strong convergence training.

The authors in [77] present *FedCS*, a protocol that optimizes FL's efficiency with a heterogeneous client in a mobile edge computing environment. *FedCS* proposes solving a resource constraint client selection problem that allows the server to aggregate as many client updates as possible and speed up the training convergence rate. In [25], the authors address the problem of FL training over wireless realist networks. The authors formulate an optimization problem considering both the user's selection and the allocation of resources to minimize the loss function. The predicted FL algorithm convergence rate, which takes wireless factors into account, is expressed in a closed-form.

The authors in [76] incorporate Deep Q Network (DQN) into a mobility-aware federated learning network for resource allocation. The authors suggest using the DQN to allow the model owner, without any a priori knowledge of the network, to find the optimal decisions in terms of energy and channels selection. They formulate the model owner's energy and channel selection decision as a stochastic optimization problem. The optimization problem's goal is to maximize the model

owner's number of successful transmissions while minimizing energy and channel costs. In [3], the investigators provide a DQN, which enables the server to learn and find optimal decisions without knowing the network dynamics in the first place. They use Mobile Crowd-Machine Learning (MCML) to address traditional machine learning data privacy.

In [73], the investigators introduce *FedAvg* in which the server collects the local stochastic gradient descents from the client devices and takes the average to produce the next global model. The authors performed extensive experiments on this algorithm, demonstrating its robustness in unbalanced and non-IID data distributions. In [60], the authors define an aggregation framework, called *FedProx*. *FedProx* addresses the system and statistical heterogeneity that arises from FL. The authors argue that *FedProx* can be viewed as a generalization and re-parametrization of *FedAvg*, the current state-of-the-art aggregation method for FL. *FedProx* allows variable amounts of work to be performed locally across devices, and relies on a proximal term to help stabilize the aggregation results.

In [130], the researchers focus on the statistical challenge of FL when local data is non-IID. To address this challenge, they propose *FedShare* whose main idea is to share parts of a small public dataset among clients to alleviate the weight divergence across the local data of the clients. In [28], the authors define *FedSGD* and provide a comprehensive study of its convergence. *FedSGD* operates under non-IID data for strongly convex and smooth FL problems. *FedSGD* is characterized by a trade-off between the number of local computation rounds and global communication rounds.

Conclusive Remarks The main limitation of these scheduling approaches is that they work in an offline fashion by trying to optimize a set of parameters each time a task is received. This entails high execution time and is inefficient for delay-critical tasks such as big data analytics that need prompt responses. Moreover, the existing

scheduling methods in edge computing, FL, and IoT concentrate primarily on the resource characteristics of participant devices.

Chapter 3

Cloud Task Scheduling based on Artificial Intelligence

Cloud computing has emanated as more than just a technology towards cooperating with large quantities of data. When mentioning the technology that provides flexible resources and IT services based on the Internet, the term cloud computing is therefore used [93]. Cloud computing is more and more popular in large-scale computing and data storage because it enables the sharing of computing resources that are distributed, thus leading to an automated system management, workload balancing and virtualization efficiency. Several applications are increasingly focusing on third-party resources hosted across the Internet, each with a varying capacity [126]. In the cloud computing environment, different load balancing and scheduling approaches (a technique which divides the workload across multiple computing resources such as computers, hard drives, and network) exist to ensure an appropriate utilization of resources. They also attempt to fix the problem that all the processors in the system and every node in the network must share an equal amount of workload

which is assigned to them. Load scheduling plays a key role in efficient resource utilization in a cloud computing environment. Task scheduling approaches can be divided into two categories:

- The first class is that of batch mode heuristic scheduling in which jobs are queued within a set and collected as batches as they arrive in the system, after which they get started after a fixed period. Some examples include First Come First Serve (FCFS), Round Robin (RR), Min-Max, and Min-Min [43, 59, 79].
- The second class is the online mode heuristic scheduling, where jobs are scheduled individually as they arrive in the system. These approaches are more feasible in a cloud environment as the systems may have different platforms and execution speeds.

Task scheduling is one of the crucial technologies in cloud computing, and proper task scheduling is required to improve the efficiency and to minimize the execution time. The overall performance of cloud systems heavily depends on the underlying scheduling algorithm. Thus, how to efficiently and rationally allocate the finite, heterogeneous and geographically distributed resources to meet the end-user's requirements is an urgent issue for cloud service providers.

In this chapter, we proposed a hybrid approach, called Multi Label Classifier Chains Swarm Intelligence (MLCCSI). This approach is based on two strategies. The first strategy is the swarm intelligence, which we applied on, Ant Colony Optimization (ACO) algorithm, Artificial Bee Colony (ABC) algorithm and, Particle Swarm Optimization (PSO) algorithm to find the optimal resource allocation for each task in the dynamic cloud system. Then, the second strategy is the application of the machine learning algorithm (Classifier Chains) on the results from the three algorithms and generate a new hybrid model considering the size of the tasks and the number

of the virtual machines. This strategy not only minimizes the makespan of a given tasks set, but it also adapts to the dynamic cloud computing system and balances the entire system load. The new scheduling strategy is simulated using the CloudSim version 2.1 toolkit package [20] and [22]. Roughly speaking, in our model, the machine learning algorithm required to predict a proper scheduling algorithm for every data center depending on the execution time. We use initial data from the iterate on the optimization algorithms to create a model for each data center in the cloud. These models can be further updated from a new test or real runs. The system can then use these models to make informed choices towards optimizing some externally defined criterion.

3.1 Swarm Intelligence

Swarm Intelligence is an Artificial Intelligence (AI) technique, which is studied based on the monitoring of the collective behavior in biological activities such as ant/bee foraging, nest building, larval sorting, a division of labor, and collaborative transport, etc. [15]. The number of applications of swarm intelligence is exponentially increasing in fields of, e.g., communications networks, combinatorial optimization, and robotics. Self-organized Services (SOS) in the Cloud with swarm intelligence as one possible solution have already exhibited their advantages over conventional SOS techniques in terms of adaptive routing, minimize the required cloud resources (i.e., virtual machines) load balancing, etc.

3.1.1 Swarm Intelligence-based Cloud Scheduling

A. Ant Colony Optimization (ACO)

The essential idea of ACO derives from the foraging behavior of ant colonies. When an ants group tries to search for food, they use a special genius kind of chemical to communicate with each other. That chemical is referred to as a pheromone. Randomly, the ants start to search their food. Once the ants find a path to the food source, they leave pheromone on the path. An ant can keep track off the trails of the other ants to the food source by sensing pheromone on the ground. As this process continues, most of the ants attract to choose the shortest path as there have been a huge amount of pheromones accumulated on this path [62]. In task scheduling, during an iteration of the ACO algorithm, each ant $k, k = 1, ..., m$ (m is the number of the ants), builds a tour executing n (n is the number of tasks) steps in which a probabilistic transition rule is applied. The k-ant chooses VM j for next task i with a probability that is computed by Equation 3.1.

$$p_{ij}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha} * [\eta_{ij}]^{\beta}}{\sum_{s \in allowed_k} [\tau_{is}(t)]^{\alpha} * [\eta_{is}]^{\beta}} & if j \in allowed_k \\ \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

where

- $\tau_{ij}(t)$ shows the pheromone concentration at the t time on the path between task i and VM j.
- $allowed_k = \{0, 1, \cdots, n-1\}$.
- $\eta_{ij} = \dfrac{1}{d_{ij}}$ is the visibility for the t moment, calculated with heuristic algorithm

and d_{ij} which expresses the expected execution time and transfer time of task i on VM j can be computed with Equation 3.7.

$$d_{ij} = \frac{TL\_Task_i}{Pe\_num_j * Pe\_mips_j\_VM_j} + \frac{InputFilleSize}{VM\_bw_j} \tag{3.2}$$

where TL_Task_i is the total length of the task that has been submitted to VM_j, Pe_num_j is the number of VM_j processors, Pe_mips_j is the MIPS of each processor of VM_j Input File Size is the length of the task before execution and VM_bw_j is the communication bandwidth ability of the VM_j. Finally the two parameters α and β control the relative weight of the pheromone trail and the visibility information respectively. Algorithm 3.1 shows the ACO scheduling [107] where $Cloudlet_{list}$ the list of tasks, VM_{list} the list of virtual machines, and *tabu* indicates the allowed VMs for ant k in next step also *tabu* records the traversed VM by ant k.

Algorithm 3.1 Scheduling based ACO

Require: α, β, $max_{iterations}$, $Cloudlet_{list}$, VM_{list}
for i in $Cloudlet_{list}$ and k in VM_{list} **do**
pair $Cloudlet_i, VM_K \longleftarrow \tau_{i,j}(0) = C$ // pheromone(C)
$VM_K \longleftarrow Ant_j \longleftarrow randomPick(Ant_{pool})$
$Ant_j^{tabu} \longleftarrow add(VM_k)$
while NOT done **do**
  for $k = 1tom$ **do**
    $VM_N \longleftarrow select(Ant_k, VM_{list}, Cloudlet_{list})$
    $Ant_i^{tabu} \longleftarrow add(VM_s)$
  end for
  for $k = 1tom$ **do**
    $L_k \longleftarrow calculate()$ // L_k the length of the current best tour done by the ants.
  end for
    $\tau_{i,j} \longleftarrow update()$ // The local Pheromone value
    $pheromone_{global} \longleftarrow update()$
    increment (iterations)
end while

B. Artificial Bee Colony (ABC)

The artificial bee colony algorithm (ABC), an optimization algorithm based on the intelligent foraging behavior of honey bee swarm was proposed by Karaboga in 2005 [52], [53]. This is nature inspired algorithm for self-organization. ABC contains three groups of bees: employed bees, onlookers, and scouts. The first half of the colony consists of the employed artificial bees and the second half includes the onlookers. For every food source, there is only one employed bee. In other words, the number of employed bees is equal to the number of food sources. The employed bee of an abandoned food source becomes a scout. The search carried out by the artificial bees can be summarized as follows:

- Employed bees determine a food source within the neighborhood of the food source in their memory.
- Employed bees share their information (distance, quality, direction and other information) with onlookers within the hive and then the onlookers select one of the food sources.
- Onlookers select a food source within the neighborhood of the food sources chosen by themselves.
- An employed bee of which the source has been abandoned becomes a scout and starts to search a new food source randomly.

This algorithm has a similar principle in balance the work of the virtual machine. The ABC algorithm calculates the virtual machine workload then it decides whether it is overloaded, light weighted or balanced. The high priority of the task is off from the overload virtual machine and tasks are waiting for the light weight virtual machine. These tasks are known as scout bee in the next step. ABC-inspired Load Balancing

technique reduces the response time of VM and also reduces the waiting time of the task. Onlooker bees calculate the probability of estimating new solution around food source using the following equation:

$$p_i(t) = \frac{fit_i(t)}{\Sigma_{i=1}^{TS} fit_i(t)} \tag{3.3}$$

where fit_i will be the Fitness value of task source i, and TS will be the total number of task sources. Algorithm 3.2 shows the ABC scheduling [2], where $Datacenter_{list}$ the list of data centers.

Algorithm 3.2 Scheduling based ABC

1: **Require:** $Cloudlet_{list}$, VM_{list}, $Datacenter_{list}$, fac_{LB}
2: $Groups(q) \longleftarrow divide(Cloudlet_{list})$
3: **for i** = 1 to q **do**
4: $length_i \longleftarrow lengthofgroupk(Groups_i)$
5: **end for**
6: **for k** = 1 to q **do**
7: $Cloudlet_L \longleftarrow max(Groups_k)$
8: **while**$Groups_k \geq Groups_i \mid i = 1...qandi \neq k$ **do**
9: **for** s = 1 to n **do**
10: $Datacenter_s \longleftarrow select(Datacenter_{list})$
11: **if** $fac_{LB} \leq VM_sAssigned(Datacenter)$ **then**
12: assign($Cloudlet_L$, $Datacenter_{i \neq s}$ ($VM_{leastLoad}$))
13: **else**
14: decrement ($length_k$)
15: **end if**
16: **end for**
17: **end while**
18: **end for**

C. Particle Swarm Optimization (PSO)

Particle Swarm Optimisation (PSO) is a swarm-based intelligence algorithm [32], influenced by the social behavior of animals such as a flock of birds finding a food source or a school of fish protecting themselves from a predator. A particle in PSO is

analogous to a bird or fish flying through a search (problem) space. The movement of each particle is co-ordinated by a velocity which has both magnitude and direction. Each particle position at any instance of time is influenced by its best position and the position of the best particle in a problem space. The performance of a particle is measured by a fitness value, and in order to measure how well the particle's position, the fitness function can be defined:

$$f = Min(\frac{VTime}{VRutilization}) \tag{3.4}$$

where $VTime$ to denote the execution time of VMs for executing all of the tasks, and $VRutilization$ to denote the resource utilization of VMs during the process of running the tasks. Particles in the search process update themselves by tracking two best-known positions. Best-known position known as a local best position is the individual best known position in terms of fitness value reached so far by the particle itself. Another best-known position known as a global best position is the best position in the entire population. Equation 3.5 shows the velocity, position, global best, local best update mechanisms. All these update mechanisms affect the search directions of PSO at later iterations. The velocity and position are updated as follows:

$$\begin{cases} v_i^{l+1} = wv_i^l + a_1\varphi_1(pb_i^l - p_i^l) + a_2\varphi_2(gb^l - p_i^l) \\ \\ \text{and} \\ \\ p_i^{l+1} = p_i^l + v_i^{l+1} \end{cases} \tag{3.5}$$

where the subscript i denotes the particle number; l the iteration number; pb_i^l the

personal best position of the *ith* particle up to iteration l; gb^l the global best position so far; φ_1 and φ_2 two uniformly distributed random numbers used to determine the influence of pb_i and gb; and a_1 and a_2 two constant values denoting, respectively, the cognitive and social learning rate. Algorithm 3.3 shows the PSO scheduling [23].

Algorithm 3.3 Scheduling based PSO

```
1: Input: Task, Particles
2: Output: gBest
3: foreach P_i do
4:     pBest = Generate_initial_position(P_i)
5: foreach pBest of particle P_i do
6:     gBest = Max(pBest_1, pBest_2, ....)
7: repeat
8:     j ⟵ 1 ;
9:     while j ≤ m do
10:        Select the task t_j;
11:        Calculate_est(t_j);
12:        Allocate_task(t_j);
13:        j + +;
14:    foreach particle P_i do
15:        Calculatecur_particle_fit; (current particle)
16:        if cur_particle_fit < pBest_i_fit then
17:            Update(pBest_i);
18:        if cur_particle_fit< gBest_i _fit then
19:            Update( gBest_i);
20:    if the termination condition is met then
21:        Output 4gBest_i; Break;
22:    else
23:    foreach particle P_i do
24:            Update( P_i_velocity);
25:            Update( P_i_position);
26: until the termination condition is met ;
```

3.1.2 Multi-Label prediction based on SI

Multi-label classification (MLC) has attracted increasing attention in the machine learning community during the past few years. In this chapter, we focus on a method

called classifier chains (CC) [83]. As its name suggests, CC selects an order on the label set, a chain of labels, and trains a binary classifier for each label in this order. The difference with binary relevance (BR) is that the feature space used to induce each classifier is extended by the previous labels in the chain. These labels are treated as additional attributes, with the goal to model conditional dependence between a label and its predecessors. CC performs particularly well when being used in an ensemble framework, usually denoted as ensemble of classifier chains (ECC), which reduces the influence of the label order.

$$P(y|x) = \Pi_{j=1}^{d} P(y_j|pa(y_j), x) \tag{3.6}$$

Where, $pa(y_j)$ represents the parent labels for y_j. Obviously, $|pa(y_j)| = p$, where p denotes the number of labels prior to y_j following the chain order. In the training phase, according to a predefined chain order, it builds d binary classifiers $h_1, ..., h_d$ such that each classifier predicts correctly the value of y_j by referring to $pa(y_j)$ in addition to x. In the testing phase, it predicts the value of y_j in a greedy manner:

$$y_j^* = argmax P(y_j|pa(y_j), x), \qquad j = 1, ..., d. \tag{3.7}$$

3.2 Performance Evaluation

In this section, we evaluate the performance of our proposed model (MLCCSI). First, we explain the implementation details and then present the experimental results that involve the makespan, and compare this results with the optimization algorithms ACO, ABC, and PSO.

3.2.1 Implementation and Setup

We implemented our framework in a 64-bit Windows 7 environment on a machine equipped with an Intel(R) i5-2400M CPU with Nvidia(Quadro) HD Graphics 3.10 GHz Processor and 8.192 GB RAM. We simulated the proposed model and the optimization Scheduling algorithms in the CloudSim toolkit [15]. This simulation mainly shows the advantage of the proposed model compared to the ACO, ABC, and PSO Algorithms in makespan term. The experiment is implemented with 3 data centers and 50 tasks between 30 and 2700 bytes with different numbers of VMs varying from 2 to 50 under the simulation platform. The resource situation is shown in Table 1. The computation workload of the task is 10000 MIPS (Million Instruction per second), and the manager of the three data centers both have space shared and time shared policy for VMs, but, for the VMs manager, we set Time shared algorithm for tasks. All the algorithms are tested by varying the number of the VMs with randomly varying the size of the tasks.

Table 3.1: Parameters setting of cloud simulator

Type	**Parameters**	**Value**
Data Center	Number of Datacenters	3
	Number of Hosts	6
	Type of Manager	Space shared and Time shared
	Datacenter Cost	1 –15
Virtual Machine (VM)	Total number of VMs	2 –50
	VM memory(RAM)	512(MB)
	Type of Manager	Time shared
	Bandwidth	1000 bit
Task	Total number of task	50
	Size of task	30 –2700 bytes
	Number of PEs requirement	2

3.2.2 Experimental Results

In this section, we analyze the performance of our model based on the results of simulation done using CloudSim. We extended the classes of the CloudSim simulator to simulate our model. In the following illustrations, we compare the makespan of our model with the basic algorithms. In the first step, we apply the optimization algorithms when the virtual machines number has been fixed to 2, and the task sizes are varying between 30, and 2700 bytes. Moreover, we iterate the optimization while varying the number of virtual machines from 2, to 50. We have run each algorithm 112 times, and the average makespan of these runs is shown in Figs. 3.1, 3.2, and 3.3.

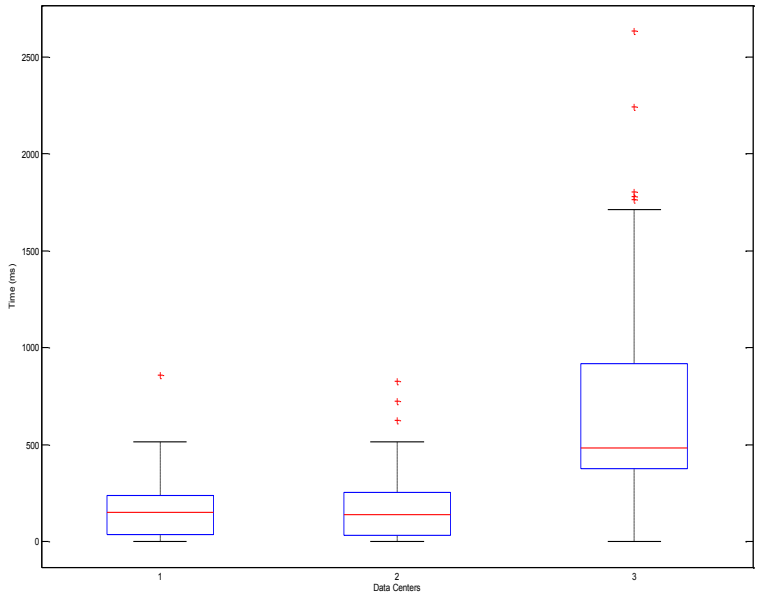


Figure 3.1: The average makespan of 3 data centers for the ant colony algorithm

We use the machine learning classifier chain algorithm to choose the future algorithm to be used to schedule the tasks in every data center while having the best task execution time, by using Gibbs sampler. The key idea of Gibbs sampling is that one only considers univariate conditional distributions, i.e. the distribution when all of the variables but one are assigned fixed values. This property makes

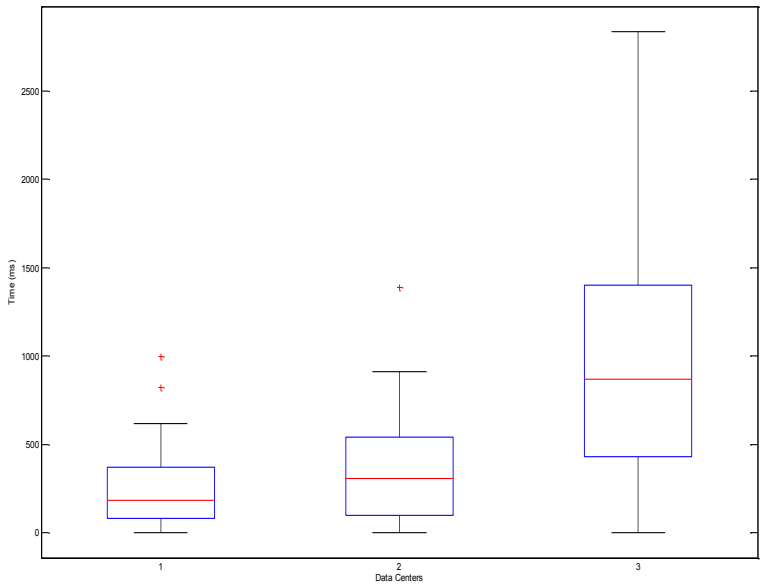


Figure 3.2: The average makespan of 3 data centers for the artificial bee honey algorithm

Gibbs sampling a perfect fit for our fully connected conditional dependency network we build, where the univariate conditional distributions needed for Gibbs sampling are directly available from the conditional probabilistic predictors associated with each variable. The inference procedure of Gibbs sampling is very simple. We first choose a random ordering of the variable r, and initialize each variable y_1 to a value y_i. In each sampling iteration, we visit every variable in the given order, $y_i(1), ..., y(k)$, where r maps the new order index into the original variable index. The new value of each variable $Y_r(i)$ is resampled according to the conditional predictor associated with it, $p(y|x, y_r(i), \theta_r(i))$. The idea behind Gibbs sampling is to approximate the joint distribution from the samples obtained from the conditional distributions. The sampler is expected to converge to a stationary distribution after some burn-in iterations. Then, it collects samples to recover the approximated joint distribution and determine the most probable explanation (MPE).

Figs. 3.4, 3.5, and 3.6 show the average makespan on the data center one, two

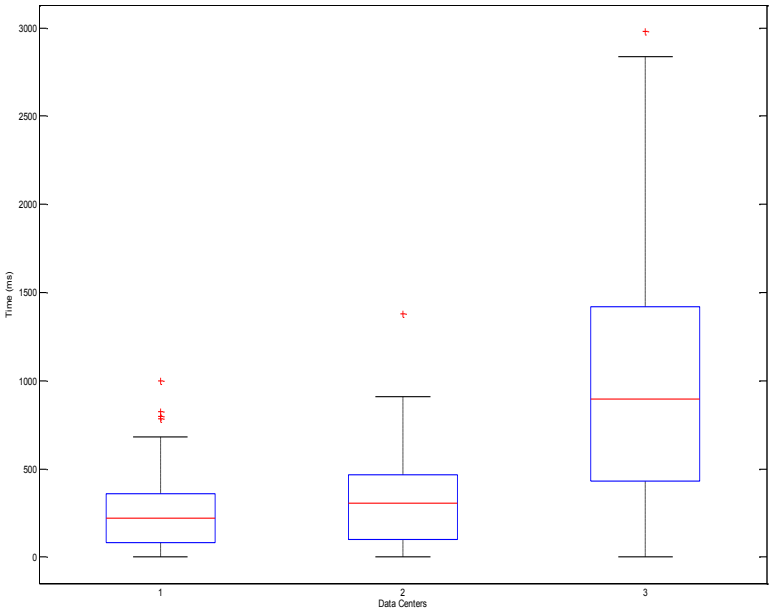


Figure 3.3: The average makespan of 3 data centers for the particle swarm optimization algorithm

and three when we used the optimization algorithms ant colony, artificial honey, and particle swarm, respectively.

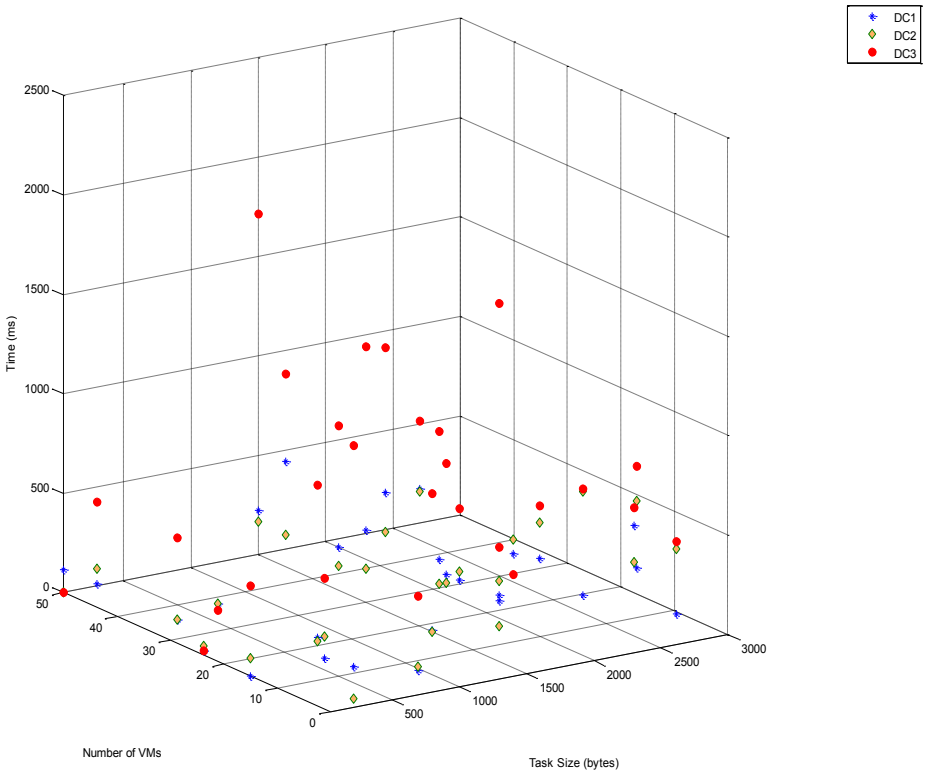


Figure 3.4: Ant colony algorithm

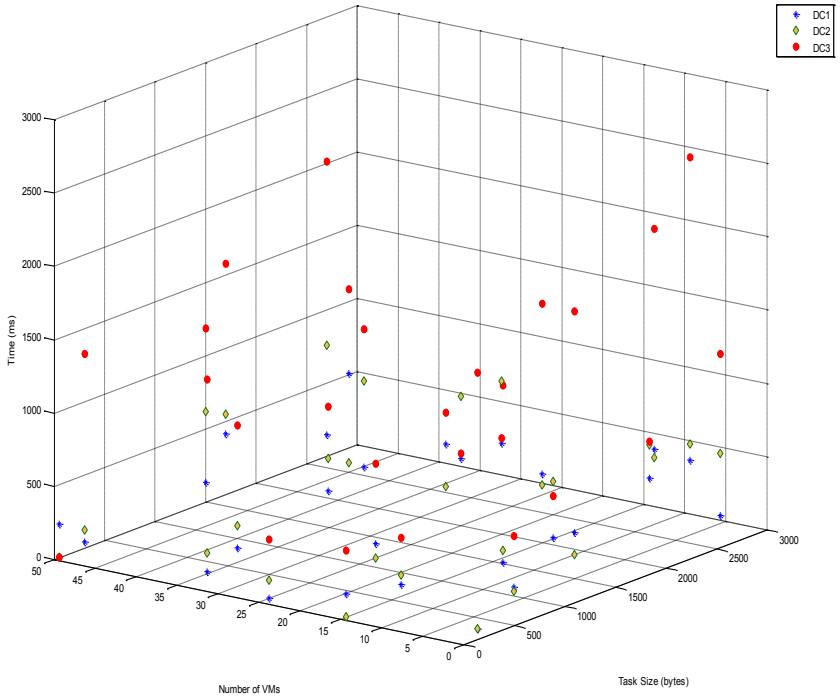


Figure 3.5: Artificial bee honey algorithm

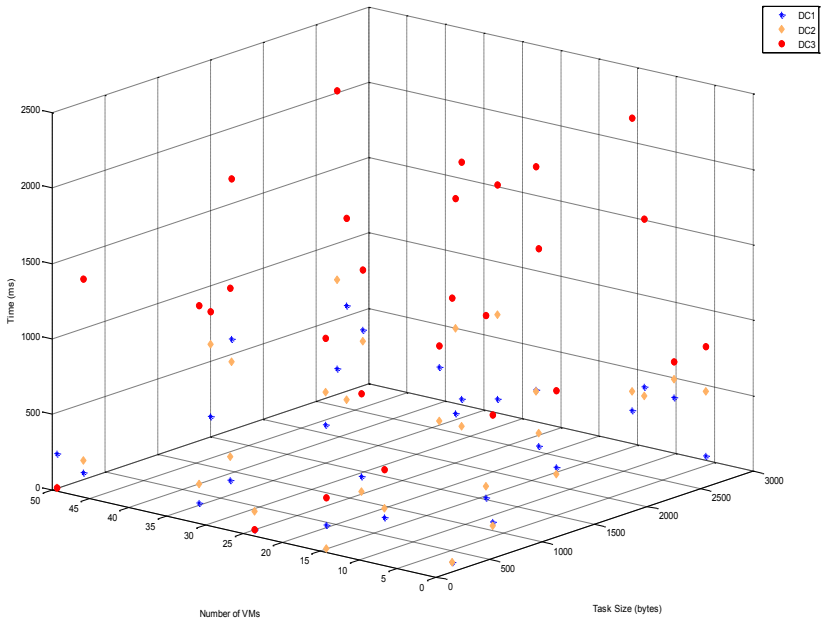


Figure 3.6: PSO algorithm

Furthermore, Fig. 3.7 shows the makespan when used with the multi-label classifier chain machine learning algorithm, and it shows that the average makespan of the basic algorithm has been roughly reduced especially for ABC, and PSO algorithms on all the data centers. When we compare the results with ant colony algorithm, it shows that the time has been reduced 10% on data center one, 7% on data center two and 13% on data center three. Furthermore, the figure shows a clear time decrease on artificial honey bee algorithm, 51% on data center one, 75% on data center two, and 60% on data center three. Moreover, the decrease on makespan happens roughly on the particle swarm optimization algorithm, and the figure shows 63% decrease on data center one, 73% on data center two, and finally 68% on data center three.

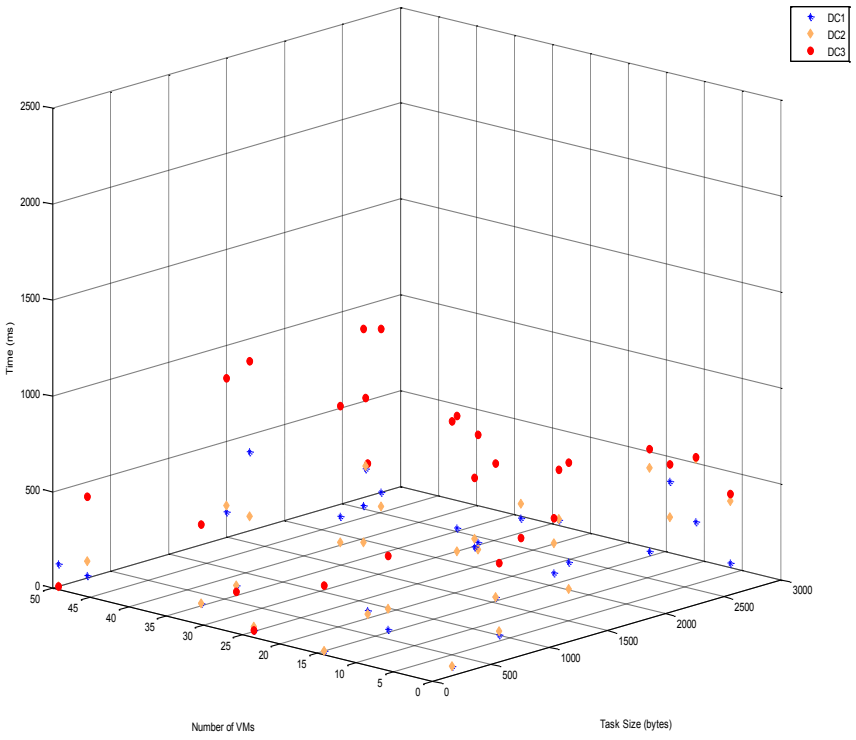


Figure 3.7: MLCCSI model

3.3 Conclusion

In this chapter, we have proposed the MLCCSI model for tasks scheduling with load balancing. We have experimentally evaluated the model in applications with the number of VMs ranging from 2 to 50 and varying tasks sizes from 30 to 2700. Three scheduling algorithms are discussed, namely ACO, ABC and PSO and a new scheduling model that uses these three algorithms is introduced. The experimental results show that the MLCCSI model minimizes the makespan and utilizes the resources effectively compared to the standard optimization algorithms. We have used the makespan as fitness criteria for checking the fitness of the scheduling results, and The experimental result shows that MLCCSI balances the entire system load effectively, and it clearly shows reduced average makespan. This method can be adapted to existing cloud computing systems for decreasing makespan and providing better resource utilization.

Promisingly, this work gives guidance to a new architecture that could be adopted to solve or mitigate several challenges that encounter the domain of cloud computing. It opens as well numerous research directions that seem worthy working on and investigating such as: (1) building efficient scheduling optimization algorithms in the cloud, (2) developing resource sharing and task scheduling models, and (3) building machine learning algorithms and dynamic models to minimize the required cloud resources and load balancing.

Since the tasks are carried out locally at the level of the VMs or IoT devices, some malicious devices might optimize for a malicious objective that aims to generate targeted misclassifications. Some other devices might not dedicate enough resources to process tasks, which could lead to poor-quality results. To address these challenges, we propose in the next chapter a scheduling solution that takes into account the resources availability and trust values of the devices.

Chapter 4

Trust-aware Big Data Task Scheduling Approach in Cloud Computing Environments

The term big data had arisen in the past few years as a building block concept, owing to the increase in the volumes of data that are generated by different businesses on a daily basis. This raised the need to adopt up-to-date solutions that could enable the storage, scheduling, and processing of large volumes of data. Cloud computing has been the number one choice for most of the businesses seeking to keep up with the big data evolution trends [11], thanks to the wide variety of advantages it offers such as multi-tenancy, elasticity, and virtualization. Thus, instead of purchasing and maintaining expensive hardware equipment to store and analyze big data, companies can now migrate these responsibilities to the cloud to reduce the

capital and operational expenditures and enjoy improved performance. This, however, entails a serious problem for cloud administrators, i.e., how to efficiently schedule the big data tasks in the virtualized environments so as to guarantee optimal performance and minimal resource wastage.

In cloud computing, resources, either software or hardware, are virtualized and allocated as services from providers to users [12]. The computing resources can be allocated dynamically upon the requirements and preferences of consumers, where the resources are located in different regions and have various processing characteristics (number of CPU cores, amount of main memory, etc.), and costs [110]. The process of allocating services to perform a set of tasks while satisfying constraints in terms of time, cost, quality of service QoS, and service availability known as task scheduling [65]. Therefore, task scheduling and resource allocation should be carefully arranged and optimized simultaneously in order to attain an overall cost and time-effective schedule. Specifically, the process of scheduling big data analytics tasks in cloud computing environments is quite challenging since it involves the optimization of several (sometimes conflicting) objectives. On the one hand, guaranteeing minimal timespan for big data tasks is crucial especially when it comes to delay-critical applications (e.g., healthcare management, intelligent transportation systems) wherein small delays might cause loss of life. Moreover, as the number of incoming big data tasks is quite large, minimizing the cost of each single task is a major concern for these data sources (e.g., IoT manufacturer) [9, 31]. From the cloud providers' site, managing the available resources so as to increase the capacity of simultaneously receiving the largest possible number of tasks is a major concern. That is, cloud providers should schedule the big data tasks in such a way to guarantee a minimal CPU, RAM, bandwidth, and disk storage consumption on each tasks, without sacrificing the overall performance.

In this chapter, we discuss our trust-aware scheduling solution called *BigTrustScheduling* that is proposed for scheduling big data tasks in cloud computing environments. The proposed solution considers the trust value of the VMs and the task priority in terms of execution time and price.

4.1 An Overview of The Proposed Approach (BigTrustScheduling)

Fig. 4.1 shows the design of *BigTrustScheduling*, our trust-aware scheduling model for big data tasks in cloud computing environments. The proposed solution consists of three stages: (1) VMs' trust level computation; (2) tasks priority levels determination; and (3) trust-aware scheduling. These stages are executed sequentially to minimize the hardware and network recourses cost while optimizing the response time delivered to customers.

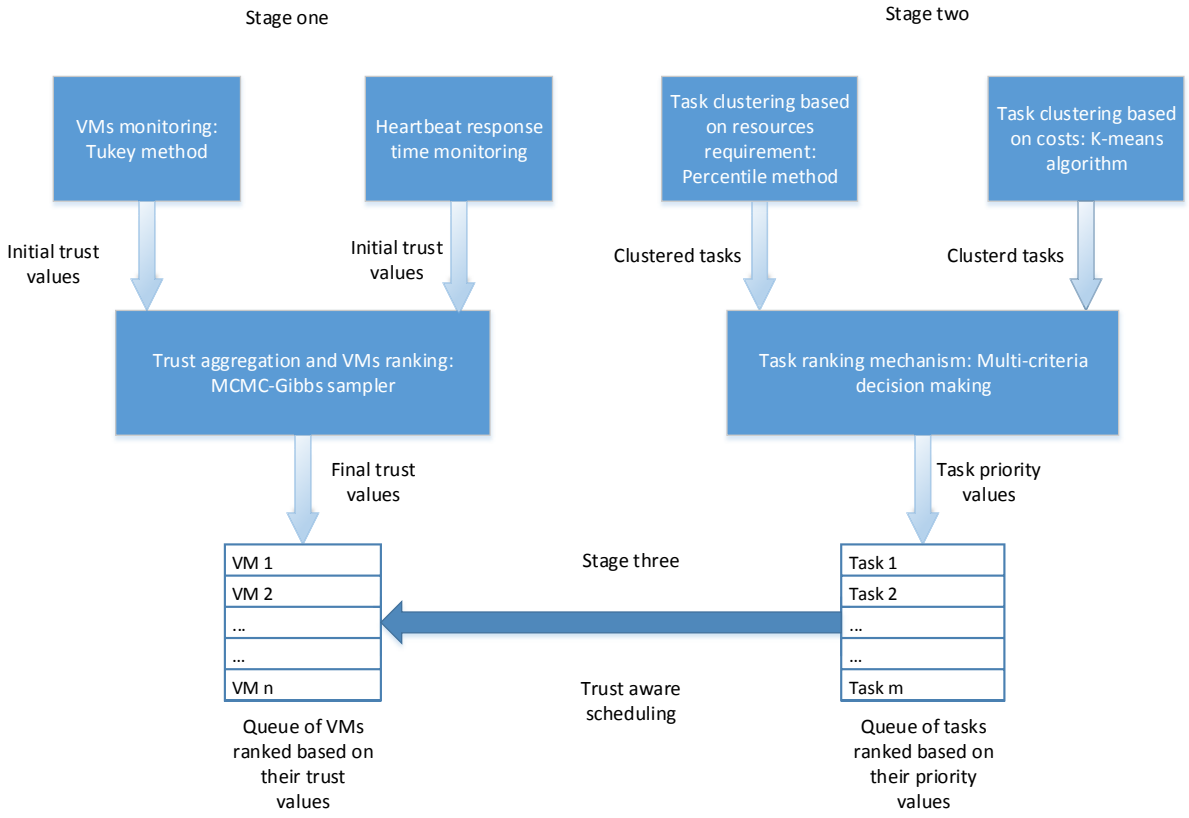


Figure 4.1: The three stages of our proposed solution (*BigTrustScheduling*)

The first stage comprises three phases, namely: hearbeat-based initial trust value computation, statistics-based trust computation, and Markov Chain Monte Carlo Gibbs Sampler (MCMC)-based trust aggregation and VMs ranking. The objective of the first phase is to derive initial trust values for the VMs based on the heartbeats average response time and average frequency ratio. Specifically, we adopt the concept of *JobTracker* from the Hadoop platform [117] whose role is to periodically collect messages (commonly called heartbeats) from tasktracker agents deployed on VMs to ensure that these VMs are still alive [119]. To further improve the accuracy of the trust establishment process, we propose in the second phase an Interquartile Range (IQR) statistical technique (Tukey method) [14] in which the cloud system continually monitors the CPU, RAM, Bandwidth, and Disk storage consumption to

capture any abnormal behavior (i.e., over-utilization or under-utilization). IQR has been chosen for the considered problem thanks to its robustness to outliers and its low overhead, which boosts its adoption in resource-constrained environments [39]. In the third phase, we employ the MCMC Gibbs sampler to aggregate the trust observations collected from the first and the second phases and come up with final trust values for each VM. The power of Gibbs sampler comes from its ability to break down the high-dimensional parameter space into several low dimensional phases, thus facilitating the convergence to the optimal distribution [36].

The second stage comprises three phases, namely: percentile-based task clustering, machine learning-based task clustering, and multi-criteria decision-making approach to task priority level determination. The first phase is interested in clustering tasks on the basis of their hardware, network, and storage requirements, by means of a statistical technique. Note that our choice of the percentile-based method for tasks clustering based on their resource requirements stems from the sparse nature of our data[1]. This limits the performance of K-means (which highly depends on the average) and results in a high number of outliers. Thereafter, in the second phase, we employ the K-means learning method to cluster the tasks according to their execution costs. The strength of K-means lies in its reasonable computational overhead for high-dimensional data compared to other clustering techniques (e.g., K-medoids, hierarchical clustering, etc.) [51]. Moreover, K-means has been chosen in this step over the percentile method since our aim is to generate five clusters (i.e., very low, low, medium, high, and very high), whereas the percentile method usually works well for generating a small number of clusters (i.e., three). On top of the two aforementioned clustering processes, we design in the third phase a multi-criteria decision-making approach to determine the priority level of each task.

[1]http://gwa.ewi.tudelft.nl/datasets/gwa-t-12-bitbrains

Having determined the trust level of each VM and the priority level of each task, we propose in the final stage a trust-aware scheduling approach to match each task to the corresponding VM. Such a trust-aware scheduling approach can be integrated as a complementary step into the state-of-the-art big data schedulers such as YARN [111] and MESOS [46]. In summary, the main contributions of this chapter are:

- Proposing a trust-aware scheduling mechanism to optimize the performance of big data services execution. To the best of our knowledge, this is the first work that advances a comprehensive trust-aware and performance-oriented scheduling approach for big data services on the cloud.
- Designing a trust establishment framework which combines performance information related to the average heartbeat response time, average heartbeat frequency ratio, and VMs resources utilization to derive trust values for each VM.
- Developing two clustering techniques from machine learning and statistics to group tasks into a set of clusters based on their resource requirements and execution costs.
- Designing a multi-criteria decision-making mechanism that capitalizes on the two proposed clustering techniques to determine the priority level of each task.

4.2 Problem Definition

We consider a finite set of jobtrackers, where each jobtracker monitors and manages a set of tasktrackers responsible for a set $V = \{v_1, v_2, ..., v_m\}$ of VMs, and let v_j be a typical VM in the set. Every tasktracker monitors and analyzes the CPU, RAM, disk storage, and network bandwidth utilization of each $v_j \in V$. As a first stage, the

jobtracker seeks to establish trust relationships toward the tasktrackers based on the average frequency and response time of the heartbeats. This allows the cloud system to build an initial trust value for each v_j denoted as $InitialTrust_j$. To do so, the cloud system collects log files from jobtrackers which contain information on the behavior of the tasktrackers (i.e., heartbeats information) to learn about the active and inactive ones. In order to complete the final trust decision, the cloud system monitors the VMs found trusted in the initial trust establishment phase to inspect their CPU, RAM, network bandwidth, and disk storage utilization, and applies the Interquartile Range (IQR) statistical measure to identify any abnormal utilization on each v_j and hence compute another initial trust score for each VM. Note that $c(v_j)$ represents the CPU consumption of v_j, $r(v_j)$ represents the RAM consumption of v_j, $s(v_j)$ represents the disk storage consumption of v_j, and $b(v_j)$ represents the bandwidth consumption of v_j. Finally, the MCMC Gibbs Sampler method is employed to aggregate both initial trust values, come up with a final trust score for each VM, and rank these VMs based on their trust scores.

Having computed the trust values of each VM and ranked them based on their trustworthiness, the next step is to determine the priority level of each task. Let $T = \{t_1, t_2, ..., t_n\}$ be a set of tasks submitted by certain users and let t_i be a typical task in the set. Moreover, let R_i^z represent the requirement of each task t_i in terms of the parameter z (CPU, RAM, bandwidth, and disk storage). We first propose a new method based on the percentile ranking to cluster the tasks based on their CPU, RAM, network bandwidth, and disk storage requirements R_i^z. Thereafter, we propose a K-means clustering technique to cluster tasks based on their monetary prices. Finally, we advocate a multi-decision criteria approach which capitalizes on the two aforementioned clustering methods to determine the priority level of each task and rank these tasks based on their priority score.

Having ranked VMs based on their trust scores and tasks based on their priority levels, the final step is to execute the actual scheduling process. To do so, we propose a new scheduling approach which takes into consideration the trust values of the VMs, the priority levels of the tasks, and the resource requirements and availabilities to match each task to the corresponding VM, so as to minimize the makespan and monetary cost. Note that the different notations used throughout the chapter are summarized in Table 4.1.

Table 4.1: Notations

Symbol		Significance
V	:	Set of virtual machines.
T	:	Set of tasks provided by users.
z	:	A performance parameter $z \in M = \{CPU, RAM, bandwidth, disk\,storage\}$.
φ_j	:	Current heartbeat frequency for VM v_j.
Φ_j	:	Average heartbeat response time for the VM v_j.
$InitialTrust_j$	:	Initial trust believed by the cloud system in the trustworthiness of the virtual machine v_j .
ψ_j	:	Habitual heartbeat response time from the historical log file for VM v_j.
Q_j	:	Heartbeat response frequency ratio for VM v_j.
γ_j	:	Historical average heartbeat frequency for VM v_j.
W_j	:	Average response time ratio for VM v_j.
η_j^x	:	Response time of heartbeat x for VM v_j.
τ_h	:	Time moment $h \in \mathbb{N}$.
P_i	:	Priority degree of task $t_i \in T$.
$\chi_{V,\tau}$	:	Matrix storing the trust values of all VMs V across time moments τ.
e_{jh}	:	Trust value of VM v_j at time moment τ_h.
μ	:	Set of trust mean values of VM v_j across time moments.
a_r	:	Resource requirement cluster $r \in \{1, 2, 3\}$ for low, medium, and high, resp.
c_u	:	Task cost cluster $u \in \{1, \ldots, 5\}$ for very low, low, medium, high, and very high, resp.
$D^z(T)$	:	A (3×5)-matrix of requirements for task $t_i \in T$ in terms of the performance parameter z.
G_{ru}	:	Position of a particular task belonging to the clusters a_r and c_u in the matrix $D^z(T)$ $(M_{ru} = D^z(T)[r, u])$.

4.3 BigTrustScheduling: Description of the Proposed Trust-Aware Scheduling Approach

4.3.1 First Stage: Building trust on VM

A. First Phase: Building Initial Trust on TaskTracker

Scheduling decisions are made by a master node called jobtracker, while the worker nodes, called tasktrackers are responsible for the task execution. The jobtracker maintains a queue of the jobs currently running, states in a tasktrackers cluster and the list of tasks assigned to each tasktracker. Each tasktracker periodically reports its status (active, inactive) to the jobtracker in the form of a heartbeat message as shown in Fig. 4.2. The content of the heartbeat message is:

- Lists of completed or failed tasks.
- Progress report of the tasks currently being carried out at the issuer tasktracker.
- Boolean flag (accept new tasks) that indicates whether the sender jobtracker can assign an additional task or not. This indicator is set to true if the number of running tasks at the tasktracker is below the VMs capacity.

The jobtracker collects the heartbeats received from the set of tasktrackers installed on each VM. If a heartbeat is not received from a tasktracker within a certain interval of time, this means that the VM should be deemed dead or down. Hence, to calculate the heartbeat response frequency ratio Q_j, let γ_j be the habitual historical average heartbeat frequency for VM v_j, and φ_j be the current heartbeat frequency for VM v_j.

$$Q_j = \frac{\varphi_j}{\gamma_j} \times 100\% \tag{4.1}$$

and the average heartbeat response time Φ_j would be:

$$\Phi_j = \frac{\sum_{x=1}^{N_j} \eta_j^x}{N_j} \tag{4.2}$$

where η_j^x represents the response time of the heartbeat x for the VM $v_j \in V$, and N_j is the total number of received heartbeats for v_j. Then, the average response time ratio can be calculated as follows:

$$W_j = \frac{\Phi_j}{\psi_j} \times 100\% \tag{4.3}$$

where ψ_j is the habitual heartbeat response time of VM v_j from the historical log file. Finally, the first trust value of VM v_j ($FirstInitialTrust_j$) is computed by dividing the summation of the average heartbeats response frequency ratio, and the average response time ratio by two. The reason behind this formula is because we assume that both the response frequency ratio and average response time ratio have equal importance.

$$FirstInitialTrust_j = \frac{Q_j + W_j}{2} \tag{4.4}$$

The second trust value of VM v_j ($SecondInitialTrust_j$) it will computed later in Algorithm 4.3.

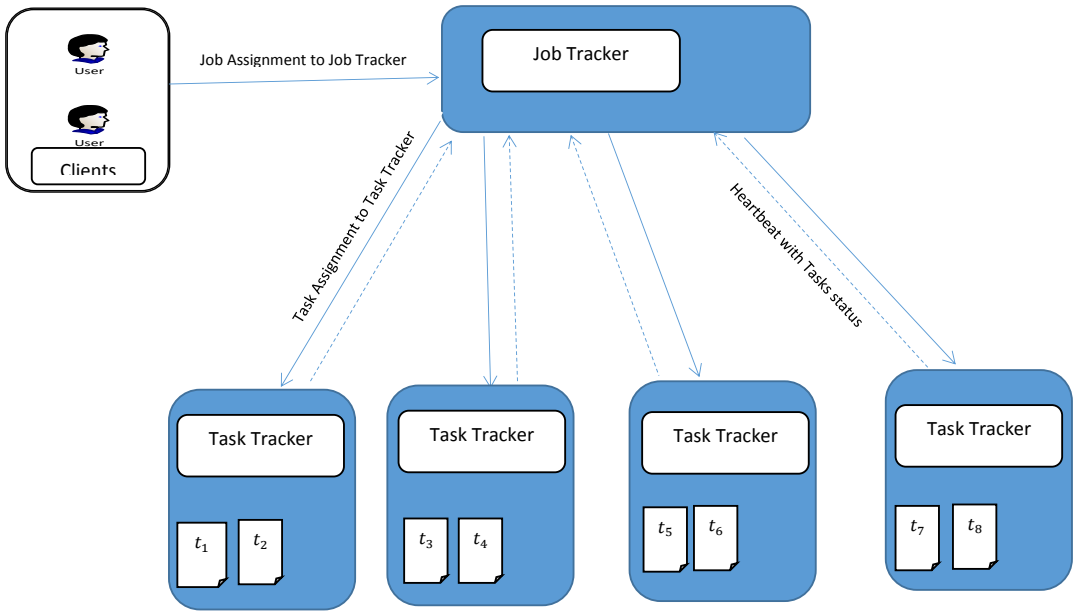


Figure 4.2: Tasks assignment and reporting process

B. Second Phase: VMs Monitoring

In this phase, the tasktracker monitors the CPU, RAM memory, network bandwidth, and disk storage consumption of the VMs to construct a consumption dataset for each VM. It then employs the IQR statistical technique to recognize any abnormal consumption. The IQR is a measure of variability whose basic idea is to divide a given set of data into disjoint quartiles (i.e., $Q1, Q2, Q3$). The first quartile Q1 is the value in the dataset that 25% of the values are smaller than it. The second quartile Q2 represents the median value of the dataset. The third quartile Q3 represents the value in the dataset that 25% of the values are greater than it. The IQR is obtained by subtracting the first quartile from the third quartile. The reasons for choosing the IQR measure for our problem are (1) its robustness to disordered and unorganized data and (2) its simple and light nature that does not require significant computational effort. Algorithms 4.1, 4.2, and 4.3 (executed by the jobtracker) are introduced to described

the aforementioned process. In particular, Algorithm 4.1 determines the VMs whose consumption exceeds the normal maximal consumption, Algorithm 4.2 is introduced to determine the VMs whose consumption goes down the normal minimal habitual consumption (e.g., failed VMs). Based on the results obtained from Algorithm 4.1 and Algorithm 4.2, Algorithm 4.3 is introduced to compute initial trust for the VMs. After having collected the CPU, memory, disk storage, and bandwidth consumption for each VM, the jobtracker calculates the median usage of each VM for each corresponding metric (e.g., CPU) (Algorithms 4.1 and 4.2 - line 15). Then, based on the calculated median value, the first and third quartiles Q1 and Q3, respectively, are derived for each metric (Algorithm 4.1 and 2 - lines 16-17). Using these quartiles, IQR is calculated by subtracting Q3 from Q1 and multiplying the value obtained by 1.5 (Algorithm 4.1 and Algorithm 4.2 - line 18). Intuitively, this means that any value greater than one and a half times the upper quartile or lower than one and a half times the lower quartile shall be considered an outlier according to Tukey's analysis [112].

Algorithm 4.1 VMs Upper Consumption Monitoring

Inputs:

v_j: a VM being monitored by the cloud system

$M = \{CPU,\ memory,\ diskstorage,\ and\ bandwidth\}$: the set of VM's metrics to be analyzed by the cloud system

δ: size of time window after which the algorithm is to be repeated

Variables:

$M_j^z(t)$: a table recording the amount of each metric $z \in M$ consumed by v_j during the time interval $[t-\delta, t]$

$\bar{x}_j^z(t)$: the median consumption of $z \in M$ by v_j during the time interval $[t-\delta, t]$

$Q1_j^z(t)$: the 1st quartile consumption of $z \in M$ by v_j during the time interval $[t-\delta, t]$

$Q3_j^z(t)$: the 3rd quartile consumption of $z \in M$ by v_j during the time interval $[t-\delta, t]$

$IQR_j^z(t)$: the IQR consumption of $z \in M$ by v_j during the time interval $[t-\delta, t]$

$L_j^z(t)$: the upper consumption limit of $z \in M$ v_j during the time interval $[t-\delta, t]$

$OverUse_j^z$: sum of VM v_j's unusual high consumption of $z \in M$ (initialized to 0)

$CountOverUse_j^z$: a counter enumerating the occurrence of unusual high consumption of $z \in M$ by v_j (initialized to 0)

$AvgOverUse_j^z$: VM v_j's average unusual high consumption of $z \in M$

Outputs:

$PropOverUse_j^z$: VM v_j's unusual consumption of $z \in M$ proportionally to the upper consumption limit of this z

$|OverusedMetrics_j|$: the number of metrics that v_j over consumed such that $|OverusedMetrics_j| \leq |M|$ **for each** metric $z \in M$ **do**

    Compute the median $\bar{x}_j^z(t)$ of $M_j^z(t)$
    Find $Q1_j^z(t)$ as the median of $M_j^z(t)$'s lower half
    Find $Q3_j^z(t)$ as the median of $M_j^z(t)$'s upper half
    Compute $IQR_j^z(t) = (Q3_j^z(t) - Q1_j^z(t)) \times 1.5$
    Compute $L_j^z(t) = IQR_j^z(t) + Q3_j^z(t)$
    for each data point $y \in M_j^z(t+\mu)$ **do**
        if $y > L_j^z(t)$ **then**
            $OverUse_j^z = OverUse_j^z + y$
            $CountOverUse_j^z = CountOverUse_j^z + 1$
        end if
    if $CountOverUse_j^z > 0$ **then**
        $AvgOverUse_j^z = OverUse_j^z / CountOverUse_j^z$
        $PropOverUse_j^z = L_j^z(t) / AvgOverUse_j^z$
        $|OverusedMetrics_j| = |OverusedMetrics_j| + 1$
    end if
    end for
end for

Algorithm 4.2 VMs Lower consumption Monitoring

Inputs:

v_j: a VM being monitored by the cloud system

$M = \{CPU,\ memory,\ diskstorage,\ and\ bandwidth\}$: the set of VM's metrics to be analyzed by the cloud system

δ: size of time window after which the algorithm is to be repeated

Variables:

$M_j^z(t)$: a table recording the amount of each metric $z \in M$ consumed by v_j during the time interval $[t-\delta, t]$

$\bar{x}_j^z(t)$: the median consumption of $z \in M$ by v_j during the time interval $[t-\delta, t]$

$Q1_j^z(t)$: the 1st quartile consumption of $z \in M$ by v_j during the time interval $[t-\delta, t]$

$Q3_j^z(t)$: the 3rd quartile consumption of $z \in M$ by v_j during the time interval $[t-\delta, t]$

$IQR_j^z(t)$: the IQR consumption of $z \in M$ by v_j during the time interval $[t-\delta, t]$

$W_j^z(t)$: the lower consumption limit of $z \in M$ by v_j during the time interval $[t-\delta, t]$

$UnderUse_j^z$: sum of VM v_j's unusual low consumption of $z \in M$ (initialized to 0)

$CountUnderUse_j^z$: a counter enumerating the occurrence of unusual low consumption of $z \in M$ by v_j (initialized to 0)

$AvgUnderUse_j^z$: VM v_j's average unusual low consumption of $z \in M$

Outputs:

$PropUnderUse_j^z$: VM v_j's unusual consumption of $z \in M$ proportionally to the lower consumption limit of this z

$|UnderusedMetrics_j|$: the number of metrics that v_j over consumed such that $|UnderusedMetrics_j| \leq |M|$

for each metric $z \in M$ **do**
    Compute the median $\bar{x}_j^z(t)$ of $M_j^z(t)$
    Find $Q1_j^z(t)$ as the median of $M_j^z(t)$'s lower half
    Find $Q3_j^z(t)$ as the median of $M_j^z(t)$'s upper half
    Compute $IQR_j^z(t) = (Q3_j^z(t) - Q1_j^z(t)) \times 1.5$
    Compute $W_j^z(t) = Q1_j^z(t) - IQR_j^z(t)$
    for each data point $y \in M_j^z(t+\mu)$ **do**
        if $y < W_j^z(t)$ **then**
            $UnderUse_j^z = UnderUse_j^z + y$
            $CountUnderUse_j^z = CountUnderUse_j^z + 1$
        end if
    if $CountUnderUse_j^z > 0$ **then**
        $AvgUndeUse_j^z = UnderUse_j^z / CountUnderUse_j^z$
        $PropUnderUse_j^z = W_j^z(t) / AvgUnderUse_j^z$
        $|UnderusedMetrics_j| = |UnderusedMetrics_j| + 1$
    end if
    end for
end for

Algorithm 4.3 Virtual Machines Monitoring

Inputs:

v_j: a VM being monitored by the cloud system

$M = \{CPU,\ memory,\ diskstorage,\ and\ bandwidth\}$: the set of VM's metrics to be analyzed by the cloud system

δ: size of time window after which the algorithm is to be repeated

Variables:

$M_j^z(t)$: a table recording the amount of each metric $z \in M$ consumed by v_j during the time interval $[t-\delta, t]$

$PropOverUse_j^z$: VM v_j's unusual consumption of $z \in M$ (obtained from algorithm 1) proportionally to the upper consumption limit of this z

$|OverusedMetrics_j|$: the number of metrics that v_j over consumed (obtained from algorithm 1) such that $|OverusedMetrics_j| \leq |M|$ obtained from algorithm 1

$PropUnderUse_j^z$: VM v_j's unusual consumption of $z \in M$ (obtained from algorithm 2) proportionally to the lower consumption limit of this z

$|UnderusedMetrics_j|$: the number of metrics that v_j under consumed (obtained from algorithm 2) such that $|UnderusedMetrics_j| \leq |M|$

$UpperInitialTrust_j$: the initial trust of cloud system regarding v_j's trustworthiness with respect to the upper limit.

$LowerInitialTrust_j$: the initial trust of cloud system regarding v_j's trustworthiness with respect to the lower limit.

Output:

$SecondInitialTrust_j$: the initial trust of cloud system in VM v_j's trustworthiness

procedure VMMonitoring
  repeat
  run Alg. 4.1 on v_j, M, δ **to obtain** $PropOverUse_j^z$ **and** $|OverusedMetrics_j|$
    if $|OverusedMetrics_j| = 0$ **then**
      $UpperInitialTrust_j = 1$
    else
      $UpperInitialTrust_j = \frac{\sum_{z \in M} PropOverUse_j^z}{|OverusedMetrics_j|}$
    end
  run Alg. 4.2 on v_j, M, δ **to obtain** $PropUnderUse_j^z$ **and** $|UnderusedMetrics_j|$
    if $|UnderusedMetrics_j| = 0$ **then**
      $LowerInitialTrust_j = 1$
    else
      $LowerInitialTrust_j = \frac{\sum_{z \in M} PropUnderUse_j^z}{|UnderusedMetrics_j|}$
    end
    else
      $SecondInitialTrust_j = \frac{UpperInitialTrust_j + LowerInitialTrust_j}{2} \times 100\%$
  until δ **elapses**
end procedure

By adding the IQR to the third quartile, the jobtracker computes the upper consumption limit for each underlying metric (Algorithm 4.1 - line 19). Similarly, by subtracting Q1 from IQR, the jobtracker determines the lower consumption limit for each underlying metric (Algorithm 4.2 - line 19). These limits describe the patterns of maximal and minimal habitual utilization of each VM at a certain time interval, where any future utilization greater/lower than the upper/lower limits would be considered unusual. The jobtracker checks then for any future consumption of the VM at time

$t + \delta$ to determine whether there exists any consumption that exceeds the computed upper limit (Algorithm 4.1 - lines 20-21). If so, this result is added to a table that registers the VM's unusual consumption (Algorithm 4.1 - line 22), and the average unusual consumption for each metric is then computed (Algorithm 4.1 - line 26). At the same time $t+\delta$, the jobtracker also checks for any consumption that falls below the computed lower limit (Algorithm 4.2 - lines 20-21). If found, these values are added to a table that stores the VM's unusual low consumption (Algorithm 4.2 - line 22), and the average unusual consumption for each metric is computed (Algorithm 4.2 - line 26). Eventually, the jobtracker computes its initial trust in each VM's trustworthiness by dividing the sum of average unusual consumptions over all the metrics by the number of metrics that the VM has overused if any (Algorithm 4.3 - line 17), and underused (Algorithm 4.3 - line 22). If no metric has been overused or underused, the initial trust in the VM's trustworthiness would be set to 1 (Algorithm 4.3 - line 15 and line 20) respectively. Then, by computing the sum of $UpperInitialTrust_j$ and $LowerInitialTrust_j$ and dividing it by two (Algorithm 4.3 - line 25), we obtain the aggregate trust of each VM. Note finally that the whole process is repeated periodically after a certain period of time δ to continuously capture the dynamism in the VMs' performance and behavior.

C. Third Phase: Rank Aggregation via MCMC Gibbs Samples

In this section, we use Gibbs sampling to rank the VMs based on there trust values. The inputs of this step are the initial trust values obtain from the monitoring algorithm and the trust values obtained from the heartbeat phase. The average of both trust sources represents the samples at every time moment used to find the likelihood distribution, and we used the last time moment to represents the prior distribution. Using this information the objective of the Gibbs sampling phase is to come up with

the posterior or aggregate trust value of each VM at the current time moment (at which we are interested in computing the trust values).

Trust is calculated dynamically based on the initial trust values obtained from the monitoring algorithm and the trust value obtained using the heartbeat data. Based on the collected data from phase one and two each VM can be classified either trusted or not.

We will be using the MCMC methods in a Bayesian framework. Bayesian statistics is an interesting topic that we cannot hope to cover in the few lines here. Historically Bayesian statistics has been quite theoretical, as until about twenty years or so ago it had not been possible to solve practical problems through the Bayesian approach due to the intractability of the integrations involved. Therefore, readers are advised to consult the following references [17, 35] to know more about Bayesian statistics. In this work, we capitalize on the Bayesian approach to combine our prior trusted values with the data collected and produce new posterior trust values for each VM. Note that the methodology followed to compute the aggregate trust values for the VMs is inspired by [125].

Let $\chi_{V,\tau}$ be a matrix storing the trust values for all VMs $V = \{v_1,\ v_2,\ \ldots,\ v_m\}$ across time moments $\tau = \{\tau_1,\ \tau_2,\ \ldots,\ \tau_n\}$. We use $j \succ j'$ to denote that VM v_j is ranked higher than VM $v_{j'}$, which means that v_j is considered to be more trusted. Let e_{jh} be the trusted value of VM v_j at time moment τ_h. Furthermore, any e_{jh} having a highly trusted value would have a small position index in the matrix $\chi_{V,\tau}$, i.e. if $e_{jh} > e_{j'h}$, this means that $j \succ j'$. For clarification purposes, in the following discussion we use index j to denote the identifier of the VM and index h to denote the corresponding time moment at which the rank is computed, with m and n denoting the total numbers of VMs being ranked and time moments, respectively. In our work, each ranked VM is associated with

relevant covariates (CPU, RAM, bandwidth, and disk storage) that are used as a reference to determine the ranks of the VMs as per Algorithm 4.3. Table 4.2 shows the trust matrix $\chi_{V,\tau}$ of VMs V in different time moments τ from Algorithm 4.3.

Table 4.2: Trusting lists of VMs in different time moments (the matrix $\chi_{V,\tau}$)

VMs / Time Moments	τ_1	τ_2	τ_3	.....	τ_n
v_1	e_{11}	e_{12}	e_{13}	.....	e_{1n}
v_2	e_{21}	e_{22}	e_{23}	.....	e_{2n}
v_3	e_{31}	e_{32}	e_{33}	.....	e_{3n}
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$
v_m	e_{m1}	e_{m2}	e_{m3}	.....	e_{mn}

In Table 4.2, e_{jh} represents the trust value of VM $v_j \in V$ from the matrix $\chi_{V,\tau}$, where $j \in \{1, 2, ..., m\}$ at time moment h, and $e_{jh} > e_{j'h}$ if and only if $j \succ j'$. The set $\mu = (\mu_1, ..., \mu_m)$ represents the trust mean values of each VM across time moments. Let λ be the trust ranking list for all VMs, where these VMs are sorted in descending order on the basis of their trust values. The trust ranking list λ should satisfy the following conditions:

$$e_{jh} = \mu_j + \epsilon_{jh}, \qquad \epsilon_{jh} \sim N(0, \sigma^2), \qquad (1 \leq h \leq n; 1 \leq j \leq m) \tag{4.5}$$

where ϵ_{jh} is a small value and the different ϵ_{jh} values for each VM v_j and time moment h are jointly independent. Thus, to compute the trust value of the VMs (Equation 4.5), we fix $\sigma = 1$. Since we only observe the trust ranking list λ, multiplying by a constant or adding a constant to all the μ_j does not influence the likelihood function, and $e_{j_1h} > e_{j_2h}$ if and only if $v_{j_1} \succ v_{j_2}$ for the time moment h. The rank model in Equation 4.5 supposes that the τ values are independent and identically distributed (i.i.d.) conditionally on μ. Hence, the likelihood function becomes:

$$p(\tau_1, ..., \tau_{n-1}|\mu_j) = \prod_{h=1}^{n-1} p(\tau_h|\mu_j) = \prod_{h=1}^{n-1} \int_{\mathbb{R}^n} p(\tau_h|e_{jh}, \mu_j) p(e_{jh}|\mu_j) de_{jh} \tag{4.6}$$

where, $p(\tau_h|e_{jh}, \mu_j) = 1$. Our goal is to generate an aggregated rank based on an estimate of μ_j in Equation 4.6. We can employ a Bayesian method, which is more convenient to incorporate prior information, to quantify estimation uncertainties, and to utilize efficient Markov chain Monte Carlo (MCMC) algorithms. With a reasonable prior value, the posterior mean of μ_j would also be a consistent estimator under the same setting as in Equation 4.6. Let $p(e_{j_n})$ denote the prior probability. The posterior distribution of μ_j and $(e_{j_1}, ..., e_{j_n})$ is given in Equation 4.7 .

$$p(\mu_j, e_1..., e_{jn}|\tau_1, ..., \tau_n) = p(e_{jn}). \prod_{h=1}^{n-1} p(e_{jh}|\mu_j). \prod_{h=1}^{n-1} 1\{\tau_h = rank(e_{jh})\} \tag{4.7}$$

4.3.2 Second Stage: Task Clustering

A. First Phase: Percentile Method for Task Clustering based on their Resource Requirements

Having computed the trust scores of the VMs and ranked them based on their trust values, we are interested in this section in clustering the tasks on the basis of their resource requirements. To do so, we take advantage of the percentile method [49], which is used to partition the observations in a certain dataset into percentiles (e.g., 10^{th} percentile) based on their values. The task clustering process is depicted in Algorithm 4.4. The first step in the Algorithm is to sort the resource requirements of the tasks for each corresponding resource metric (i.e., CPU, RAM, storage, and bandwidth) (Step 10 in Algorithm 4.4). Then, we compute the lower rank (Step 11

in Algorithm 4.4) and higher rank (Step 12 in Algorithm 4.4) indices. The lower rank index represents the index that 25% of the observations fall below it. On the other hand, the higher rank index represents the index that 25% of the observations fall above it. These rank indexes are then split into integer (i.e, $Rank_{Integer}$) and fractional parts ($Rank_{Fraction}$). For example, the integer part of 3.4 would be 3 and the fractional part would be 0.4. Next, we determine the observations in the sorted dataset that correspond to the $Rank_{Integer}$ and $Rank_{Integer} + 1$ (Steps 13-16 in Algorithm 4.4) and save them into the variables $element_{value}$ and $element_{PlusOne}$ respectively (Steps 17-20 in Algorithm 4.4) . The lower and higher percentile values are then computed by interpolating between $element_{value}$ and $element_{PlusOne}$ values according to the $Rank_{Fraction}$. Finally, the actual clustering steps are executed by assigning the tasks whose resource requirements fall under the lower percentile value to the low requirement cluster k_{low} and the tasks whose resource requirements fall above the higher percentile value to the high requirement cluster k_{high} (Steps 24-27 in Algorithm 4.4). The remaining values are automatically assigned to the medium requirement percentile k_{medium} (Step 28-30 in Algorithm 4.4).

Algorithm 4.4 Task Clustering Algorithm

Inputs:

n: size of the dataset containing the requirements

T: set of task where $t_i \in T$

$M = \{CPU,\ memory,\ diskstorage,\ and\ bandwidth\}$: the set of VM's metrics to be analyzed by the cloud system

R_T^z: requirement array in terms of the metric $z \in M$ for the set T of tasks

R_i^z: requirement in terms of the metric $z \in M$ for the task $t_i \in T$

Outputs: Clusters $k_{high}^z, k_{medium}^z, k_{low}^z$, for each $z \in M$

Procedure Task Clustering

for each metric $z \in M$ **do**

Sort R_T^z

Compute Rank1 $E_1(R_T^z)$ such that $E_1(R_T^z) = 0.25 * (n-1) + 1$

Compute Rank2 $E_2(R_T^z)$ such that $E_2(R_T^z) = 0.75 * (n-1) + 1$

$Rank_{Integer1} = \lceil E_1(R_T^z) \rceil$

$Rank_{Integer2} = \lceil E_2(R_T^z) \rceil$

$Rank_{Fraction1} = E_1(R_T^z) - \lceil E_1(R_T^z) \rceil$

$Rank_{Fraction2} = E_2(R_T^z) - \lceil E_2(R_T^z) \rceil$

$element_{value1} = R_T^z[\lceil E_1(R_T^z) \rceil]$

$element_{value2} = R_T^z[\lceil E_2(R_T^z) \rceil]$

$element_{PlusOne1} = R_T^z[\lceil E_1(R_T^z) \rceil + 1]$

$element_{PlusOne2} = R_T^z[\lceil E_2(R_T^z) \rceil + 1]$

Compute the lower percentile value such that

$lower_{percentile} = element_{value1} + Rank_{Fraction1} \times (element_{PlusOne1} - element_{value1})$

Compute the higher percentile value such that

$higher_{percentile} = element_{value2} + Rank_{Fraction2} \times (element_{PlusOne2} - element_{value2})$

for each $t_i \in T$ **do**

if $R_i^z \leq lower_{percentile}$ **then**

Assign t_i to k_{low}^z

else if $R_i^z \geq higher_{percentile}$ **then**

Assign t_i to k_{high}^z

else

Assign t_i to k_{medium}^z

end if

end for

end for

end procedure

B. Second Phase: K-Means Clustering

K-means is one of the most widely adopted unsupervised learning techniques. It aims at clustering a set of observations in a certain dataset based on their degree of similarity. In this work, we employ K-means to group the tasks into five different clusters (i.e., very low, low, average, high, and very high) based on their monetary costs. K-means operates in an iterative manner through assigning each data observation to one of the K (in our case 5) groups based on some feature

similarity criteria (the feature is the cost in our case). Technically speaking, the K-means algorithm starts by defining K centroids, each pertaining to a given cluster. These centroids are chosen in such a way to be the farthest possible from one another. Then, each data observation is assigned to the nearest centroid, according to the *Euclidean distance*. Then, the mean of all the observations in each cluster is computed and identified to be the new centroid of the underlying cluster. This process keeps repeating until reaching the case where the same points are being assigned to each cluster in consecutive rounds.

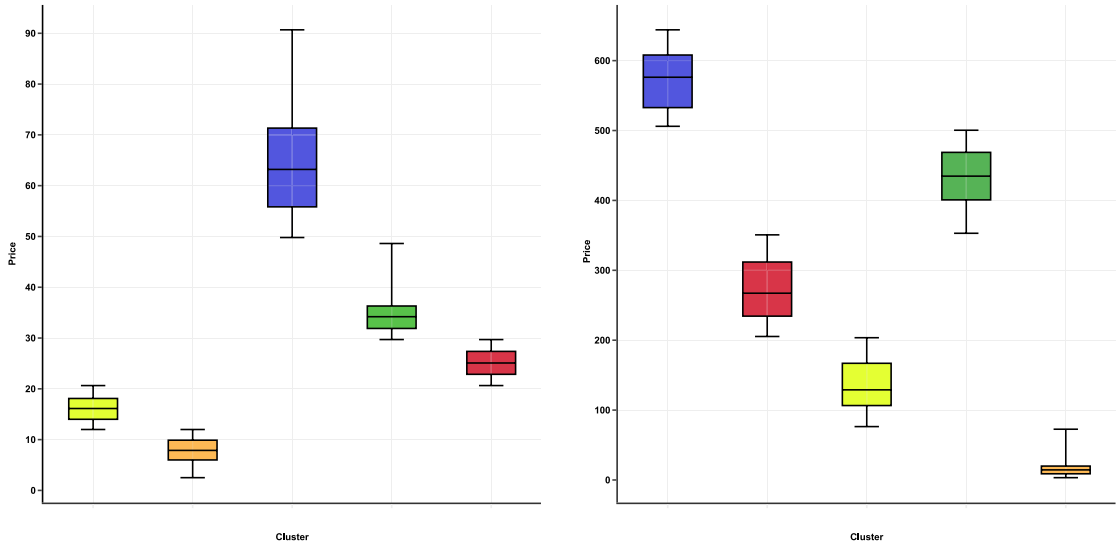
Let $X = \{x_j \mid j = 1, 2, \ldots, n\}$, be the set of n d-dimensional points to be clustered into a set of K clusters, $C = \{c_k \mid k = 1, 2, \ldots, K\}$. Mathematically, the K-means algorithm seeks to minimize the squared error function given by:

$$J(C) = \sum_{k=1}^{K} \sum_{x_j \in c_k} (\parallel x_j - \mu_k \parallel)^2 \tag{4.8}$$

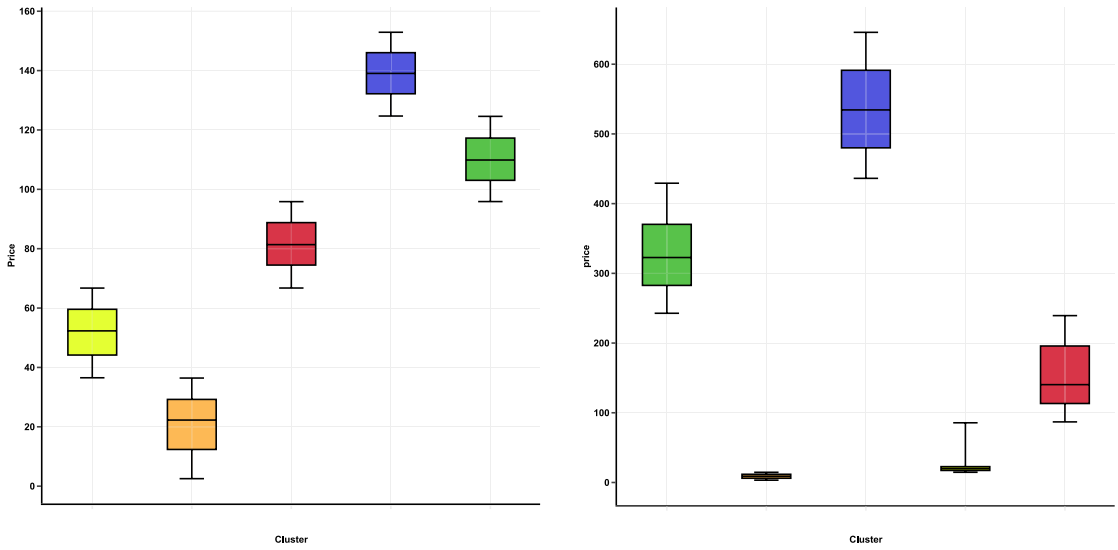
where $(\parallel x_j - \mu_k \parallel)^2$ is a chosen distance measure between a data point x_j and the mean value μ_k of cluster c_k , for all the data points from their respective cluster centers.

In Fig. 4.3, we show the output yielded after applying the K-means clustering technique on the cost values of our tasks, which were computed according to Google Cloud pricing list [2] for the CPU, RAM, bandwidth, and disk storage. By observing Fig. 4.3, we can notice that the cost range for the very high cluster (in blue) is between 50$ and 91$ for the CPU, 510$ and 850$ for the RAM, 125$ and 150$ for the bandwidth, and between 440$ and 830$ for the disk storage. The cost range for the high cluster (in green) is between 30$ and 42$ for the CPU, between 360$ and 500$ for the RAM, between 95$ and 125$ for the bandwidth, and between 240$ and 415$ for the disk storage. The cost range for the medium cluster (in red) is between

[2]https://cloud.google.com/pricing/list



a) Cost of tasks in terms of CPU b) Cost of tasks in terms of RAM



c) Cost of tasks in terms of bandwidth d) Cost of tasks in terms of disk storage

Figure 4.3: Clustering-based costs

21$ and 29$ for the CPU, between 205$ and 340$ for the RAM, between 54$ and 97$ for the bandwidth, and between 90$ and 230$ for the disk storage. The cost range for the low cluster (in yellow) is between 13$ and 21$ for the CPU, between 80$ and 205$ for the RAM, between 37$ and 68$ for the bandwidth, and between 40$ and 49$ for the disk storage. Finally, the cost range for the very low cluster (in orange) is between 5$ and 12$ for the CPU, between 2$ and 47$ for the RAM, between 10$ and 36$ for the bandwidth, and between 10$ and 30$ for the disk storage.

C. Third Phase: Multi-Criteria Task Priority Level Determination

Having clustered the tasks based on their resource requirements (Section 4.3.2.A) as well as their associated costs (Section 4.3.2.B), we are now ready to determine the priority level of each task. For this end, we adopt a multi-criteria decision-making technique [109] by following the subsequent steps:

- Determine the potential criteria (task cost clusters derived in Section 4.3.2.B) and alternatives (task requirement clusters derived in Section 4.3.2.A);
- Put the tasks in the decision matrix based on the clusters to which they belong;
- Compute the weight of each task, for each parameter (i.e., CPU, RAM, bandwidth, and disk storage), based on its position in the matrix; and
- Compute the overall priority for each task, for all the parameters.

Formally, consider a priority determination problem with a set $A = \{a_1, a_2, a_3\}$ of alternatives and a set $C = \{c_1, c_2, c_3, c_4, c_5\}$ of criteria. In the set A, a_1 represents the low resource requirements cluster. a_2 represents the medium resource requirements cluster, and a_3 represents the hight resource requirements cluster. In the set C, c_1 represents the very low task cost cluster, c_2 represents the low task cost cluster, c_3

represents the medium task cost cluster, c_4 represents the high task cost cluster, and c_5 represents the very high task cost cluster. The decision matrix representing the mapping between the set of potential criteria and set of alternatives is depicted in Matrix (4.9).

$$D^z(T) = \begin{array}{c} \\ a_1 \\ a_2 \\ a_3 \end{array} \begin{array}{c} \begin{array}{ccccc} c_1 & c_2 & c_3 & c_4 & c_5 \end{array} \\ \begin{pmatrix} G_{11} & G_{12} & G_{13} & G_{14} & G_{15} \\ G_{21} & G_{22} & G_{23} & G_{24} & G_{25} \\ G_{31} & G_{32} & G_{33} & G_{34} & G_{35} \end{pmatrix} \end{array} \tag{4.9}$$

In Matrix (4.9), each G_{ru} represents the position of the the task belonging to cluster a_r and cluster c_u in the decision matrix. Using this matrix, the priority degree formula for a task t_i is given in Eq. (4.10).

$$p_i = \frac{\sum_{z \in M} G^z_{ru}}{4 \times G_{35}} \times 100\%, \tag{4.10}$$

where G^z_{ru} is the position of the task t_i in the instantiation of the matrix $D^z(T)$, $z \in M$ and 4 represents the number of task resource parameters considered in this work. For example, assume that we have fifteen tasks $T = \{t_1, \cdots, t_{15}\}$. Then, we would have four different instantiations of the decision matrix $D^z(T)$: $D^c(T)$, $D^r(T)$, $D^b(T)$, and $D^s(T)$ for the CPU, RAM, bandwidth, and disk storage respectively. Assume that the fifteen tasks are distributed in the four decision matrices as follows:

$$D^c(T) = \begin{array}{c} \\ a_1 \\ a_2 \\ a_3 \end{array} \begin{array}{c} \begin{array}{ccccc} c_1 & c_2 & c_3 & c_4 & c_5 \end{array} \\ \begin{pmatrix} t_{11} & t_2 & t_3 & t_5 & t_9 \\ t_{15} & t_1 & t_6 & t_{14} & t_{12} \\ t_4 & t_8 & t_{13} & t_7 & t_{10} \end{pmatrix} \end{array}$$

$$D^r(T) = \begin{array}{c} \\ a_1 \\ a_2 \\ a_3 \end{array} \begin{array}{c} \begin{array}{ccccc} c_1 & c_2 & c_3 & c_4 & c_5 \end{array} \\ \begin{pmatrix} t_{13} & t_9 & t_5 & t_1 & t_2 \\ t_8 & t_3 & t_4 & t_{11} & t_6 \\ t_{12} & t_{15} & t_{14} & t_{10} & t_7 \end{pmatrix} \end{array}$$

$$D^b(T) = \begin{array}{c} \\ a_1 \\ a_2 \\ a_3 \end{array} \begin{array}{c} \begin{array}{ccccc} c_1 & c_2 & c_3 & c_4 & c_5 \end{array} \\ \begin{pmatrix} t_{14} & t_6 & t_{10} & t_{13} & t_8 \\ t_5 & t_{12} & t_1 & t_4 & t_9 \\ t_3 & t_7 & t_{15} & t_{11} & t_2 \end{pmatrix} \end{array}$$

$$D^s(T) = \begin{array}{c} \\ a_1 \\ a_2 \\ a_3 \end{array} \begin{array}{c} \begin{array}{ccccc} c_1 & c_2 & c_3 & c_4 & c_5 \end{array} \\ \begin{pmatrix} t_{11} & t_{13} & t_{15} & t_5 & t_3 \\ t_2 & t_9 & t_{10} & t_6 & t_7 \\ t_4 & t_{12} & t_8 & t_1 & t_{14} \end{pmatrix} \end{array}$$

Then, for the sake of simplicity and without loss of generality, we show in the following how the priority degree of three random tasks t_1, t_7 and t_{13} would be computed as per Eq. (4.10)

$$\begin{aligned} P_1 &= (\frac{4}{15}) + (\frac{4}{15}) + (\frac{6}{15}) + (\frac{12}{15})/4 \times 100\% \\ &= (0.27 + 0.27 + 0.4 + 0.8)/4 \times 100\% = 44\% \end{aligned}$$

$$\begin{aligned} P_7 &= (\frac{12}{15}) + (\frac{15}{15}) + (\frac{6}{15}) + (\frac{10}{15})/4 \times 100\% \\ &= (0.8 + 1 + 0.4 + 0.67)/4 \times 100\% = 72\% \end{aligned}$$

$$\begin{aligned} P_{13} &= (\frac{9}{15}) + (\frac{1}{15}) + (\frac{4}{15}) + (\frac{2}{15})/4 \times 100\% \\ &= (0.6 + 0.07 + 0.27 + 0.13)/4 \times 100\% = 27\% \end{aligned}$$

Thus, we can conclude that task t_1 has a priority degree of 44%, task t_7 has a priority degree of 72%, and task t_{13} has a priority degree of 27%. This means that, according to our scheduling algorithm described in Section 4.3.2.C, task t_7 should be scheduled first, followed by task t_1, and then task t_{13}.

4.3.3 Third Stage: Trust-Aware Task Scheduling Approach

Algorithm 4.5 is introduced to illustrate the task scheduling process of the tasks on the VMs. The Algorithm takes as inputs the queue of tasks ranked based on their priority degree (as explained in Section 4.3.2.B), the queue of VMs ranked based on their trust levels (as explained in Section 4.3.2.A), the VMs specifications in terms of CPU, RAM, bandwidth, and disk storage, and the task resource requirements in terms of CPU, RAM, band width, and disk storage.

The algorithm tries to match the tasks having the highest priority degrees with the VMs having the highest trust values, if and only if those VMs have enough hardware and network resources that satisfy the task requirements. This process keeps repeated until all tasks are assigned to VMs. In this way, we guarantee that the untrusted VMs would be the last choice of the scheduling algorithm to assign tasks to, and that the tasks enjoying high priorities would be mapped to the VMs that are the most trusted.

Algorithm 4.5 Scheduling Algorithm

Inputs:

T**:** set of task where $t_i \in T$

V**:** set of VMs where $v_j \in V$

Q_T: queue of tasks ranked based on their priority values

Q_V: queue of VMs ranked based on their trust values

CPU_j^x: current CPU available on VM v_j at time x

RAM_j^x: current RAM available on VM v_j at time x

BW_j^x: current Bandwidth available on VM v_j at time x

DS_j^x: current Disk Storage available on VM v_j at time x

R_i^c**:** the CPU requirements of task t_i

R_i^r**:** the RAM requirements of task t_i

R_i^s**:** the Disk Storage requirements of task t_i

R_i^b**:** the Bandwidth requirements of task t_i

procedure Task Scheduling

    for each $t_i \in Q_T$

        repeat

            $h_v = head(Q_v)$

            if $R_i^c \leq CPU_{h_v}^x$ **and** $R_i^r \leq RAM_{h_v}^x$ **and** $R_i^b \leq BW_{h_v}^x$ **and** $R_i^s \leq DS_{h_v}^x$

                assign t_i to h_v

            else

                $h_v = h_v.next$

            end if

        until t_i is assigned to a VM

    end for

end procedure

4.4 Experiments and Empirical Analysis

4.4.1 Experimental Setup

To conduct our experiments, we employ a datset collected by Bitbrains[3], a service provider that is specialized in managed hosting and business computation for enterprises. The dataset consists of $1,750$ VMs running in a distributed datacenter and serving up to $8,260$ tasks. The information contained in the dataset include: timestamps, number of provisioned virtual CPU cores, CPU capacity of each VM, CPU usage of each VM, amount of memory provisioned to each VM, memory usage per VM, disk read and write throughput per VM, and network received/transmitted throughput. We consider both trusted and untrusted VMs, and vary the percentage of untrusted VMs from 10% to 50%. Untrusted VMs are considered those that exhibit poor performance in serving customers' requests and consume abnormal amounts of resources (e.g., CPU, RAM, etc.). To implement the k-means clustering approach, we vary the number of clusters from 1 to 5 to determine the optimal number of clusters that minimizes the percentage of outliers. Based on the experiment results, we could conclude that a number of 5 clusters achieves the best performance on the considered dataset.

The objective of the first set of experiments is to study the performance of our solution in terms of execution time and scheduling time. Minimizing these metrics is of prime importance to guarantee efficient execution of big data requests in realistic industrial environments. In the second set of experiments, we measure the monetary cost entailed by our approach for serving tasks. This is also an important factor for both providers and customers to help them save resources and money. We compare our approach with three algorithms i.e, SJF, RR, and improved PSO. The SJF approach

[3]http://gwa.ewi.tudelft.nl/datasets/gwa-t-12-bitbrains

considers only the size of the tasks to derive the order of their execution. The main purpose of the RR approach is to support time sharing system among tasks on the VMs. The main idea of the improved PSO approach is to change the weights of the particles with the increase in the number of iterations and to inject some random weights in the final stages to avoid the case where the PSO algorithm generates local optimum solutions.

Note that the experiments have been conducted using the CloudSim simulator, which allows us to obtain a realistic cloud environment. The experiments have been performed in a 64-bit Windows 7 environment on a machine equipped with an Intel Core i7-6700 CPU 3.40 GHz Processor and 16 GB RAM.

4.4.2 Experimental Results

In Fig. 4.4, we measure the total makespan of the tasks entailed by the different studied solutions, while varying the number of deployed VMs from 10 to 50. For this experiment, the number of tasks has been fixed to $7,884$ and assume that all the considered VMs are all trustworthy. By observing the figure, we notice that our solution decreases the total makespan compared to the other solutions especially in a low-density VMs environment (i.e., when the number of deployed VMs is relatively small). The reason is that in our solution, we provide a matching algorithm that intelligently maps the resource requirements of the tasks with the resource capabilities of the VMs. In other words, we decrease the probability of assigning tasks that require significant amounts of resources to low-performing VMs that would not be capable of efficiently serving those tasks. The second observation from this figure is that by increasing the number of deployed VMs, the performance gap between the different solutions tends to be smaller. This can be justified by the fact that the larger the number of VMs is (assuming that all these VMs are trustworthy), the

larger the resources available to serve tasks would be. Nonetheless, by improving the performance in low-density VMs, our solution would aid cloud providers with decreasing their overall costs through decreasing their need to deploy more VMs.

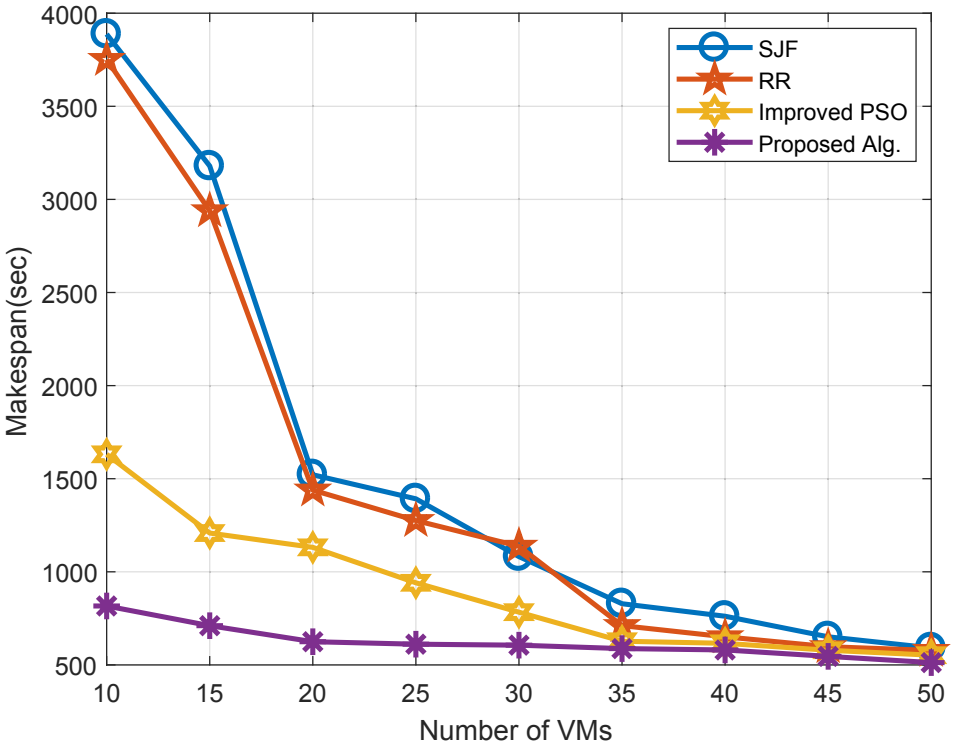


Figure 4.4: **Task makespan**: Our solution significantly decreases the tasks makespan compared to the three considered solutions

In Fig. 4.5, we study the monetary cost entailed by the different studied solutions for the cloud providers, while varying the number of deployed VMs from 10 to 50. For this experiment, the number of tasks has been fixed to $7,884$ and assume that the considered VMs are all trustworthy. By observing the figure, we notice that our solution considerably help providers reduce their costs compared to the other solutions. The reason for this observation is that our solution decreases the

total makespan of the tasks as shown in Fig. 4.4.

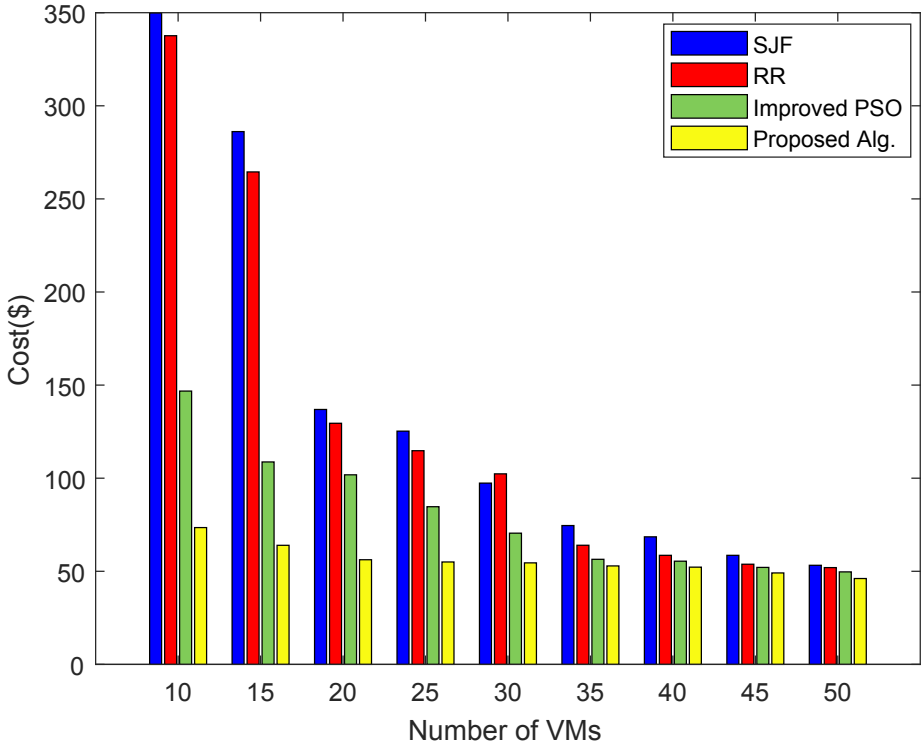


Figure 4.5: **Task costs**: Our solution is able to decrease the cost of task execution in comparison with the three considered models

In Fig. 4.6, we measure the time spent by the tasks waiting in the queue to be assigned to VMs. Like the previous experiments, we vary the number of deployed VMs from 10 to 50, while fixing the number of tasks to $7,884$ and assuming that all the considered VMs are all trustworthy. The main observation that can be drawn from this figure is that increasing the number of deployed VMs leads to a considerable decrease in the waiting times. While this result is expected, the objective of this figure is to show that even for a small number of deployed VMs and a quite large number of tasks (i.e, $7,884$), the waiting time entailed by our model is still acceptable. For

instance, the total waiting time for 7,884 to be assigned to 10 VMs is 18 seconds.

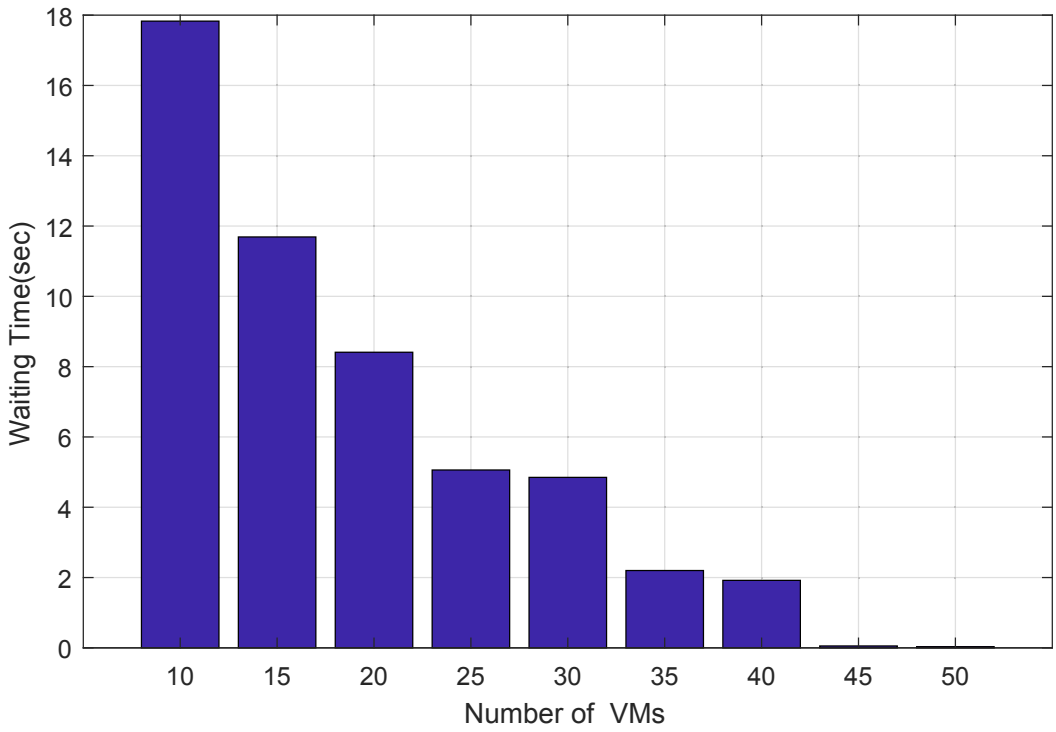


Figure 4.6: **Waiting time**: The waiting time of tasks prior to being assigned to VMs significantly decreases with the increase in the number of deployed VMs

In Fig. 4.7, we measure the makespan of the tasks while varying the percentage of untrusted VMs from 10% to 50%. For this experiment, we fix the number of deployed VMs to 50 and the number of tasks to 7,884. We notice from this figure that starting from 10% of untrusted VMs, our solution can considerably reduce the makespan compared to the other approaches. The reason behind this improvement is that in our solution, we employ a two-step trust establishment mechanism and MCMC Gibbs Sampler-based aggregation technique to derive the trust value of each VM and rank them based on their degree of trustworthiness. This is important to

avoid assigning tasks to untrusted VMs whose poor performance might affect the overall timespan and success chances of the task execution process. In other words, our scheduling technique maps the tasks to the VMs that enjoy high trust values (i.e., highly performing VMs) to ensure minimal makespan and cost. This means that untrusted VMs would have quite poor chances of being assigned any tasks. On the other hand, despite the effectiveness of the other compared approaches, these approaches ignore the trustworthiness of the VMs in the scheduling process, thus increasing the chances of tasks being assigned to poorly performing VMs.

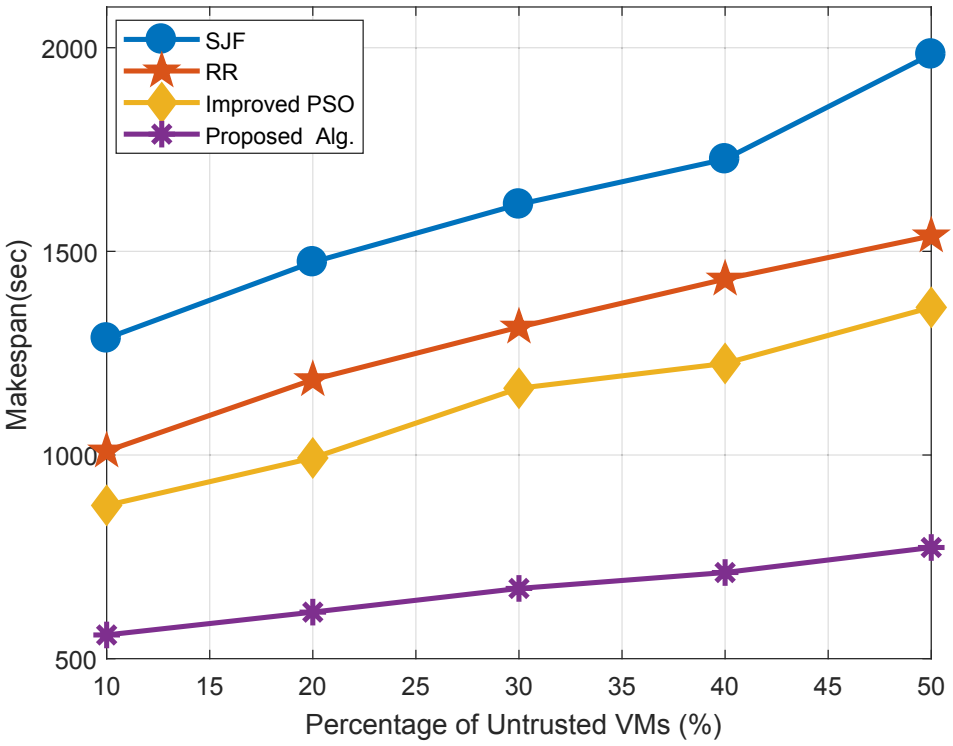


Figure 4.7: **Task makespan**: Our solution significantly decreases the tasks makespan compared to the three considered solutions in the presence of untrusted VMs

In Fig. 4.8, we quantify the monetary cost of the tasks at the side of the

providers, while varying the percentage of untrusted VMs from 10% to 50%. For this experiment, like the previous experiments we fix the number of deployed VMs to 50 and the number of tasks to $7,884$. The figure reveals that our solution aids providers in reducing their monetary costs compared to the other approaches in the presence of untrusted VMs. This can be justified by the fact that, according to Fig. 4.7, our trust-based approach could decrease the makespan of the task scheduling and execution processes, in the presence of untrusted VMs. This, in turn, results in reducing the amount of resources spent on serving the tasks, which leads to decreasing the monetary costs of the providers.

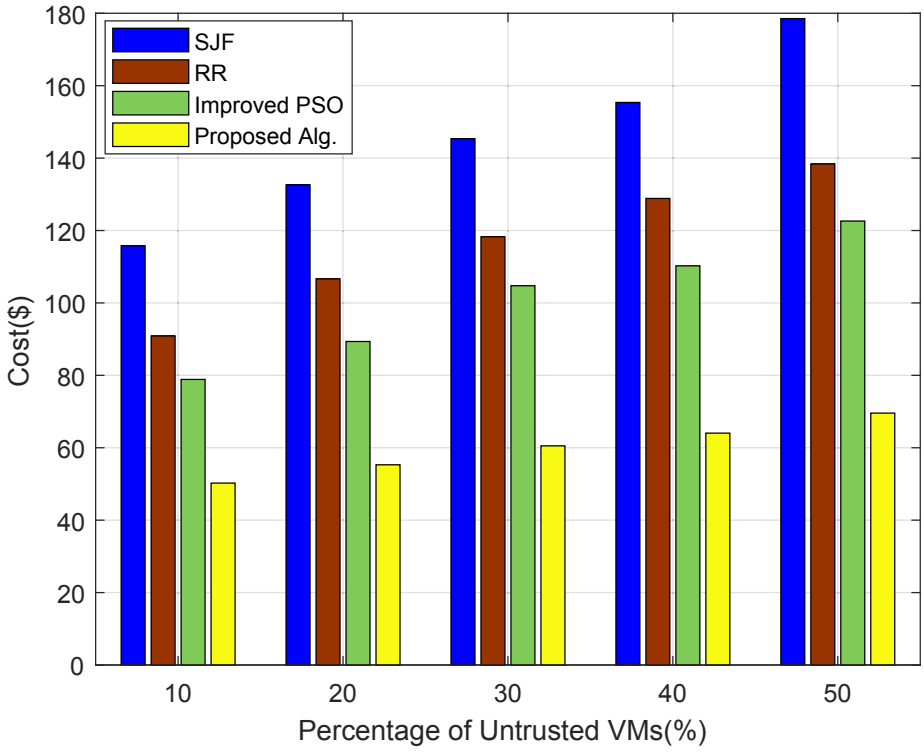


Figure 4.8: **Task cost**: Our solution significantly decreases the costs of tasks compared to the three considered solutions in the presence of untrusted VMs

In Fig. 4.9, we study the runtime of the four compared solutions, while varying the number of VMs from 10 to 50. For this experiment, we fix the number of tasks to $7,884$. Execution time refers to the time needed by the whole set of algorithms that make up the solution to execute. According to the figure, our approach enjoys lower runtime than the other solutions. The reason is that, although our solution consists of many phases (i.e., initial trust computation, VMs monitoring, MCMC-based trust aggregation, K-means-based task cost clustering, percentile-based task requirement clustering, multi-criteria task priority determination, and trust-based task scheduling), adopting a trust-based scheduling approach reduces the chances of tasks failing on some untrusted VMs and their rescheduling. Moreover, the VMs' trust-based clustering and tasks priority determination helps our solution improve the mapping between tasks' resource requirements and VMs' resource availabilities. On the other hand, the SJF solution, for example, focuses only on the size of the tasks to determine the order of their execution. This has the limitation of increasing the probability of some tasks being allocated to untrusted VMs, thus raising the need for rescheduling those tasks in cases of VM failures. By carefully observing Fig. 4.9, also reveals that the improved PSO exhibits the highest runtime amongst the other solutions. The reason is that the improved PSO needs to go through many iterations in order to converge to the desired solution that minimizes the scheduling process makespan.

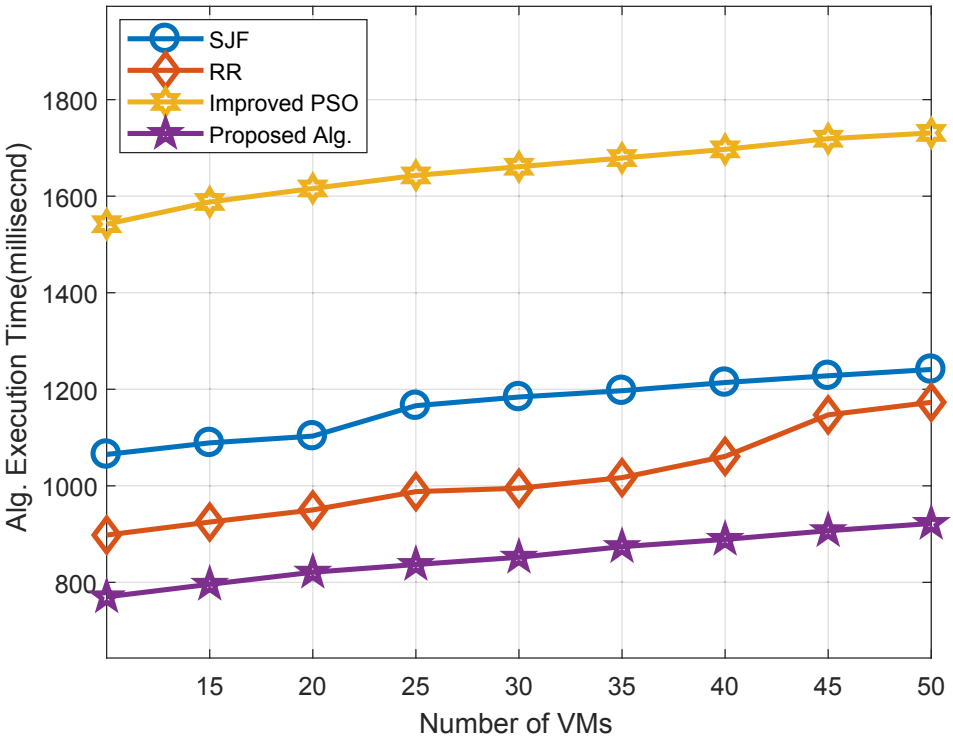
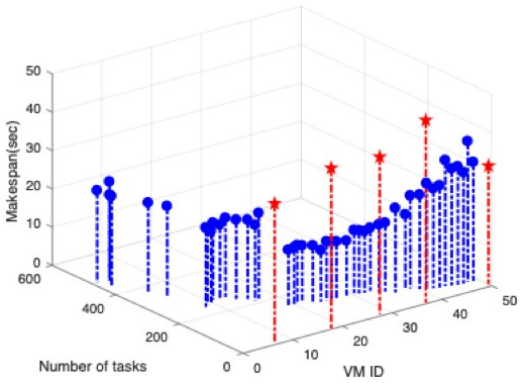


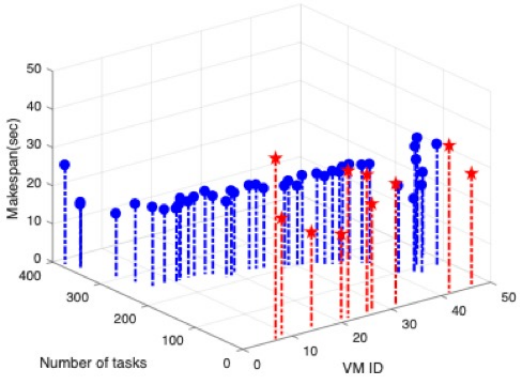
Figure 4.9: **Execution time**: The execution time of our solution is lower than that of the three other compared approaches.

In Fig. 4.10, we provide an in-depth breakdown of our scheduling solution by showing, for each VM, the number of assigned tasks and the makespan of the process of serving these tasks, under a varying percentage of untrusted VMs. In this figure, the red plots refer to the untrusted VMs, while the blue plots refer to the trusted ones. The main observation that can be drawn from this figure is that our solution assigns only a small number of tasks to the untrusted VMs compared to the trusted ones (for example, in Fig. 4.10a, the highest number of tasks that were assigned to an untrusted VM was 10 out of $7,884$). The second important observation is that the trusted VMs (in blue) entail small makespan even when assigned a large number

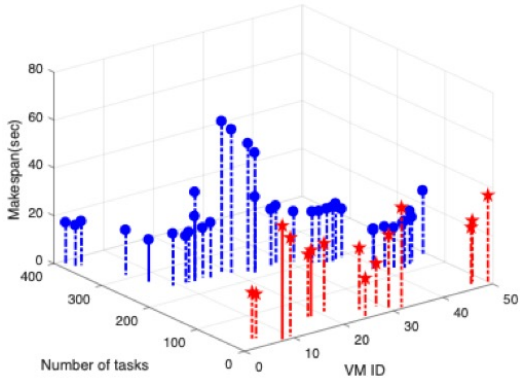
of tasks, compared to the untrusted ones that are assigned a negligible number of tasks. For example, we can notice from Fig. 4.10a that, among the trusted VMs, the makespan of the VM that was assigned the largest number of tasks ($\approx$ 490 tasks) has been $\approx$ 22s. On the other hand, among the untrusted VMs, the makespan of the VM that was assigned the largest number of tasks ($\approx$ 15 tasks) has been $\approx$ 35s. This shows that even when assigned a considerably smaller number of tasks, untrusted VMs entail higher makespans compared to the trusted VMs that serve larger pools of tasks.



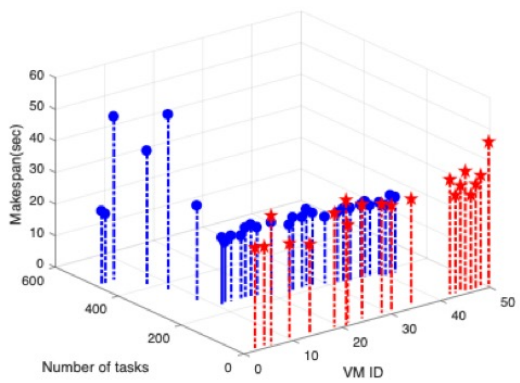
(a) 10% of untrusted VM



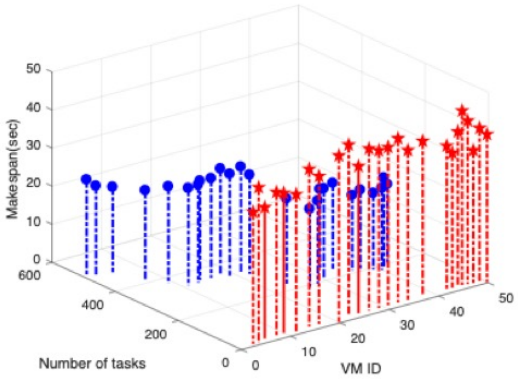
(b) 20% of untrusted VM



(c) 30% of untrusted VM



(d) 40% of untrusted VM

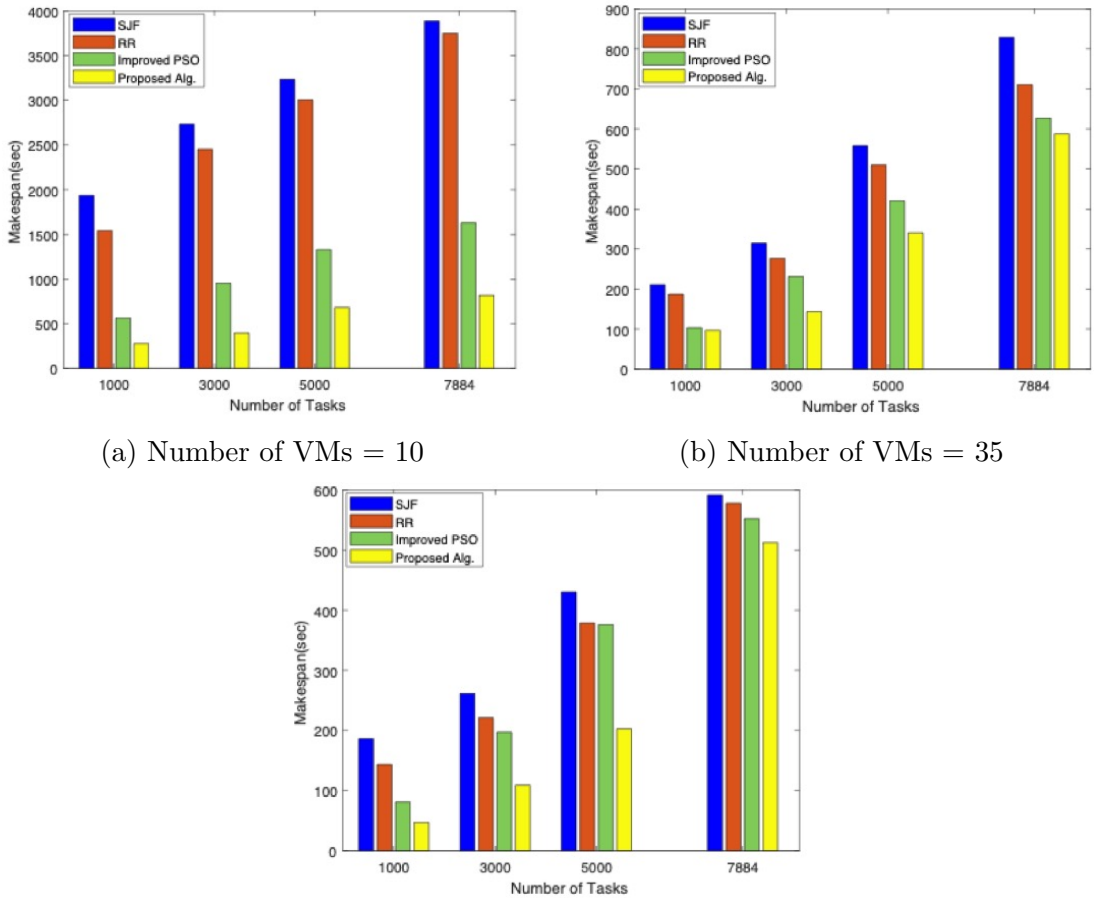


(e) 50% of untrusted VM

Figure 4.10: Clustering-based costs

In Fig. 4.11, we study the total makespan while varying both the number of tasks and number of deployed VMs. We notice the total makespan keeps increasing

with the increase in the number of tasks. The reason is that the larger the number of tasks that need to be served is, the more the time it would take to serve them, for a fixed number of VMs. However, by increasing the number of deployed VMs, we can reduce the makespan needed to serve an increasing number of tasks. Under this scenario, our solution shows the ability to reduce the makespan compared to the other solutions, while showing a greater scalability to the increase in both the number of tasks and number of VMs. This is particularly useful in production environments that require serving huge numbers of tasks.



(a) Number of VMs = 10

(b) Number of VMs = 35

(c) Number of VMs = 50

Figure 4.11: **Task makespan**: We study in this figure the impact of varying both the number of tasks and number of VMs on the overall makespan of the tasks

4.4.3 Conclusion

In this chapter, we investigated the challenges of achieving high-performance and trustworthy big data task scheduling in cloud computing environments. In particular, we proposed *BigTrustScheduling*, a trust-based scheduling approach that is particularly useful for big data tasks. The main idea is to derive a trust value for each VM based on its underlying performance, and then prioritize tasks based on their resource requirements and associated prices.

The proposed scheduling solution intelligently maps the tasks to the appropriate VMs in such a way that minimizes the makespan and cost of tasks execution. While this trust-based approach is important to improve the QoS of the big data analytics process, more challenges have to be taken into consideration to further optimize the big data task execution in cloud environments. Specifically, with the increasing number of big data tasks received by cloud providers, there should be an effective and efficient approach that would help providers automate the scheduling process so as to reduce the manual intervention, which often results in degraded performance and is error-prone. Therefore, the results of this chapter are used in Chapter 5 and Chapter 6 for the purpose of building an automated big data scheduling approach to be used toward the accomplishments of Objective 2 and Objective 3 discussed in Chapter 1.

Chapter 5

Deep and Reinforcement Learning for Automated Task Scheduling in Large-Scale Cloud Computing Systems

Cloud computing is undeniably becoming the main computing and storage platform for today's major workloads. From IoT and Industry workloads to big data analytics and decision-making jobs, cloud systems daily receive a massive number of tasks that need to be simultaneously and efficiently mapped onto the cloud resources. Therefore, deriving an appropriate task scheduling mechanism that can both minimize tasks' execution delay and cloud resources utilization is of prime importance. Recently, the concept of cloud automation has emerged to reduce the manual intervention and improve the resource management in large-scale cloud computing workloads [8]. In this chapter, we capitalize on this concept and propose four deep and

reinforcement learning-based scheduling approaches to automate the process of scheduling large-scale workloads onto cloud computing resources, while reducing both the resource consumption and task waiting time. These used approaches are: reinforcement learning (RL), deep Q networks (DQN), recurrent neural network long short-term memory (RNN-LSTM) and deep reinforcement learning combined with LSTM (DRL-LSTM).

5.1 An Overview of The Proposed Approach

5.1.1 Solution Overview

The topic of task scheduling in cloud computing environments has been extensively addressed in the literature. The current scheduling approaches can be categorized into two main classes, i.e., traditional approaches and intelligent approaches. Traditional approaches focus on tuning and extending conventional scheduling approaches such as First-In-First-Out (FIFO), Shortest Job First (SJF), Round-Robin (RR), Min-Min and Max-Min [13,24,37] to fit the cloud computing settings. The main limitation of the traditional approaches is that they can support only a limited number of parameters (e.g., makespan) to optimize. This makes them unsuitable for the cloud computing environment in which many parameters such as task Makespan and CPU, memory, and bandwidth costs need to be simultaneously optimized.

Intelligent approaches, on the other hand, [7, 10, 38, 40, 80, 86, 100, 116, 133] capitalize on artificial intelligence techniques such as fuzzy logic, Particle Swarm Optimization (PSO) and Genetic Algorithm (GA) to devise more solid scheduling techniques optimizing several parameters simultaneously. However, similar to the traditional approaches, the intelligent scheduling approaches operate in an offline fashion through attempting to optimize a series of parameters upon the receipt

of a certain task. This causes high execution time, which makes it inefficient for delay-critical tasks such as Internet of Things (IoT) and big data analytics tasks.

Recently, many attempts [6, 64, 81, 92, 101, 113, 114] have been made to leverage the booming advancements in the field of machine learning, especially deep learning, to automate the resource management process in the cloud system. These approaches are mainly based on the idea of examining historical resource data from VMs in order to predict the future workload. The objective is to improve the resource management and avoid the under and over-provisioning cases. In this chapter, we investigate the application of four deep and reinforcement learning approaches to automate the process of scheduling tasks over the cloud.

5.1.2 Reinforcement Learning (RL)

The main idea of RL [105] is to teach a certain agent how to behave and adapt to the changes that take place in its environment. Specifically, we employ the Q-learning algorithm to learn an optimized scheduling policy by considering the future decisions and evaluating the feedback from the cloud environment. Let $E = \{e_1, e_3, \ldots, e_n\}$ be a set of tasks submitted by a number of users and $V = \{v_1, v_2, \cdots, v_m\}$ be a set of VMs. Moreover, let s_t represent the cloud scheduler state at time moment t in the state space S and a_t be an action from the action space A at time moment t with probability $\mathcal{P}(\grave{s}|s, a) = \mathcal{P}[s_{t+1} = \grave{s}|s_t = s, a_t = a]$, where $\sum_{\grave{s} \in S} \mathcal{P}(\grave{s}|s, a) = 1$ [67]. The cloud scheduler policy in our model $\pi(a|s)$, which maps states to actions, assigns every task e_i to a VM v_j. The immediate reward of taking action a_t in state s_t is r_t. The objective of the cloud scheduler is to find an optimum scheduling policy that minimizes the cumulative reward value (i.e., cost) for all the considered VMs and tasks. The states, actions, and reward function of our RL solution are as follows.

- **State Space**

  At each time t, the state s_t represents the current scheduling of the tasks on the VMs and each VM v_j is described in terms of the available resources (CPU, RAM, bandwidth, and disk storage). A task e_i can be assigned to any VM v_j which meets the resource constraints that will be defined latter in this section.

- **Action Space**

  The action a_t represents the scheduling action at time t of all the considered tasks on the available VMs. For each task, the action can be presented as (0 or 1), which means the cloud scheduler can assign a task e_i to a VM v_j or not. Technically speaking, when a task e_i is assigned to a VM v_j, the action space w.r.t that task is presented as a vector of size m, e.g.: $(0, 1, 0, 0, \ldots, 0)$, which indicates the task e_i is assigned to the second VM.

- **Reward**

  The reward function is used to represent the task scheduling process efficiency. If task e_i is assigned to VM v_j, we define the execution cost $\zeta_{i,j}$ as an immediate individual reward. The overall reward at time t r_t is the sum of all the costs. The individual reward is defined in terms of the amounts of CPU, RAM, bandwidth, and disk storage as follows:

$$\zeta_{i,j} = (\psi_{i,j} + \varphi_{i,j}) \times P_j \tag{5.1}$$

  where $\varphi_{i,j}$ is the waiting time for e_i to be assigned to v_j, P_j is the unit price of each virtual machine v_j, and $\psi_{i,j}$ is the execution time of e_i on v_j.

In our optimization scheduling model, we employ the Q-learning method to evaluate the feedback from the cloud system environment to optimize future decision-making.

After collecting each reward, the mean Q-value of an action a on state s following the policy π is denoted $Q_\pi(s,a)$ and the optimal value function is:

$$Q^*(s,a) = min_\pi Q_\pi(s,a) \tag{5.2}$$

This optimal value function can be nested within the Bellman optimality equation as follows:

$$Q^*(s,a) = \sum_{\grave{s}} \Upsilon(\grave{s}|s,a)[r + \gamma \min_{\grave{a}} Q^*(\grave{s},\grave{a})] \tag{5.3}$$

where γ is the discount factor to what degree the future reward is affected by the past actions, and Υ denotes the transition probability of going from the current state s to the next state $\grave{s}$ under action a. We assume that the resource demands of each task is known upon arrival and for a task e_i to be assigned to VM v_j the following conditions should be met:

$$K_i^{CPU} \leq CPU_j^t; K_i^{RAM} \leq RAM_j^t; K_i^{BW} \leq BW_j^t; K_i^{DS} \leq DS_j^t \tag{5.4}$$

where $K_i^{CPU}, K_i^{RAM}, K_i^{BW}$, and K_i^{DS} are the task CPU, RAM, bandwidth, and disk-storage requirements respectively, and CPU_j^t, RAM_j^t, BW_j^t, and DS_j^t are the current amounts of available VM CPU, RAM, bandwidth, and disk-storage specification respectively. After that, the cloud scheduler evaluates $Q_\pi(s,a)$ for the current policy π. Then, the policy is updated as follows,

$$\grave{\pi} = arg \min_a Q_\pi(s,a) \tag{5.5}$$

Finally, the main objective of this model is to determine the optimal policy π^* leading

to minimize the reward of any state s:

$$\min_{\pi^*} E[Q_{\pi^*}(s,a)], \qquad \forall s \in S \tag{5.6}$$

5.1.3 Deep Q Networks (DQN)

The goal of deep reinforcement learning (DRL) is to learn the optimal policy that maximizes the total discounted reward through a process of exploration and exploitation. The discounted reward is considered since the aim is to maximize the future reward in the long run, rather than the immediate next reward. In this chapter, we use DQN, a specific type of DRL that combines Q-learning and deep learning. In our problem, If we memorize all the Q-values in the Q-table, the imaginable state may be more than ten thousands, and the matrix $Q(s,a)$ would be very large. Therefore, we use a deep neural network (DNN) as approximation to estimate $Q(s,a)$ instead of computing Q-value for each state action pair (s,a). This is important to model large-scale scheduling scenarios that are characterized by a large number of actions-state pairs. In our training process, the DRL cloud scheduler selects a random scheduling action (i.e., assigning tasks e_i to VMs v_j) with a high probability to explore the effect of the unknown scheduling alternatives and obtain a better strategy. The cloud scheduler increases the probability of choosing the action with the highest Q value during the training process, to minimize the expected cumulative reward (execution cost), which employs the Bellman equation (Equation 5.3). Technically speaking, the cloud scheduler chooses to schedule one or more waiting tasks at each time moment t subject to the conditions in Equation 5.4. The optimal $Q-value$ function indicates that at time t, each scheduling policy π selects a valid VM from the set V to execute each task from the set E so as to minimize the overall execution cost. As in our RL model (Section 5.1.2), the cloud scheduler chooses an action $a \in A$

on a state $s \in S$ depending on the behavioral policy $\pi(a|s)$. The goal of our model is to optimize π^* that minimizes the expected cumulative reward (i.e., cost), where the immediate reward at time t of taking action a_t in state s_t is r_t. We obtain the actual Q-value of action a, by using the state s as input of the online network, and $\grave{s}$ as input to the target network to obtain the minimum Q-value of all actions in the target network. We use the Mean Square Error (MSE) to define the loss function and the Bellman equation (Equation 5.3) to minimize the loss function $L_u(\beta_u)$ as follows:

$$L_u(\beta_u) = E_{(s,a,r,\grave{s})}[(r + \gamma \min_{\grave{a}} Q(\grave{s}, \grave{a}|\grave{\beta}_u) - Q(s, a|\beta_u))^2] \tag{5.7}$$

where β represents the parameters of the online network in the u^{th} iteration, $\grave{\beta}$ represents the parameters of the target network in the u^{th} iteration [74], and $E_{(s,a,r,\grave{s})}[.]$ denotes the expected value of the next reward given the current state s and action a, together with the next state $\grave{s}$.

5.1.4 RNN-LSTM

Long short-term memory (LSTM) is a recurrent neural network (RNN) equipped with an input gate, output gate and a forget gate as shown in Fig. 5.1 along with three cells: state, output and input. The network structure of the LSTM units that we integrate into our DRL approach is depicted in Fig. 5.2. Let p denote the input gate, o denote the output gate and f denote the forget gate. Moreover, we denote by C the cell state, h the cell output, and by x the cell input. The equations that are used to compute the LSTM gates and states are given as follows:

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f) \tag{5.8}$$

$$p_t = \sigma(W_p.[h_{t-1}, x_t] + b_p) \tag{5.9}$$

$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o) \tag{5.10}$$

where W are the weights for each of the gates, the b terms denote bias vectors, and σ is the logistic sigmoid function.

$$\tilde{C}_t = \tanh(W_C.[h_{t-1}, x_t] + b_C) \tag{5.11}$$

where $\tilde{C}$ is the updated cell state and tanh is the hyperbolic tangent function. Given the value of the input gate activation p_t, the forget gate activation f_t and the candidate state value $\tilde{C}_t$, we can compute C_t, the memory cells' new state at time t as follows:

$$C_t = f_t \times C_{t-1} + p_t \times \tilde{C}_t \tag{5.12}$$

$$h_t = o_t \times \tanh C_t \tag{5.13}$$

The proposed RNN-LSTM prediction method has three parts: data pre-processing, model training, and model prediction. First, the collected data, i.e., (CPU_j^t, RAM_j^t, BW_j^t, and DS_j^t) are re-sampled to match the time moments. Second, the feature vectors of each resource type data are extracted. By training the LSTM recurrent neural network, the error between output o_t and real value is continuously reduced. Moreover, the LSTM unit can store long-term information and is suitable for long-term training. In our model, we focus on learning the output of the next state given the current state of the model. Thus, the model represents the probability distribution of sequences in the most general form unlike other models that assume

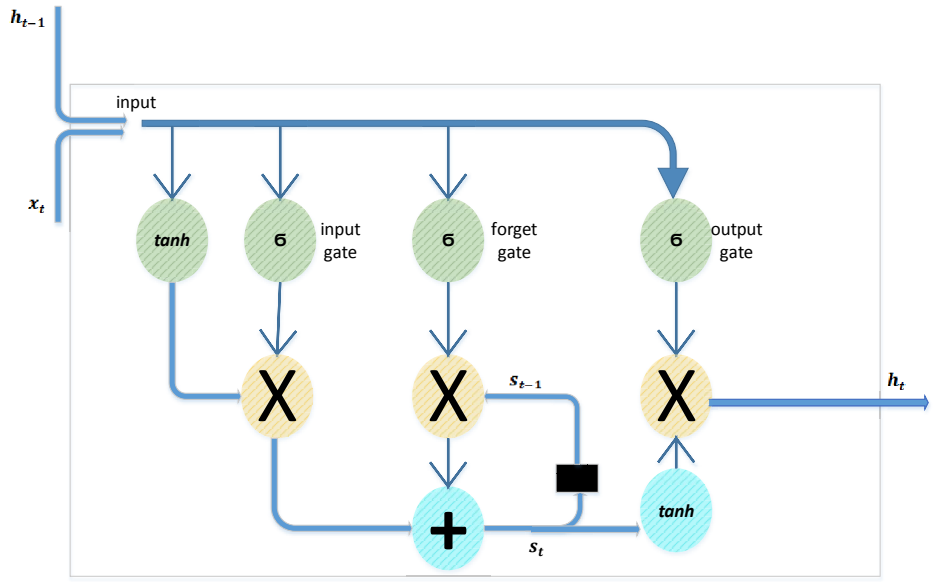


Figure 5.1: Architecture of an LSTM cell

independence between outputs at different time steps, given latent variable states.

We use two neural network layers in our model. At each time moment t, the first LSTM-based network layer is employed for cost prediction, where the outputs are all the VMs predicted states based on the history information. The predicted states are then used as inputs to the second layer. Next, with the output of the second layer, the cloud scheduler assigns the tasks to the VMs. Thereafter, the selected VMs execute

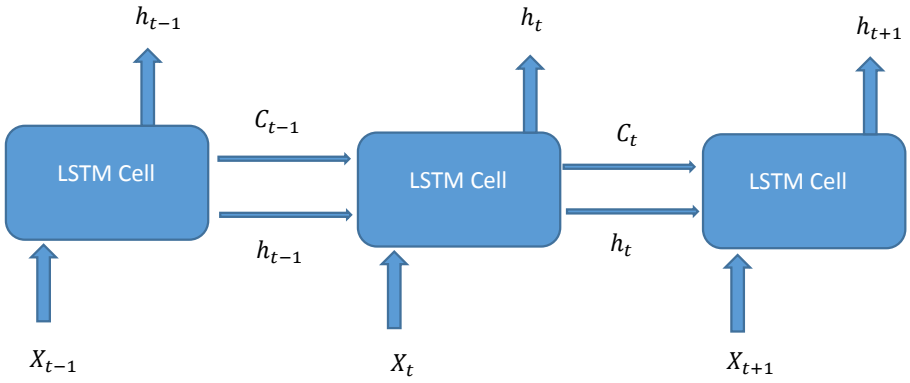


Figure 5.2: Reinforcement learning with LSTM (RL-LSTM)

the policy and transmit their data to the cloud scheduler, along with their current true states, which will be stored into the history information for future prediction usage. The cloud scheduler finally receives rewards. All the VMs complete there tasks and store the cost and the resources usage for future exploitation. We repeat this process in the next time moments, until the process converges when all the tasks get assigned with a minimum cost.

5.2 Experiments and Empirical Analysis

5.2.1 Dataset

To carry out our experiments, we employ the Google cluster dataset, which contains data on the resource requirements and availability for both tasks and VMs. The Google cluster consists of many machines that are connected via a high-speed network. The dataset includes $670,000$ traces, where approximately 40 million task events across over 12000 machines for 30 days have been recorded [84, 118]. The dataset consists of the following features: start time, end time, job ID, machine ID, task index, CPU rate, maximum CPU rate, assigned memory usage, maximum memory usage, canonical memory usage, unmapped page cache, total page cache, disk I/O time, local disk space usage, maximum disk I/O time, cycles per instruction, aggregation type, memory accesses per instruction, sample portion, and sampled CPU usage. In the training part of the LSTM, we first initialize the weights of both the input and output layers as a normal distribution whose mean is 0 and standard deviation is 1. The bias for both layers is set to be 0.1. The Multi-Layer Perceptron (MLP) structure consists of three hidden layers and one LSTM layer. The three hidden layers have 512, 256, and 128 output dimensions respectively. In addition, a Rectified Linear Unit (ReLU) is used for the activation function. The output of the MLP is the input to the LSTM.

5.2.2 Validation Metrics

In the first set of experiments, we aim to study and compare the training and test accuracy of the different proposed deep and reinforcement learning-based scheduling approaches (i.e., RL, DQN, RNN-LSTM, and DRL-LSTM). In both cases, the accuracy means the accuracy of predicting the appropriate VMs to host each incoming task. It is computed with regard to a ground Truth.

For the reinforcement and deep reinforcement learning, our scheduling agent performs actions in the MDP environment populated by the dataset and learns from the obtained reward at each state to select the next one using our policy function. The training is also done on the data generated by the agent. At each state, the agent tries to find the best action that minimizes the reward. The actions that the agent made at the corresponding states in one learning run, will become part of the training dataset for the next run. For the testing, the MDP is populated by new data.

We train the scheduling policy in an episodic setting. In each episode, we consider a varying number of VMs (from 10 to 100), and a fixed number of tasks $(78,597)$. The tasks arrive and are scheduled based on the policy, as described in Section 5.1.2. The episode terminates when all the tasks are executed. During the training, we simulate fixed number of episodes (100) for each VM set to explore the probabilistic space of possible actions using the current policy, and use the resulting data to improve the policy for all the tasks. Technically speaking, we record the state, action, and reward information for each episode, and use these values to compute the cumulative reward of each episode. We simulate a large number of iterations (1000 iterations) and then compute the average reward value. The minimum reward value (cost) from the ground truth corresponds to the full accuracy. The other accuracy values are computed subsequently. For the testing, 20% of new data (from the dataset) is used. During testing, the agent follows the learned policy by selecting the action

with lowest reward value at each step.

In the second set of experiments, we measure the execution cost entailed by the different scheduling approaches in terms of CPU and RAM spent on running the tasks. This perspective is important for both cloud providers and customers in the sense that it enables them to pick the scheduling approach that reduces their overall monetary costs. In the third series of experiments, we compare the best identified candidate with three other scheduling approaches, namely, SJF, RR, and improved PSO.

It is worth mentioning that in some cases, variable values of the accuracy and resource utilization metrics are obtained at some simulation rounds. To deal with this problem, We made sure to run each single simulation for a large number of iterations (1000 iterations) and then average over these iterations to get a stable and representative value for each corresponding metric. Our program is written in the python language, version 3, RapidMiner Studio version 9.3, and performed in a 64-bit Windows 7 environment on a machine equipped with an Intel Core i7-6700 CPU 3.40 GHz Processor and 16 GB RAM.

5.2.3 Experimental Results

In Fig.5.3, we measure the training accuracy of the different studied approaches. The main observation that can be drawn from this figure is that increasing the number of deployed VMs leads to a modest decrease in the accuracy. This can be justified by the fact that having a larger number of VMs to select from might increase the probability of mistakenly assigning tasks to some inappropriate VMs. The second observation that can be drawn is that the DRL-LSTM yields the highest training accuracy (between 94.2% and 96.2%). DQN yields the second highest training accuracy (between 91% and 96%) followed by RNN-LSTM (between 88% and 96%) and finally RL (between

82.1% and 92%).

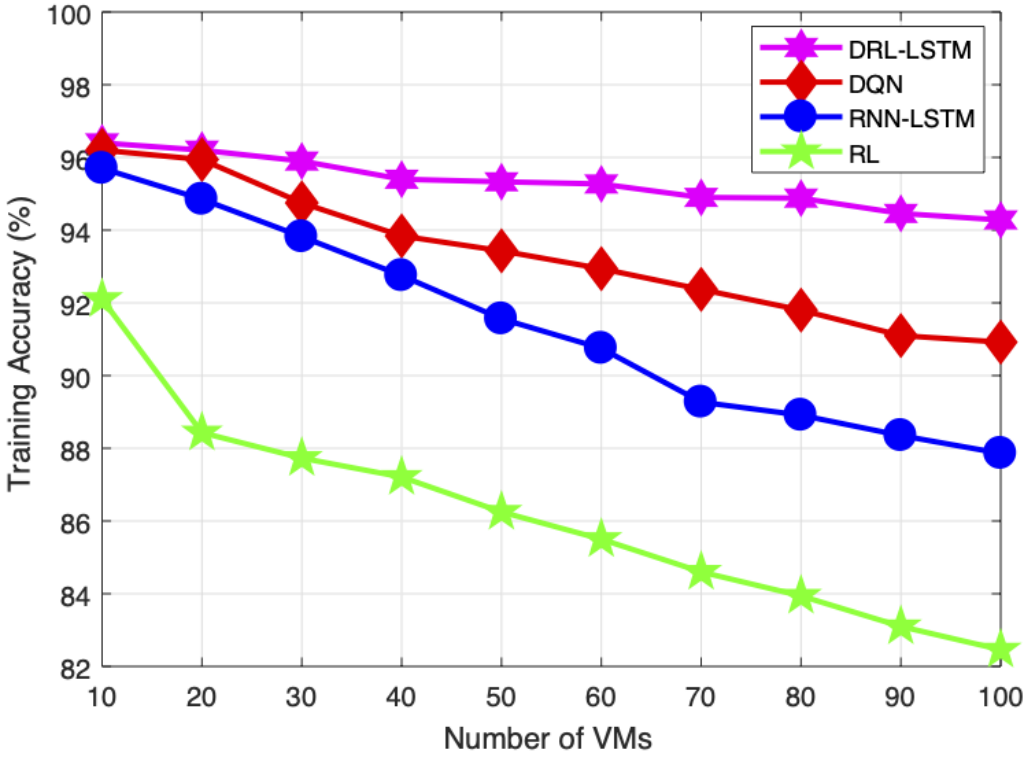


Figure 5.3: Training accuracy of machine learning algorithm

In Fig. 5.4, we measure the test accuracy of our four approaches. This metric is of prime importance since it can inform us about the accuracy of the different approaches on data examples they have not seen yet, thus giving an indication about the overfitting rate of the machine learning approaches. Again, DRL-LSTM yields the highest test accuracy (between 89.8% and 95.1%). However, different from the case of training accuracy, DQN and RNN-LSTM yield similar test accuracy (between 86% and 92%). This indicates that DQN (which recorded higher training accuracy than RNN-LTSM) was overfitting the training data. Finally, as is the case for the training accuracy, RL yields the least test accuracy varying from 67% to 86%.

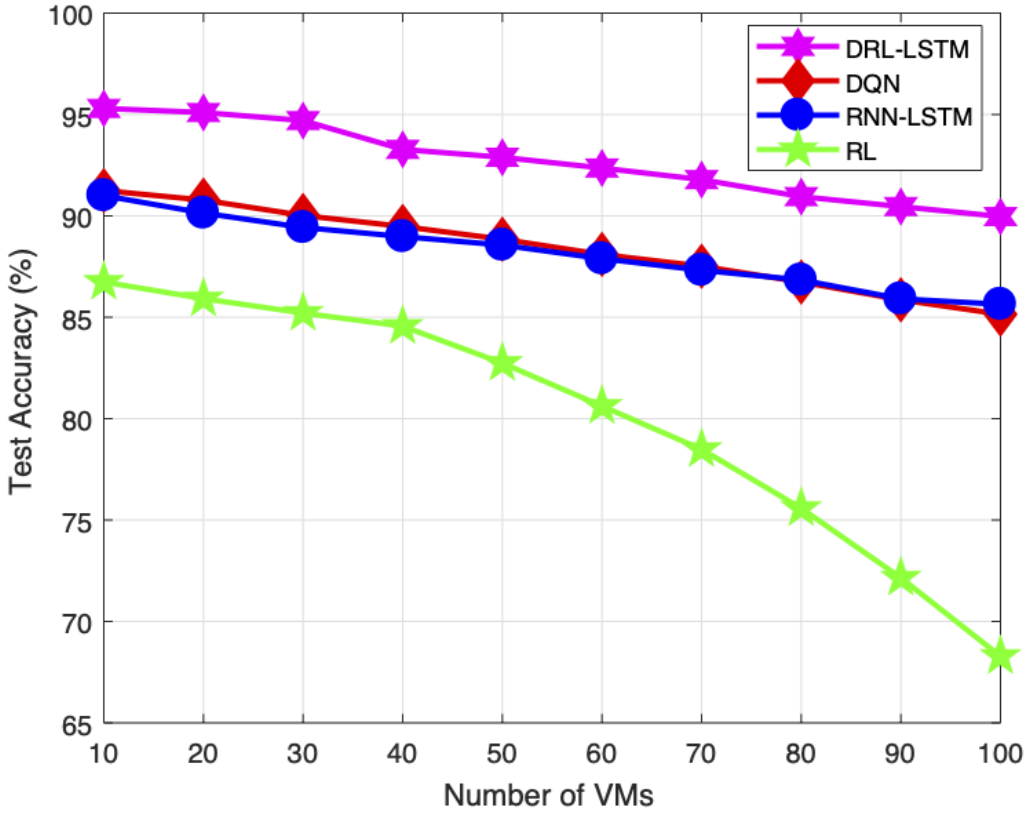
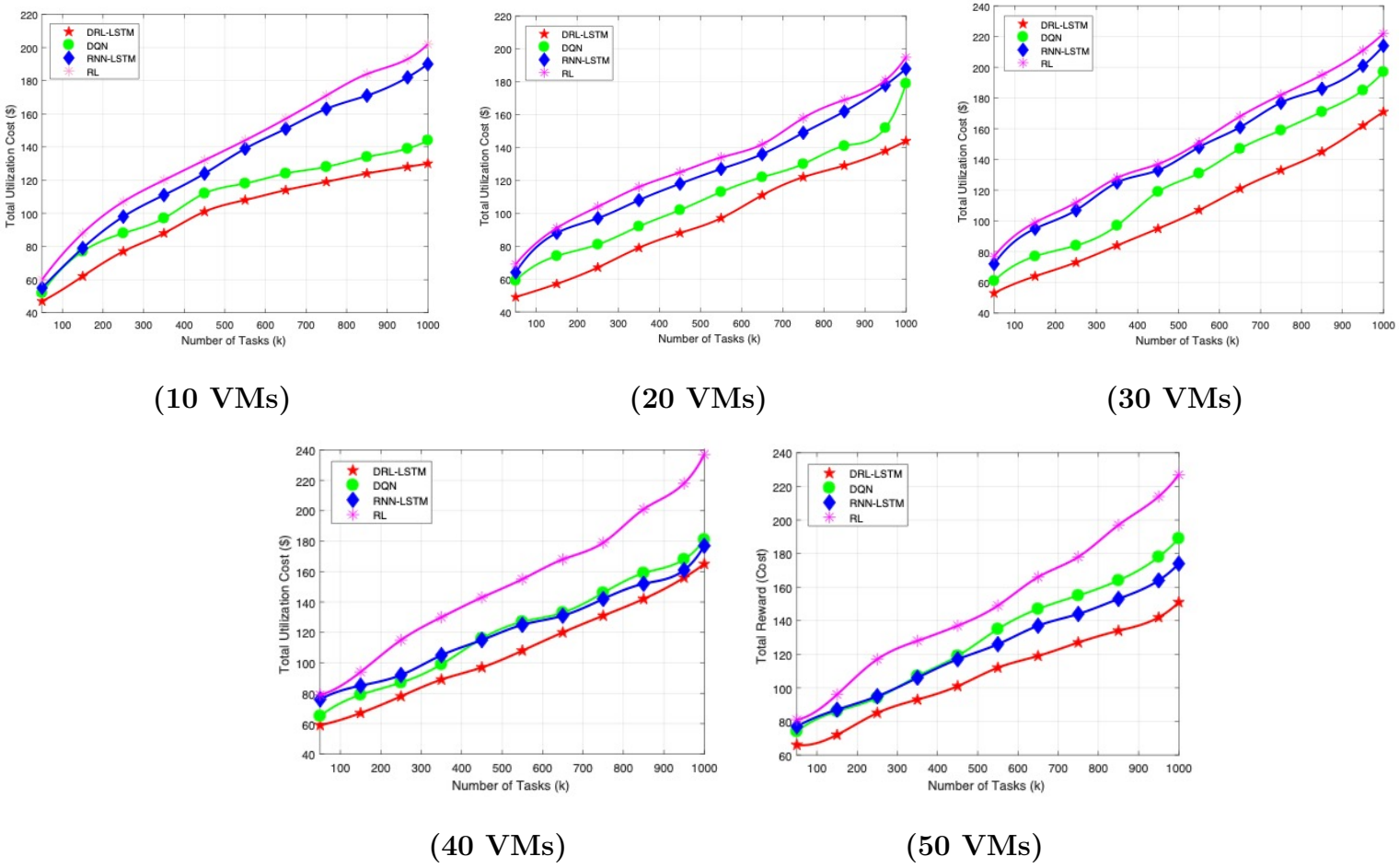


Figure 5.4: Test accuracy of machine learning algorithm

In Fig. 5.5, we present a detailed study on the total utilization cost entailed by our solutions. In this series of experiments, we vary the number of deployed VMs from 10 to 50, and vary the number of tasks from $50,000$ to 1 million. The objective is to study the scalability of the different solutions w.r.t the variation in the number of VMs and number of tasks. We notice from the figure that increasing the number of VMs leads to an increase in the utilization cost for the different approaches. This is because deploying more VMs leads to consuming higher amounts of resources. Moreover, we notice that DRL-LSTM shows a better scalability compared to the other solutions, followed by DQN and RNN-LSTM interchangeably and then finally RL.



(10 VMs) **(20 VMs)** **(30 VMs)**

(40 VMs) **(50 VMs)**

Figure 5.5: **Total utilization cost:** We study in this figure the the impact of varying both the number of tasks and number of VMs on the overall usage cost

We provide in Table 5.1 the confusion matrix of the DRL-LSTM approach, which scored the best amongst the other reinforcement and deep learning solutions in terms of training and test accuracy. In the matrix, for each underlying VM, the rows refer to the optimal VM that the tasks should ideally be assigned to. The columns refer to the VMs chosen (predicted) by DRL-LSTM to receive the tasks. For example, by examining the matrix, we notice that out of the total number of tasks that should be scheduled on VM1, DRL-LSTM has actually scheduled 94.46% (second column and second row) of these tasks on that VM, yielding an insignificant error rate of 5.54%

(last column and second row).

Table 5.1: Confusion matrix

	VM0	VM1	VM2	VM3	VM4	VM5	VM6	VM7	VM8	VM9	VM10	Error Rate
VM0	100%	0	0	0	0	0	0	0	0	0	0	0
VM1	0	94.46%	0.4%	0	1.72%	3.42%	0	0	0	0	0	5.54%
VM2	0	0.05%	99.95%	0	0	0	0	0	0	0	0	0.05%
VM3	0	0	0	99.92%	0	0	0.08%	0	0	0	0	0.8%
VM4	0	0	0	0	94.08%	0	0	0	0.79%	0	5.13%	5.3%
VM5	0	0	0	0	0.39%	95.45%	0	0	0	4.16%	0	4.77%
VM6	0	0	11.1%	0.77%	0	1.2%	86.93%	0	0	0	0	13%
VM7	0	0	0	0	0	0	0	100%	0	0	0	0
VM8	0.07%	0	0	0	0	0	0	0	99.03%	0	0	0.97%
VM9	0	0	0	0	0	0	0	0	0	100%	0	0
VM10	0	0	1.67%	0	0	0	0	3.07%	0	0	95.26%	4.7%

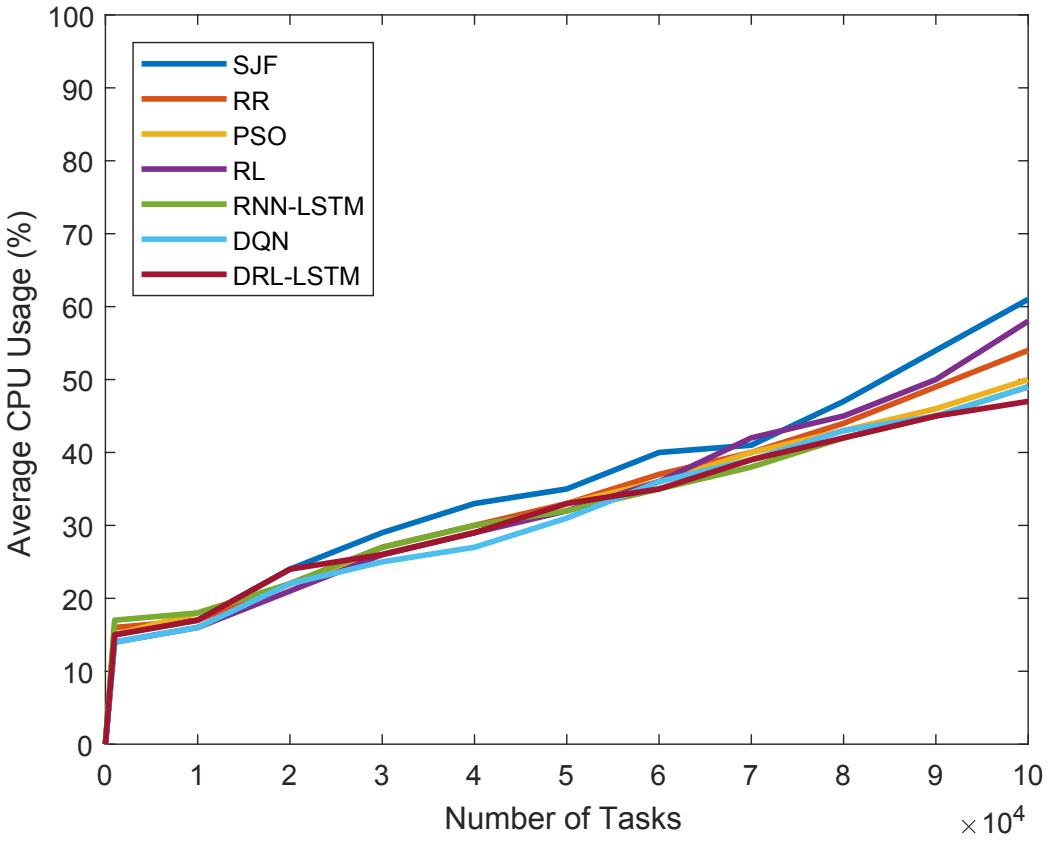


Figure 5.6: Average CPU usage

In the next set of experiments, we are interested in comparing our scheduling solutions with the traditional scheduling approaches, namely SJF, RR and PSO. Specifically, in Fig. 5.6, Fig. 5.7, Fig. 5.8 and Fig. 5.9, we measure the average CPU and RAM utilization entailed on the VMs by our solutions as well as the traditional scheduling approaches. In these figures, we fix the number of VMs to 100 and vary the number of tasks up to $100,000$. We observe that increasing the number of tasks leads to increasing the average CPU and RAM utilization. The second observation that can be taken from Fig. 5.6 and Fig. 5.8 is that our deep and reinforcement

learning-based scheduling solutions reduce the CPU and RAM consumption compared to the traditional scheduling proposals. The reason is that our learning-based solutions consist of a learning component that intelligently predicts the appropriate VM for each incoming task while considering a multitude of metrics at a time.

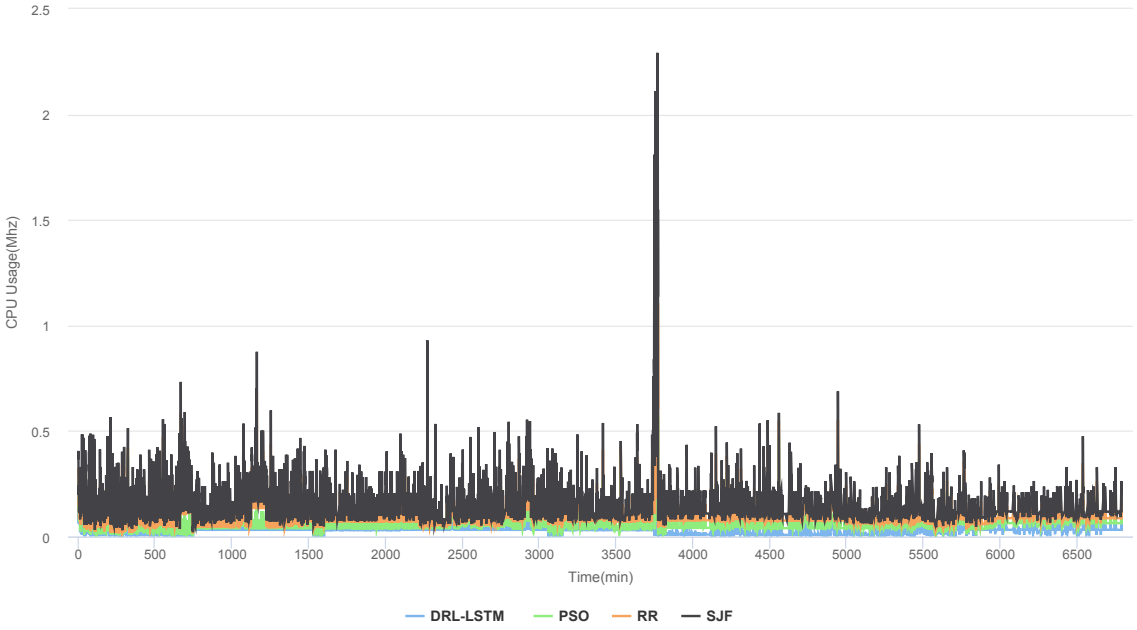


Figure 5.7: CPU usage (Mhz)

On the other hand, traditional scheduling approaches consider only a small set of metrics (e.g., waiting time, etc.) when assigning tasks to VMs, which degrades the quality of their decisions. Moreover, DRL-LSTM minimizes further the CPU and RAM utilization compared to the rest of our solutions. This outcome is a natural result of the training and testing accuracy results explained in Fig. 5.3 and Fig. 5.4.

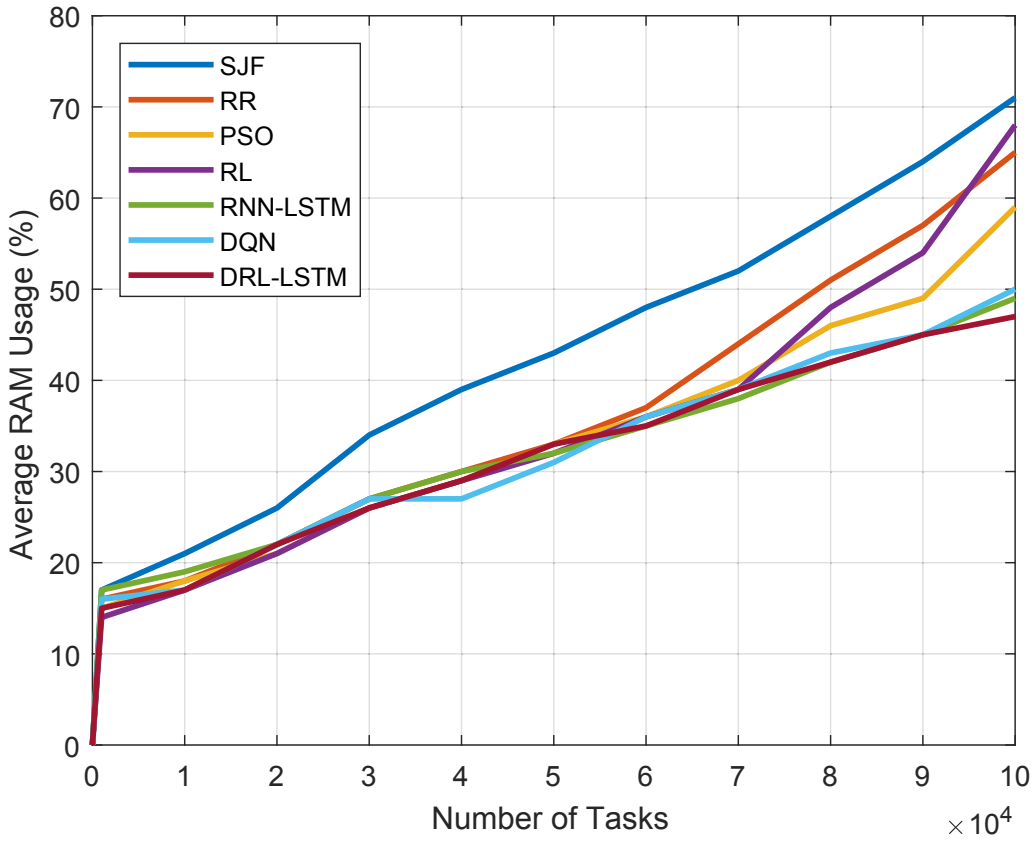


Figure 5.8: Average RAM usage

Finally, in Fig. 5.10 and Fig. 5.11, we provide a detailed description of the CPU and RAM monetary costs entailed by DRL-LSTM (the best approach identified so far) compared to the traditional scheduling approaches (i.e., PSO, RR and SJF). The experiments have been conducted on a sample of 11 VMs and a number of tasks fixed to $78,597$. For instance, we can notice from Fig. 5.10 that the CPU cost varies from 0 to \$6.5 in the case of DRL-LSTM (Fig. 5.10a), from 0 to \$10 in the case of RR (Fig. 5.10c) and improved PSO (Fig. 5.10b) and from 0 to \$20 in the case of SJF (Fig. 5.10d).

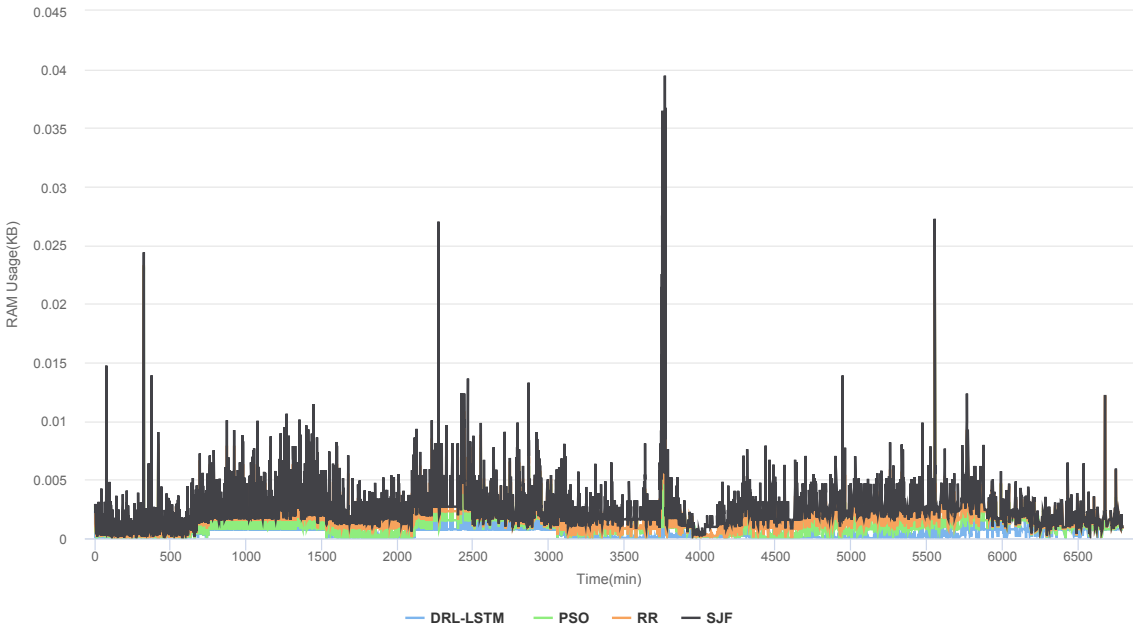
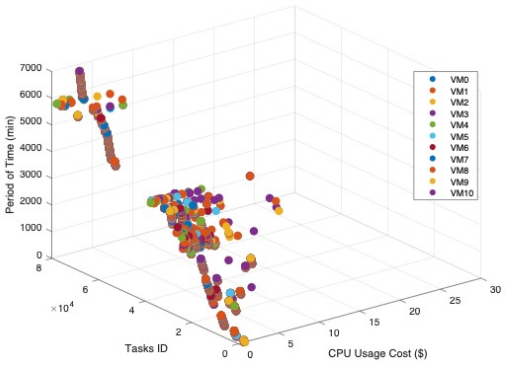
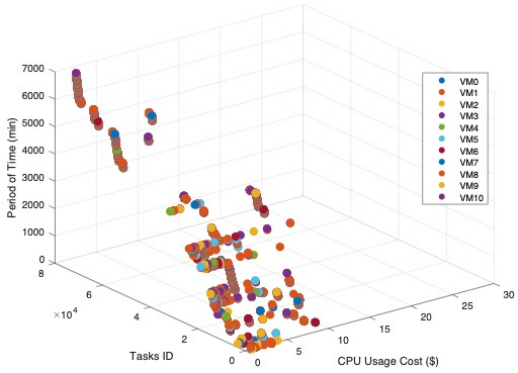


Figure 5.9: RAM usage (KB)

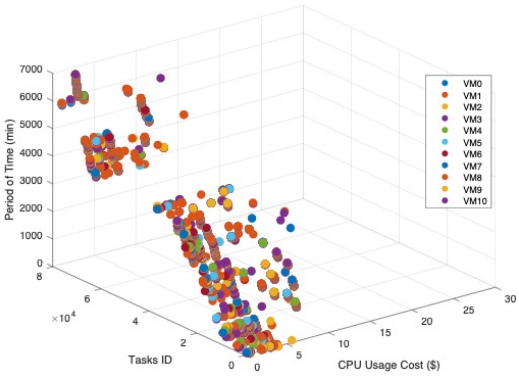
Furthermore, by examining Fig. 5.11, we notice that the RAM cost varies from 0 to $7 in the case of DRL-LSTM (Fig. 5.11a), from 0 to $15 in the case of the improved PSO (Fig. 5.11b), from 0 to $20 in the case of RR (Fig. 5.11c), and from 0 to $25 in the case of SJF (Fig. 5.11d). Overall, we can conclude the DRL-LSTM solution achieves the best results in terms of training and test accuracy, total utilization cost, and CPU and RAM utilization compared to both the other three learning-based approaches and the traditional scheduling solutions.



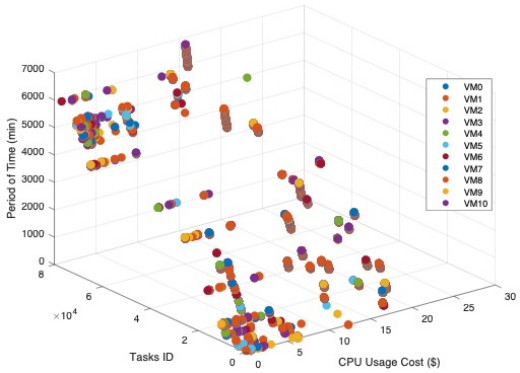
(a) CPU usage cost by using DRL-LSTM



(b) CPU usage cost by using PSO

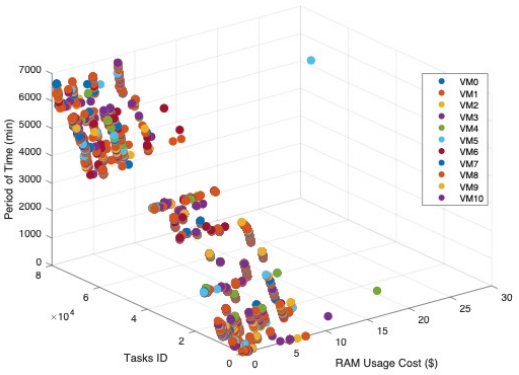


(c) CPU usage cost by using RR

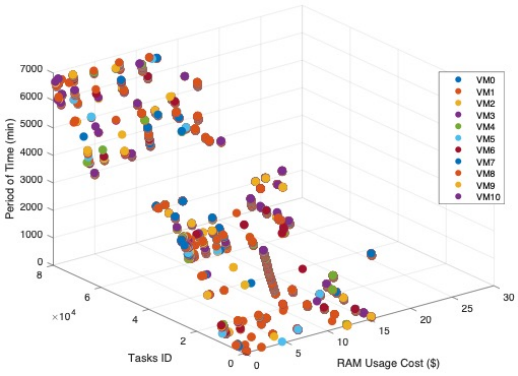


(d) CPU usage cost by using SJF

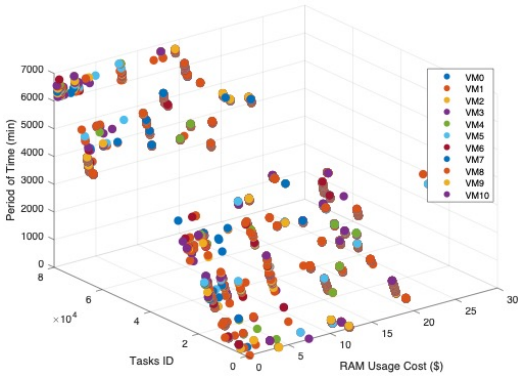
Figure 5.10: **CPU usage cost**: we give in this figure a detailed breakdown of the CPU usage cost of our DRL-LSTM approach compared to PSO, RR, and SJF



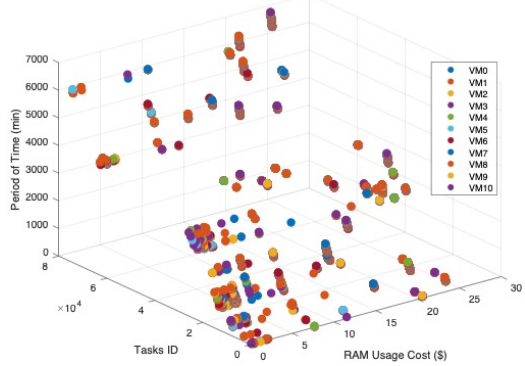
(a) RAM usage cost by using DRL-LSTM



(b) RAM usage cost by using PSO



(c) RAM usage cost by using RR



(d) RAM usage cost by using SJF

Figure 5.11: **RAM usage cost**: we give in this figure a detailed breakdown of the RAM usage cost of our DRL-LSTM approach compared to PSO, RR, and SJF

5.3 Conclusion

In this chapter, we proposed four automated task scheduling approaches in cloud computing environments using deep and reinforcement learning. The comparison of the results of these approaches revealed that the most efficient approach is the one that combines deep reinforcement learning with LSTM to accurately predict the appropriate VMs that should host each incoming task. Experiments conducted using the dataset from GCP pricing and Google cluster resource and task requirements

revealed that this solution minimizes the CPU utilization cost up to 67% compared to the Shortest Job First (SJF), up to 35% compared to both the Round Robin (RR) and improved Particle Swarm Optimization (PSO) approaches. Besides, our solution reduces the RAM utilization cost by 72% compared to the SJF, by 65% compared to the RR, and by 31% compared to the improved PSO.

Although very promising results are achieved, the work has still some limitations. The first issue to be considered is the high computation time of our approach. To address this problem, we investigate the application of federated learning techniques in the next chapter, which help reduce the time of learning the model. Moreover, the proposed solution does not take into account the reliability of the selected IoT devices, which increases the risk of assigning the tasks to malicious or poorly performing nodes. To address this problem, in the next chapter, we include the performance and trust metrics of the IoT devices in the reward function of the different learning models to better learn the behavior of these devices and avoid selecting the bad ones.

Chapter 6

Trust-driven Reinforcement Selection Strategy for Federated Learning on IoT Devices

Federated learning is a distributed machine learning approach that enables a large number of edge/end devices to perform on-device training for a single machine learning model, without having to share their own raw data. We consider in this chapter a federated learning scenario wherein the local training is carried out on IoT devices and the global aggregation is done at the level of an edge server. One essential challenge in this emerging approach is IoT selection (also called scheduling), i.e., how to select the IoT devices to participate in the distributed training process [91]. The existing approaches suggest to base the scheduling decision on the resource characteristics of the devices to guarantee that the selected devices would have enough resources to

carry out the training. In this chapter, we argue that trust should be an integral part of the decision-making process and therefore design a trust establishment mechanism between the edge server and IoT devices. The trust mechanism aims to detect those IoT devices that over-utilize or under-utilize their resources during the local training. Thereafter, we introduce DDQN-Trust, a Double Deep Q Learning-based selection algorithm that takes into account the trust scores and energy levels of the IoT devices to make appropriate scheduling decisions. Finally, we integrate our solution into four federated learning aggregation approaches, namely, *FedAvg*, *FedProx*, *FedShare* and *FedSGD*.

6.1 Trust-Aware IoT Scheduling for Federated Learning

6.1.1 Trust Establishment Mechanism

In Algorithm 6.1, we propose a statistical trust establishment method for IoT devices based on monitoring the CPU and RAM consumption of the devices to identify the ones that exhibit some abnormal resource consumption behavior, and the devices whose consumption goes down the normal minimal habitual consumption (e.g., failed IoTs). This is important to detect those devices that do not dedicate enough resources to serve the FL tasks as well as those that exhibit some overly high consumption which could be an indication of some malicious behavior. For example, some malicious devices might optimize for a malicious objective that aims to generate targeted misclassification. Such devices are expected to spend more resources than the regular devices that only try to optimize for the underlying federated task. Note that, every edge server monitors IoT devices that are located within its range. Thus, Algorithm

6.1 is executed by each edge server. The proposed method capitalizes on the modified Z-score statistical technique. Modified Z-score is a standardized score that measures outlier strength, i.e., how much a particular score differs from the typical score by checking the dependability of a particular score on a certain typical score. This method shows a greater robustness to outliers compared to some other statistical techniques (e.g, traditional Z-Score, Tukey method, etc.) since it capitalizes on the median $\bar{x}$ instead of the mean μ. In our algorithm, this method approximates the difference of a certain score from the median using the median absolute deviation $MAD_j^z(t)$ of a metric z (e.g., CPU, RAM) consumed by a device j during a time window $[t-\delta, t]$ (Algorithm 6.1 line 6).

More specifically, the modified Z-score $\alpha_j^z(i,t)$ is calculated through dividing the difference between the consumption $x_j^z(i)$ of the device j in terms of the resource metric z at time moment $i \in [t-\delta, t]$ and the median consumption of that device in terms of that metric within the time interval $[t-\delta, t]$ by the median absolute deviation of the metric z (Algorithm 6.1 line 20). The constant $\varrho = 0.6745$ is needed because $\mathbb{E}(MAD_j^z(t)) = 0.6745\sigma$ for a large number n of samples. Observations will be labeled outliers when $\alpha_j^z(i,t) \geq \varphi$, where $\varphi = 3.5$ as argued in [50]. This limit quantifies the patterns of maximal and minimal habitual utilization of each IoT device within a certain time interval. Thus, any future consumption that exceeds or falls under this limit is deemed to be unusual.

Algorithm 6.1 IoT Trust

Inputs:

j: an IoT being monitored by the edge computing server

$M = \{CPU,\ memory\}$: the set of IoT's metrics to be analyzed by the edge server

δ: size of time window after which the algorithm is to be repeated

Variables:

$M_j^z(t)$: a table recording $x_j^z(i)$ $(i = t-\delta, t-\delta+1, \ldots, t)$, the amounts of $z \in M$ consumed by j during the time interval $[t-\delta, t]$

$\bar{x}_j^z(t)$: the median of $M_j^z(t)$ (median consumption of $z \in M$ by j during the time interval $[t-\delta, t]$)

$MAD_j^z(t)$: the median absolute deviation of $M_j^z(t)$, i.e., $MAD_j^z(t) = median\left\{\left|x_j^z(i) - \bar{x}_j^z(t)\right|\right\}$ for all $t-\delta \leq i \leq t$

$\alpha_j^z(i,t)$: the modified Z-score of $x_j^z(i) \in M_j^z(t)$

$AbnormalMetrics_j^z$: sum of unusual consumption of $z \in M$ by j

$CountAbnormalMetrics_j^z$: a counter enumerating the occurrence of unusual consumption of $z \in M$ by j

$AvgAbnormalMetrics_j^z$: j's average unusual consumption of $z \in M$

$PropAbnormalMetrics_j^z$: j's unusual consumption of $z \in M$ proportionally to the upper and lower consumption limits of this z

$AbnormalMetrics_j$: the number of abnormal usages of all the metrics by j.

Output:

Γ_j: trust value of j

Initialize $AbnormalMetrics_j$ **to** 0
for each metric $z \in M$ **do**
  Initialize $AbnormalMetrics_j^z$ **and** $CountAbnormalMetrics_j^z$ **to** 0
  Initialize $AvgAbnormalMetrics_j^z$ **and** $PropAbnormalMetrics_j^z$ **to** 0
  Compute the median $\bar{x}_j^z(t)$ of $M_j^z(t)$
  Compute the $MAD_j^z(t)$ of $M_j^z(t)$
  Compute $\alpha_j^z(i,t) = \dfrac{\varrho(x_j^z(i) - \bar{x}_j^z(t))}{MAD_j^z(t)}$
  for each data point $x_j^z(i) \in M_j^z(t)$ **do**
    if $\alpha_j^z(i,t) \geq \varphi$ **then**
      $AbnormalMetrics_j^z = AbnormalMetrics_j^z + x_j^z(i)$
      $CountAbnormalMetrics_j^z = CountAbnormalMetrics_j^z + 1$
    end if
  end for
    if $CountAbnormalMetrics_j^z > 0$ **then**
      $AvgAbnormalMetrics_j^z = AbnormalMetrics_j^z / CountAbnormalMetrics_j^z$
      $PropAbnormalMetrics_j^z = \frac{\varphi}{AvgAbnormalMetrics_j^z}$
      $AbnormalMetrics_j = AbnormalMetrics_j + 1$
    end if
end for
    if $AbnormalMetrics_j = 0$ **then**
      $\Gamma_j = 1$
    else
      $\Gamma_j = \dfrac{\sum_{z \in M} PropAbnormalMetrics_j^z}{AbnormalMetrics_j}$
    end if
return Γ_j

The Algorithm then checks for any future consumption of the IoT to determine whether there exists any consumption that exceeds or falls under the computed abnormal limit (Algorithm 6.1 - lines $22-23$). If such a case is encountered, this

observation is added to a table that registers each IoT's unusual consumption (if any) (Algorithm 6.1 - line 24). The average unusual consumption for each metric is then computed (Algorithm 6.1 - line 28). The Algorithm then computes the trust value of each IoT by dividing the sum of the proportional abnormal consumption over all the metrics by the number of metrics that the device has overused/underused (if any) (Algorithm 6.1 - line 36). If no metric has been overused/underused, the initial trust in the IoT's trustworthiness would be set to 1 (Algorithm 6.1 - line 34), which represents a full trust in that device.

6.1.2 DDQN-Trust Scheduling Policy

Reinforcement learning [87,104] is an active research and application area of machine learning that has been applied to solve uncertainty-driven problems wherein exact models are often infeasible. It aims at guiding a certain agent on how to react to the changes that take place in the environment. The agent performs the appropriate actions that maximize its cumulative reward according to the current state of the environment. In this work, we propose DDQN-Trust, a trust and energy-aware dynamic Double Deep Q Network scheduling method. The proposed method consists of a multi-layered neural network that, for a given state outputs a vector of actions given a set of parameters of the network. The problem is formulated as a global Markov Decision Process (MDP) where the system global states and global actions are formulated as the combination of IoT devices local states and actions. It is defined by the tuple $\left\langle S, A, T, R, \gamma \right\rangle$, where:

- S: the set of global states of the system.
- A: the set of joint actions of all the IoT devices.
- T: the transition probability function defined as: $T(s, a, s') = Pr(s'|s, a)$, where

$s, s' \in S$ and $a \in A$.

- $R : S \times A \times S \mapsto \mathbb{R}$: the reward function of the model.
- γ: a discount factor that decreases the impact of the past reward.

Let S_j be the set of local states of the IoT device j and J the set of all the devices. The global state space S is obtained through the Cartesian product of IoT devices local states: $S = \prod_{j \in J} S_j$. Each local state $s_j \in S_j$ is as follows:

$$s_j = (\Gamma_j, \chi_j); \qquad \Gamma_j \in [0, 1], \chi_j \in \{0, 1, \ldots, \chi^{max}\} \tag{6.1}$$

where Γ_j is the trust value of the IoT device j computed in Algorithm 6.1 and χ_j is the energy state of j. Trust and energy state are dynamic, so they could change from state to state. The global action space of the parameter edge server is the joint action space of each device: $A = \prod_{j \in J} A_j$ where A_j is the set of local actions of j. A local action $a_j \in A_j$ is as follows:

$$a_j = (\sigma_j, l_j^{\chi}, \xi_j); \quad \sigma_j \in \{0, 1\}, l_j^{\chi} \in \{0, 1, \ldots, \chi^{max}\}, \xi_j \in \mathbb{R} \tag{6.2}$$

where $\sigma_j = 1$ means the parameter server assigns a training task to the IoT device j; $\sigma_j = 0$ otherwise, l_j^{χ} refers to the amount of energy needed by the IoT device j to download, train and upload the model, and ξ_j is the cost of transmitting the model from the parameter server to the device j and running the model. For an action to be feasible from a global state s to s', the following condition should hold:

$$l_j^{\chi}(s') \leq \chi_j(s) \quad \forall j \in J \tag{6.3}$$

where $l_j^{\chi}(s')$ refers to l_j^{χ} in the action leading to s' and $\chi_j(s)$ is χ_j in s. Finally, to define the reward function R, the objective of maximizing the selection of trusted IoT

devices having enough energy to receive and perform the training task is considered. The cost ξ_j is also considered proportional to the maximum cost ξ^{max}. The reward ψ_j for the device j is a function of state $s \in S$ and action $a \in A$ as follows:

$$\psi_j(s,a) = \begin{cases} \Gamma_j \cdot \chi_j - \frac{\xi_j}{\xi^{max}}, & \text{if } \ l_j^{\chi} \leq \chi_j. \\ -\frac{\xi_j}{\xi^{max}}, & \text{otherwise.} \end{cases} \tag{6.4}$$

Thus, along with the trust scores of the IoT devices, the edge server accounts for the available energy level of the devices to make sure that these devices have enough battery capacity to download, train and upload the model.

The global reward of the parameter server is as follows:

$$R(s,a) = \sum_{j \in J} \psi_j(s,a) \tag{6.5}$$

The parameter edge server determines the optimal policy $\pi^* : S \rightarrow A$ that indicates the actions to be taken at each state to maximize the cumulative reward. The essential goal of the Q-learning (QL) algorithm used to find π^* is to update the Q-value of a state-action pair, $Q(s,a)$, which encodes the expected future discounted reward for taking action a in state s. The optimal action-value function $Q^*(s,a)$ is $Q^*(s,a) = \max_{\pi} Q_\pi(s,a)$. This optimal value function can be nested within the Bellman optimality equation as follows:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} Pr(s'|s,a). \max_{a' \in A} Q^*(s',a') \tag{6.6}$$

Depending on the Q-table that results from updating the $Q(s,a)$ values, the parameter server determines the optimal action from any state to maximize the cumulative reward. The QL algorithm is practical for networks with small state and action

spaces only, but when the number of network participants increases (which is the case of IoT networks that consist of a large number of devices), the problem of assigning training tasks to the IoT devices becomes high dimensional. The Deep QL (DQL) algorithm (a combination of QL and deep neural network DNN) comes into play to solve the high dimensionality problem. The input of the DNN is one of states of the online network, and the outputs are the Q-values $Q(s, a; \theta)$ of all the possible actions, with θ being the weight matrix of the DNN. The DNN needs to be trained by using experiences $(s, a, R(s, a), s')$ to obtain the approximate values $Q^*(s, a)$. We use the Mean Square Error (MSE) to define the loss function and DNN uses the Bellman equation to minimize this loss function as follows:

$$L(\theta_i) = E[(R(s, a) + \gamma \arg \max_{a' \in A} Q(s', a'; \theta'_i) - Q(s, a; \theta_i))^2] \tag{6.7}$$

where θ_i represents the parameters of the online network at the i^{th} iteration, θ'_i represents the parameters of the target network at the i^{th} iteration, and $E[.]$ denotes the expected value. Note that the action a is selected based on the ϵ-greedy policy. By using the max operator (which uses the same Q-values to select and to evaluate an action in standard QL and DQN), we observe that it is more likely that this operator selects overestimated values, resulting in overoptimistic estimates. To prevent such a problem, we should decouple the action selection from the evaluation by employing the Double Deep Q-network (DDQN) [96]. The main feature of DDQN is the use of two separate DNNs, i.e., an online network with weight set θ and a target network with weight set θ''. The DDQN employs two valuation functions for two autonomous DNNs learned through randomly assigning experiences to update one of the two value functions, resulting in two sets of weights θ for the first DNN and θ'' for the second DNN. At each iteration, the weights of the online network are updated, while those of the target network are kept constant to determine the greedy policy. The target

function of our DDQN-Trust error is defined by:

$$T_{DDQN-Trust}(s, a, s') = R(s, a) + \gamma Q(s', \arg\max_{a' \in A} Q(s', a'; \theta); \theta'') \tag{6.8}$$

To compute the optimal value $Q(s', a'; \theta)$, the weight θ of the online network uses the next state s' to select an action, while the target network θ'' uses the next state s' to evaluate the action. Then, a stochastic gradient descent step is performed to update the weights of the online networks θ based on the loss

6.1.3 DDQN-Trust-based Federated Learning Model

In this section, we describe how the FL process can be executed after integrating our trust establishment and scheduling mechanisms. A DNN model is distributed over the IoT devices to be collaboratively trained following the FL framework. Let D_j be a local dataset collected by the IoT device j, $D_j = \{(x_{1_j}, y_{1_j}), \ldots, (x_{n_j}, y_{n_j})\}$, where x_{i_j} is the i^{th} training sample and y_{i_j} represents the corresponding ground-truth label. In this work, we take a general Convolutional Neural Network (CNN) model for analysis. The edge server receives the local gradient vectors from the trusted IoT devices and then aggregates (averages) them to obtain the global gradient using Equation (6.9):

$$g\left[\nu\right] = \frac{1}{\sum\limits_{j \in J} |\vartheta_j|} \sum_{j \in J} |\vartheta_j| g_j\left[\nu\right] \tag{6.9}$$

where ϑ_j is a subset of local data collected from the IoT device j for a training period ν, with $\vartheta_j \subseteq D_j$, and $g_j\left[\nu\right]$ being the local gradient which is computed as per Equation (6.10).

$$g_j\left[\nu\right] = \nabla_{w_j} L_j\left(w_j, \vartheta_j\right) \tag{6.10}$$

where w_j is the local parameter set of the CNN model, L_j is the local loss function on the IoT device j to measure the training error and $\nabla_{w_j} L_j(.)$ is the gradient of the loss function L_j with respect to w_j.

Algorithm 6.2 DDQN-Trust-based Federated Learning Algorithm for IoT Selection

Initialize the global parameter set of the CNN model
for each round $\tau = 1, 2, \ldots$ **do**
    Use Algorithm 6.1 to compute the trust scores of all the IoT devices
    Use DDQN-Trust to select a subset $\mathcal{E} \subseteq J$ of IoT devices to participate in the training
    Send W_τ to each selected IoT
    for each IoT device $j \in \mathcal{E}$ **do** % $\mathcal{E} = \{1, 2, \ldots, E\}$
        Execute **IoTLocalUpdate**(W_τ) % See Algorithm 7.2
    end for
    $W_\tau = \frac{1}{n} \sum_{j=1}^{E} n_j w_j$
end for

In Algorithms 6.2 and 6.3, we describe the federated learning process after embedding our proposed trust establishment mechanism and DDQN-Trust scheduling policy to improve the selection of IoT devices. In Algorithm 6.2, n_j is the data size available on IoT device j, n is the size of the whole data across all devices, E is the total number of selected devices, τ is the training communication round index and W_τ is the global parameter set at round τ.

Algorithm 6.3 IoT Local Training

IoTLocalUpdate(W_τ)
    $w_j = W_\tau$
    for each local iteration $t = 1$ to T **do**
        $w_j = w_j - \eta \nabla_{w_j} L_j(w_j, \vartheta_j)$ % η is the learning rate
    end for
    return w_j to the edge server

Each IoT device runs the stochastic gradient descent (SGD) algorithm based on the received global gradient. The local loss function on each device j is defined as per Equation (6.11):

$$L_j(w_j) = \frac{1}{N_j} \sum_{(x,y)\in D_j} \ell(w_j, x, y) \tag{6.11}$$

where $\ell(w_j, x, y)$ is the sample-wise loss function that quantifies the prediction error between the learning output (via input x and parameter w_j) and the ground-truth label y, and N_j is the number of data samples of the device j. Each device seeks to minimize the local loss function defined in Equation (6.11) to minimize the training error. On a global level, the main target of the training task at the edge server is to optimize the parameters towards minimizing the global loss function $L(W)$ via the SGD algorithm expressed as follows:

$$L(W) = \frac{1}{\sum_{j=1}^{E} N_j} \sum_{j=1}^{E} N_j L_j(w_j) \tag{6.12}$$

6.1.4 Federated Learning Aggregation Approaches

We integrate our solution with four existing federated learning approaches, namely, *FedProx*, *FedShare*, *FedSGD*, and *FedAvg*. The objective is to pick the approach that best suits our solution.

A. FedAvg

The *FedAvg* approach relies on a single-model strategy that leverages an averaged results across many clients. In *FedAvg*, the server chooses a subset of IoT devices in each communication round and sends the global model back to them. Each device will perform a predefined number of gradient descent iterations on its local data, before pushing the model's weight to the server. Finally, the server averages these weights to generate a new global model. Technically speaking, after receiving the local model w_j^τ and gradient g, the server aggregates them using Algorithm 6.2

(Line 9) and Equation (6.9) and shares w and g with the IoT devices.

B. FedSGD

The main idea of *FedSGD* is to let the IoT devices minimize a surrogate function Φ_j^τ [28] after each global round, as follows:

$$\Phi_j^\tau(w) = \Xi_j(w) + \langle \eta \nabla g^{\tau-1} - \nabla g_j(w^{\tau-1}), w \rangle \tag{6.13}$$

with Ξ_j being L-smooth and β strongly convex, $\forall j$ [75], and Φ inspired by the Distributed Approximate NEwton (DANE) scheme introduced in [97]. Furthermore, we include both local and global gradient estimates weighted by a controllable parameter η. Thereafter, the IoTs send not only the local model w_j, but also local gradient estimates $\nabla g^{\tau-1}$ to speed the convergence up in the experiments.

C. FedShare

In *FedShare*, a globally accessed dataset G with a uniform distribution is used to mitigate the impact of data heterogeneity across the client devices. Specifically, a sample of this dataset is given to each IoT device in the initialization phase. The shared data from G is combined with the private data of each device to form the training set of the device's local model. The server then aggregates the local models from the IoT devices using *FedAvg* to construct the global model. In *FedShare*, two trade-offs are considered. The first one is the trade-off between the test accuracy and the size of G, which is quantified as $\beta = \dfrac{\|G\|}{\|\prod_{j \in J} D_j\|} \times 100\%$. The second trade-off is between the test accuracy and the random distributed fraction [130].

D. FedProx

To a certain extent, *FedProx* is similar to *FedAvg* in that devices are chosen at each round, which means that local updates are made to a subset of devices, and these are then averaged to obtain a global model. Therefore, to reiterate, instead of all training devices having the same number of rounds, *FedProx* indirectly trains for various devices having varying rounds. In other words, instead of assuming a uniform number of local rounds Ω for all devices throughout the training process, *FedProx* implicitly accommodates variable Ω's for different devices and at different iterations. We adopt Ω_j^τ-inexact solution [60] to find w_j^τ for each IoT device j at round τ where Ω_j^τ-inexact minimizer minimizes the following objective h_j:

$$w_j^{\tau+1} \approx argmin_w \ h_j(w; w^\tau) = L_j(w_j) + \frac{T}{2}\|w - w^\tau\|^2 \tag{6.14}$$

6.2 Experimental Results and Analysis

6.2.1 Experimental Setup

To carry out our experiments, we used TensorFlow Federated (TFF), which is an open-source framework for machine learning on decentralized data. TFF supports a variety of distributed learning scenarios executed on a large number of heterogeneous devices having diverse capabilities. We trained a CNN model on the CIFAR-10 dataset[1] to evaluate the performance and efficiency of our solution. The dataset consists of $50,000$ training images and 10,000 testing images divided across 10 object classes. The employed CNN model consists of six 3×3 convolution layers as follows: 32, 32, 64, 64, 128, 128. Each layer is activated by a Rectified Linear Unit (ReLU) and batch normalized. Every pair of convolution layers is followed by a 2×2 max

[1] `https://www.cs.toronto.edu/~kriz/cifar.html`

pooling layer, followed by three fully-connected layers (where each fully connected layer takes a 2D input of 382 and 192 units) with ReLU activation and another 10 units activated by soft-max. The model is trained on IoT devices using the Stochastic Gradient Descent (SGD) algorithm with a batch size of 128 rows. The training dataset was distributed over a set of 1000 IoT devices (i.e., $|J| = 1000$) of 4 types: type-1 with 1 CPU core and 1.75GB RAM, type-2 with 2 CPU cores and 3.5GB RAM, type-3 with 4 CPU cores and 7GB RAM, and type-4 with 8 CPU cores and 14GB RAM. At each iteration, the edge server selects the top 50 IoT devices returned by the scheduling algorithm (i.e., $E = 50$).

We evaluate the performance of the proposed DDQN-Trust solution against the traditional DQN [76] which has lately been used for client selection in federated learning and with the random scheduling approach, the default approach in federated learning. The proposed DDQN-Trust model consists of two Deep Neural Networks (DNNs), where each DNN has a size of $32 \times 32 \times 32$. The Adam optimizer is used to adjust the learning rate during the training. The learning rate η is initially set to 0.01 to avoid losing the local minima. In general, the deep Q learning approach prefers the long-term reward; therefore, we set the value of the discount factor γ to 0.9. We use the ϵ-greedy policy with $\epsilon = 0.9$ that balances between the exploration and exploitation. During the training phase, ϵ is linearly reduced to zero to move from exploration to exploitation. Our application is written in Python, version 3, and executed in a 64-bit Windows 7 environment on a machine equipped with an Intel Core i7-6700 CPU 3.40 GHz Processor and 16 GB RAM.

6.2.2 Experimental Results

In Fig. 6.1, we measure the accuracy of our DDQN-Trust approach against the classic DDQN that does not include our trust algorithm. The considered accuracy is yielded

by the *FedProx*, *FedShare*, *FedSGD* and *FedAvg* approaches. We ran the experiments over 1000 iterations to analyse the scalabilty of the different considered solutions. The first observation that can be drawn from the figure is that the trust-based approaches, i.e., *FedProx*, *FedShare*, *FedSGD* and *FedAvg* with DDQN-Trust achieve higher accuracy compared to the approaches that do not consider trust, i.e., *FedProx*, *FedShare*, *FedSGD*, and *FedAvg* with DDQN. In particular, the accuracy levels obtained by the *FedProx*, *FedShare*, *FedSGD*, and *FedAvg* approaches with DDQN-Trust are 91%, 85%, 83%, and 81% respectively, whereas the accuracy levels obtained by the *FedProx*, *FedShare*, *FedSGD*, and *FedAvg* approaches with DDQN are 77%, 75%, 70%, and 68% respectively. The second observation that can be drawn from Fig. 6.1 is that the trust-based approaches converge faster to a stable accuracy level compared to the non-trust approaches. The improvements brought by the trust-based approaches mainly stem from their consideration of the trust and energy values when selecting the devices that will participate in the federated training.

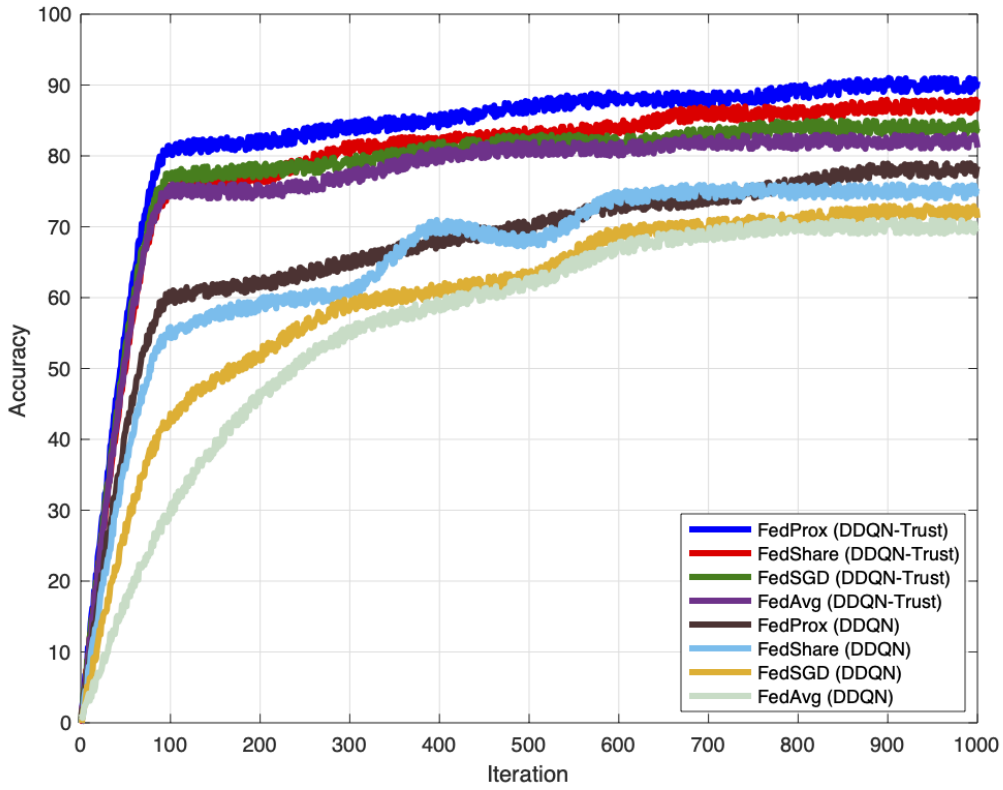
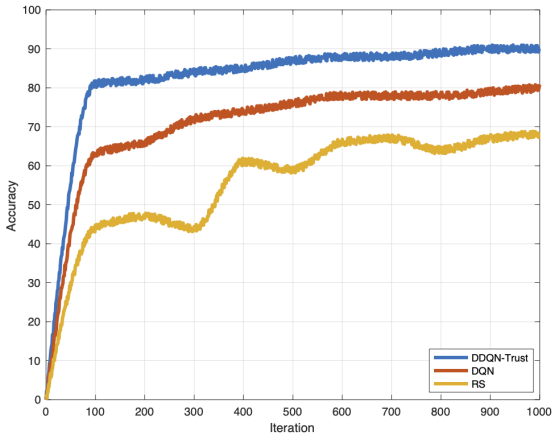


Figure 6.1: Accuracy over iteration rounds of different aggregation methods with the CNN model of our DDQN-Trust and classic DDQN

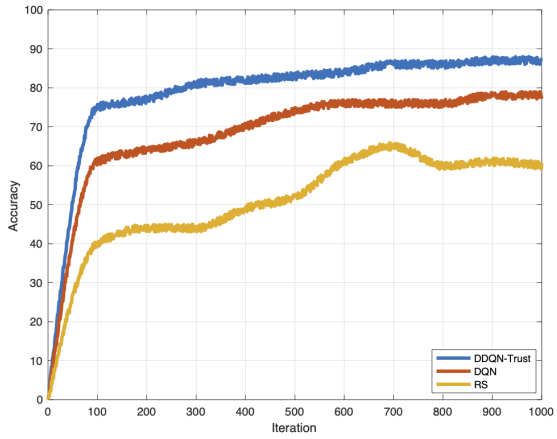
In Fig. 6.2, we compare the accuracy of our trust-based approach against the DQN and random scheduling approaches used to select the IoT devices while varying the federation approaches. Specifically, we compare the studied approaches (i.e., *FedProx*, *FedShare*, *FedSGD*, and *FedAvg*) with the DDQN-Trust, DQN, and random scheduling approaches over 1000 iterations. We notice from the sub-figures that the accuracy obtained by the DDQN-Trust scheduling approach is higher than that obtained by the DQN and random approaches under the different aggregation methods. In particular, the accuracy levels obtained by the DDQN-Trust, DQN, and random scheduling are of 91%, 80%, and 68% respectively with the *FedProx* aggregation framework, 88%, 77%, and 61% respectively with the *FedShare*

framework, 85%, 76%, and 63% respectively with the *FedSGD* framework, and 82%, 74%, and 60% respectively with the *FedAvg* framework. We also notice from the sub-figures that DDQN-Trust converges to a stable accuracy level faster than the DQN and random approaches. The improvements with regard to the random scheduling approach mainly stem from the fact that DDQN-Trust leverages the trust and energy values of the IoT devices rather than making random selections. Compared to the traditional DQN, DDQN-Trust improves the accuracy since it relies on a double Q learning model that provides a better estimation of the potential actions. This is thanks to its second Q-function approximator, which helps avoid overoptimism. In DQN, on the other hand, the Q values are noisy; thus when we take the maximum over all the actions, there is a considerable risk of obtaining an overestimated value.

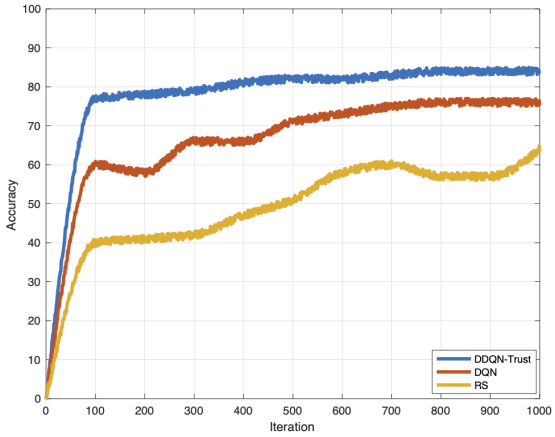
One important observation from Fig. 6.1 and Fig. 6.2 is that *FedProx* has the highest accuracy and the fastest convergence compared to the other three aggregation approaches. This is because *FedProx* addresses the system and statistical heterogeneity across the IoT devices, which makes its aggregation results more accurate. On the other hand, the classic *FedAvg* yields the lowest accuracy level and the slowest convergence, may be due to the fact that when the local data distributions across the devices are highly heterogeneous, the local updating schemes may allow local models to move too far away from the initial global model, potentially damaging the convergence. FedShare and *FedSGD* are characterized by an acceptable accuracy level. In fact, *FedShare* depends on a data-sharing strategy that seeks to distribute a small subset of independent global data, characterized by a uniform distribution over classes, to the local devices. This helps the federated learning model be less influenced by the data heterogeneity across the IoT devices. On the other hand, *FedSGD* relies on a local surrogate function that is designed for each IoT device to allow it to solve its local optimization problem approximately up to a local accuracy level.



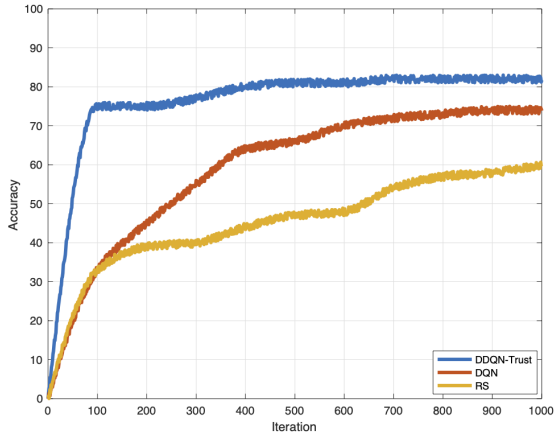
(a) **FedProx**



(b) **FedShare**



(c) **FedSGD**

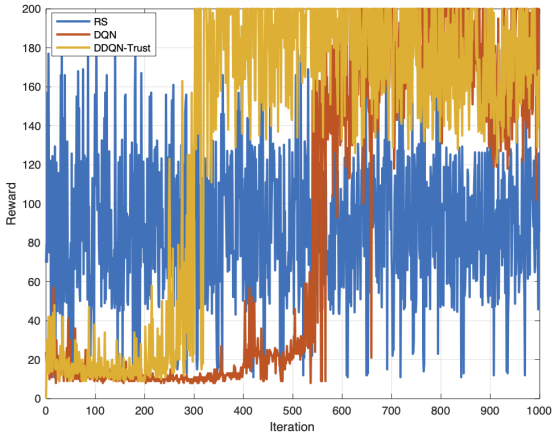


(d) **FedAvg**

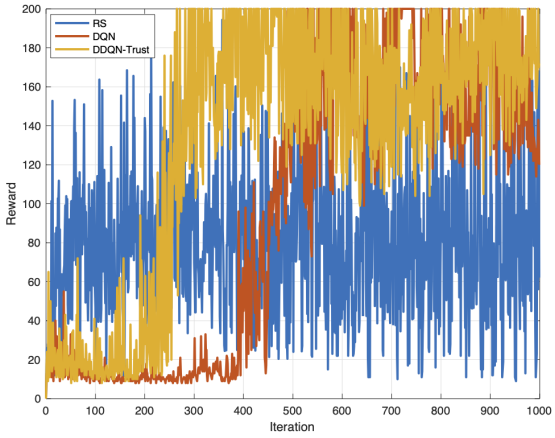
Figure 6.2: Performance of the trained CNNs with DDQN-Trust, DQN, and RS scheduling models

In Fig. 6.3, we provide experimental comparisons in terms of cumulative reward. We ran the experiments over 1000 iterations. We notice from the sub-figures that the reward obtained by the DDQN-Trust is much higher than those obtained by the DQN and random scheduling approaches. In particular, the average rewards obtained by the DDQN-Trust, DQN, and random approaches are 188, 174, and 107, respectively with the *FedProx* framework (Fig. 6.3a), 181, 167, and 97 respectively with the *FedShare* framework (Fig. 6.3b), 177, 162, and 95 respectively with *FedSGD* (Fig.

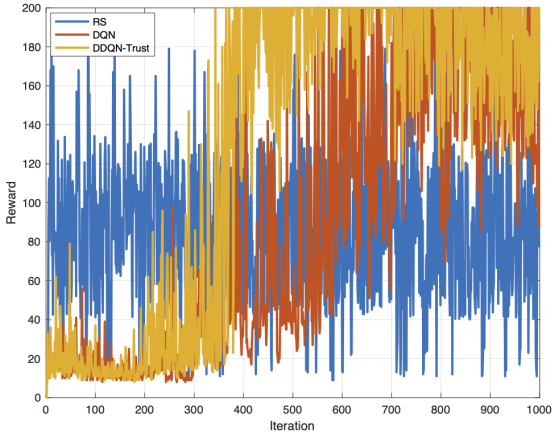
6.3c), and 162, 160, and 81 respectively with *FedAvg* (Fig. 6.3d). This means the proposed DDQN-Trust approach enables the edge server to better learn the scheduling policy that best maximizes the reward. In the random approach, the edge server randomly selects IoT devices, which increases the risk of selecting unreliable devices or devices that have insufficient energy levels and resources. This endangers the whole collaborative training process and makes the performance unstable. Moving to the traditional DQN approach, its overestimation of the future actions leads to a natural reduction in the overall reward that results from the chosen actions.



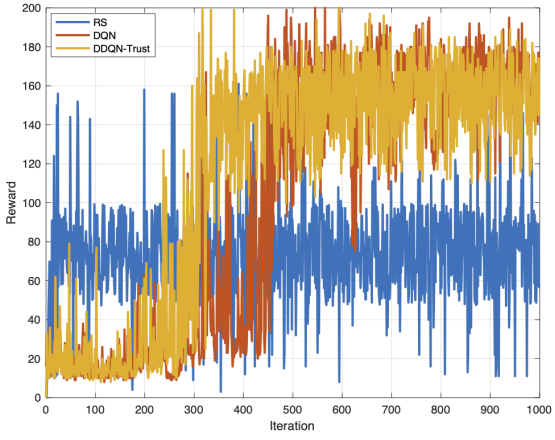
(a) **FedProx**



(b) **FedShare**



(c) **FedSGD**



(d) **FedAvg**

Figure 6.3: Reward values in DDQN-Trust, DQN, and Random scheduling policies

6.3 Conclusion

In this chapter, we designed and formulated a trust and energy-aware FL scheduling approach in IoT environments using DDQN while considering four aggregation approaches namely, *FedAvg*, *FedProx*, *FedShare*, and *FedSGD*. Experiments conducted on the CIFAR-10 real-world dataset reveal that our DDQN-Trust solution outperforms, in terms of accuracy and cumulative reward, the most commonly used scheduling approaches in FL, i.e., DQN and random scheduling. Our solution accurately selects the appropriate set of IoT devices whose participation in the federated training improves the machine learning model's accuracy. We studied the accuracy of the three models by implementing a CNN model in a federated fashion on the IoT devices and varying the aggregation approaches. The results revealed that our DDQN-Trust solution, DQN, and random scheduling yield respectively an accuracy of 91%, 80%, and 68% with the *FedProx* aggregation framework, 88%, 77%, and 61% respectively with *FedShare*, 85%, 76%, and 63% with *FedSGD*, and 82%, 74%, and 60% with *FedAvg*. Besides, our DDQN-Trust converges faster to a stable accuracy level. Finally, the results revealed that the reward obtained by our proposed solution is much higher than those obtained by the DQN and random scheduling approaches. In particular, the average rewards obtained by the DDQN-Trust, DQN, and random approaches are 188, 174, and 107, respectively with *FedProx*, 81, 167, and 97 respectively with *FedShare*, 77, 162, and 95 respectively with *FedSGD*, and 162, 160, and 81 respectively with *FedAvg*.

A major challenge for this model stems from the variability in the availability and volumes of data from one region to another and the overhead of constantly training from scratch. To tackle this challenge, we extend our environment in the next chapter by varying the number of edge server devices to share the knowledge among them instead of employing just one device. Technically speaking, we adapt and test our

solution in a COVID-19 scenario; and integrate a transfer learning method into our solution. This will enable inter-server knowledge sharing to handle the problem of COVID-19-related data scarcity in some regions.

Chapter 7

COVID-FED: Applying Smart Scheduling Approach in the Healthcare Domain

COVID-19 is undoubtedly the talk of the town across the world, owing to the sudden and non-stop outbreak of this new generation of coronavirus in most of the countries. This pandemic can, in some cases, entail some progressive respiratory failures and massive alveolar damage [4], and might sometimes lead to death. Early and automatic diagnosis is undeniably helpful to achieve timely treatment and even prevention (e.g., referral to quarantine). Several studies [19, 29, 78] have been conducted for detecting and fighting against COVID-19. In this chapter, we intend to apply our scheduling strategy in the healthcare domain to investigate its applicability in real-world scenarios. We aim to complement these studies by combining several state-of-the-art concepts from computer science and Artificial Intelligence (AI) to build a comprehensive solution for health monitoring and tracking (using IoT and

edge computing), trust-aware participant selection and job scheduling (using trust management and Deep Reinforcement Learning (DRL)), privacy-preserving machine learning (using Federated Learning (FL)) and long training time and data scarcity problems reduction (using Transfer Learning (TL)).

We propose an all-encompassing solution named COVID-FED. COVID-FED is a multi-faceted COVID-19 detection approach which incorporates federated learning, trust management, and Deep Reinforcement Learning (DRL) in an edge computing setting that considers IoT devices and medical imaging. We capitalize on federated transfer learning over IoT and edge devices for dynamic detection of COVID-19 from X-ray images as shown in Fig. 7.1. Edge servers collaborate with IoT devices to train the COVID-19 detection model using federated learning without exchanging raw confidential data. Transfer learning is important to handle the scarcity of data in some regions and compensate potential lack of learning at some servers. DRL and trust management are combined to assign the COVID-19 detection tasks to the most trusted and resource-efficient IoT devices.

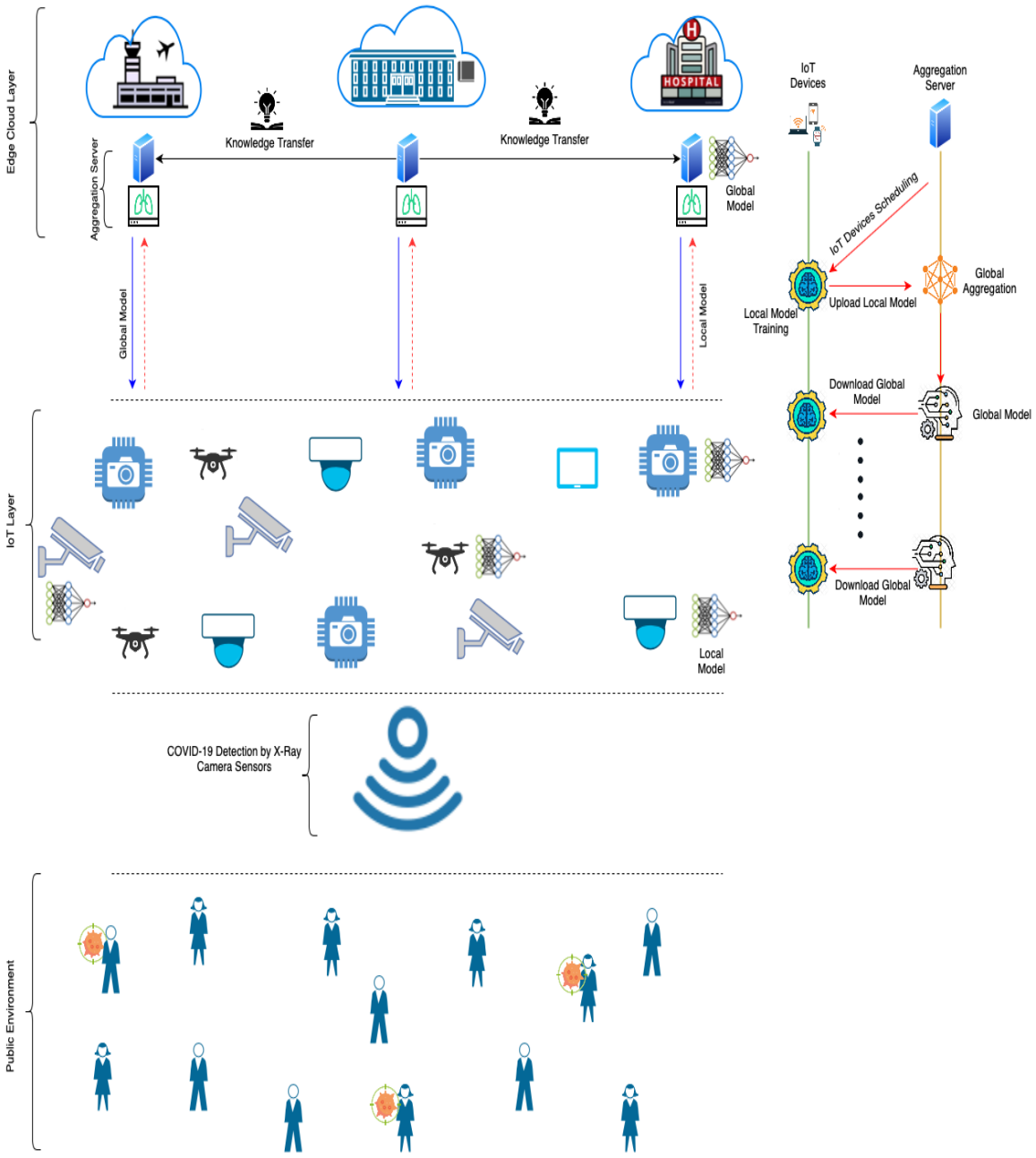


Figure 7.1: System architecture and communication process of federated transfer learning in edge cloud.

7.1 Problem Formulation

7.1.1 System Model

Let $M = \{m_1, m_2, \ldots, m_x\}$ be a set of x IoT devices charged for COVID-19 detection. Each IoT device is equipped with X-ray scanning and machine learning capabilities, which enable it to train a COVID-19 detection model. $E = \{e_1, e_2, \ldots, e_r\}$ is a set of r edge servers (ESs) responsible for aggregating the model updates received from the IoT devices. Let $T_m = \{t_{m1}, t_{m2}, \ldots, t_{ml}\}$ be a set of l training models to be analyzed on the IoT device $m \in M$, which forms a tuple and α_m be a trust value assigned by an ES from E to this IoT device. An ES $e \in E$ chooses to schedule one or more tasks over the IoT devices at each time step τ.

Let ε_m be the time needed to train a local model on the device m. We introduce two functions: $\Theta_1(\varepsilon_1, \varepsilon_2, \ldots \varepsilon_n)$ an aggregation function that computes the execution time of the overall FL process involving the selected n IoT devices participating in the federated training, and $\Theta_2(\alpha_1, \alpha_2, \ldots \alpha_n)$ an aggregation function that computes the overall trust of the selected n IoT devices. The sum of ε_i and α_i ($i := 1 \ldots n$) are examples of these aggregation functions. The objective of our scheduling solution is to minimize Θ_1 and maximize Θ_2 subject to the constraints (7.1), (7.2), (7.3), and (7.4).

$$K_{mi}^{CPU} \leq CPU_m^{\tau}, \qquad \forall t_{mi} \in T_m \tag{7.1}$$

$$K_{mi}^{RAM} \leq RAM_m^{\tau}, \qquad \forall t_{mi} \in T_m \tag{7.2}$$

$$K_{mi}^{BW} \leq BW_m^{\tau}, \qquad \forall t_{mi} \in T_m \tag{7.3}$$

$$K_{mi}^{DS} \leq DS_m^{\tau}, \qquad \forall t_{mi} \in T_m \tag{7.4}$$

where K_{mi}^{z} represents a requirement of the training model t_{mi} where each z represents a certain resource parameter, i.e., CPU, RAM, bandwidth (BW), and disk storage (DS). $CPU_{m}^{\tau}, RAM_{m}^{\tau}, BW_{m}^{\tau}, and\, DS_{m}^{\tau}$ are the current amounts of available CPU, RAM, Bandwidth, and disk storage on IoT device m.

7.2 COVID-FED Description

7.2.1 Trust Management

To model the trust relationships between an ES $e_i \in E$ and the IoT devices in terms of efficiently and honestly executing the COVID-19 detection duties, we employ the trust establishment algorithm that we recently proposed in 6.1.

The algorithm monitors the IoT devices' resource consumption over time and uses a modified Z-score approach to classify the IoT devices that exhibit some suspicious behavior in terms of over-consumption or under-consumption. This is highly important to consider in our case to identify those devices that (1) do not allocate sufficient resources to perform the COVID-19 detection tasks or (2) perform additional computations to embed some malicious goals (e.g., optimize for a malicious objective that seeks to cause misclassification) into their local training problems. Note that each ES monitors the IoT devices that are located within its range.

The main idea of the trust approach is to approximate the difference of a given score from the median using the median absolute deviation in our algorithm $MAD_{m}^{z}(\tau)$ of a metric z (e.g., CPU, RAM, BW, DS) consumed by the IoT device m at the current time τ. More specifically, the modified Z-score (β) is calculated through dividing the difference between the consumption of the device m in terms of the resource metric z at time moment τ and the median consumption of that device in terms of that metric within the time interval τ by the median absolute deviation

of the metric z as follows:

$$\beta_m^z(\tau) = \frac{\varrho(x_m^z(\tau) - \bar{x}_m^z(\tau))}{MAD_m^z(\tau)} \tag{7.5}$$

where constant $\varrho = 0.6745$ is needed because $\mathbb{E}(MAD_m^z(\tau)) = 0.6745\sigma$ for a large number n of samples.

Then the algorithm tests to decide if there is any consumption that exceeds or falls below the computed abnormal limit for any potential consumption of the IoT device.

Afterwards, by dividing the sum of this relative abnormal consumption $PropAbnormalMetrics_m^z$ over all the $z \in Z$ metrics by the number of $AbnormalMetrics_m$ of metrics that the device has overused/underused, we derive the trust value of each device α_m. If no metric has been overused/underused, the initial trust in the IoT device's trustworthiness will be 100%.

$$\alpha_m = \frac{\sum_{z \in Z} PropAbnormalMetrics_m^z}{AbnormalMetrics_m} \tag{7.6}$$

7.2.2 DRL Scheduling Policy

The proposed scheduling policy consists of a multi-layered neural network that, for a given state, outputs a vector of actions, given a set of parameters of the network [90]. The problem is formulated as a global Markov Decision Process (MDP) where the global states and global actions of the system are formulated as a combination of local states and actions of the IoT devices. The scheduling policy is defined by the tuple $\left\langle S, A, \mathcal{T}, R, \gamma \right\rangle$, where:

- S: Set of global states of the system.
- A: Set of joint actions over all the IoT devices.

- $\mathcal{T}$: Transition probability function defined as: $\mathcal{T}(s, a, s') = Pr(s'|s, a)$, where $s, s' \in S$ and $a \in A$.
- $R : S \times A \mapsto \mathbb{R}$: Reward function of the model.
- γ: Discount factor that decreases the impact of the past reward.

Let S_m be the set of local states of the IoT device $m \in M$. The global state space S is obtained through the Cartesian product of the IoT devices' local states, i.e., $S = \prod_{m \in M} S_m$. Each local state $s_m \in S_m$ is computed as follows:

$$\begin{aligned} s_m =& (\alpha_m, \vartheta_m, \delta_m^{||}); \quad \alpha_m \in \{0, 1, \ldots, \alpha^{max}\}, \\ & \vartheta_m \in \{0, 1, \ldots, \vartheta^{max}\}, \\ & \delta_m^{||} \in \{0, 1, \ldots, \delta^{max}\} \end{aligned} \tag{7.7}$$

where α_m is the trust value of the IoT device m computed in Equation (7.6), ϑ_m is the time needed to run the local model on the device m, and $\delta_m^{||}$ is the normalized amount of resources (i.e., CPU, RAM, bandwidth, and disk storage) on the device m. Trust value and execution time are dynamic, so they could change from state to state. The global action space of the parameter server is the joint action space of each device: $A = \prod_{m \in M} A_m$ where A_m is the set of local actions of m. A local action $a_m \in A_m$ is as follows:

$$\begin{aligned} a_m =& (\sigma_m, \varepsilon_m, K_m^{||}, \zeta_m); \\ & \sigma_m \in \{0, 1\}, \varepsilon_m \in \{0, 1, \ldots, \varepsilon^{max}\}, \\ & K_m^{||} \in \{0, 1, \ldots, K^{max}\}, \zeta_m \in \mathbb{R} \end{aligned} \tag{7.8}$$

where $\sigma_m = 1$ means that the parameter server assigns a training task to the IoT device m and $\sigma_m = 0$ means that the server does not assign the task to m. ε_m refers to the time needed by the IoT device m to download, train and upload the model, and

$K_m^{||}$ is the normalized amount of resources needed to assign the model from the ES to the IoT device m, and ζ_m is the cost of transmitting the model from the parameter server to the device m and running the model. For an action to be feasible from a global state s to s', the following condition should hold:

$$\varepsilon_m(s') \leq \vartheta_m(s) \qquad \forall m \in M \tag{7.9}$$

$$K_m^{||}(s') \leq \delta_m^{||}(s) \qquad \forall m \in M \ \& \ \forall z \in Z \tag{7.10}$$

where $\varepsilon_m(s')$ and $K_m^{||}(s')$ refer to ε_m and $K_m^{||}$ in the action leading to s'; and $\vartheta_m(s)$ and $\delta_m^{||}(s)$ are ϑ_m in s and $\delta_m^{||}$ in s consecutively.

The reward function R is defined in such a way to maximize the selection of trustworthy IoT devices to perform the federated training and to minimize overall training time. The cost ζ_m is also considered proportional to the maximum cost ζ^{max}. The reward Ψ_m for the device m is a function of state $s \in S$ and action $a \in A$ and is computed as follows:

$$\Psi_m(s,a) = \begin{cases} \sigma_m.\varepsilon_m.K_m^{||} - \frac{\zeta_m}{\zeta^{max}}, & \text{if } \ K_m^{||}(s') \leq \delta_m^{||}(s). \\ & \textit{and} \quad \varepsilon_m(s') \leq \vartheta_m(s). \\ -\frac{\zeta_m}{\zeta^{max}}, & \text{otherwise.} \end{cases} \tag{7.11}$$

The ES accounts for the available resources on the IoT devices in addition to the trust scores of these devices to ensure that they have sufficient capacity to download, train and upload the machine learning model. Thus, the global reward of the parameter server is given by the following equation:

$$R(s,a) = \sum_{m \in M} \Psi_m(s,a) \tag{7.12}$$

The parameter ES determines the optimal policy $\pi^* : S \rightarrow A$ that indicates the actions to be taken at each state to maximize the cumulative reward. The Q-learning (QL) algorithm's essential goal to find π^* is to update the Q-value of a state-action pair, $Q(s,a)$, which encodes the expected future discounted reward for taking action a in certain state s. The optimal action-value function $Q^*(s,a)$ is $Q^*(s,a) = \max_{\pi} Q_\pi(s,a)$. This optimal value function can be nested within the Bellman optimality equation as follows:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} Pr(s'|s,a). \max_{a' \in A} Q^*(s',a') \tag{7.13}$$

The parameter server calculates the optimal action from any state to maximize the cumulative reward, based on the Q-table that emerges from updating the $Q(s,a)$ values. The QL algorithm is only feasible for networks with limited state and action spaces, but the issue of assigning training tasks to IoT devices becomes highly dimensional when the number of network participants increases (as is the case for IoT networks consisting of a large number of devices). To solve the high dimensionality problem, the Deep QL (DQL) algorithm (a combination of QL and Deep Neural Network DNN) comes into play.

The DNN takes as input one of the online network's states, and outputs the Q-values $Q(s,a;\theta)$, as well as the weight-matrix θ of the DNN of all eventual actions. In order to obtain approximate values $Q^*(s,a)$, DNN needs to be trained using experience $(s,a,R(s,a),s')$. To define the loss function, we use the mean square error (MSE), and DNN uses the Bellman equation to minimize the following loss function:

$$L(\theta_i) = E[(R(s,a) + \gamma \arg\max_{a' \in A} Q(s', a'; \theta_i') - Q(s, a; \theta_i))^2] \tag{7.14}$$

where θ_i denotes the online network parameters of the i^{th} iteration, θ_i' represents the goal network parameters of the i^{th} iteration, and $E[.]$ represents the expected value. Note that the action a is chosen on the basis of the ϵ-greedy policy [66].

7.2.3 Federated Learning Model

We adopt the FL to achieve privacy-preserving on-device COVID-19 data analytics. Let D_m be a local X-ray images dataset collected by IoT device m. $D_m = \{(x_{1_m}, y_{1_m}), \ldots, (x_{n_m}, y_{n_m})\}$, where x_{i_m} is the i^{th} training sample and y_{i_m} represents the corresponding ground-truth label. In this work, we employ a general CNN model to perform our analysis on the X-ray data. The ES first trains a global CNN model on a publicly available X-ray dataset and then sends the initial parameters to the set of IoT devices selected as per our scheduling solution discussed in Section 7.2.2. These IoT devices capitalize on the shared parameters to locally train the CNN on their own set of collected X-ray images and hence derive an updated set of the parameters. Upon receiving the updated parameters from the IoT devices, the server aggregates (using Equation (7.15)) these parameters to derive a global aggregate model.

$$g\left[\nu\right] = \frac{1}{\sum\limits_{m \in M} |\lambda_m|} \sum_{m \in M} |\lambda_m| g_m\left[\nu\right] \tag{7.15}$$

where $\lambda_m \subseteq D_m$ is a subset of local data collected by IoT device m for a training period ν and $g_m\left[\nu\right]$ is the local gradient which is computed as per Equation (7.16).

$$g_m\left[\nu\right] = \nabla_{w_m} L_m\left(w_m, \lambda_m\right) \tag{7.16}$$

where w_m is the set of local parameters of the CNN model, L_m is the local loss function (in terms of training error) to be minimized on IoT device m and $\nabla_{w_m} L_m(.)$ is the gradient of the loss function L_m with respect to w_m.

Algorithm 7.1 DRL-based Federated Learning Algorithm for COVID-19 Detection

1: Initialize the global parameter set of the CNN model on a publicly available X-ray image data
2: **for** each round $\varphi = 1, 2, \ldots$ **do**
3: **Use** our scheduling solution described in Section 7.2.2 to select a subset $\mathcal{E} \subseteq M$ of IoT devices to participate in the training
4: **Send** W_φ to each selected IoT
5: **for** each IoT device $m \in \mathcal{E}$ **do**
% $\mathcal{E} = \{1, 2, \ldots, S\}$
6: Execute **IoTLocalUpdate**(m, W_φ)
% See Algorithm 7.2
7: **end for**
8: $W_\varphi = \frac{1}{n} \sum_{m=1}^{S} n_m w_m$
9: **end for**

We explain in Algorithms 7.1 and 7.2 the FL process we propose to recognize COVID-19. In Algorithm 7.1, n_m is the volume of the data that are available on IoT device m, n is the volume of the overall data across all IoT devices, S is the total number of selected IoT devices to participate in the FL process, φ is an index that represents a training communication round and W_φ is the set of global parameters at training round φ.

The Stochastic Gradient Descent (SGD) algorithm is run by each IoT device based on the obtained global gradient. The local loss function $L_m(w_m)$, which has to be minimized on each device m, is calculated as shown in Equation (7.17):

$$L_m(w_m) = \frac{1}{N_m} \sum_{(x,y) \in D_m} \ell(w_m, x, y) \tag{7.17}$$

where $\ell(w_m, x, y)$ is the sample-level loss function that quantifies the prediction error between the learning output (via the input x and parameter w_m) and ground-truth

label y, and N_m is the number of data samples on device m. Each device attempts to minimize this local loss function, thereby reducing the error of the training process. The main objective of the global model at the ES level is to optimize the set of parameters to minimize the global loss function $L(W)$ using the SGD algorithm as shown in Equation (7.18):

$$L(W) = \frac{1}{\sum_{m=1}^{S} N_m} \sum_{m=1}^{S} N_m L_m(w_m) \tag{7.18}$$

Algorithm 7.2 IoT Local Training

1: **IoTLocalUpdate**(m, W_φ)
2: $w_m = W_\varphi$
3: **for** each local iteration $i = 1$ to T **do**
4: $w_m = w_m - \eta \nabla_{w_m} L_m(w_m, \lambda_m)$
% η is the learning rate
5: **end for**
6: return w_m to the ES

7.2.4 Inter-Edge Transfer Learning

Our solution includes a TL component to allow the ESs to share their knowledge with one another, without revealing their raw data. Doing so is useful in many situations such as:

- One ES is newly deployed, but another server has already explored some knowledge.
- Some ESs do not have enough IoT devices in their vicinity or have IoT devices that do not have enough data to obtain efficient learning.
- Some ESs couldn't obtain enough knowledge for a given task compared to the knowledge obtained by other servers.

Algorithm 7.3 ESs Knowledge Transfer

1: **Input:** $E = \{e_1, e_2, \ldots, e_r\}$
2: **Receive local model updates**(w_m)
3: **for** each round $\varphi = 1, 2, \ldots$ **do**
4: **for** each edge server $e \in E$ **do**
5: Aggregate global parameters W_{φ}^{e}
6: Calculate global loss function $L\left(W_{\varphi}^{e}\right)$
7: **end for**
8: Calculate the optimal Global Model over E:
9: $W^* = \underset{W_{\varphi}^{e}, \forall e \in E}{\arg\min}(L(W_{\varphi}^{e}))$
10: **Send** W^* to each selected IoT
11: **end for**

The intuition is that knowledge transferred among the aggregation servers can boost the optimization on the edge. As such, we propose to transfer knowledge bidirectionally. We explain in Algorithm 7.3 the TL process that is proposed to share the knowledge among ESs. In this algorithm, each ES $e \in E$ aggregates the local models received from the selected IoTs (Algorithm 7.1) and then calculates the global loss function (Equation (7.18)), where W_{φ}^{e} is the global model W_{φ} on the ES e. The optimal global model W^* is calculated over all the ESs based on the minimum loss function value (line 9) before sending it back to the selected IoT devices.

7.3 Experimental Results and Analysis

We explain in this section the environment employed to perform our experiments and present and analyze the experimental results.

7.3.1 Experimental Setup

To carry out our experiments, we capitalize on the dataset[1] which consists of chest X-ray images for individuals infected with COVID-19, individuals with viral

[1] `https://www.kaggle.com/tawsifurrahman/covid19-radiography-database`

Pneumonia, as well as normal images. In total, the dataset consists of 219 COVID-19 positive images, 1341 normal images and 1345 viral pneumonia images. The data distribution is non-IID and unbalanced, reflecting the characteristics of real-world FL scenarios.

We train a CNN model to determine our algorithm's efficiency and effectiveness. The CNN model used consists of six 3×3 convolution layers as follows: 32, 32, 64, 64, 128, 128. The Rectified Linear Unit (ReLU) activates each layer and normalizes the batch. Every pair of convolution layers is followed by a 2×2 max pooling layer, then by three fully-connected layers (where each fully connected layer takes a 2D input of 382 and 192 units) with ReLU activation and another 10 units activated by the soft-max. We employ the TensorFlow Federated (TFF) platform, which provides an open source framework for decentralized data learning. TFF facilitates a variety of collaborative learning scenarios on a number of heterogeneous devices with different resources. The SGD algorithm is used to train the model on IoT devices with a batch size of 128 rows per IoT device for every training round. We distributed the training data on 1000 IoT devices (i.e., $|M| = 1000$) of four various types:

- Type-1 with 1 CPU core and 1.75GB RAM,
- Type-2 with 2 CPU cores and 3.5GB RAM,
- Type-3 with 4 CPU cores and 7GB RAM, and
- Type-4 with 8 CPU cores and 14GB RAM.

The ES selects the top 50 IoT devices returned by the scheduling algorithm (e.g., $E = 50$), at each iteration. Our program is written in Python 3 and executed on a 64-bit Windows 7 threaded environment on an Intel Core i7 3.40 GHz CPU and 16 GB of RAM.

7.3.2 Experimental Results

In Fig. 7.2, we compare the accuracy of our solution under different combinations, i.e., Trust, Deep Reinforcement learning, Federated and Transfer learning (TDRFT); Trust, Deep Reinforcement learning, and Federated learning (TDRF); and Deep Reinforcement learning, Federated and Transfer learning (DRFT). We also compare these different combinations of our solution with two common scheduling approaches, i.e., Round Robin (RR) and Random Scheduling (RS) while integrating our trust establishment solution into them. The different approaches are executed at five different ESs to closely inspect the accuracy values.

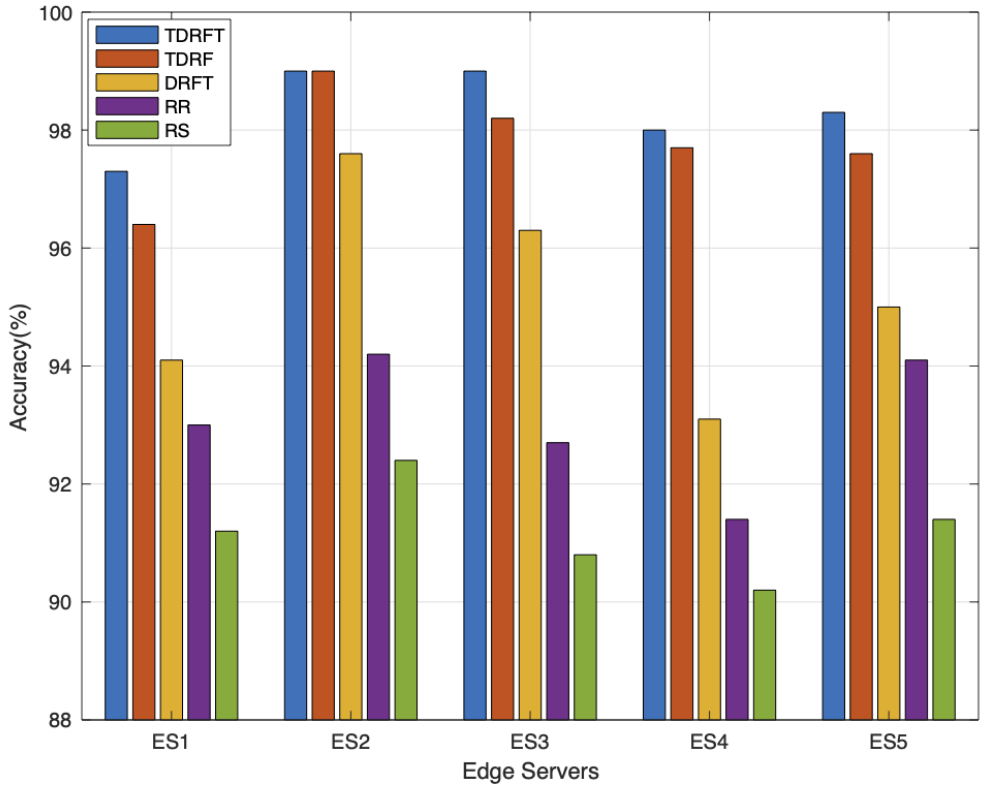
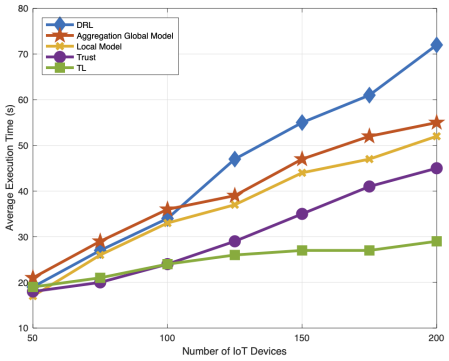


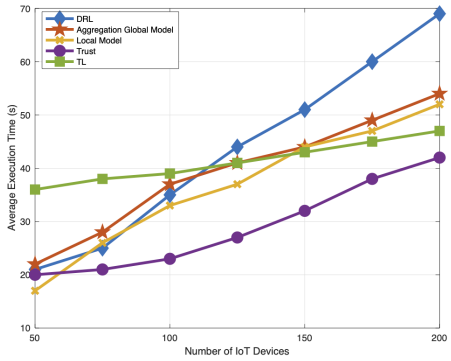
Figure 7.2: Comparison of accuracy of final global model at five different ESs

We notice from the figure that the accuracy obtained with TDRFT is higher

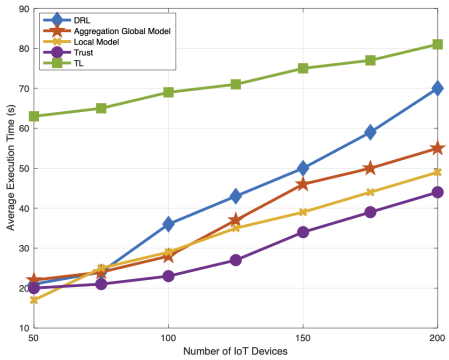
than that obtained with the other combinations and approaches. In particular, the accuracy levels obtained with TDRFT varies between 97.2 and 99.1 across the five edge servers. With TDRF, the accuracy level varies between 96.4 and 99.1. With DRFT, the accuracy level varies between 93 and 97.7. With RR, the accuracy level varies between 91.6 and 94.2. Finally, with RS, the accuracy level varies between 90.2 and 91.6. Thus, we conclude that our solution with all of its components improves the accuracy of detecting COVID-19 cases. The reason is that it employs deep Q-learning to select the IoT devices that achieve the best combinations in terms of resource availability and trust maximization, and includes a TL component to compensate the lack of learning from which some edge servers might suffer.



(a) 5 edge servers



(b) 25 edge servers



(c) 50 edge servers

Figure 7.3: Average execution time of the proposed model phases

In Fig. 7.3, we measure the execution time of each single component of our solution. To do so, we vary the number of IoT devices from 50 to 200, while also varying the number of ESs from 5, to 25, to 50. The main observation that can be drawn from this figure is that increasing the number of IoT devices leads to a modest increase in the execution time, especially when it comes to the DRL phase. This can be justified by the fact that having a larger number of IoTs to assign the tasks to increases the search space of the most of the components, except for the TL phase, which is independent from the number of IoT devices. The second observation is that the increase in the number of the ESs leads to increasing the execution time of the TL component, without having any significant impact on the other components. The reason is that having more servers means that more TL processes might need to be performed among these servers.

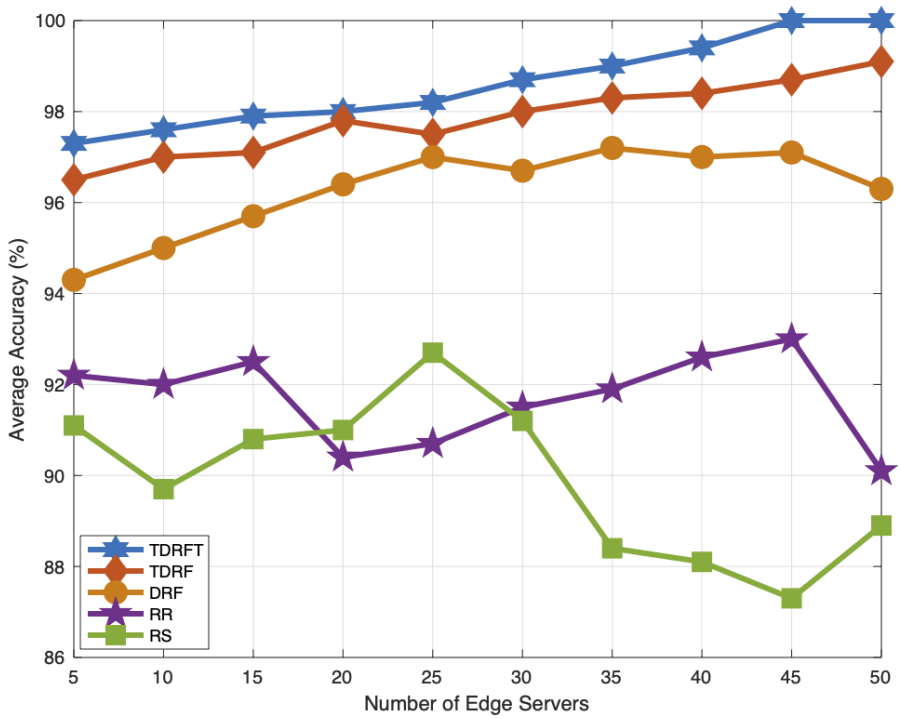


Figure 7.4: Comparison of average accuracy of final global model of varying number of ESs

In Fig. 7.4, we provide experimental comparisons in terms of average accuracy while varying the number of edge server between 5 and 50. Again, in this scenario, the accuracy obtained with TDRFT is much higher than that obtained than the rest of the compared approaches. In particular, the average accuracy obtained by the TDRFT, TDRF, DRF, RR and RS approaches are $97.3-100$, $96.4-98.8$, $94.2-97.1$ $90-93.2$ and $87.3-92.6$, respectively. This means that the proposed TDRFT approach enables the edge server to better learn and recognize COVID-19 by adopting our solution with all of its components.

In Fig. 7.5, we measure the execution time of the different studied approaches, while varying the number of IoT devices from 50 to 200 and varying the number of servers from 5 to 100. The main observation that can be drawn from this simulation is that increasing the number of IoT devices leads to a modest increase in the execution time in our solution (i.e., TDRF) compared to the other models. This is because our solution employs deep Q-learning to select the IoT devices that achieve the best combinations in terms of resource availability and trust maximization. On the other hand, increasing the number of ESs leads to a modest increase in the execution time in all the studied approaches, even in our model as we use TL to exchange knowledge among the servers.

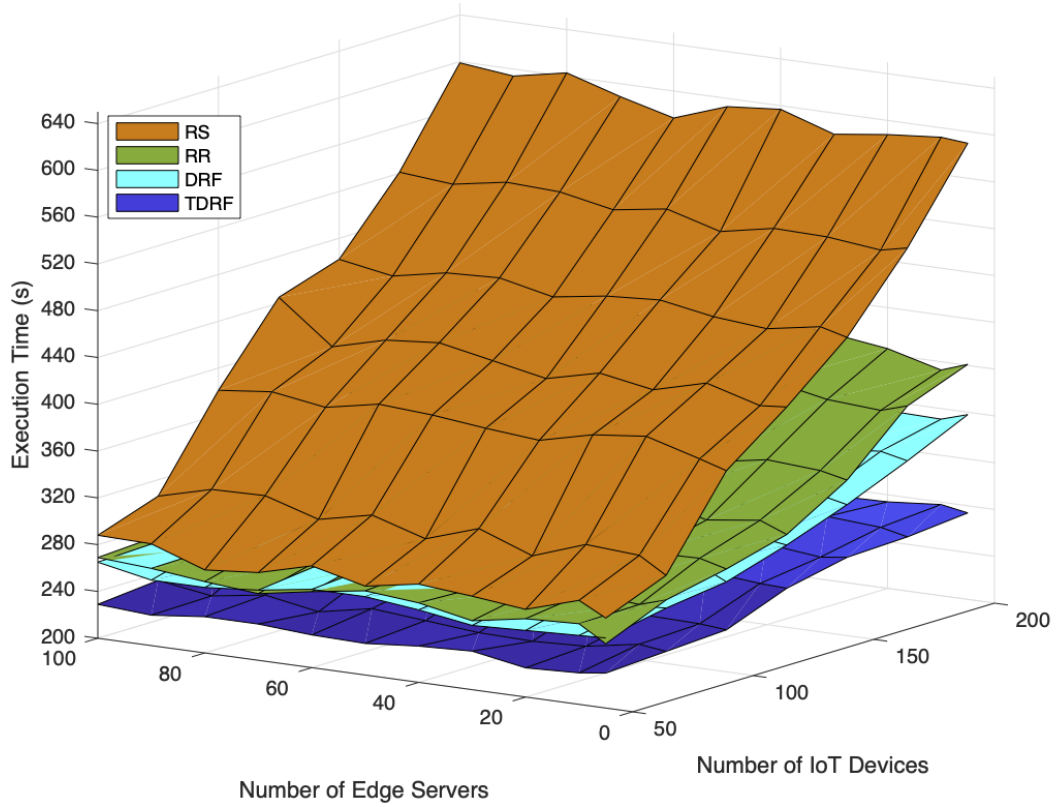
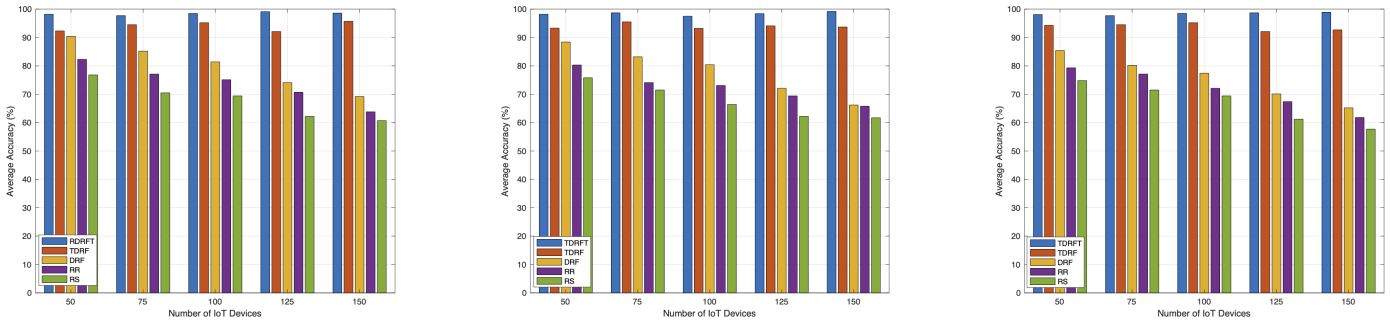
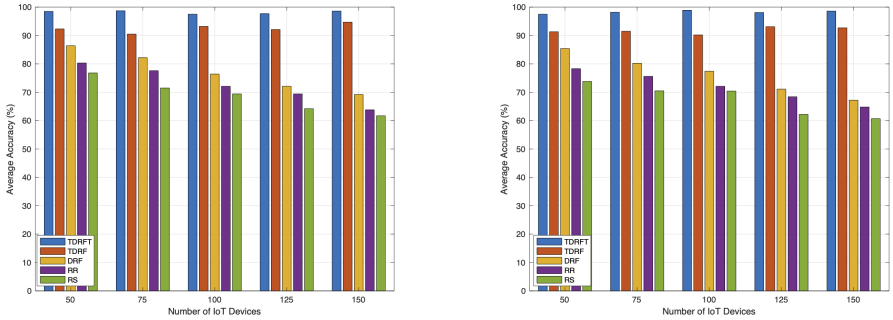


Figure 7.5: **Execution time:** We study in this figure the impact of varying both the number of IoT devices and number of ESs on the execution time

In Fig. 7.6, we measure the average accuracy of the CNN that was trained by IoT devices selected by the TDRFL, TDRF, DRF, RR, and RS approaches. We ran the experiments over 1000 iterations (i.e., $T = 1000$) to study the scalability of the different considered solutions, while varying the number of ESs from 10 to 50 and also varying the number of IoT devices from 50 to 150. The main observation that can be drawn from this experiment is that our proposed solution, with all of its components, achieves the highest accuracy level compared to the other approaches and exhibits a better scalability to an increasing number of IoT devices and ESs. This is justified by the fact that having a larger number of IoT devices to select from might increase the probability of mistakenly assigning global learning to some inappropriate IoT devices.



(a) 10 Edge Servers **(b)** 20 Edge Servers **(c)** 30 Edge Servers

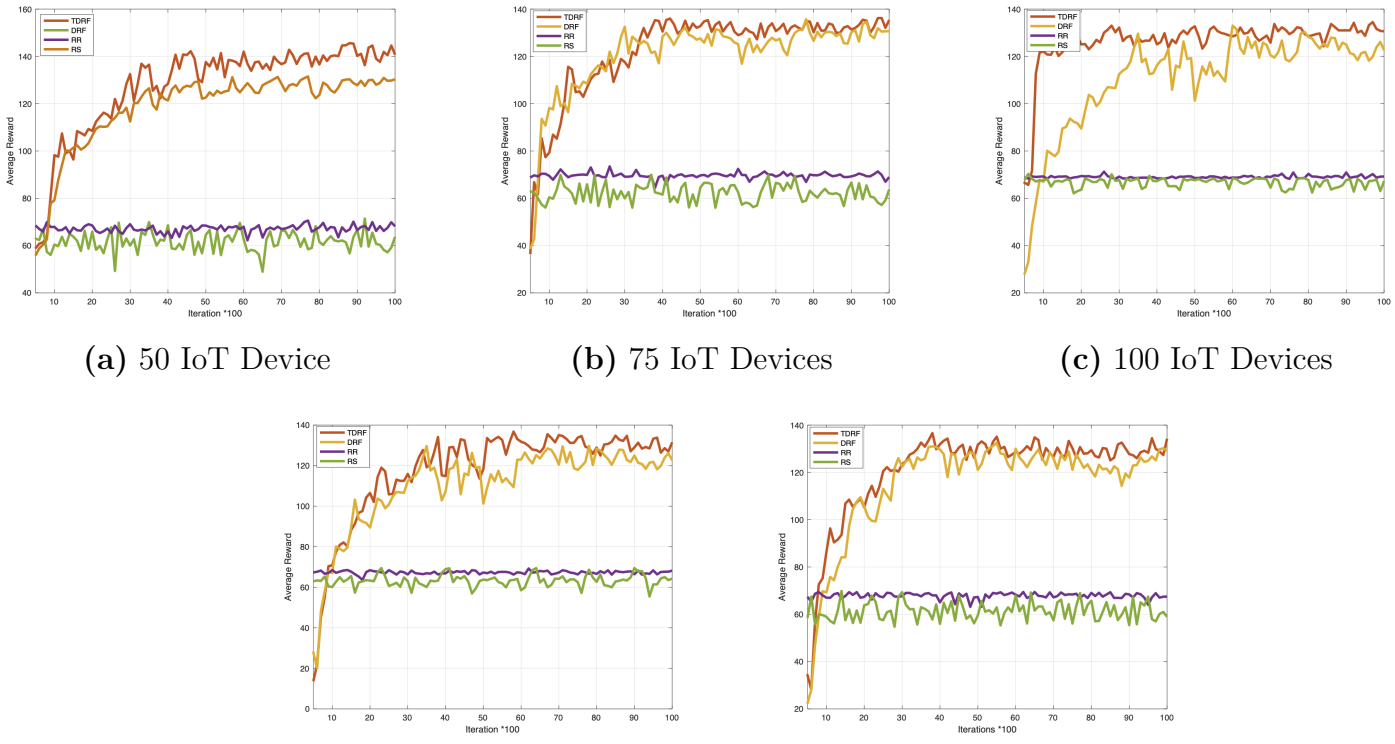


(d) 40 Edge Servers **(e)** 50 Edge Servers

Figure 7.6: Average accuracy values in TDRFT, TDRF, DRF, RR, and RS

Yet, the TL component that we integrate in our solution, which enables sharing the knowledge among servers, leads to increasing the accuracy at the level of some servers that might have made some poor selections in terms of IoT devices.

In Fig. 7.7, we provide experimental comparisons in terms of average reward. We ran the experiments over 10000 (i.e., $T = 10000$) iterations. We observe from this figure that the average rewards obtained by TDRF and DRF are much higher than those obtained by the RR and RS approaches. This means that TDRF enables the ES to better learn how to schedule the COVID-19 detection tasks in such a way that best maximizes the reward in terms of minimizing the execution time and maximizing the trust.



(a) 50 IoT Device **(b)** 75 IoT Devices **(c)** 100 IoT Devices

(d) 125 IoT Devices **(e)** 150 IoT Devices

Figure 7.7: Average reward values in TDRF, DRF, RR, and RS

7.4 Conclusion

In this chapter, we proposed a multi-faceted and comprehensive COVID-19 detection approach called *COVID-FED*, that combines medical imaging, federated transfer learning, trust management, DRL, IoT and edge computing. To the best of our knowledge, no existing approach has yet considered the integration and interconnection between all these technologies for COVID-19 detection. This makes our approach the most holistic in the literature through considering the different aspects that are necessary for COVID-19 detection such as health monitoring and tracking (using IoT and edge devices), trust-aware participant selection and job scheduling (using trust management and DRL), privacy-preserving machine learning (using FL) and training time reduction (using TL). Experiments conducted

on a real-world COVID-19 dataset reveal that our solution achieves a good trade-off between detection accuracy and model execution time compared to existing approaches. The results show as well that the components of our solution are important to the success of our approach.

Chapter 8

Conclusion

8.1 Summary and Discussion

In this thesis, we proposed a new automated scheduling approach in cloud and edge computing environments using swarm intelligence and machine learning, in particular deep, reinforcement and federated learning. We addressed the resource management automation process in the cloud system and introduced intelligent techniques that help cloud providers schedule tasks to the trusted IoT devices while minimizing resources utilization and the overall cost. For each of our contributions, we conducted an in-depth literature review to guarantee the originality of our solutions and their effectiveness in filling the state-of-the-art research gaps.

First, we proposed a hybrid approach, called Multi Label Classifier Chains Swarm Intelligence (MLCCSI) to find the optimal resource allocation for each task in the dynamic cloud system. We conducted experiments using the ACO, ABC and PSO algorithms and a new scheduling model that uses these three algorithms. The experiments revealed that this solution improves the load balancing scheduling performance and minimizes the average makespan by 75% compared to the ACO

algorithm, by 61% compared to the ABC algorithm, and by 53% compared to the PSO algorithm.

Second, we elaborated a novel trust-aware scheduling solution for big data tasks called *BigTrustScheduling* that consists of three stages: VMs' trust level computation, tasks priority level determination, and trust-aware scheduling. Experiments conducted on a real Hadoop cluster environment using real-world datasets collected from the Google Cloud Platform pricing and Bitbrains task and resource requirements revealed that our solution minimizes the makespan up to 48% and reduces the monetary cost by 58% compared to the state-of-the-art big data tasks scheduling approaches.

Third, we advanced four automated task scheduling approaches in cloud computing environments using deep and reinforcement learning, while reducing both the resource consumption and task waiting time. Experiments conducted using real-world dataset from GCP pricing and Google cluster resource and task requirements revealed that this solution minimizes the CPU utilization cost up to 32%, and reduces the RAM utilization cost by 41% compared to the state-of-the-art scheduling strategies.

Fourth, we developed a trust establishment technique for IoT devices to find the optimal FL scheduling decisions while considering four aggregation approaches namely, *FedAvg*, *FedProx*, *FedShare*, and *FedSGD*. We studied the accuracy of the state-of-the-art models by implementing a CNN model in a federated fashion on IoT devices and varying the aggregation approaches. Experiments conducted on the CIFAR-10 real-world dataset revealed that our solution outperforms the baselines in terms of accuracy and cumulative reward. Technically speaking, our model maximizes the accuracy between 19% and 27% and the reward between 33% and 81%.

Finally, we applied our model in the context of a real health care case

(COVID-19). Experiments conducted on a concrete COVID-19 dataset showed that our solution achieves a good trade-off between detection accuracy and model execution time compared to relevant existing approaches.

8.2 Contributions

A summary of this thesis contributions is as follows:

1. We proposed a hybrid approach, called Multi Label Classifier Chains Swarm Intelligence (MLCCSI). This approach is based on two strategies. The first strategy is the swarm intelligence, which we applied on the Ant Colony Optimization (ACO) algorithm, Artificial Bee Colony (ABC) algorithm and, Particle Swarm Optimization (PSO) algorithm to find the optimal resource allocation for each task in the dynamic cloud system. Then, the second strategy is the application of the machine learning algorithm (Classifier Chains) on the results from the three algorithms, which generates a new hybrid model considering the size of the tasks and the number of the virtual machines.

2. We put forward a comprehensive inter-cloud trust-aware scheduling mechanism to increase the performance of big data services execution. The proposed trust framework combines performance information related to the average heartbeat response time, average heartbeat frequency ratio, and VMs resources utilization to derive trust values for each VM. To study the performance of our proposed trust framework, we introduced four machine learning approaches for automated task scheduling in cloud computing environments. We studied and compared the performance of these approaches and identified the best one in terms of minimizing the task execution cost and delay.

3. We designed a trust establishment technique for IoT devices. The algorithm monitors the CPU and memory consumption of the IoT devices and employs a modified Z-score statistical method to identify the devices that exhibit any abnormal behavior in terms of over-consumption or under-consumption.

4. We introduced DDQN-Trust, an algorithm that enables the edge servers to find the optimal scheduling decisions in terms of energy efficiency and trustworthiness. The algorithm is designed to solve the optimization problem while modeling the uncertainty that the server faces regarding the resource and trust levels of the IoT devices. We integrated four aggregation approaches, namely *FedAvg*, *FedProx*, *FedShare* and *FedSGD* into our DDQN-Trust solution. This is important to broaden the applicability of our solution to a wider set of federated learning scenarios.

5. We proposed a multi-faceted approach which integrates federated transfer learning, IoT, edge computing, trust management and DRL. We considered the integration and interconnection among all these technologies for COVID-19 detection.

The first and second contributions are proposed to answer our first research objective (**Objective** 1), which aims to guide the cloud choose the scheduling technique by using multi criteria decision to optimize the performance. The second and third contributions are contributing to answer our second research objective (**Objective** 2), which is about enabling the edge servers to find the optimal scheduling decisions in terms of trustworthiness. The third and fourth contributions answer our third research objective (**Objective** 3), which targets the long-term dependencies in the process of automating the scheduling of large-scale workloads onto cloud computing resources. Finally, the fourth and fifth contributions aim to answer the

fourth research objective (**Objective** 4), which is about building a distributed and automated big data scheduling model in the edge computing environments.

8.3 Directions for Future Work

The above-discussed thesis contributions are effective in solving some interesting research gaps in the literature. However, some points still need further study and investigation. We summarize in the following list the main persisting gaps that we believe, based on our literature reviews, are worth investigating in the future:

- There is a need to develop and study task optimization scheduling algorithms with different challenges such as migration and quality of service constraints. This study will include investigating several machine learning techniques to handle multi-label data such as k-nearest neighbors, decision trees, and neural networks with consideration of other relevant metrics such as the capacity of CPU, RAM, and bandwidth.

- The current federated learning scheduling models disregard the combination between cybersecurity and trust based resource-aware problem in the formation process. This raises the need to develop some security and resource-aware formation solutions that guide each cloud provider to decide about assigning the learning model to the trusted IoT devices based on resource availability and security.

- The existing trust models in the domain of cloud computing are focusing only on the trust relationships between edge servers and IoT devices but disregard the trust relationships among the edge cloud's internal components and between the IoT devices. In this thesis, we made a first step toward investigating the

intra-cloud trust relationships by exploring the trust connections between the edge servers and IoT devices. However, further efforts are needed to investigate the relationships among the other components of the cloud centers (e.g., servers, IoT devices ,databases, etc.).

- There is a need to build smarter detection and scheduling strategies, including specific threat detection of patterns such as attack networks and trust IoT devices based-resource availability. Deep reinforcement learning for the detection of malicious behaviors combined with federated learning for a distributed scheduling is a potential investigation path.

Bibliography

[1] SP Abirami and Shalini Ramanathan. Linear scheduling strategy for resource allocation in cloud environment. *International Journal on Cloud Computing: Services and Architecture (IJCCSA)*, 2(1):9–17, 2012.

[2] Ali Al Buhussain, E Robson, and Azzedine Boukerche. Performance analysis of bio-inspired scheduling algorithms for cloud environments. In *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*, pages 776–785. IEEE, 2016.

[3] Tran The Anh, Nguyen Cong Luong, Dusit Niyato, Dong In Kim, and Li-Chun Wang. Efficient training management for mobile crowd-machine learning: A deep reinforcement learning approach. *IEEE Wireless Communications Letters*, 8(5):1345–1348, 2019.

[4] Ioannis D Apostolopoulos and Tzani A Mpesiana. COVID-19: automatic detection from x-ray images utilizing transfer learning with convolutional neural networks. *Physical and Engineering Sciences in Medicine*, 34(4):98–105, 2020.

[5] Wei Bao, Dong Yuan, Zhengjie Yang, Shen Wang, Wei Li, Bing Bing Zhou, and Albert Y Zomaya. Follow me fog: Toward seamless handover timing schemes in a fog computing environment. *IEEE Communications Magazine*, 55(11):72–78, 2017.

[6] Enda Barrett, Enda Howley, and Jim Duggan. Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. *Concurrency and Computation: Practice and Experience*, 25(12):1656–1674, 2013.

[7] Sayantani Basu, Marimuthu Karuppiah, K Selvakumar, Kuan-Ching Li, SK Hafizul Islam, Mohammad Mehedi Hassan, and Md Zakirul Alam Bhuiyan. An intelligent/cognitive model of task scheduling for iot applications in cloud computing environment. *Future Generation Computer Systems*, 2018.

[8] Ahmed Saleh Bataineh, Jamal Bentahar, Rabeb Mizouni, Omar Abdel Wahab, Gaith Rjoub, and May El Barachi. Cloud computing as a platform for monetizing data services: A two-sided game business model. *arXiv preprint arXiv:2104.12762*, 2021.

[9] Ahmed Saleh Bataineh, Jamal Bentahar, Omar Abdel Wahab, Rabeb Mizouni, and Gaith Rjoub. A game-based secure trading of big data and iot services: Blockchain as a two-sided market. In *International Conference on Service-Oriented Computing*, pages 85–100. Springer, 2020.

[10] Ahmed Saleh Bataineh, Rabeb Mizouni, Jamal Bentahar, and May El Barachi. Toward monetizing personal data: A two-sided market analysis. *Future Generation Computer Systems*, 2019.

[11] Ahmed Saleh Bataineh, Rabeb Mizouni, Jamal Bentahar, and May El Barachi. Toward monetizing personal data: A two-sided market analysis. *Future Generation Computer Systems*, 111:435–459, 2020.

[12] Ahmed Saleh Bataineh, Rabeb Mizouni, May El Barachi, and Jamal Bentahar. Monetizing personal data: a two-sided market approach. *Procedia Computer Science*, 83:472–479, 2016.

[13] Upendra Bhoi, Purvi N Ramanuj, et al. Enhanced Max-Min task scheduling algorithm in cloud computing. *International Journal of Application or Innovation in Engineering and Management (IJAIEM)*, 2(4):259–264, 2013.

[14] William M Bolstad and James M Curran. *Introduction to Bayesian statistics*. John Wiley & Sons, 2016.

[15] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm intelligence: from natural to artificial systems*. Number 1. Oxford university press, 1999.

[16] George Michailidis BoonyarithSaovapakhiran and Michael Devetsikiotis. Aggregated-dag scheduling for job flow maximization in heterogeneous cloud computing. In *Proc. IEEE Global Telecommunication Conference, Houston*, 2011.

[17] George EP Box and George C Tiao. *Bayesian inference in statistical analysis*, volume 40. John Wiley & Sons, 2011.

[18] Theodora S Brisimi, Ruidi Chen, Theofanie Mela, Alex Olshevsky, Ioannis Ch Paschalidis, and Wei Shi. Federated learning of predictive models from federated electronic health records. *International journal of medical informatics*, 112:59–67, 2018.

[19] Luca Brunese, Francesco Mercaldo, Alfonso Reginelli, and Antonella Santone. Explainable deep learning for pulmonary disease and coronavirus COVID-19 detection from X-rays. *Computer Methods and Programs in Biomedicine*, 196:105608, 2020.

[20] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N Calheiros. Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities. In *High Performance Computing & Simulation, 2009. HPCS'09. International Conference on*, pages 1–11. IEEE, 2009.

[21] Yi Cai, Zhutian Chen, and Huaqing Min. Improving particle swarm optimization algorithm for distributed sensing and search. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2013 Eighth International Conference on*, pages 373–379. IEEE, 2013.

[22] Rodrigo N Calheiros, Rajiv Ranjan, César AF De Rose, and Rajkumar Buyya. Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services. *arXiv preprint arXiv:0903.2525*, 2009.

[23] Huangning Chen and Wenzhong Guo. Real-time task scheduling algorithm for cloud computing based on particle swarm optimization. In *International Conference on Cloud Computing and Big Data in Asia*, pages 141–152. Springer, 2015.

[24] Huankai Chen, Frank Wang, Na Helian, and Gbola Akanmu. User-priority guided Min-Min scheduling algorithm for load balancing in cloud computing. In *2013 National Conference on Parallel computing technologies*, pages 1–8. National Conference on Parallel Computing Technologies, 2013.

[25] Mingzhe Chen, Zhaohui Yang, Walid Saad, Changchuan Yin, H Vincent Poor, and Shuguang Cui. A joint learning and communications framework for federated learning over wireless networks. *arXiv preprint arXiv:1909.07972*, 2019.

[26] Yuxia Cheng, Zhiwei Wu, Kui Liu, Qing Wu, and Yu Wang. Smart dag tasks scheduling between trusted and untrusted entities using the mcts method. *Sustainability*, 11(7):1826, 2019.

[27] Rodrigo Fernandes de Mello, Luciano Jose Senger, and Laurence Tianruo Yang. A routing load balancing policy for grid computing environments. In *Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on*, volume 1, pages 6–pp. IEEE, 2006.

[28] Canh T Dinh, Nguyen H Tran, Minh NH Nguyen, Choong Seon Hong, Wei Bao, Albert Y Zomaya, and Vincent Gramoli. Federated learning over wireless networks: Convergence analysis and resource allocation. *IEEE/ACM Transactions on Networking*, 2020.

[29] Abhishek Dixit, Ashish Mani, and Rohit Bansal. CoV2-Detect-Net: Design of COVID-19 prediction model based on hybrid DE-PSO with SVM using chest X-ray images. *Information Sciences*, 2021.

[30] Tingting Dong, Fei Xue, Chuangbai Xiao, and Juntao Li. Task scheduling based on deep reinforcement learning in a cloud manufacturing environment. *Concurrency and Computation: Practice and Experience*, page e5654, 2020.

[31] Nagat Drawel, Jamal Bentahar, Amine Laarej, and Gaith Rjoub. Formalizing group and propagated trust in multi-agent systems. In *IJCAI*, pages 60–66, 2020.

[32] Ke-Lin Du and MNS Swamy. Particle swarm optimization. In *Search and Optimization by Metaheuristics*, pages 153–173. Springer, 2016.

[33] Amir Gandomi and Murtaza Haider. Beyond the hype: Big data concepts, methods, and analytics. *International journal of information management*, 35(2):137–144, 2015.

[34] Abdullah Gani, Aisha Siddiqa, Shahaboddin Shamshirband, and Fariza Hanum. A survey on indexing techniques for big data: taxonomy and performance evaluation. *Knowledge and information systems*, 46(2):241–284, 2016.

[35] Seymour Geisser. *Predictive inference*. Routledge, 2017.

[36] Alan E Gelfand. Gibbs sampling. *Journal of the American statistical Association*, 95(452):1300–1304, 2000.

[37] Einollah Jafarnejad Ghomi, Amir Masoud Rahmani, and Nooruldeen Nasih Qader. Load-balancing algorithms in cloud computing: A survey. *Journal of Network and Computer Applications*, 88:50–71, 2017.

[38] B Gomathi, Karthikeyan Krishnasamy, and B Saravana Balaji. Epsilon-fuzzy dominance sort-based composite discrete artificial bee colony optimisation for multi-objective cloud task scheduling problem. *International Journal of Business Intelligence and Data Mining*, 13(1-3):247–266, 2018.

[39] David F Groebner, Patrick W Shannon, Phillip C Fry, and Kent D Smith. *Business statistics: A decision making approach*. Prentice Hall/Pearson, 2011.

[40] Daniel Grzonka, Agnieszka Jakobik, Joanna Kołodziej, and Sabri Pllana. Using a multi-agent system and artificial intelligence for monitoring and improving the cloud performance and security. *Future Generation Computer Systems*, 86:1106–1117, 2018.

[41] Manish Gupta and Govind Sharma. An efficient modified artificial bee colony algorithm for job scheduling problem. In *International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-1, Issue6*. Citeseer, 2012.

[42] Ahmad Hammoud, Azzam Mourad, Hadi Otrok, Omar Abdel Wahab, and Haidar Harmanani. Cloud federation formation using genetic and evolutionary game theoretical models. *Future Generation Computer Systems*, 104:92–104, 2020.

[43] Yaojun Han and Xuemei Luo. An effective algorithm and modeling for information resources scheduling in cloud computing. In *Advanced Cloud and Big Data (CBD), 2013 International Conference on*, pages 14–19. IEEE, 2013.

[44] Ibrahim Abaker Targio Hashem, Ibrar Yaqoob, Nor Badrul Anuar, Salimah Mokhtar, Abdullah Gani, and Samee Ullah Khan. The rise of "big data" on cloud computing: Review and open research issues. *Information systems*, 47:98–115, 2015.

[45] Hamdan Hejazi, Husam Rajab, Tibor Cinkler, and László Lengyel. Survey of platforms for massive iot. In *2018 IEEE International Conference on Future IoT Technologies (Future IoT)*, pages 1–8. IEEE, 2018.

[46] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D Joseph, Randy H Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, volume 11, pages 22–22, 2011.

[47] Chenghao Hu, Jingyan Jiang, and Zhi Wang. Decentralized federated learning: A segmented gossip approach. *arXiv preprint arXiv:1908.07782*, 2019.

[48] Shihong Hu and Guanghui Li. Dynamic request scheduling optimization in mobile edge computing for IoT applications. *IEEE Internet of Things Journal*, 7(2):1426–1437, 2019.

[49] Rob J Hyndman and Yanan Fan. Sample quantiles in statistical packages. *The American Statistician*, 50(4):361–365, 1996.

[50] Boris Iglewicz and David Caster Hoaglin. *How to detect and handle outliers*, volume 16. Asq Press, 1993.

[51] Anil K Jain. Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 31(8):651–666, 2010.

[52] Dervis Karaboga. An idea based on honey bee swarm for numerical optimization. Technical report, Technical report-tr06, Erciyes university, engineering faculty, computer engineering department, 2005.

[53] Dervis Karaboga and Bahriye Basturk. On the performance of artificial bee colony (abc) algorithm. *Applied soft computing*, 8(1):687–697, 2008.

[54] Wazir Zada Khan, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob, and Arif Ahmed. Edge computing: A survey. *Future Generation Computer Systems*, 97:219–235, 2019.

[55] P Venkata Krishna. Honey bee behavior inspired load balancing of tasks in cloud computing environments. *Applied Soft Computing*, 13(5):2292–2303, 2013.

[56] Dhinesh Babu LD and P Venkata Krishna. Honey bee behavior inspired load balancing of tasks in cloud computing environments. *Applied Soft Computing*, 13(5):2292–2303, 2013.

[57] Suk Kyu Lee, Mungyu Bae, and Hwangnam Kim. Future of iot networks: A survey. *Applied Sciences*, 7(10):1072, 2017.

[58] Lei Lei, Huijuan Xu, Xiong Xiong, Kan Zheng, and Wei Xiang. Joint computation offloading and multiuser scheduling using approximate dynamic programming in NB-IoT edge computing system. *IEEE Internet of Things Journal*, 6(3):5345–5362, 2019.

[59] Jiayin Li, Meikang Qiu, Zhong Ming, Gang Quan, Xiao Qin, and Zonghua Gu. Online optimization for scheduling preemptable tasks on iaas cloud systems. *Journal of Parallel and Distributed Computing*, 72(5):666–677, 2012.

[60] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 2018.

[61] Yibin Li, Keke Gai, Longfei Qiu, Meikang Qiu, and Hui Zhao. Intelligent cryptography approach for secure distributed big data storage in cloud computing. *Information Sciences*, 387:103–115, 2017.

[62] Tianjun Liao, Thomas Stützle, Marco A Montes de Oca, and Marco Dorigo. A unified ant colony optimization algorithm for continuous optimization. *European Journal of Operational Research*, 234(3):597–609, 2014.

[63] Chun-Yan Liu, Cheng-Ming Zou, and Pei Wu. A task scheduling algorithm based on genetic algorithm and ant colony optimization in cloud computing. In *Distributed Computing and Applications to Business, Engineering and Science (DCABES), 2014 13th International Symposium on*, pages 68–72. IEEE, 2014.

[64] Ning Liu, Zhe Li, Jielong Xu, Zhiyuan Xu, Sheng Lin, Qinru Qiu, Jian Tang, and Yanzhi Wang. A hierarchical framework of cloud resource allocation and

power management using deep reinforcement learning. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pages 372–382. IEEE, 2017.

[65] Yongkui Liu, Xun Xu, Lin Zhang, Long Wang, and Ray Y Zhong. Workload-based multi-task scheduling in cloud manufacturing. *Robotics and Computer-Integrated Manufacturing*, 45:3–20, 2017.

[66] Manuel Lopez-Martin, Belen Carro, and Antonio Sanchez-Esguevillas. Application of deep reinforcement learning to intrusion detection for supervised problems. *Expert Systems with Applications*, 141:112963, 2020.

[67] Liuyang Lu, Yanxiang Jiang, Mehdi Bennis, Zhiguo Ding, Fu-Chun Zheng, and Xiaohu You. Distributed edge caching via reinforcement learning in fog radio access networks. In *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, pages 1–6. IEEE 89th Vehicular Technology Conference (VTC2019-Spring), 2019.

[68] Xin Lu and Zilong Gu. A load-adapative cloud resource scheduling model based on ant colony algorithm. In *2011 IEEE International Conference on Cloud Computing and Intelligence Systems*, pages 296–300. IEEE, 2011.

[69] Fei Luo, Ye Yuan, Weichao Ding, and Haifeng Lu. An improved particle swarm optimization algorithm based on adaptive weight for task scheduling in cloud computing. In *Proceedings of the 2nd International Conference on Computer Science and Application Engineering*, page 142. ACM, 2018.

[70] Juan Luo, Luxiu Yin, Jinyu Hu, Chun Wang, Xuan Liu, Xin Fan, and Haibo Luo. Container-based fog computing architecture and energy-balancing

scheduling algorithm for energy IoT. *Future Generation Computer Systems*, 97:50–60, 2019.

[71] Siva Theja Maguluri, R Srikant, and Lei Ying. Stochastic models of load balancing and scheduling in cloud computing clusters. In *INFOCOM, 2012 Proceedings IEEE*, pages 702–710. IEEE, 2012.

[72] Najme Mansouri, Behnam Mohammad Hasani Zade, and Mohammad Masoud Javidi. Hybrid task scheduling strategy for cloud computing by modified particle swarm optimization and fuzzy theory. *Computers & Industrial Engineering*, 130:597–633, 2019.

[73] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282. PMLR, 2017.

[74] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[75] Yurii Nesterov et al. *Lectures on convex optimization*, volume 137. Springer, 2018.

[76] Huy T Nguyen, Nguyen Cong Luong, Jun Zhao, Chau Yuen, and Dusit Niyato. Resource allocation in mobility-aware federated learning networks: A deep reinforcement learning approach. *arXiv preprint arXiv:1910.09172*, 2019.

[77] Takayuki Nishio and Ryo Yonetani. Client selection for federated learning with heterogeneous resources in mobile edge. In *IEEE International Conference on Communications (ICC)*, pages 1–7, 2019.

[78] Liwei Ouyang, Yong Yuan, Yumeng Cao, and Fei-Yue Wang. A novel framework of collaborative early warning for COVID-19 based on blockchain and smart contracts. *Information Sciences*, 570:124–143, 2021.

[79] Chandrashekhar S Pawar and Rajnikant B Wagh. Priority based dynamic resource allocation in cloud computing. In *Cloud and Services Computing (ISCOS), 2012 International Symposium on*, pages 1–6. IEEE, 2012.

[80] Zhiping Peng, Jianpeng Lin, Delong Cui, Qirui Li, and Jieguang He. A multi-objective trade-off framework for cloud resource scheduling based on the deep Q-network algorithm. *Cluster Computing*, pages 1–15, 2020.

[81] Feng Qiu, Bin Zhang, and Jun Guo. A deep learning approach for vm workload prediction in the cloud. In *2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 319–324. IEEE, 2016.

[82] Fahimeh Ramezani, Jie Lu, and Farookh Hussain. Task scheduling optimization in cloud computing applying multi-objective particle swarm optimization. In *International Conference on Service-Oriented Computing*, pages 237–251. Springer, 2013.

[83] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. *Machine learning*, 85(3):333–359, 2011.

[84] Charles Reiss, John Wilkes, and Joseph L. Hellerstein. Google cluster-usage traces: format + schema. Technical report, Google Inc., Mountain View, CA,

USA, November 2011. Revised 2014-11-17 for version 2.1. Posted at `https://github.com/google/cluster-data`.

[85] John W Rittinghouse and James F Ransome. *Cloud computing: implementation, management, and security*. CRC press, 2016.

[86] Gaith Rjoub and Jamal Bentahar. Cloud task scheduling based on swarm intelligence and machine learning. In *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 272–279. IEEE, 2017.

[87] Gaith Rjoub, Jamal Bentahar, Omar Abdel Wahab, and Ahmed Saleh Bataineh. Deep and reinforcement learning for automated task scheduling in large-scale cloud computing systems. *Concurrency and Computation: Practice and Experience*, page e5919, 2020.

[88] Gaith Rjoub, Jamal Bentahar, and Omar Abdel Wahab. Bigtrustscheduling: Trust-aware big data task scheduling approach in cloud computing environments. *Future Generation Computer Systems*, 110:1079–1097, 2020.

[89] Gaith Rjoub, Jamal Bentahar, Omar Abdel Wahab, and Ahmed Bataineh. Deep smart scheduling: A deep learning approach for automated big data scheduling over the cloud. In *2019 7th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 189–196. IEEE, 2019.

[90] Gaith Rjoub, Omar Abdel Wahab, Jamal Bentahar, and Ahmed Bataineh. A trust and energy-aware double deep reinforcement learning scheduling strategy for federated learning on iot devices. In *International Conference on Service-Oriented Computing*, pages 319–333. Springer, 2020.

[91] Gaith Rjoub, Omar Abdel Wahab, Jamal Bentahar, and Ahmed Saleh Bataineh. Improving autonomous vehicles safety in snow weather using federated yolo cnn

learning. In *Mobile Web and Intelligent Information Systems*, pages 121–134. Springer International Publishing, 2021.

[92] Katty Rohoden, Rebeca Estrada, Hadi Otrok, and Zbigniew Dziong. Stable femtocells cluster formation and resource allocation based on cooperative game theory. *Computer Communications*, 134:30–41, 2019.

[93] Ola Salman, Imad Elhajj, Ali Chehab, and Ayman Kayssi. Iot survey: An sdn and fog computing perspective. *Computer Networks*, 143:221–246, 2018.

[94] M Schroeck, R Shockley, J Smart, D Romero-Morales, and P Tufano. Analytics: the real-world use of big data: How innovative enterprises extract value from uncertain data, executive report. *IBM Institute for Business Value and Said Business School at the University of Oxford*, 2012.

[95] S Selvarani and G Sudha Sadhasivam. Improved cost-based algorithm for task scheduling in cloud computing. In *Computational intelligence and computing research (iccic), 2010 ieee international conference on*, pages 1–5. IEEE, 2010.

[96] Mohit Sewak. Deep q network (dqn), double dqn, and dueling dqn. In *Deep Reinforcement Learning*, pages 95–108. Springer, 2019.

[97] Ohad Shamir, Nati Srebro, and Tong Zhang. Communication-efficient distributed optimization using an approximate newton-type method. In *International conference on machine learning*, pages 1000–1008. PMLR, 2014.

[98] Usman Shaukat, Ejaz Ahmed, Zahid Anwar, and Feng Xia. Cloudlet deployment in local wireless networks: Motivation, architectures, applications, and open challenges. *Journal of Network and Computer Applications*, 62:18–40, 2016.

[99] Sukhpal Singh and Inderveer Chana. Qos-aware autonomic resource management in cloud computing: a systematic review. *ACM Computing Surveys (CSUR)*, 48(3):42, 2016.

[100] A Sathya Sofia and P GaneshKumar. Multi-objective task scheduling to minimize energy consumption and makespan of cloud computing using nsga-ii. *Journal of Network and Systems Management*, 26(2):463–485, 2018.

[101] Binbin Song, Yao Yu, Yu Zhou, Ziqiang Wang, and Sidan Du. Host load prediction with long short-term memory in cloud computing. *The Journal of Supercomputing*, 74(12):6554–6568, 2018.

[102] Borja Sotomayor, Rubén S Montero, Ignacio M Llorente, and Ian Foster. Virtual infrastructure management in private and hybrid clouds. *IEEE Internet computing*, 13(5), 2009.

[103] J Angela Jennifa Sujana, M Geethanjali, R Venitta Raj, and T Revathi. Trust model based scheduling of stochastic workflows in cloud and fog computing. In *Cloud Computing for Geospatial Big Data Analytics*, pages 29–54. Springer, 2019.

[104] Yaohua Sun, Mugen Peng, and Shiwen Mao. Deep reinforcement learning-based mode selection and resource management for green fog radio access networks. *IEEE Internet of Things Journal*, 6(2):1960–1971, 2018.

[105] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[106] Mona Taghavi, Jamal Bentahar, and Hadi Otrok. Two-stage game theoretical framework for iaas market share dynamics. *Future Generation Computer Systems*, 102:173–189, 2020.

[107] Medhat A Tawfeek, Ashraf El-Sisi, Arabi E Keshk, and Fawzy A Torkey. Cloud task scheduling based on ant colony optimization. In *Computer Engineering & Systems (ICCES), 2013 8th International Conference on*, pages 64–69. IEEE, 2013.

[108] Preeti Thakur and Manish Mahajan. Different scheduling algorithm in cloud computing: A survey. *International Journal of modern computer science*, 5(1), 2017.

[109] Evangelos Triantaphyllou. Multi-criteria decision making methods. In *Multi-criteria decision making methods: A comparative study*, pages 5–21. Springer, 2000.

[110] Jinn-Tsong Tsai, Jia-Cen Fang, and Jyh-Horng Chou. Optimized task scheduling and resource allocation on cloud computing environment using improved differential evolution algorithm. *Computers & Operations Research*, 40(12):3045–3055, 2013.

[111] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, et al. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 5. ACM, 2013.

[112] Omar Abdel Wahab, Jamal Bentahar, Hadi Otrok, and Azzam Mourad. Optimal load distribution for the detection of vm-based ddos attacks in the cloud. *IEEE Transactions on Services Computing*, (1):1–1, 2017.

[113] Omar Abdel Wahab, Robin Cohen, Jamal Bentahar, Hadi Otrok, Azzam Mourad, and Gaith Rjoub. An endorsement-based trust bootstrapping approach for newcomer cloud services. *Information Sciences*, 2020.

[114] Omar Abdel Wahab, Nadjia Kara, Claes Edstrom, and Yves Lemieux. MAPLE: A machine learning approach for efficient placement and adjustment of virtual network functions. *Journal of Network and Computer Applications*, 142:37–50, 2019.

[115] Omar Abdel Wahab, Azzam Mourad, Hadi Otrok, and Tarik Taleb. Federated machine learning: Survey, multi-level classification, desirable criteria and future directions in communication and networking systems. *IEEE Communications Surveys & Tutorials*, 2021.

[116] Wei Wang, Guosun Zeng, Daizhong Tang, and Jing Yao. Cloud-dls: Dynamic trusted scheduling for cloud computing. *Expert Systems with Applications*, 39(3):2321–2329, 2012.

[117] Tom White. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[118] John Wilkes. More Google cluster data. Google research blog, November 2011. Posted at `http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html`.

[119] Chao-Tung Yang, Jung-Chun Liu, Ching-Hsien Hsu, and Wei-Li Chou. On improvement of cloud virtual machine availability with virtualization fault tolerance mechanism. *The Journal of Supercomputing*, 69(3):1103–1122, 2014.

[120] Chaowei Yang, Qunying Huang, Zhenlong Li, Kai Liu, and Fei Hu. Big data and cloud computing: innovation opportunities and challenges. *International Journal of Digital Earth*, 10(1):13–53, 2017.

[121] Hai Yang. Improved ant colony algorithm based on pso and its application on cloud computing resource scheduling. In *Advanced Materials Research*, volume 989, pages 2192–2195. Trans Tech Publ, 2014.

[122] Jiachen Yang, Bin Jiang, Zhihan Lv, and Kim-Kwang Raymond Choo. A task scheduling algorithm considering game theory designed for energy management in cloud computing. *Future Generation computer systems*, 2017.

[123] Jiachen Yang, Bin Jiang, Zhihan Lv, and Kim-Kwang Raymond Choo. A task scheduling algorithm considering game theory designed for energy management in cloud computing. *Future Generation Computer Systems*, 105:985–992, 2020.

[124] Yuli Yang and Xinguang Peng. Trust-based scheduling strategy for workflow applications in cloud environment. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2013 Eighth International Conference on*, pages 316–320. IEEE, 2013.

[125] Dingdong Yi, Xinran Li, and Jun S Liu. Bayesian aggregation of rank data with covariates and heterogeneous rankers. *arXiv preprint arXiv:1607.06051*, 2016.

[126] Lamia Youseff, Maria Butrico, and Dilma Da Silva. Toward a unified ontology of cloud computing. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. IEEE, 2008.

[127] Yanlong Zhai, Tianhong Bao, Liehuang Zhu, Meng Shen, Xiaojiang Du, and Mohsen Guizani. Toward reinforcement-learning-based service deployment of 5g mobile edge computing with request-aware scheduling. *IEEE Wireless Communications*, 27(1):84–91, 2020.

[128] Shaobin Zhan and Hongying Huo. Improved pso-based task scheduling algorithm in cloud computing. *Journal of Information & Computational Science*, 9(13):3821–3829, 2012.

[129] PeiYun Zhang and MengChu Zhou. Dynamic cloud task scheduling based on a two-stage strategy. *IEEE Transactions on Automation Science and Engineering*, 15(2):772–783, 2018.

[130] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.

[131] Gao Zhong-wen and Zhang Kai. The research on cloud computing resource scheduling method based on time-cost-trust model. In *Computer Science and Network Technology (ICCSNT), 2012 2nd International Conference on*, pages 939–942. IEEE, 2012.

[132] Zhou Zhou, Fangmin Li, Huaxi Zhu, Houliang Xie, Jemal H Abawajy, and Morshed U Chowdhury. An improved genetic algorithm using greedy strategy toward task scheduling optimization in cloud environments. *Neural Computing and Applications*, pages 1–11, 2019.

[133] Xingquan Zuo, Guoxiang Zhang, and Wei Tan. Self-adaptive learning pso-based deadline constrained task scheduling for hybrid iaas cloud. *IEEE Transactions on Automation Science and Engineering*, 11(2):564–573, 2014.