OPTIMIZATION OF TASK SCHEDULING IN HETEROGENEOUS CLOUD ENVIRONMENT USING GENETIC ALGORITHM

Supervisor: MARCO GRIBAUDO

M.Sc. Thesis by: LIDONG ZHANG, 873625

Academic Year 2017-2018
- Ideas without actions ain’t ideas -

- Le idee senza azioni non sono idee -

- 不付诸于行动的想法不是想法 -
Abstract

With the innovation of computer technology and the rapid development of the Internet, cloud computing came into being. In cloud computing, resources and tasks are dynamic and heterogeneous. In this thesis, after studying the task scheduling problem in cloud computing and analyzing the existing problems of resource allocation algorithm, we found that cloud computing usually needs to deal with a large number of computing tasks. Therefore, how to allocate computing resources reasonably and efficiently to schedule tasks, so that the tasks can be completed in a shorter time and at a lower cost, is an important issue.

During this studying, a task scheduling algorithm based on genetic algorithm is proposed and implemented. The results generated by this algorithm can make the task completion time shorter and at the same time also cost less. We also introduce time and cost factors into the fitness function to control the evolution direction of the population so that we can meet the different needs and preferences of users.

Finally, the simulation of the above proposed algorithm is executed by expanding the cloud computing simulation platform, CloudSim. Then we analyze its performance through experiments. Experimental results show that our algorithm is an effective task scheduling algorithm in cloud computing.

Keywords: Cloud Computing, GA, Task Scheduling, Resource Allocation, VMs, Optimization Problem, Performance, Heterogeneous, Time, Cost, Optimal Solution, CloudSim.
Sommario

Con l’innovazione della tecnologia informatica e il rapido sviluppo di Internet, è nato il cloud computing. Nel cloud computing, risorse e attività sono dinamiche ed eterogenee. In questa tesi, dopo aver studiato il problema della pianificazione delle attività nel cloud computing e dopo aver analizzato i problemi esistenti riguardanti gli algoritmi di allocazione delle risorse, abbiamo scoperto che il cloud computing, in genere, ha bisogno di gestire un gran numero di attività di elaborazione. Pertanto, come allocare le risorse di elaborazione in modo ragionevole ed efficiente per pianificare le attività, in modo che tutte le attività possano essere completate in un tempo più breve e ad un costo inferiore, è una questione importante.

Durante questo studio, viene proposto e implementato un algoritmo di pianificazione delle attività basato sugli algoritmi genetici. I risultati generati da questo algoritmo possono ridurre il tempo di completamento dell’attività e allo stesso tempo anche costare meno. Introduciamo anche fattori di tempo e costo nella funzione di fitness per controllare la direzione evolutiva della popolazione in modo da soddisfare le diverse esigenze e preferenze degli utenti.

Infine, la simulazione dell’algoritmo sopra proposto viene eseguita esandendo la piattaforma di simulazione del cloud computing, CloudSim. Poi analizziamo le sue prestazioni attraverso esperimenti. I risultati ottenuti mostrano che l’algoritmo di pianificazione delle attività risulta efficace nel cloud computing.

**Keywords:** Cloud Computing, GA, Pianificazione delle attività, Assegnazione delle risorse, VM, Problema di ottimizzazione, Prestazioni, Eterogeneo, Tempo, Costo, Soluzione ottimale, CloudSim.
Acknowledgements

Time is like running water. In a twinkling of an eye, two years have passed. Looking back on these two years, I have gained a lot in my study and life. Here, I am very grateful to those people who helped and cared about me.

I particularly want to thank my mentor, Professor Gribaudo. He is a very responsible professor and also a kind superior. His help is indispensable to the success of my thesis. During the researching of my thesis, he has gave me a lot of valuable suggestions which helped me to complete my thesis efficiently.

Secondly, I want to thank my family for their support and concern. They have always got my back and I feel lucky to be a family with them.

Moreover, I also want to thank my friends who have helped me to translate the abstract of my thesis into italian.

Finally, I have to thank Politecnico di Milano for giving me this platform to learn new knowledge and show myself.
## Contents

Abstract I

Sommario III

Acknowledgements V

1 Introduction 1
   1.1 Research Background and Significance 2
   1.2 Scenario and Problem Statement 3
   1.3 Methodology 5
   1.4 Contributions 6
   1.5 Structure of Thesis 7

2 Preliminaries and State of the Art 9
   2.1 Introduction to Cloud Computing 9
      2.1.1 Cloud Computing Concept 9
      2.1.2 Cloud Computing Features 10
      2.1.3 Cloud Computing Architecture 12
      2.1.4 Cloud Computing Platforms 13
   2.2 Could Computing Task Scheduling 16
      2.2.1 Task Scheduling Concept 16
      2.2.2 Task Scheduling Problems 17
      2.2.3 Task Scheduling Model 18
      2.2.4 Common Task Scheduling Algorithms 18
      2.2.5 Performance Evaluation of Task Scheduling Algorithms 20
   2.3 Optimization Problems 23
      2.3.1 Function or Trial-and-Error 24
      2.3.2 Single or Multi-Objective 24
2.3.3 Static or Dynamic ........................................ 25
2.3.4 Continuous or Discrete ................................. 26
2.3.5 Constrained or Unconstrained ......................... 26
2.3.6 Linear or Non-linear ..................................... 27
2.4 Genetic Algorithms ........................................ 27
   2.4.1 Rationale .............................................. 27
   2.4.2 Applications .......................................... 28
   2.4.3 General Research Based on GA ....................... 28
2.5 Cloud Computing Simulation ........................... 29
2.6 Cloud Computing Simulators Evaluation .............. 30
2.7 Review of Related Work .................................. 32
2.8 Summary .................................................... 34

3 Design and Implementation: Goals and Task Scheduling Model Design ........................................ 35
   3.1 Concepts ................................................... 35
   3.2 Goals and Tasks Scheduling Model Design .......... 37

4 Design and Implementation : Approach .................. 41
   4.1 Task Scheduling Strategy Based on GA Algorithms ...... 42
      4.1.1 Encoding and Decoding ............................ 42
      4.1.2 Initialization of the Population .................. 43
      4.1.3 Fitness Function Design ........................... 44
      4.1.4 Selection Model .................................. 46
      4.1.5 Crossover Operation ................................ 46
      4.1.6 Mutation Operation ................................ 46
   4.2 Algorithm Architecture and Implementation ........ 47
   4.3 Simulation Tool : CloudSim Toolkit ................... 47
      4.3.1 CloudSim Architecture ............................ 51
      4.3.2 CloudSim Scheduling Policies ................... 52

5 Experimental Simulation and Performance Evaluation 57
   5.1 Experimental Design .................................... 57
      5.1.1 Experimental Parameters .......................... 58
      5.1.2 Simulation Flow ................................... 61
   5.2 Experimental Result and Performance Analysis ....... 61

6 Exploitation of GA in Task Scheduling .................. 67

VIII
7 Conclusion and Future Work

References
List of Tables

5.1 Basic HD and SW Configuration ........................................ 58
5.2 Simulation Parameters .................................................. 59
5.3 Algorithm Parameters ................................................. 59

6.1 Heterogeneous VMs Configuration .................................... 67
6.2 Time-optimized Performance Comparison .......................... 68
6.3 Cost-optimized Performance Comparison ......................... 68
6.4 Performance Comparison in Heterogeneous Cloud Environment 71
List of Figures

1.1 Steps of Task Scheduling in Cloud Computing ............ 4
2.1 The Architecture of Cloud Computing ................. 13
2.2 Framework of Task Scheduling in Cloud Computing ....... 16
2.3 A Set of Points and Non-dominated Front .............. 25
2.4 Cloud Computing Simulation Frameworks .............. 31
2.5 Layered cloud computing architecture .................. 32
3.1 Cloud Task Scheduling Principles ....................... 36
4.1 Resources-Tasks Encoding ............................. 43
4.2 Genetic Algorithm Architecture ....................... 48
4.3 CloudSim Architecture ................................ 52
4.4 Space Shared Policy for VMs and Tasks ............... 53
4.5 Space Shared Policy for VMs and Time Shared for Tasks .. 54
4.6 Time Shared Policy for VMs and Space Shared for Tasks .. 54
4.7 Time Shared Policy for VMs and Tasks .................. 55
5.1 The Simulation process sequence diagram of CloudSim .... 61
5.2 Number of Calls to Fitness Function ................... 62
5.3 Fitness Value Discrepancy ............................. 63
5.4 Run Time Comparison ................................ 64
5.5 Optimum the Iterations ............................... 64
5.6 Comparison of Optimum Iterations ..................... 66
6.1 Time Performance Comparison A ....................... 72
6.2 Time Performance Comparison B ....................... 73
6.3 Time Performance Comparison C ....................... 73
6.4 Comprehensive Time Performance Comparison .......... 74
6.5 Cost Performance Comparison A ....................... 74
6.6 Cost Performance Comparison B . . . . . . . . . . . . . . . . 75
6.7 Cost Performance Comparison C . . . . . . . . . . . . . . . . 75
## List of pseudo codes

1. Initial population ........................................ 49
2. Calculation Fitness ........................................ 49
3. Selection Operation ....................................... 50
4. Crossover Operation ....................................... 50
5. Mutation Operation ....................................... 50
6. Simulation Steps .......................................... 60
Chapter 1

Introduction

From the birth of the first computer to present, computer model has gone through three significant phases, namely the single host mode, server/client mode and cloud computing mode. In the meanwhile, people’s requirements for computational power, storage facilities, transmission and interaction capabilities of computers are also getting higher and higher. And the traditional computing model can not fully meet the needs of users at this stage. Cloud computing is as a trendy computing mode developed from grid computing, distributed computing and parallel computing. In its environment, the server is virtualized into a huge resource pool by applying virtualization technology and the Virtual Machines (VMs) are dynamically invoked to users. With the increasing popularity and commercialization of cloud computing, it has also gained a very wide range of user groups, on the other hand, this also means that cloud computing resources are almost always dealing with a large number of tasks. Therefore, it is very important assign user tasks to different resources, so that the performance of the entire system can achieve the best effect with also considering the cost saving. Furthermore, cloud computing users should choose what kind of cloud environment configuration to maximize their own interests when using cloud computing products. These have been hot issues that cloud computing providers and cloud computing users are very concerned about. Hence, in this thesis we are going to propose and implement an algorithm that can be applied into cloud computing for optimizing task scheduling problems. The proposed algorithm should also reduce the total makespan execution time and balances the load over the cloud resources with minimum total monetary cost.
1.1 Research Background and Significance

With the development of virtualization and Internet technologies[1], cloud computing has been developed rapidly, meanwhile, the needs for computing and huge storage resources are fast growing as well. Currently, large IT companies such as Google, Microsoft, Amazon, IBM, Baidu and Alibaba have already lunched their own cloud computing platforms as one of the most important development strategies in the future, while Facebook, Youtube, Myspace, etc, also start using cloud computing services.

Cloud computing as a trendy computing service model, it has changed the initial mode which centralizes tasks delivering them to large processors and realized the computing mode of on-demand allocation. Its core idea is to connect a large number of computing resources, storage resources and service resources through a network to form a resource pool and then uniformly schedule and manage resources according to user needs. The user does not have to care which machine the task is running on, but only pays on demand. Therefore, the goal of cloud computing is to dynamically provide data-centric computing service systems based on user needs and to maximize the reliability, customization, and security of the services provided.

However, with the continuous growth of Internet users, the commercialization of cloud computing is becoming more and more prominent and the demand for services is becoming more and more diverse. This requires cloud service providers to pay more attention to user needs, such as the reliability of tasks, the execution makespan and monetary cost of tasks. Though, cloud computing has already become a booming area and has been emerging as an commercial reality in the information technology domain. However, the technology is still not fully developed. There are still some areas that are needed to be focused on.

- Resource allocation
- Task scheduling

Task scheduling and provision of resources are main problem areas in both Grid as well as in cloud computing. Cloud computing is an emerging technology in IT domain. The scheduling of the cloud services to the consumers by service providers influences the cost benefit of these computing paradigms. The key of the whole cloud computing resource allocation is to
choose a reasonable resource allocation algorithm. A good resource allocation algorithm can not only improve the computing transmission speed, but also reduce the load and energy consumption of the cloud computing network, so as to balance the network load and improve the performance of the whole cloud computing system.

Therefore, the main objective of this paper is to propose an algorithm that addresses the task scheduling problem then it will be evaluated in comparison with exhaustive algorithms to prove its effectiveness in solving the task scheduling problem in the cloud environment.

### 1.2 Scenario and Problem Statement

Some people regard cloud computing technology as the third wave of technology innovation after personal computers and the Internet. They believe that it will soon or has fundamentally changed the pattern of the entire information industry and the habits and ways of human use of computers with the rapid development of cloud computing. However, exactly due to its fast growing, its own shortcomings are becoming more and more noticeable along with people’s increasingly requirements. For example, now cloud computing is facing many obstacles and challenges such as security, performance and resource management [2]. Task scheduling is one of the main problems related to resource management, which has a curious impact on efficiency, throughput and resource utilization in cloud environment. Task scheduling in cloud computing involves allocating user tasks to available resources, thereby increasing system utilization and throughput without violating Service-Level Agreement (SLA) requirements [3]. In fact, task scheduling is a matter of mapping user’s tasks flow to available resources in cloud computing environment. This is an optimization problem, because the scheduler tries to find the best task-VMs mapping (optimal matching) related to scheduling time such as response time, generation time and completion time.

Assume that on a cloud platform, there are $m$ tasks $T = T_1, T_2, ..., T_m$ and $n$ available resources $R = R_1, R_2, ..., R_n$ and $m$ tasks need to be scheduled to execute on $n$ executable resources in a reasonable and efficient way. Here, cloud resources refer to the virtual resources [4].

Tasks scheduling steps are modeled as illustrated in Figure 1.1.

1. A user of cloud computing submits a task to a scheduler.
2. A scheduler communicates with Cloud Information System (CIS) for getting information about resources.

3. CIS provides the resources information to the scheduler.

4. The scheduling algorithm does its role for mapping task to the suitable resource and submits the task to the winner resource (decision process for allocating a resource).

5. The user gets the identification (id) of the resource and uses it through cloud interface.

6. The user sends the input data to the resource according to the schedule.

7. The scheduler gets over time updated information about the status of a cloud to manage the schedule.

8. The information is sent to the user.

Above is the process of the task scheduling in cloud computing. Frankly speaking, task scheduling main goal is to fairly allocate virtual machines between tasks so as to obtain the shortest total execution time and to improve the utilization of resources and the overall performance of the whole system.

However, execution time is considered complex, especially because there are many factors to be considered, such as task completion time, cost, response time, power consumption, etc. [5]. For example, the order of task execution must be considered when assigning the tasks to VM processors in
a multiprocessor environment since task schedulers are interdependent, in which the output of some tasks is the input of another task. For instance assigning the dependent tasks to the most appropriate VM processors is known to be an non-deterministic polynomial complete (NP-complete) problem as discussed by Verma and Kaushal [6]. The scheduling processes of the workflow applications are a multi-objective optimization problem (also known as Pareto optimization), where users might wish to minimize the money cost and the execution time for the whole workflow application with efficient load balancing over the VMs in the cloud environment. The optimal decision for the multi-objective workflow optimization is the trade-off between the three objectives; therefore, the objectives must be rated based on their importance to the users to select the best Pareto solutions because, for instance, minimizing the overall cost may lead to maximizing the execution time and the load over a specific VM [7, 8]. The workflow scheduling problem is an inherited problem from the heterogeneous computing environments, for which different research efforts were made to address the scheduling problem [9–11]. However, heterogeneous computing environments are not easy to set up, and their capability of giving more uniform performance with less failure is quite limited in comparison to the cloud environments [12, 13]. Moreover, the main objective of the various previous efforts in addressing the workflow scheduling problem in heterogeneous environments is to only minimize the finish time.

So far, no algorithm with polynomial time complexity can solve these optimization problems. Among them, genetic algorithm (GA) [6-8] is a widely used meta-heuristic algorithm for sampling large and highly complex search space which is difficult to handle. GA is often used to solve many allocation, planning and scheduling optimization problems [9-14]. In short, the optimal task scheduling in cloud computing is a challenging problem. There is a need to improve the performance and quality of services and fairly reduce execution costs and time consumption. And it also should take into account the interests of both users and service providers.

1.3 Methodology

We propose and implement GA Algorithm coding by Java then to apply it to task scheduling in cloud computing. Using the simulation tool cloudsim (discussed in chapter 2) to simulate the task scheduling process in cloud computing. We create different number of cloud tasks on cloudsim for simulation
experiments. At the same time, we use exhaustive algorithm to obtain the optimal solution of cloud task scheduling according to the objective function and condition constraints that we created. The result serves as the final reference criterion to judge whether GA algorithm achieves the desired goal and whether it has a greater performance improvement than exhaustive algorithm. Then Excel is used to make data charts to analyze data.

Finally, we create as many cloud task scenarios as possible according to some pre-set cloud task characteristics and apply GA algorithm to cloud task scheduling problem. For example, we fix cloud task configuration to see whether there will be significant and effective performance improvement by changing different available resource allocations. After that we hope to acquire some constructive conclusions for cloud users when they need to purchase or configure cloud resources to perform cloud task scheduling. In the end, we use cloudsim to do simulation experiments to get the total completion time and estimated cost of cloud tasks. And for proposed algorithm, each experiment will be executed 20 times then the average result of GA will be compared with exhaustive algorithm optimum solution correspond to each experiment. Last but not the least a detailed analysis and discussion will be provided.

1.4 Contributions

The effort of this thesis is mainly twofold. One is analyzing and studying existing theory of Genetic algorithm in order to design the modules of our proposed algorithm, implement it and then assess the performance of GA through theoretical analysis. The other is to apply GA to simulate the task scheduling process in designed cloud task scenarios in order to find the optimum resource allocation schema with the low monetary cost and short makespan execution time.

For the part of the first contribution, a specific type of assignment problem, where tasks are assigned to cloud resources, is considered. Mathematical formulation of the optimization problem is presented and Exhaustive algorithm is implemented and applied for finding the optimal solution. An objective function is formulated using a formal method for measuring the performance of generated solution of suggested GA. Simulation experiments are performed on task scheduling problems and the results are discussed with respect to accuracy, scalability, efficiency, etc.
Another part of contribution one of the thesis is to design and implement a similarity function for initializing population so as to make sure the initial population is distributed over the widest possible solution domain. The goal is enhancing the optimization ability of genetic algorithm and accelerating the convergence speed to the optimal solution.

Our second contribution regards creating as many as possible different cloud tasks scenarios and cloud computing environment configurations scenarios to simulate the task scheduling based on GA. Analyzing the result from all aspects. At last, solving a scenarized task scheduling case in cloud computing based on reality by applying the proposed algorithm.

1.5 Structure of Thesis

The thesis is organized as follows:

Chapter 2 Addresses introduction to cloud computing, concept and issues of task scheduling in cloud computing, the optimization concepts, categories of optimization models, common algorithm used for task scheduling and Genetic Algorithms (GAs), simulation toolkit and evaluation, related works.

Chapter 3 Formalizes goals and task scheduling model design.

Chapter 4 Proposes the application of GA algorithm to cloud task scheduling and demonstrates some implementation details of the algorithm and discusses the selection of simulation toolkit.

Chapter 5 Designs the simulation steps and analyzes the performance of GA.

Chapter 6 Applies GA into different cloud task scenarios.

Chapter 7 Concludes the thesis, summarizing the finished studies and discussing possible future studies.
Chapter 2

Preliminaries and State of the Art

2.1 Introduction to Cloud Computing

2.1.1 Cloud Computing Concept

Cloud computing is a trendy Internet-based computing method. With the support of this computing method, the Internet can realize the on-demand allocation of various resources, including software and hardware resources that can be shared. This concept has made cloud computing become another revolution of internet industry after the mainframe to C/S model in the late 20th century. Since then, the data coming from the Internet is transparent to users, as described by the cloud concept. Users do not need to pay attention to the details of the infrastructure behind it, nor do they need professional technologies to control the back-end. Cloud computing has built the Internet into a self-service platform that increases IT services to meet user needs and it also develops unified usage delivery models to provide users with dynamic and easily scalable virtualization resources [15]. Cloud computing itself is not created from nothing. Its technical framework is based on the traditional computer technologies such as distributed computing, parallel computing and virtualization, which have been developed for a long time in the past. Therefore, cloud computing is a product combination of tradition and modernity.
2.1.2 Cloud Computing Features

Cloud computing is the development of Parallel Computing, Distributed Computing, and Grid Computing. From the service level, it mainly provides three kinds of services [16], namely Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Cloud computing can dynamically allocate resources according to users’ needs and realize rational utilization of resources. The main features of cloud computing are as follows:

- **Flexible Resources**
  Cloud computing relies on the core concept of parallel computing [14], which dispatches computing requests and resource requests of all parties freely according to their needs. It also provides complete data processing services, such as computing model settings, the output of calculation results, multi-form presentation, application API, etc. Ultimately, it can be used in the development of scientific research, public utilities, government and other platforms.

- **High Reliability and Safety**
  Cloud computing uses distributed storage system [16], which has the characteristics of high fault tolerance, interchangeability and fast data backup and recovery, ensuring the reliability and security of data. It can satisfy the computing needs of different fields and characteristics, support various data processing and computing models. Moreover, it also has massive storage and infinite space, in another word, cloud computing provides the most reliable data storage center. Cloud computing shows good performance of security. It isolates access to data in groups and customizes the firewall strategy with the natural capabilities of anti-ARP spoofing and anti-DDOS attack.

- **Low Cost and High Cost Performance**
  Cloud computing is a ”cloud” consisting of connecting PCs from different environments through the server. Cloud computing uses computing and storage resources in a service-oriented manner. It enables on-demand access and pay-as-you-go to resources, which eliminates the needs of purchasing large amounts of equipment. Cloud computing connects low-cost PCs, but its computing power can exceed super com-
puters, in such way cloud computing reduces the cost of computing resources to become cost-effective.

- **Virtualization**

  In the Internet, physical resources have certain differences. How to integrate these heterogeneous and discrete physical resources has always been a difficult problem. To solve this problem, virtualization technology emerges as the times require. Virtualization technology is the foundation of cloud computing. It realizes the interconnection of heterogeneous resources and the separation between upper and lower layers. It improves the utilization of resources and flexibly allocates tasks to different physical resources according to the needs of users. In the traditional IT architecture the computing powers of different regions are different. Different application scenarios have different requirements for computing resources in different periods, but they use the richest ideas to allocate computing resources. This inevitably leads to a lack of flexibility in the scheduling of computing power, causing resource bottlenecks or even system paralysis, which in turn leads to an incalculable waste. After using virtualization technology, the platform does not need to add any computing resources and can still significantly improve the utilization of computer resources. Therefore, there is a strong demand for virtualization integration. As the utilization of existing IT infrastructure has been improved, computing resources such as a large number of PCs can be fully released.

- **Unified management**

  With the integration of cloud computing, the traditional IT management model has changed dramatically. Cloud computing reduces the additional cost of manual management and puts all physical resources into the cloud. With unified engine scheduling and consistent platform entrance, it loosens and decouples the work of IT managers and brings great convenience to IT managers.

- **Low-priced Fault Tolerance**

  In the traditional IT architecture, the strategies of dual-machine HA and backup are usually used to ensure the high reliability of the business system. However, these methods have some shortcomings, such as
easily causing a large amount of resources to be idle and fault tolerance is not very good. In the era of cloud computing, with the packaging of cloud computing, the granularity of resource allocation becomes smaller and fault tolerance can be placed at the virtual machine level, which greatly increases resource utilization with reduced costs, at the same time, security level also gets promoted.

2.1.3 Cloud Computing Architecture

Cloud computing is currently the hottest Internet frontier technology, and it has caused many well-known enterprises around the world to rush into research and development. Most of the theories indicate that the cloud computing architecture [15] can be divided into four layers, namely resource layer and virtualization layer, management and service layers. See Figure 2.1 for details.

- Resource Layer
  Resource layer is a level that integrates physical resources, including servers, storage devices, network devices and so on.

- Virtualization Layer
  Virtualization layer is to use virtualization technology to integrate and abstract heterogeneous hardware resources and application resources of resource layer, and provide virtual resources to management for use. As one of the core of the cloud computing platform, it mainly provides the abstraction of physical devices and the virtualization ability for applications in the future. Virtualization layer provides a variety of virtualization technologies and integrates multiple client operating systems.

- Management Level
  Management is mainly responsible for the management of virtualized resources, physical resources and personalized network services as required. For example, the automatic allocation of computing resources, on-demand allocation and backup of the system need the participation of management. In general, management is only open to its upper layer (service layer), which not only avoids the operator’s requirements
for different resource management, but also realizes the integration of resource management capabilities of service layer.

- **Service Level**

  Service layer provides system management capability service and self-service portal. The system management of service layer is mainly responsible for resource allocation, user application audit, service opening, system security and so on. Self-service portal is mainly open to users, users can order personalized products through self-service, submit service applications, adjust resource allocation and so on.

### 2.1.4 Cloud Computing Platforms

As an IT technology, cloud computing has developed rapidly. Currently, all major IT companies in the world have launched their own cloud computing service platforms. Here are a few examples of famous cloud computing platforms.
• IBM Blue Cloud

The Blue Cloud computing platform [16] is a cloud computing platform launched by IBM in 2007 to provide customers with an out-of-the-box approach. It mainly consists of a global network resource structure, so that data computing operations are not limited to local machines or regional service clusters, but in a larger data environment similar to the Internet. Specifically, ”Blue Cloud” has the following characteristics: First, its development and application is based on IBM’s professional technology in the field of large-scale computing, and includes IBM’s other related software systems (such as Linux operating system image, Hadoop file system, etc.) which are built in parallel with it. Secondly, “Blue Cloud” can provide customers with a safer and more stable seamless experience, because “Blue Cloud” internal software can allocate resources in real time across multiple servers to ensure the best performance. Finally, ”Blue Cloud” plan has a great effect on enterprise development. Blue Cloud integrates DB2, WebSphere, Hadoop and other software systems with hardware products, which brings great convenience for users to build cloud computing environment.

• Google’s Cloud Computing Platform

Google is the largest researcher and practitioner of cloud computing [17]. With the rapid development of Internet technology, the scale of network data is becoming larger and larger. Node failures often occur when processing cluster data. For this reason, Google proposes a distributed parallel cluster infrastructure to solve this problem. Google’s cloud computing architecture generally consists of three parts, namely, the Google File System, Map/Reduce programming mode, and BigTable, a large-scale distributed database.

1. GFS

In order to meet the growing data processing needs of Google, Google has designed and implemented an extensible distributed file system based on Linux, which can be referred to as GFS (Google File System), mainly used in distributed applications and applications requiring reading and writing of large amounts of data. The operating carrier of the system is the common hardware with low value, but it has fault-tolerant function, so it can provide high overall performance services for a large
number of users. The merit of GFS is that it builds distributed file system based on low-cost commercial machine, combines closely with the characteristics of Google application, simplifies its implementation process, and provides an innovative, useful and feasible solution.

2. Map/Reduce

Designed and implemented by Google, Map/Reduce is a programming model for parallel computing on large data sets (greater than 1TB). The emergence of Map/Reduce brings great convenience to programmers, making distributed systems easier to learn and research. The implementation of Map/Reduce is divided into Map phase and Reduce phase. The Map phase is responsible for segmenting the input files, and then grouping them into Reduce for processing to achieve efficient distributed computing efficiency.

3. BigTable

BigTable is a key/value distributed database that stores structured data. BigTable is designed by Google to handle PB-level data reliably and has excellent deployability. At present, BigTable has been applied to many products and projects, such as Google Analytics, Orkut, Personalized Search, Google Earth and so on. From these applications, it can be found that BigTable has strong adaptability and scalability. It can meet the different needs of products and provide a flexible, high-performance solution.

- Amazon’s Elastic Computing Cloud

Amazon is the first company to provide cloud computing services. Its cloud computing platform is called Elastic Computing Cloud [18] (Elastic Compute Cloud, EC2). Flexible Computing Cloud provides developers with cloud computing platform to support developers to develop their own cloud computing applications, which is completely different from Google. Unlike Blue Cloud, Elastic Computing Cloud does not have a global resource architecture similar to Blue Cloud, nor does it sell physical cloud computing service platforms. Simply put, the main feature of EC2 is that it is based on Amazon’s large-scale cluster computing platform. With the help of Web services, users can selectively run their desired software or applications on their Amazon machine image files, making cloud computing easier and more adjustable.
2.2 Could Computing Task Scheduling

2.2.1 Task Scheduling Concept

Task scheduling and resource allocation in cloud computing environment is a NP-complete problem. It aims at improving the overall performance of the system and reducing the total execution time and resource consumption. According to the resource allocation strategy, tasks are sent to the corresponding resource nodes for orderly execution in order to get a better allocation scheme [19]. In cloud computing environment, the essence of resource allocation is to allocate n independent tasks to m heterogeneous and effective available resources ($m < n$). The ultimate goal is to achieve the process of cloud resource allocation with minimum task processing cost and to make full use of resources. The allocation mentioned here actually includes two processes: task scheduling and resource allocation. Task scheduling refers to assigning parallel tasks to specific resources, and resource allocation refers to determining the execution order of tasks according to the pre-defined resource allocation strategy. In order to express these two processes conveniently, this thesis unifies "task scheduling" to describe them. The infrastructure of cloud computing task scheduling includes resource layer, service layer and task layer. These three modules work together from bottom to top to achieve the management and allocation of cloud computing resources. The cloud computing task scheduling framework is shown in Figure 2.2.

![Figure 2.2: Framework of Task Scheduling in Cloud Computing](image-url)
2.2.2 Task Scheduling Problems

With the deepening of research, in order to solve the problem of task scheduling, there are more and more allocation models and algorithms emerging. However, cloud computing resources are heterogeneous, virtualized, large-scale and so on. Direct using of these algorithms is not enough to meet the needs of services. Improving the stability of the algorithm, guaranteeing secure task scheduling and enhancing the trust of users are still the focused researching areas. When solving the problem of cloud computing task scheduling, there are the following difficulties:

- The service-oriented features of cloud computing are usually ignored by researchers [20]. Cloud computing provides on-demand allocation to meet different needs of users, for example, some users emphasize the confidentiality of cloud computing data, some users pay attention to the speed of data transmission, while others want to reduce service costs, which requires cloud computing to provide different services to users on demand. In fact, the existing allocation strategies of cloud computing are relatively simple, do not fully consider the characteristics of service-oriented, and do not achieve a better balance in the overall performance of services, user service level and cost.

- There is no ideal test environment and prediction model. For any algorithm, to evaluate its performance, it must be applied to the actual environment for repeated experiments, and the success of the algorithm is directly related to the quality of the prediction model. Therefore, how to build a real test environment and an accurate model is one of major obstacle to verify the allocation algorithm.

- Lack of uniform service quality [18] measurement standards. For example, the transmission speed is the measurement standard of CPU resources, while the bandwidth and delay are the measurement indexes of network transmission quality of service. Usually, the value of parameters in the algorithm is determined by these quality indicators. The uncertainty of parameters brings some difficulties to the realization of the algorithm.

- Further research on mainstream algorithms is needed. Currently, the mainstream algorithms for resource allocation are still widely used, but
each algorithm has its own shortcomings. For example, genetic algorithm lacks the ability of local search, which makes it easy to fall into local optimal solution prematurely. Ant colony algorithm has no initial pheromone, which results in low search efficiency. How to improve the existing problems in these mainstream algorithms is also a major direction to seek the optimal resource allocation strategy.

2.2.3 Task Scheduling Model

A key technology of the cloud computing system is the server virtualization technology, which realizes the separation of the service layer and the physical resource layer of the cloud computing. It also can flexibly distribute workload to different physical resources to achieve efficient utilization of resources. Server virtualization allows multiple virtual machines (VMs) to execute concurrently on a physical machine (PM), and each VM has a complete software system and a specified physical resource configuration (CPU processing capacity, memory RAM size, network bandwidth, etc.). Virtualization technology allows virtual machine resources to be dynamically allocated and migrated as needed. The task scheduling problem describes how many servers are needed in general and how virtual machines (VMs) are allocated to servers at various times. The optimal allocation of resources is to minimize the number of physical machines participating in virtual machine allocation.

2.2.4 Common Task Scheduling Algorithms

In cloud computing, intelligent optimization algorithms are used to solve NP-hard problems such as task scheduling. Generally used algorithms include ant colony algorithm, simulated annealing algorithm and artificial neural network. The following will briefly introduce the principles and advantages and disadvantages of these three algorithms.

- **Ant Colony Algorithm**

  Ant colony algorithm, commonly known as ant algorithm, is a simulated evolutionary algorithm[21]. It mainly finds the optimal solution by finding the optimal path. In 1992, Marco Dorigo proposed the algorithm for the first time in his doctoral dissertation, based on the
behavior of ants in choosing the shortest path during their search for food. Ant colony algorithm is a probabilistic algorithm. Its probabilities are mainly manifested in that the current ants depend on the choice of the antecedent ants for the path selection between fixed points, that is, the more times they are selected before a certain path, the easier they will be selected. A series of studies have shown that the ant colony algorithm has many advantages, such as good stability, positive feedback, parallelism, intelligent search and easy integration with other algorithms. In some classical optimization problems, such as the famous TSP (Travelling Salesman Problem) and QAP (Quadratic Assignment Problem), are usually solved by the algorithm. With the advancement of research, ant colony algorithm has been widely used in task scheduling, traffic planning and combinatorial optimization. At the same time, the algorithm also exposes the shortcomings of long accumulation time of pheromones and easy to fall into local optimal solution. Its optimization problem is still worth studying.

Simulated Annealing Algorithm

Simulated annealing algorithm is a stochastic optimization algorithm based on Monte Carlo iteration strategy. In the 1950s, N. Metropolis et al. proposed the algorithm. Until the 1980s, S. Kirkpatrick successfully applied the algorithm to the field of combinatorial optimization by utilizing the similarity between combinatorial optimization and physical annealing process[16]. The basic idea is: starting from a higher initial temperature, the algorithm uses the probability mutation characteristics to find the global optimal solution randomly from the solution space in the process of decreasing temperature, and achieves the effect of jumping out from the local optimal probability until gradually reaching the global optimal.

Simulated annealing algorithm also has its unique characteristics, its use is not limited by the areas involved in optimization problems, it has good adaptability that can accept sub-optimal solutions probabilistically. Its design process is simple and can handle complex nonlinear combinatorial optimization problems in parallel. Algorithms with universality and global optimization performance are widely used in production scheduling, wireless sensor networks, control engineering and other fields. For the simulated annealing algorithm, because it does
not have the ability to control the most possible search direction, the convergence speed is slow, and the search ability in the whole solution space is weak, and the performance of the algorithm is vulnerable to the influence of initial parameters and initial solutions.

Artificial Neural Network

Artificial neural network (ANN), as its name implies, it is a mathematical model that simulates the processing of information by brain neural network. It is often referred to as neural network. In the 1940s, W.S. McCulloch and W.Pitts first proposed the mathematical model of neurons and the method of network structure; in the early 1980s, with the vigorous development of optimization algorithms, American physicist J.J. Hopfield established a fully interconnected neural network model, which represents the introduction of neural networks into the field of optimization algorithms[17]. J. J. Hopfield, an American physicist, introduced neural networks into the field of optimal computing. A large number of processing neurons are connected to each other to form a neural network, which is essentially an operational model. Each neuron can generate its specific excitation function; there is a weighted value between two connected neurons, that is, the weighted value through the connected signal. The output of the neural network depends on the connection mode of the network, the excitation function and the weights of the neurons.

Artificial neural network has the ability to process large-scale data in parallel and has good learning ability. In addition, the artificial neural network has good fault-tolerance and robustness, which enables it to integrate well with other fields of technology. However, the neural network itself is highly complex and unsuitable for solving high-precision problems.

2.2.5 Performance Evaluation of Task Scheduling Algorithms

In recent years, with the rapid development of cloud computing, the types and usage of cloud computing services are also increasing rapidly. In cloud computing system, there are large-scale service requests waiting to be processed every day. In order to ensure the efficient operation of the system, it
is very important to choose the appropriate task scheduling method. In the process of resource allocation, how to improve the overall performance of the network and the utilization of resources must be considered by the allocation algorithm. At the same time, the allocation algorithm should meet the needs of users as much as possible.

**Cloud Computing Scheduling Target**

In a cloud computing system, each task has its own needs for cloud computing services. On the premise of guaranteeing the performance of the whole system, cloud computing provides users with a matching scheduling strategy according to the specific needs of different tasks, which is called cloud scheduling. A robust job scheduling algorithm usually meets the following objectives:

- **Span Optimal**
  In cloud computing, the time period from the start of the first task to the completion of the last task is called span. In different cloud computing environments, the span is uncertain because task execution time is susceptible to the environment, and different cloud environments can lead to unpredictable execution times. Due to the uncertainty of the span, in order to estimate its value, it is necessary to comprehensively estimate the three factors by considering the overhead of task execution and the processing power of a certain node and the running time of other services at the node. In cloud computing, although the span is difficult to estimate, the span should be guaranteed as much as possible.

- **Quality of Service (QOS)**
  With the development of information technology, cloud computing has gradually become a new business model. As a new computing mode, cloud computing has its own characteristics and evaluation criteria. Whether resources can be allocated according to users’ needs is usually taken as an important indicator to measure cloud computing services. Therefore, in the design of task scheduling algorithm, the needs of end users should be put in the first place and the quality of service should be improved as much as possible.

- **Cost Optimization**
Commercial cloud computing is a paid service, which requires end users to pay a certain fee for the resources they need, such as processors, hard disks, bandwidth, etc. Therefore, it is a worth considering problem, for the job scheduling algorithm, which minimizing the user’s cost without affecting the user’s computing requirements.

- **Optimal Overhead**

  Job scheduling algorithm makes full use of resources, but it brings communication overhead and scheduling overhead. After using scheduling algorithm, the total communication overhead caused by data exchange among communication nodes is communication overhead; the overhead caused by node change due to scheduling is scheduling overhead. Finding the appropriate scheduling node and reducing the overhead is the key to the algorithm.

- **Load Balancing**

  In the process of job scheduling, the load threshold of each node should be considered. If the scheduling algorithm makes the load of a node exceed the limit value, the performance of the whole cloud system will be unstable, and resources will be wasted at the same time. Job scheduling algorithm should take into account the load threshold of nodes and dispatch the service to the node with light load to maintain the balance.

**Performance Indicators of Scheduling Target Resource Allocation Algorithms**

The performance metrics of the scheduling algorithm are roughly the same as those for evaluating the performance of the cloud system [23]. There are five main aspects: Average Completion Time (Average Completion Time), Dissatisfaction, Fairness, Locality and Scheduling Time.

- Average completion time, which is the average of the time required for a task to complete from submission to execution in cloud computing.

- The degree of dissatisfaction is usually expressed by the ratio of the user’s demand for resources to the actual devaluation of the demand. It represents the probability that the algorithm meets the user’s minimum demand for resources.
• Fairness, as the name implies, it is used to indicate whether the algorithm is fair in the process of resource allocation. Fairness requires that the algorithm has the following performance: users with the same privileges allocate the same resources; users with different privileges allocate resources according to the weight of each user.

• The degree of localization, that is, the probability that task data is saved on the resource during the execution of the task. If task T is executed on resource R, task T does not copy the data to resource R, indicating that the task is not localized. In cloud computing, because the data processed by the data center is massive, if a large amount of data is not localized, it still needs to be copied when processing such massive data, which will lead to severe loss of performance of the entire system. Therefore, the localization of data is very important.

• Scheduling time, that is, the overhead generated by the scheduling algorithm, is the time taken to complete the scheduling of all tasks. The length of the scheduling time directly reflects the complexity of the scheduling algorithm. Longer scheduling time will directly affect the throughput of the system. Therefore, keeping the scheduling time as small as possible in the resource allocation process is an important indicator.

In summary, the performance indicators of the algorithm are interactional and restrictive. Over-improvement of one performance may lead to the decline of other performance, so that the algorithm with the best performance indicators does not exist. The research on job scheduling algorithm should focus on finding an ideal balance point, which can improve one or more of its performance while minimizing the impact on other performance of the algorithm.

2.3 Optimization Problems

An optimization problem is the problem of finding the extreme values (best or worst solutions) of one or more functions (often called objective functions or cost functions) over a set of (decision) variables subject to a set of constraints [12, 16, 18]. An optimization problem can be mathematically represented as
Maximize/Minimize $f_m(x) \quad m = 1, \ldots, M$
subject to $g_j(x) \leq 0 \quad j = 1, \ldots, J$
$h_k(x) = 0 \quad k = 1, \ldots, K$
$x_i^{(L)} \leq x_i \leq x_i^{(U)} \quad i = 1, \ldots, N$ (2.1)

The number of objective functions are represented by $M$ and a solution $x$ is a vector of $n$ decision variables such that $x \in \mathbb{R}^n$. In case of single variable optimization $n = 1$, which is known as one-dimensional optimization. The constraints and variable bounds determine the feasible region of the solution space $S$ [17].

Next subsections describe the categories of optimization models.

### 2.3.1 Function or Trial-and-Error

In function optimization, mathematical model is built, which describes the objective function or cost function of the problem. The optimal solution of the problem is dependent on the mathematical formula. On the other hand, trial-and-error optimization tries to find the optimal solution by adjusting the values of given input variables without knowing much about the process involved. [8, 20]

### 2.3.2 Single or Multi-Objective

**Single Objective Optimization:** If there is one objective function ($M = 1$) in equation (2.1), the problem is single objective optimization [18] problem. For example in travelling salesman problem (TSP) [21], given a list of cities and distance between pairs of cities, the aim is to find the shortest possible route such that each city is visited once and we return to the origin city. In this problem the objective is to minimize the length of the tour and constraints are that every city should be visited exactly once and return to the origin.

**Multi-Objective Optimization:** Multi-Objective optimization problem deals with two or more objective functions, which are often conflicting [18]. General mathematical model of multi-objective problems is shown in equation (2.1) with $M > 1$. The objective function vectors in multi-objective optimization belong to a multi-dimensional objective space $\mathbb{R}^M$ [12]. In these optimization problems, there are usually more than one optimal solutions.
and these solutions can be defined from a mathematical concept of partial ordering [25]. The solutions in multi-objective optimization are compared using dominance relationship. A solution $x$ dominates another solution $y$ if $x$ is no worse than $y$ in all objectives and $x$ is strictly better than $y$ in at least one of the objectives [23–25]. A solution $x$ is said to be Pareto-optimal [26, 27] if there is no other solution $y \in S$ that dominates $x$. Considering a given set of solutions in Figure 2.1, the points which are not dominated by any other member of the solution space are non-dominated solutions. Solution space is the set of all possible outcomes with regard to the objective functions. A non-dominated solution is one in which it is impossible to improve an objective without worsening at least one other objective. In Figure 2.3, the points 1, 3, 6 and 9 are Pareto-optimal solutions.

### 2.3.3 Static or Dynamic

Static optimization problems are the problems where the output does not change with time [8, 20]. On the other hand, in dynamic optimization problems, the output is a function of time [8, 20]. In other words at different times and circumstances the output can be different. For example, if we want to travel from destination $A$ to destination $B$ and we want to follow the best route. There can be several routes from $A$ to $B$ and from distance point of
view the problem is static optimization problem where the shortest route $R$ will be the optimal solution. But suppose in different times of the day the route $R$ may be crowded or not suitable and our objective is to minimize the traveling time. Considering the traveling time as an objective, the problem is dynamic and the optimal solution can be different during different times of the day. [8]

2.3.4 Continuous or Discrete

Optimization problems can also be categorized into discrete or continuous based on the variables. Continuous variables have an infinite number of possible values, while discrete variables have a finite number of possible values [8, 20]. Discrete optimization is also known as combinatorial optimization [3]. In combinatorial optimization problems, the desired optimal solution is a certain combination of variable settings from the finite pool of possibilities (variable settings). Most of the real-world optimization problems are combinatorial optimization problems. Some of the examples of combinatorial optimization problems, which can be solved using well-known polynomial-time algorithms are metroid, matching, shortest path, and spanning trees [24]. There is also a class of combinatorial optimization problems, where the search space becomes intractable even for moderate sized instances and many of them are NP-hard [27] combinatorial optimization problems. Some of the examples of combinatorial NP-hard optimization problems are generalized assignment problems (GAP) [9, 28], travelling salesman problems, nurse scheduling problems and the large scale assignment, planning and rescheduling problems discussed in this thesis. For these kinds of problems, there is no known polynomial-time algorithm. Meta-heuristics [22] play an important role in sampling such intractably large and highly complex search spaces.

2.3.5 Constrained or Unconstrained

Unconstrained optimization problems are only concerned with the objective function to be optimized without any restrictions on the problem variables [8, 20]. For example, a firm wants to outsource a project to a subcontractor who maximizes the quality of the projects. On the other hand, constrained optimization problems optimize the objective function subject to some restrictions (constraints) on the involved variables [8, 20]. For instance, the
firm maximizing the quality will be usually subject to a cost constraint.

2.3.6 Linear or Non-linear

In Linear optimization problems, the objective functions and constraints are linear [8, 23]. Due to linear nature of the objective functions, the optimum is always located at the boundaries of feasible area, which is defined by constraints and local optimum is also a global optimum. Non-linear optimization problems are those problems in which one or more constraints or the objective function are non-linear [8, 23]. Non-linear means that the output of the function is not directly proportional to the input. The optimum of non-linear problems is not necessarily located at the boundaries of the feasible region, it can also be interior of the region. Further, a local optimum is not necessarily to be the global optimum. Non-linear optimization problems are harder than linear problems. Most of the real-world optimization problems are non-linear due to the nature of the physical systems [24].

2.4 Genetic Algorithms

2.4.1 Rationale

Genetic Algorithm (GA), first proposed by John Holland in 1975 [10], are a type of meta-heuristic search and optimization algorithms inspired by Darwin’s principle of natural selection. The central idea of natural selection is the fittest survive. Through the process of natural selection, organisms adapt to optimize their chances for survival in a given environment. Random mutations occur to the genetic description of an organism, which is then passed on to its children. Should a mutation prove helpful, these children are more likely to survive to reproduce. Should it be harmful, these children are less likely to reproduce, so the bad trait will die with them [12].

In analogy, GA maintains a “population” of solution candidates for the given problem. Elements are drawn at random from this population and allowed to “reproduce”, by combining some aspects of the two parent solutions. The key is that the probability that an element is chosen to reproduce is based on its “fitness”, essentially an objective function related to the solution. Eventually, unfit elements die from the population, to be replaced by successful solution offspring [12].
In GA, solutions are parametrically represented in strings of code (e.g., binary). Fitness value is defined to evaluate solutions. The general procedures of a GA include: 1) Create a population of random individuals; 2) Evaluate each individual’s fitness; 3) Select individuals to be parents; 4) Produce children; 5) Evaluate children; 6) Repeat steps 3 to 5 until a solution with satisfied fitness is found or some predetermined number of generations is met [13]. More details on GA procedures are further treated in Chapter 4.

2.4.2 Applications

Since the inception of GA, they have found applications in numerous areas. In the area of engineering design, Yao (1992) used GA to estimate parameters for nonlinear systems [14]; Joines (1996) applied GA to manufacturing cell design [15]; Gold (1998) introduced GA to kinematic design of turbine blade fixtures [16]. In the area of scheduling and planning, Timothy (1993) optimized sequencing problems using GA [17]; Davern (1994) designed the architecture for job shop scheduling with GA [18]. In the area of computer science, Rho (1995) used GA in distributed database design [19]. In the area of image processing, Tadikonda (1993) used GA to realize automated image segmentation and interpretation [20]; Huang (1998) designed detection strategies for face recognition with GA [21].

Due to the fact that GA is non-problem-specific, its application is not confined with the problems’ physical background, and hence can be applied to many combinatorial optimization problems in different disciplines.

2.4.3 General Research Based on GA

The interest in understanding and promoting GA’s performance motivated considerable theoretical research of GA. The ultimate goal is to be able to design efficient and robust GAs. To achieve this goal, however, two fundamental questions should be fully understood:

1) How do GAs work?

2) What types of problems are suitable for GAs to solve [22]?

Generally speaking, almost all the theoretical work in GA stems from these two fundamental questions.

To answer the first of the two questions, mathematical tools are adopted to model the evolution process and explain the improvement of solutions.
over generations. The first relatively rigorous model should be credited to John Holland [10]. He developed Schema Theorem to describe how certain pieces of code thrive or diminish depending on their fitness relative to the average. Although there are some criticisms against Schema Theorem, it still serves as the basis for many theoretical studies of GA. Some research related to Schema Theorem includes the modification of the theorem [23]. Another attempt is to model the GA process as a Markov process [24]. The Markov models aim at describing the convergence behavior of GA. It is more precise, but unfortunately, this model is usually very complicated and offers little practical guide to designing competent GAs. Based on the study of the mechanisms of GA, Goldberg proposed Building Block Hypothesis. He then decomposed the design of GA and provided several guidelines to the design of competent GAs [25]. From there, a series of GA was designed [26, 27, 28] with improved efficiency and/or robustness. GA does not work well for all the problems. Thus, it is important to understand what type of problem GA is capable in solving, or alternatively what makes it difficult for GA. The central idea to address this question is the idea of epistasis. Simply put, epistasis refers to the interdependency between the parameters of a solution, which incurs nonlinearity and hence makes the problem hard for GA. Davidor proposed a method to measure epistasis [29]. Vose and Liepins showed that in principle epistasis in any 6 problem can be reduced through different encoding schemes [30]. However, for complicate problems, devising such a coding scheme can be a formidable task.

2.5 Cloud Computing Simulation

A Cloud Computing offer ranges from proposing a specific IT infrastructure to deploying complicated applications and software solutions. By studying the Cloud Computing service delivery model originates the challenge of managing hundreds of thousands of users and applications requests. Therefore, a Cloud Computing provider should consider intelligent infrastructure deployment in order to establish a Cloud Computing offer, which insures transparency, scalability, security and foremost celerity (QoS) [23].

Cloud Computing assessment and evaluation is mandatory for both Cloud providers that are planning a specific service delivery and Cloud users who are intending to shift their IT infrastructure, platform or software into the Cloud (Internet in case of public cloud or Intranet in case of private cloud).
Despite the fact that using real infrastructure for testing and evaluating cloud deployment can give the investigators a real world approach to make critical decision about moving forward with this model of computing, in most cases it can be very expensive:

- Infrastructures, platforms and software high costs
- Necessity to test on scalable environments (more infrastructure)
- Management and maintenance expenses

In addition to this critical factor, we can add the time consumption in order to test a specific scenario:

- Infrastructures Installations and configurations
- Repeatable and variable tests
- Debugging and troubleshooting

A more viable alternative is the use of simulation tools. These tools open up the possibility of evaluating the hypothesis (application benchmark study) in a controlled environment where one can easily reproduce results [22]. There are variety of simulator tools for modelling and simulation of large-scale Cloud computing environments [24] (Figure 2.4). Generally, we can designate between two types of simulators: graphical user interface (GUI) simulators or programming language based simulators (like Java for example).

2.6 Cloud Computing Simulators Evaluation

In the past decade, Grids [14] have evolved as the infrastructure for delivering high-performance services for compute- and data-intensive scientific applications. To support research, development, and testing of new Grid components, policies, and middleware, several Grid simulators, such as GridSim [10], SimGrid [9], OptorSim [15], and GangSim [28], have been proposed. SimGrid is a generic framework for simulation of distributed applications on Grid platforms. Similarly, GangSim is a Grid simulation toolkit that provides support for modeling of Grid-based virtual organizations and resources.
On the other hand, GridSim is an event-driven simulation toolkit for heterogeneous Grid resources. It supports comprehensive modeling of grid entities, users, machines, and network, including network traffic.

Although the aforementioned toolkits are capable of modeling and simulating the Grid application management behaviors (execution, provisioning, discovery, and monitoring), none of them are able to clearly isolate the multi-layer service abstractions (SaaS, PaaS, and IaaS) differentiation required by Cloud computing environments. In particular, there is very little or no support in existing Grid simulation toolkits for modeling of virtualization-enabled resource and application management environment. Clouds promise to deliver services on subscription-basis in a pay-as-you-go model to SaaS providers. Therefore, Cloud environment modeling and simulation toolkits must provide support for economic entities, such as Cloud brokers and CEx, for enabling real-time trading of services between customers and providers. Among the currently available simulators discussed in this paper, only GridSim offers support for economic-driven resource management and application provisioning simulation. Moreover, none of the currently available Grid simulators offer support for simulation of virtualized infrastructures, neither have they provided tools for modeling data-center type of environments that can consist of hundred-of-thousands of computing servers.

Recently, Yahoo and HP have led the establishment of a global Cloud computing testbed, called Open Cirrus, supporting a federation of data centers located in 10 organizations [26]. Building such experimental environ-
ments is expensive and hard to conduct repeatable experiments as resource conditions vary from time to time due to its shared nature. Also, their accessibility is limited to members of this collaboration. Hence, simulation environments play an important role.

As Cloud computing RD is still in the infancy stage [31], a number of important issues need detailed investigation along the layered Cloud computing architecture (see Figure 2.5). Topics of interest include economic and also energy-efficient strategies for provisioning of virtualized resources to end-user’s requests, inter-cloud negotiations, and federation of clouds[32,33]. To support and accelerate the research related to Cloud computing systems, applications and services, it is important that the necessary software tools are designed and developed to aid researchers and industrial developers.

2.7 Review of Related Work

As users increase, tasks to be scheduled also increase proportionately. The core of resources scheduling technology lies in the scheduling algorithm. In the other hand, scheduling of tasks is considered a critical issue in the Cloud computing environment by considering different factors like completion time,
the total cost for executing all users’ tasks, utilization of the resource, power consumption, and fault tolerance. So, better algorithms to schedule tasks on such systems are required.

At present, most cloud computing platforms adopt the Map/Reduce programming model proposed by Google for parallel computing of large-scale data sets. Through Map and Reduce phases, the larger tasks are divided into several smaller sub-tasks, and then allocated to multiple computing resources for parallel execution, so as to obtain the final running result. In Map/Reduce programming model, how to schedule a large number of sub-tasks is a complex problem. Cloud computing needs to provide services to multiple users at the same time, taking into account the response time of each user, as well as the cost of services. Some existing scheduling algorithms usually only consider the optimization of one of the factors, which easily leads to the situation that the task completion time is relatively short and the cost is relatively high. Therefore, task scheduling is considered as a NP-complete problem in cloud computing environment. To solve NP-complete and NP-hard problems, in general, heuristic approaches are used. Heuristic techniques used are local heuristics, meta-heuristics and hyper-heuristics. Hyper-heuristics operate at a higher abstraction levels. Meta-heuristic techniques are expensive techniques needing knowledge in problem domain and heuristic technique.

In recent years, genetic algorithm (GA), particle swarm optimization algorithm (PSO) and ant colony algorithm (ACO) have been proposed to solve task scheduling problems in cloud computing. However, the problem becomes obvious when the tasks are dependent on each other (i.e., workflow application). The dependent tasks require a specific execution order due to the relationship between them. There are two types of workflow scheduling: the best-effort workflow scheduling and the quality of services (QoS) constraint workflow scheduling [5, 18]. However, the best-effort workflow scheduling focuses on reducing the execution time of the whole workflow tasks regardless of other factors. Many types of research were based on the best-effort workflow scheduling to reduce the execution time, such as Yongjun Song [6] has applied genetic algorithm to effectively schedule independent tasks in cloud computing environment.

One of the scheduling issues is to allocate the correct resource to the arriving tasks. The dynamic scheduling process is considered complex if several tasks arrive at the same time, Therefore, S. Ravichandran and D. E.
Naganathan [7] have introduced a system to avoid this problem by allowing the arrived tasks to wait in a queue and the scheduling will recompute and sort these tasks. Therefore, the scheduling is done by taking the first task from the queue and allocated to the resource that will be the best fit using GA. The objective of this system is to maximize utilization of resources as also reduce execution time.

In [9] Ji Wang Zhengjun Zhai have proposed a cloud computing task scheduling algorithm based on particle swarm optimization, which optimizes the transmission time and processing time of tasks. Ezio Todini [10] has applied ant colony algorithm to schedule cloud computing tasks, so that the total task completion time and the average task completion time are minimized.

However, any algorithm has its own advantages and disadvantages. For example, although genetic algorithm has fast and random global search ability, it requires more parameters, more complex programming implementation and easy to fall into local optimum. Particle swarm optimization algorithm has fast convergence speed in the initial stage, insufficient local search ability in the later stage but slow convergence speed, still compared with genetic algorithm, the convergence speed of particle swarm optimization algorithm is faster. First, the optimization performance is basically better than the later one since the programming is easy to implement and it can be easily extended or applied to solve multi-objective problems; while the ant colony algorithm has good optimization ability, but the initial pheromone is scarce and the convergence speed is slow.

2.8 Summary

This chapter first introduces the concept, characteristics, architecture and platform of cloud computing in detail; then introduces the concept of cloud computing task scheduling and its problems and task scheduling model; and introduces common resource allocation algorithms; introduces the performance evaluation index of cloud computing resource allocation algorithm; and then introduces the concept and types of problem optimization; then the simulation platform is introduced. Finally, summaries the related works.
Chapter 3

Design and Implementation: Goals and Task Scheduling
Model Design

According to SLA, the Cloud provider should guarantee optimal scheduling of user’s tasks in the Cloud computing environment. At the same time, the provider should guarantee the best throughput and optimum utilization of the Cloud resources.

Generally, by increasing the users’ tasks, the complexity of scheduling these tasks in the Cloud computing environment will be increased proportionally. Therefore, the Cloud provider needs a good algorithm to schedule the users’ tasks on the Cloud. The algorithm should satisfy Quality of Service (QoS), minimizing makespan and guaranteeing good utilization of the Cloud resources [33]. Therefore, task scheduling is classified as an optimization problem.

Fig. 3.1 illustrates the task scheduling process where each user apply his application’s tasks and the Cloud provider uses the appropriate approaches to schedule these tasks by considering those optimization parameters, such as minimum makespan, resources utilization and minimum cost.

3.1 Concepts

Currently, task scheduling algorithms in cloud computing can be divided into three categories:
Traditional Task Scheduling Algorithms

Traditional task scheduling algorithms mainly refer to task scheduling algorithms derived from grid computing, such as Min-min algorithm, Max-min algorithm, Sufferage algorithm and so on. The basic idea of Min-min algorithm [30] is to allocate tasks to resources with the earliest execution start time and the fastest execution speed. Max-min algorithm is similar to Min-min scheduling method but has its own characteristics. The Min-min algorithm first completes the task with short execution time and then completes the execution time for a long time, while the Max-min algorithm is the opposite. Sufferage algorithm defines the difference between the earliest completion time and the second earliest completion time as Sufferage value. When scheduling with this algorithm, resources are always allocated to resource nodes with larger Sufferage value to avoid the big loss.

Because of the similarity between grid computing and cloud computing, it is feasible to study task scheduling in cloud computing by using task scheduling algorithm in grid computing.

Hadoop task scheduling algorithm model

FIFO scheduling algorithm is the most widely used algorithm in Hadoop system, which is scheduled according to the time and priority of user submitting jobs. In the new version of Hadoop, engineers of Facebook and Yahoo have proposed two algorithms, namely fair scheduling algorithm and computing power scheduling algorithm. Among them, the core idea of Yahoo’s Fair Scheduling [27] is to provide as much as pos-
sible equal access to system resources to users who might have different needs. The core idea of Facebook’s Capacity Scheduling algorithm is to ensure that each job can occupy its own resources by establishing job queues to manage and maintain jobs.

Intelligent Task Scheduling Algorithms

In recent years, experts and scholars around the world have proposed many inspiring intelligent algorithms, such as genetic algorithm, ant colony algorithm, particle swarm algorithm and the fusion of these algorithms. Genetic algorithm [29] is an adaptive probability optimization algorithm based on biological genetic and evolutionary mechanism created by Professor Holland. The algorithm automatically acquires and accumulates knowledge about the search space in the search process, and adaptively controls the search process to obtain the optimal solution. Ant colony algorithm and Particle swarm optimization are two new type of simulated evolutionary algorithms that simulate the foraging behavior of animals in nature. They are the main algorithms studied in the field of swarm intelligence theory. The so-called swarm intelligence refers to the agent with simple intelligence, which shows complex intelligent behavior characteristics through cooperation.

3.2 Goals and Tasks Scheduling Model Design

Resources in cloud computing include processors, memory, networks, etc. The use of resources is on-demand and pay-per-use. In this thesis, resources in cloud computing are unified as computing resources and for the convenience of research work, the following assumptions are made:

- Ignoring the impact of broadband and data transmission on task execution time, assuming that the execution time of a task is equal to the length of the task divided by the running speed (MIPS) of the VM that executes the task.

- Cloud platform forms a task set and allocates its resources at intervals. Each task is independent of each other and when multiple tasks are
allocated to the same virtual machine, tasks are executed according to the first-in-first-out (FIFO) principle.

- The number of tasks is greater than the number of resources.
- The cost of running tasks for a unit time on each computing resource node is known.

Serving from the assumption, a mathematical model has been made in order to better formulate the cloud computing task scheduling problem. The mathematical model is as follows:

1. Assuming that there are \( m \) cloud tasks submitted by users to be scheduled \( T = \{t_1, t_2, \ldots, t_j, \ldots, t_m\} \), where \( t_j \) represents \( j^{th} \) task, which in turn contains \( k \) different feature attributes as \( t_j = \{t_j(1), t_j(2), \ldots, t_j(k)\} \).

2. \( n \) resources with \( n \ll m \), where resources refer to VMs in cloud computing data sets, \( V = \{v_1, v_2, \ldots, v_i, \ldots, v_n\} \) and \( v_i \) represents \( i^{th} \) virtual machine, which also contains \( t \) different attributes as \( v_i = \{v_i(1), v_i(2), \ldots, v_i(t)\} \).

3. Each task can only be executed on one VM, so the assignment relationship between tasks and VMs can be represented by a \( m \times n \) matrix \( X \):

\[
X = \begin{pmatrix}
    x_{11} & x_{12} & \ldots & x_{1n} \\
    x_{21} & x_{22} & \ldots & x_{2n} \\
    \vdots & \vdots & \ddots & \vdots \\
    x_{m1} & x_{m2} & \ldots & x_{mn}
\end{pmatrix}
\]  

(3.1)

Where \( x_{ji}, j \in [1, m] \) and \( i \in [1, n] \), indicates that the \( j^{th} \) cloud task runs on the \( i^{th} \) VM.

4. We use ETC (Expect Time to Complete) matrix to calculate the time required to complete the tasks on each computing resource, where ETC \( (i, j) \) represents the time required for the \( j^{th} \) task to complete on the \( i^{th} \) computing resource.

\[
ETC = \begin{pmatrix}
    etc_{11} & etc_{12} & \ldots & etc_{1n} \\
    etc_{21} & etc_{22} & \ldots & etc_{2n} \\
    \vdots & \vdots & \ddots & \vdots \\
    etc_{m1} & etc_{m2} & \ldots & etc_{mn}
\end{pmatrix}
\]  

(3.2)

and \( etc(i, j) = \frac{t_j, \text{Length}}{v_i, \text{MIPS}} \) and Length is the size of \( j^{th} \) task while MIPS represents the speed of \( i^{th} \) VM(virtual machine). And PCC (Predicted Cost to Complete) matrix counts the cost required to complete the tasks on each
computing resource, where $PCC_{(i,j)}$ represents the execution cost of the $j^{th}$ task running on the $i^{th}$ VM.

$$PCC = \begin{pmatrix}
pcc_{11} & \ldots & pcc_{1n} \\
\vdots & \ddots & \vdots \\
pcc_{m1} & \ldots & pcc_{mn}
\end{pmatrix} \quad (3.3)$$

Note that the predicted cost for each unit will be different according to the attributes of each VM. We have also created a $n \times n$ matrix called $CCU$ (communication cost per unit), $CCU(e,g)$ and $e,g \in [1,N]$ represents the communication cost between the $e^{th}$ virtual machine and the $g^{th}$ virtual machine.

$$CCU = \begin{pmatrix}
0 & ccu_{12} & \ldots & ccu_{1n} \\
ccu_{21} & 0 & \ldots & ccu_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
ccu_{n1} & ccu_{n2} & \ldots & 0
\end{pmatrix} \quad (3.4)$$

where, $ccu_{eg} = 0$ menas that communication cost is 0 between $e^{th}$ and $g^{th}$ virtual machine when $e = g$, that is, different tasks are continuously executed on the same virtual machine.

5. Based on the above assumptions and constructions, hereby, we can obtain the following functions.

Since the VM runs in parallel, the total time for task scheduling is defined as

$$\max\left\{ \sum_{j=1}^{m} ETC_{ji} \right\} , \quad (3.5)$$

and the total predicted cost is

$$cost = \sum_{j=1}^{m} \sum_{i=1}^{n} ETC_{ji} \times PCC_{ji} + \sum_{e,g=1}^{n} CCU_{eg} . \quad (3.6)$$

6. Based on the above assumptions and conditions, we define our goal as how to rationally allocate tasks to various computing resources, so that the tasks can be completed in the shortest time at the lowest cost, that are $\text{Minimize}\{\max\sum_{j=1}^{m} ETC_{ji}\}$ and $\text{Minimize}\{cost\}$.

Then we conclude that the whole problem is actually the same as seeking the shortest working time and the least cost for the longest running virtual machines, as virtual machines run in parallel way.
Chapter 4

Design and Implementation: Approach

The main interest of cloud providers is to increase profits by achieving high levels of users’ satisfaction, which comes with providing the user with the best experience. Hence, choosing the finest scheduling algorithms for resources allocation and task scheduling is very important.

As we have discussed in the previous chapter the whole problem is considered as an optimization problem; we acquired the object function by establishing a mathematics model that can be solved using heuristic algorithm such as Genetic Algorithm (GA), Particle Swarm Optimization (PSO), and Ant Colony Optimization (ACO).

Comparing with other algorithms. In GA Algorithm, chromosomes share information with each other, so the whole population moves evenly to the optimal region. The coding technology and genetic operation of GA are easy to implement. In addition, GA can be applied to a wide range of optimization problems and also to discrete problems such as TSP problem, salesman problem, job shop scheduling and so on.

Hence, in this thesis, we proposed an algorithm to solve task scheduling optimization problem in Cloud Computing environments based on the GA algorithm.
4.1 Task Scheduling Strategy Based on GA Algorithms

As we introduced in Chapter 2, GA is a flexible approach, which is based on the biological concept of generating a population. It is considered a rapidly growing area of Artificial intelligence [32, 33]. GA was inspired by Darwin’s theory of evolution. According to the theory, the term “Survival of the fittest” is used as the method of scheduling in which the tasks are assigned to resources according to the value of the fitness function [33]. This heuristic algorithm is routinely used to generate useful solutions for optimization and search problems. Genetic algorithms belong to the larger class of evolutionary algorithms (EA), which generate solutions for optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection and crossover. The output of GA is an array list of individuals from population (called chromosomes or the genotype of the genome), which are encoded candidate solutions for an optimization problem. And the result of the algorithm always evolves toward better and better solutions. As benefiting from GA, time minimization will give profit to service provider and less maintenance cost to the resources. It will also provide benefit to cloud’s service users as their application will be executed at reduced cost. The main principles of the GA are described as follows:

4.1.1 Encoding and Decoding

Encoding is the first key step of genetic algorithm. The encoding method directly affects genetic operations such as selection, crossover and mutation as a consequence it determines the efficiency of the genetic algorithm to some extent. There are many ways of coding chromosome, either directly or indirectly. In this thesis, the indirect encoding method of resource-task proposed in paper [31] is used to encode the resources occupied by tasks. The length of the chromosome is the number of tasks, and the value of each gene in the chromosome is the id of VM assigned to execute the specific task, as shown in Figure. 4.1

Where, $T_j$ denotes the task id and $P_i$ denotes the resource id where the task $T_j$ is executed. When the initial population is generated, the resource id $P_i$ in each chromosome is generated randomly. After crossover and mutation operators, task $T_j$ may take up any available resource. The optimal solution
must correspond to a chromosome coding. Assuming that there are 10 tasks and 3 available resources, the chromosome length is 10 and each gene is taken as a random number between 1 and 3. For example, the following chromosome coding is generated randomly:

\{2, 1, 2, 1, 3, 1, 3, 2, 3\}

Where, the top line of the instance represents the cloud tasks, and the following line represents the individual genes (VMs), the corresponding relation: the task 1 is assigned to VM 2, the task 2 is assigned to VM 1, and so on, the task 10 is assigned to VM 3.

However, encoding and decoding are a pair of reciprocal processes. After the generation of chromosomes, they must be decoded to obtain the distribution of tasks on different resources. Tasks are categorized according to their occupied resources then a number of sequences tasks are generated. The above chromosomes are decoded as follows:

\[ P_1 : \{T_2, T_4, T_6\} \]
\[ P_2 : \{T_1, T_3, T_9\} \]
\[ P_3 : \{T_5, T_7, T_8, T_{10}\} \]

Finally, acquiring the sequence of assigned tasks on the respective computing resources via decoding process. Then applying ETC, PCC, ... matrices (discussed in Chapter 3) to calculate the time and estimated cost for each computing resource to complete the task sequence.

### 4.1.2 Initialization of the Population

The genetic algorithm is very sensitive to the initial population which has a great influence on the convergence speed and global optimization of the algorithm. In order to accelerate the convergence speed of the algorithm and
generate the global optimal solution, the similarity between individuals is introduced to ensure that the initial population is evenly distributed in the solution space.

For computing the similarity between two chromosomes, we set the population size as $S$, the length of individual chromosome as $L$, and the number of VMs as $n$:

Hamming code distance is applied and the distance of two individual chromosomes:

$$D(i_1, i_2) = \sum_{j=1}^{L} |G_{i_1j} - G_{i_2j}| \quad \text{and} \quad i_1, i_2 \in (1, 2, \ldots, n); \quad (4.1)$$

where, $|G_{i_1j} - G_{i_2j}| = \begin{cases} 1, & G_{i_1j} \neq G_{i_2j} \\ 0, & G_{i_1j} = G_{i_2j}. \end{cases}$

Similarity:

$$Sim(i_1, i_2) = \frac{D(i_1, i_2)}{L} \quad (4.2)$$

Note that the bigger Hamming code distance, the greater value of similarity, the more diverse of population.

Detection threshold: $\mu = \frac{L-C}{L}, \ C$ represents adjusting parameter.

In order to be selected into the initial population, two individual chromosomes with similarity $Sim(i_1, i_2) > \mu$ must be satisfied.

The initial population generated by the above method can ensure that there are great differences among individuals in the population. When the population size is large, it can be distributed in the solution space in a large range and uniformly, thus reducing the probability of local optimization and improving the ability of global search.

4.1.3 Fitness Function Design

The fitness function [27] is usually introduced in genetic algorithm to evaluate the individual’s fitness in order to ensure that the population evolves in a better direction. In the population, the larger the fitness, the better the individual, and vice versa, the smaller the fitness, the worse the individual. Finally, the fitter individual will be selected and inherited to the next generation. Individuals with poor fitness will be eliminated, which also confirms the evolutionary theory of "natural selection, survival of the fittest". In genetic algorithm, fitness function is usually set to positive value, because
fitness value need to be ranked in order of size in genetic operation. Therefore, the maximum fitness value represents the fittest chromosome. However, the cloud computing resource allocation or task scheduling problem should be solved by seeking the minimum total cost (time and monetary cost) under constraints. As a result we define our fitness function to be the reciprocal of the minimum total cost. Therefore, when designing fitness function, we need to set it synthetically according to the above constraints and ultimately transform it into a function which finds the maximum value. We define fitness function as:

The fitness function of time:

$$F_{\text{time}} = \frac{1}{\sum_{j=1}^{m} ETC_{ji}}$$ (4.3)

The fitness function of monetary cost:

$$F_{\text{cost}} = \frac{1}{\sum_{j=1}^{m} \sum_{i=1}^{n} ETC_{ji} \times PCC_{ji} + \sum_{e,g=1}^{n} CCU_{eg}}$$ (4.4)

In the fitness function only considering time constraints, the higher the utilization rate of computing resources, the shorter the time required to complete all tasks, the greater the fitness value, whereas only considering cost constraints, the lower the cost required to complete all tasks, the greater the fitness value. So, we define the fitness function respect to time and cost constraints as:

$$Fitness = \alpha \times F_{\text{time}} + \beta \times F_{\text{cost}}$$ (4.5)

where, $\alpha \in [0, 1], \beta \in [0, 1], \alpha + \beta = 1$. $\alpha$ is the time factor and $\beta$ is the cost factor. Their values are specified by users according to their needs and $\alpha$ is inversely proportional to $\beta$. For time-sensitive applications, $\alpha$ should be assigned to a smaller value between [0.2, 0.5], while $\beta$ should be assigned to a larger value between [0.5, 0.8], vice versa, for cost-sensitive applications, $\beta$ should be assigned to a smaller value between [0.2, 0.5], and $\alpha$ should be assigned to a larger value between [0.5, 0.8]. For instance, when $\alpha = 1, \beta = 0$, the result of algorithm scheduling is the shortest time to complete all tasks, while $\alpha = 0, \beta = 1$, the result of algorithm scheduling is the least cost scheduling to complete all tasks.
4.1.4 Selection Model

Selection is the process of selecting individuals with strong adaptability in a population to produce a new population. According to the principle of "survival of the fittest", the higher the fitness of an individual, the greater the probability of being selected to inherit to the next generation, which makes the fitness value of an individual in the population keep approaching the optimal solution. In this stage, roulette algorithm is usually used for individual selection. As a result, some local optimal individuals are easily eliminated in the process of crossover and mutation, which might reduce the convergence speed of the algorithm. In this thesis, an optimal preservation strategy is used to implement the selection operation. In the selection operation, the fitness of each individual in the population is calculated and sorted in descending order. The 10% individuals with the highest fitness in the parent population are directly selected into the offspring population while the remaining 90% individuals are selected according to the roulette algorithm. This method can ensure that the optimal local individuals are not destroyed in the crossover and mutation operations of genetic operations ensuring the diversity of offspring and, as a result enhancing the global search ability of the algorithm.

4.1.5 Crossover Operation

The crossover operator aims to generate new chromosomes through changing the position of the genes inside every two chromosomes. In the crossover, a random number is selected in the range of the number of the chromosome genes, to represent the division point of each chromosome into two parts. The crossover returns an offspring chromosome of two parts that contains both chromosomes genes, that is, VMs. The first group of VMs takes the first chromosome until the index, which is determined by the random number. The second chromosome has the second group of the VMs starting from the index, which is determined by the random number, until the end of the chromosome.

4.1.6 Mutation Operation

The mutation operator aims to make unusual modifications in the new chromosomes that are generated from the previous crossover operator with better
fitness value than the existing chromosomes. The mutation operator operates over the returned chromosome from the selection method, and the occurrence of the mutation is based on the mutation rate variable. The mutation process starts with a number that is randomly generated to be less than or equal to the mutation rate. Two genes, that is, VMs, are selected randomly from the same chromosome and checked to be different. If they are not the same, their places are swapped to generate new chromosome, which represents a different distribution of the tasks over the available VMs. The generated chromosome is then passed to the next stage of the algorithm.

4.2 Algorithm Architecture and Implementation

GA architecture is shown in Picture. 4.2:

- Input: population size $S$, mutation rate $x$, crossover rate $y$, elitism number $m$, cloud size $j$, virtual machine number (vms) $i$, maximum number of iterations $z$, weighting factors $\alpha$ and $\beta$.

- Output: optimal solution of task scheduling

The algorithm has been implemented with Java language, the main Java pseudo code is shown in next page.

4.3 Simulation Tool: CloudSim Toolkit

As described in the Chapter 2, there are a diversity of Cloud Computing simulators, each with specific characteristics and oriented for a specific objective. Choosing the finest Cloud Computing simulator is a challenging mission. But to the best of our knowledge, we found that CloudSim is spotted as the core platform for the most used Cloud simulators up to this moment. CloudSim was established as an extension of the GridSim simulator in order to introduce the Cloud Computing virtualization layer that was not present on the original simulator.

CloudSim is a programming language based simulator and even though it does not support a graphical user interface for simulation, it proposes the CloudAnalyst (which is an extension of CloudSim) for investigators who
Figure 4.2: Genetic Algorithm Architecture
Input: chromosomeLength, populationSize
Output: population

GeneticAlgorithm ga = new GeneticAlgorithm(15, 0.30, 0.95, 2,
cloudletList, vmlist)
Population[ ] = Individual[populationSize]
for i = 0 to populationSize
    Individual individual = Individual(chromosomeLength)
    if (i=0){
        Population[i]= individual
    } else {
        If (Sim(i,i-1) > u){
            Population[i]= individual
        }else i--;
    }
}

Listing 1: Initial population

Input: individual
Output: fitness

for i = 0 to chromosome.length
    executionCostArray [] = cloudletList.get(i).executioncost;
    executionCost += executionCostArray [ individual.chromosome[i]] * etc[i]
end for
for j=1 to chromosome.length
    communicationCostArray = vmList.get(individual.getGene(j)).ccu[individual.getGene(j-1)];
    communicationCost+ = communicationCostArray
end for
    cost = executionCost + communicationCost
for k=0 to chromosome.length
    Makespans[individual.getGene(k)] + = etc[k]
end for
    Makespan = sortNumber(Makespans)
    fitness = a*1/cost+b*1/makespan

Listing 2: Calculation Fitness
Input: the chromosomes
Output: fitness chromosome
Set the tournamentSize = n
for i = 0 to tournamentSize
  id = Math.random() * chromosome.size() # select chromosome randomly
  tournament[i] = get chromosome (id)
End for
fitness = tournament.getFitness() # return the fitness value.

Listing 3: Selection Operation

Input: chromosome
Output: offspring chromosome
chromosome1 = this.chromosome
chromosome2 = population.getFittest(0)
r = (Math.random() * chromosome.length)
for i = 0, j = 0 to r
  offspring chromosome[j] = chromosome1[i]
End for
for i = r to chromosome.length
  offspring chromosome[j] = chromosome2[r]
End for

Listing 4: Crossover Operation

Input: offspring chromosome # returned from crossover operator
Output: New chromosome
Set mutationRate = 0.3
if (Math.random() <= mutationRate)
t1 = Rand [0, 1] * offspring chromosome.length # select a random number t1
t2 = Rand [0, 1] * offspring chromosome.length # select a random number t2
if offspring chromosome [t1] != offspring chromosome[t2]
  Swap (offspring chromosome [t1], offspring chromosome[t2])
End if
End if

Listing 5: Mutation Operation
prefers using a user-friendly interface to carry out their researches. CloudSim presents itself to the cloud-computing researchers as a Java based framework that supports the main characteristics of Cloud Computing (IaaS) with virtualization support and task scheduling (PaaS and SaaS) and open up the door for emerging, integrating and testing new algorithms for task scheduling or new characteristics development, which helped on delivering new simulators.

The reason behind using CloudSim stems from it openness and clear logic which is deficient on the other simulators specifically with GUI based simulators where we were not able to tackle the Cloud infrastructure layer to better test algorithms related to resource allocation and introduce new algorithms. CloudSim was the accurate selection for our research, which is focused on evaluating and assessing the task scheduling algorithms for resources allocation in order to help Cloud providers and users make precise decision about Cloud Computing model adoption.

4.3.1 CloudSim Architecture

As we know CloudSim is a Java application that was founded on GridSim, which is a simulator and a toolkit for modeling and simulation of entities in parallel and distributed computing [27],[28]. CloudSim [28] was designed in a layered architecture as showed on Figure 4.3. At the lowest layer, we find the "SimJava" (Discrete Event Simulation) which implements the core functionalities needed by the higher level of simulation (Data Center, Host, Virtual machine...). Just above the SimJava we find the "GridSim" toolkit for modeling multiple Grid infrastructures, including networks and associated traffic. At the next layer, we find the CloudSim simulation layer, which provides support for modeling and simulation of virtualized Cloud-based data center environments including dedicated management interfaces for VMs, memory, storage, and bandwidth. This layer handles the fundamental issues, such as provisioning of hosts to VMs, managing application execution, and monitoring dynamic system state [29]. The top layer in the CloudSim simulation toolkit is the “User Code” which is the main interface for simulation specifications and characteristics configuration (number of machines, applications, tasks, users, scheduling policies and their basic structure).
4.3.2 CloudSim Scheduling Policies

The virtualization technology is one of the fundamental concepts of Cloud Computing infrastructures. CloudSim deploy enormously the virtualization technology in order to simulate IaaS and PaaS provisioning and to use it as a base for users’ applications execution. On the same perspective comes the challenge of deploying the finest resources allocation and scheduling algorithm. For example, one Data center that consists of one host with two processing units and where the cloud user is trying to instantiate two virtual machines with two processing units each. Logically, there is a separation between the two virtual machines, but in reality, each virtual machine is limited to the processing power offered by the physical host, therefore we cannot instantiate both virtual machine on the same host at the same time without an appropriate scheduling algorithm [28].

In reference to this critical factor, CloudSim proposes two levels of resources allocation policies based on two basic scheduling policies, which are the time-shared and space-shared allocation policies. These allocation policies are implemented during the virtual machines construction and throughout the application execution. The Space-Shared policy and Time-Shared policies are depictions of the ”FCFS = First Come First Served” and ”RR = Round Robin” algorithms respectively.

In order to illustrate clearly the concept of each allocation policy [33,34],
we propose the following example:

- One data Center with one host
- The host has two processing units
- The user instantiate two virtual machines that require one processing unit each
- The user then try to execute two tasks (Cloudlets) in each virtual machine (each task requires one processing unit for execution)

Figure 4.4 represents a space-shared policy for both virtual machines and tasks. While each virtual machine requires one processing unit, each virtual machine will reserve one of the two processing units of the host, nevertheless only one task can get executed at a specific time and the second one will wait for the first task to end in order to get executed.

![Figure 4.4: Space Shared Policy for VMs and Tasks](image)

Figure 4.5 presents a space-shared policy for virtual machines and time-shared policy for tasks. Each virtual machine will reserve one of the two processing units of the host, and while each tasks needs one processing unit to get executed, the policy algorithm will give each task a slice of the processing unit time until both tasks are executed.
Figure 4.5: Space Shared Policy for VMs and Time Shared for Tasks

Figure 4.6 presents a time-shared policy for virtual machines and space-shared policy for tasks. Each virtual machine gets a slice of the processing unit time. The first virtual machine get to hold the first processing unit of the host in order to execute the first task and the same thing goes for the second virtual machine. As a result, both first tasks of both virtual machines get to run simultaneously. The second task of both virtual machines will hold until the first task is executed for both VMs.

Figure 4.6: Time Shared Policy for VMs and Space Shared for Tasks

Figure 4.7 presents a time-shared policy for both virtual machines and tasks. The time of both processing units of the host will be shared simultaneously by the four tasks deployed on the two virtual machines.
In this thesis, space-shared scheduling policy has been applied for simulation experiments.
Chapter 5

Experimental Simulation and Performance Evaluation

For the purpose of evaluating the proposed algorithm, the GA algorithm was implemented using CloudSim. Furthermore, to evaluate the performance of the proposed GA algorithm, the obtained results have been compared with Exhaustive algorithm implemented by us to evaluate the GA’s degree of optimization, accuracy of the optimal solution and efficiency. In addition, the performance of the our algorithm was also estimated by applying it into several scenarios to estimate its performance and capabilities for further details, as discussed in next chapter.

Exhaustive method consist in taking all the relevant conditions into consideration and let the computer search until the results meet all the requirements. However, the algorithm consumes a lot of computer resources. If the conditions are too complex, the convergence speed to optimal result will be very slow. To solve this problem, we increase the level of computation for the exhaustive method in turn until the overhead limit exceeded error is showing. So the maximum computable task load on our limited hardware resources is by $3^{16}$.

5.1 Experimental Design

- Experimental goal: Comparing the number of calling fitness function, algorithm run time with Exhaustive(EX) algorithm, the error of Optimal Solution with respect to EX and GA, the minimum critical value
of iteration in GA to get the optimal solution.

Experimental object: Genetic algorithm for task scheduling in cloud computing

Experimental tool: CloudSim

Expected result: GA has good performance in finding optimum solution. And at the beginning EX is better or slightly better than GA.

5.1.1 Experimental Parameters

To evaluate the impact and performance of the proposed algorithm on the task scheduling problem, exhaustive algorithm result is taken as the performance index then the performance and feasibility of GA algorithm were determined by the error of its optimal solution compared to the optimal solution of exhaustive algorithm. We ran extensive experiments on simulation toolkit (CloudSim) using the simulation parameters set as follow.

1. Basic hardware and software configuration is showed in Table 5.1:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>Win10</td>
</tr>
<tr>
<td>CPU</td>
<td>intel core i5-6200U 4-core</td>
</tr>
<tr>
<td>RAM</td>
<td>8GB</td>
</tr>
<tr>
<td>HD</td>
<td>500GB</td>
</tr>
<tr>
<td>IDE</td>
<td>MyEclipse Kepler</td>
</tr>
<tr>
<td>Software Toolkit</td>
<td>CloudSim 3.0.3</td>
</tr>
</tbody>
</table>

2. Simulation (cloud environment) parameters are showing in Table 5.2

These parameters were used to identify the characteristics of the VMs and the task scheduling in the experiments. A workflow was created with different numbers of tasks to examine the objectives of implemented GA in comparison with Exhaustive algorithm: (1) Does the algorithm optimize the cloud task scheduling process? (2) Is the result of the algorithm the optimal solution? (3) The performance of the algorithm. (4) Run time with respect to the different heterogeneous environment characteristics. The parameters defined in Table 5.2 were used throughout the GA evaluation experiments.
Table 5.2: Simulation Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Tasks</td>
<td>5-15</td>
</tr>
<tr>
<td>Number of VMs</td>
<td>3</td>
</tr>
<tr>
<td>MIPS</td>
<td>500-1000</td>
</tr>
<tr>
<td>RAM</td>
<td>512</td>
</tr>
<tr>
<td>BW</td>
<td>1000</td>
</tr>
<tr>
<td>Number of Processors</td>
<td>4</td>
</tr>
<tr>
<td>Processor Speed</td>
<td>1000</td>
</tr>
<tr>
<td>VM Policy</td>
<td>Space Shared</td>
</tr>
</tbody>
</table>

3. GA Algorithm parameters are showing in Table 5.3

Table 5.3: Algorithm Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population Size</td>
<td>$10 \times \text{Task.size}$</td>
</tr>
<tr>
<td>Crossover Rate</td>
<td>0.85</td>
</tr>
<tr>
<td>Mutation Rate</td>
<td>0.3</td>
</tr>
<tr>
<td>Crossover</td>
<td>Single Point</td>
</tr>
<tr>
<td>Number of Iterations</td>
<td>100</td>
</tr>
<tr>
<td>Number of Executions</td>
<td>20</td>
</tr>
<tr>
<td>$r_1, r_2$</td>
<td>[0,1]</td>
</tr>
<tr>
<td>$\alpha, \beta$</td>
<td>0.5</td>
</tr>
<tr>
<td>Number of Elites</td>
<td>Task.size</td>
</tr>
</tbody>
</table>

The algorithm starts with $10 \times n$ random solutions, called initial population. And $n$ is the number of tasks. The $10n$ theory is based on the research in paper [28] by R. Storn. The single point crossover method was chosen in the GA crossover phase (Section 3.2). The mutation operator rate was defined as 0.3 in the mutation stage. And the elite number is $n$ which means the first $n$ ranked chromosome will be kept to next iteration. Furthermore, the degree of importance of each objective in the fitness function was defined as “$\alpha = 0.5$ and $\beta = 0.5$” for the makespan, execution cost respectively. The experiment is used in the evaluation with a different number of tasks (5–15), to enlarge the size of the workflow and evaluate the algorithm under these
different cases. The number of iterations for the GA algorithm was defined to 100 iterations to reach the optimal solution. The experiments were repeated 20 times for each case, then the average results were compared with Exhaustive algorithm. Five experiments were conducted based on the characteristic of the VMs and tasks, as in Table 5.2. The size of the tasks have been changed to examine the ability of the proposed algorithm in convergence speed to optimal solution for the small to large size of tasks compared with Exhaustive algorithm so that to estimate whether the proposed algorithm works or not. The characteristics of the task scheduling algorithm (GA) that were used in the experiments are summarized in Table 5.3.

Listing 6: Simulation Steps

```
Initialize CloudSim library
CloudSim.init(num_user, calendar, trace_flag);
Create Data Center
Data center data center0 = createData center(" Datacenter_0 ");
Create agent Broker, code is as following:
Data center Broker broker = createBroker();
Create VMs
//define VM parameters
Vm vm1 =newVm(vmid, brokerId, mips, pesNumber, ram, bw, size,vmm,
new CloudletSchedulerSpaceShared(),comcost[i]);
Create cloud tasks
//define Cloudlet parameters
Utilization Model utilization Model = new UtilizationModelFull();
Cloudlet cloudlet1 = new Cloudlet(id, length, pesNumber,
fileSize, outputSize, utilizationModel, utilizationModel,
utilizationModel);
cloudlet1.setUserId( brokerId );
Start simulation
CloudSim.startSimulation ();
Stop simulation and print result
CloudSim.stopSimulation();
printCloudletList(newList);
dataCenter0.printDebts();
```
5.1.2 Simulation Flow

Before starting the simulation, first, we need to create a data center then create virtual machines and cloud tasks in the data center, select appropriate scheduling strategies and set resource parameters, and finally register resource information with the agent center, so that users can use the resources of the data center for simulation. In the simulation resource allocation experiment, we add our own scheduling method by rewriting the bindCloudletToVm() method in the DataCenterBroker class. The details of simulation steps are as shown in the Listing 6.

Sequence diagram of simulation process is showing in Figure 5.1

5.2 Experimental Result and Performance Analysis

The results of the executed experiments for the experimental parameters setting above are reported as following:
Figure 5.2: Number of Calls to Fitness Function

(1) Number of calls to fitness function From Figure 5.2, we can see that the number of calls to fitness function by GA and EX increases with the number of tasks. When the number of tasks is less than 10, the number of calls to fitness function by EX is less than GA, but when the number of tasks is more than 10, the number of calls by EX is much larger than GA and shows an exponential growth trend. This also shows that EX algorithm can be directly used to find the best solution and always get the best result when dealing with small-scale cloud tasks. The overall performance of EX algorithm is better than GA or other evolutionary algorithms (EAs). But when dealing with large-scale tasks or searching for optimal schemes in a large number of task scheduling schemes, the advantages of GA algorithm are obvious.

(2) Fitness value discrepancy As all we know, Exhaustive algorithm can always find the best solution if it exists. So in experiment 2, Exhaustive algorithm finds the optimum solution and we compare GA’s result with it to calculate the discrepancy. As the result showing in Figure 5.3, when it’s dealing with small-scale cloud tasks, the accuracy of GA algorithm in finding optimum solutions can reach almost 99.99%, compared with the optimal
solution provided by EX. However, when the cloud tasks are more than 10, the discrepancy between GA’s result and optimal result is growing up, but the two values are still close. In all, the proposed GA algorithm is working properly while fulfilling requirements.

(3) Run time comparison In experiment 3, we continue to compare the run-time results of GA and EX algorithms, however, the result will be affected by the hardware environment. From Figure 5.4, we can see that EX runs faster than GA when dealing with small-scale cloud tasks, but EX becomes more time consuming when the tasks number goes bigger while for GA it’s just slightly increased. On the other hand, this experiment also confirms the results of Experiment 1 from another angle.

(4) Optimizing the iterations From the results of experiment 2, we can see that the accuracy of GA algorithm results slightly decreases with to the increase of the number of tasks. This is because the convergence speed of GA algorithm to find the optimal result is decreasing. The results are influenced by the size of the initial population and the number of iterations of the algorithm. But in experiment 2, the number of iterations of the algorithm
Figure 5.4: Run Time Comparison

Figure 5.5: Optimum the Iterations
has been set up as a constant number, 100, while the population size is 10 times as much as the number of tasks. In fact, the setup of iteration times is a waste of performance for small tasks, but yields inadequate performance for large tasks. Therefore, the discrepancy of the results shows an increasing trend. So in experiment 4, we use the optimal solution found by EX algorithm in Experiment 2 as the control value and try to find the correlation between the optimum number of iterations and the number of tasks as well as the size of the initial population by adjusting the number of iterations of each task level (5, 7, 9, 11, ...). Then we hope to control the error between GA results and the optimal results by applying the law found in experiment 4. During the experiment, the other parameters are kept unchanged while the number of iterations for each task level are constantly reduced or increased until GA’s result is close enough to control value. It is worth mentioning that the GA algorithm is executed 20 times for each experimental scenario and when there are more than 10 times of the same results as control value, then we consider that the experimental conditions are successful. The result is showing in Figure 5.5, in order to ensure the optimum of GA’s result, it is necessary that the number of iterations of the algorithm and the initial size of the population should be increased as the amount of tasks increases and the law is showing in the picture.

(5) Comparison of optimum iterations Finally, based on the results obtained in Experiment 4, we performed the second experiment again, and we obtained the comparison of the experimental error results as shown in the Figure 5.6. The optimum iterations of the algorithm effectively reduces the error of the GA optimal result, especially for large number of tasks and keeps the total discrepancy in a stable or slowly increasing level.

In conclusion, the overall performance of GA algorithm is worthy of affirmation. It can be used for optimization problems task scheduling in cloud computing, but the convergence speed to optimal solution slows down with the increase of global solution domain. If we want to maintain the accuracy of its optimization ability, we need to increase the number of iterations and the initial population size with the increasing of task numbers.
Figure 5.6: Comparison of Optimum Iterations
Chapter 6

Exploitation of GA in Task Scheduling

According to the characteristics of cloud tasks and the needs of users, we created some scenarios to test the performance of the implemented GA in task scheduling. The cloud tasks have been basically classified into two types:

1. Cloud tasks divided equally into several sub-tasks.
2. Continuous and random-size cloud tasks.

Therefore, we have set up two categories of scenarios and each cloud task scenario will be executed by 3 group of computing resources which are different configurations as showing Table 6.1. After that we analyze the best performed configuration.

<table>
<thead>
<tr>
<th>No.</th>
<th>VMs</th>
<th>MIPS</th>
<th>Schedule Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case A</td>
<td>3</td>
<td>1000</td>
<td>Space-shared</td>
</tr>
<tr>
<td>Case B</td>
<td>4</td>
<td>2<em>1000,2</em>500</td>
<td>Space-shared</td>
</tr>
<tr>
<td>Case C</td>
<td>6</td>
<td>500</td>
<td>Space-shared</td>
</tr>
</tbody>
</table>

Table 6.1: Heterogeneous VMs Configuration

Continuous and random-size cloud tasks

Scenario A.1 User A has 15 cloud tasks and the size of each task varies from 500 to 30,000 and the user is sensitive to the execution time of the overall task, so the weight factors of the fitness function are set to $\alpha = 0.2$
and $\beta = 0.8$. Then applying the scenario to configuration caseA, caseB, caseC to execute task scheduling to get the overall completion time and estimated cost of cloud tasks. Each case is executed 20 times and we take the average value eventually, analyze the best running configuration. The result is showing in Table 6.2.

<table>
<thead>
<tr>
<th>VMs Configuration</th>
<th>Execution Time</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case A</td>
<td>76.89</td>
<td>813.91</td>
</tr>
<tr>
<td>Case B</td>
<td>59.55</td>
<td>826.52</td>
</tr>
<tr>
<td>Case C</td>
<td>72.76</td>
<td>823.82</td>
</tr>
</tbody>
</table>

Table 6.2: Time-optimized Performance Comparison

**Scenario A.2** User B has 15 cloud tasks and the size of each task varies from 500 to 30,000 and the user is sensitive to the monetary cost of the overall task, so the weight factors of the fitness function are set to $\alpha = 0.8$ and $\beta = 0.2$. Then applying the scenario to configuration caseA, caseB, caseC to execute task scheduling to get the overall completion time and estimated cost of cloud tasks. Each case is executed 20 times and we take the average value eventually, analyze the best running configuration. The result is showing in Table 6.3.

<table>
<thead>
<tr>
<th>VMs Configuration</th>
<th>Execution Time</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case A</td>
<td>77.77</td>
<td>813.92</td>
</tr>
<tr>
<td>Case B</td>
<td>81.41</td>
<td>817.46</td>
</tr>
<tr>
<td>Case C</td>
<td>78.63</td>
<td>824.48</td>
</tr>
</tbody>
</table>

Table 6.3: Cost-optimized Performance Comparison

In Scenario A.1, the results shown in Table 6.2 show that if the cloud task is allocated to configuration Case B, the shortest task execution time but also the most expensive can be obtained by the schema. The schema of lowest cost is to schedule the cloud task to configuration Case A but with the longest execution time, while the result of configuration Case C is between A and B. But when we compare the Case A and B, we can get that compared with A, the total execution time of B is reduced by 23%, but the cost is only
increased by 1.5%. In addition, the user is sensitive to the execution time of the overall task, so the best resource allocation scheme for the user is B.

In terms of scenario A.2, the result is much simpler to analyze compared to the previous scenario. As the result is showing in Table 6.3. Because users tend to spend the least on executing the cloud task, the optimal resource allocation scheme is A and the execution time is the shortest compared with the other two schemes.

Summarizing the results of A.1 and A.2, the GA algorithm proposed in this paper has significant effect on the optimization of the total execution time by adjusting the weight factor, but has little effect on the cost.

Sub-tasks with same size

Scenario B.1 User A has a set of cloud tasks with a total size of 22500 instructions equally divided into:
1. 15 sub-tasks and each one has 1500 instructions.
2. 30 sub-tasks and each one has 750 instructions.
3. 60 sub-tasks and each one has 375 instructions.

Scenario B.2 User B has a set of cloud tasks with a total size of 45000 instructions equally divided into:
1. 15 sub-tasks and each one has 3000 instructions.
2. 30 sub-tasks and each one has 1500 instructions.
3. 60 sub-tasks and each one has 750 instructions.

Scenario B.3 User C has a set of cloud tasks with a total size of 90000 instructions equally divided into:
1. 15 sub-tasks and each one has 6000 instructions.
2. 30 sub-tasks and each one has 3000 instructions.
3. 60 sub-tasks and each one has 1500 instructions.

Scenario B.4 User D has a set of cloud tasks with a total size of 180000 instructions equally divided into:
1. 15 sub-tasks and each one has 12000 instructions.
2. 30 sub-tasks and each one has 6000 instructions.
3. 60 sub-tasks and each one has 3000 instructions.
Scenario B.5  User E has a set of cloud tasks with a total size of 360000 instructions equally divided into:

1. 15 sub-tasks and each one has 24000 instructions.
2. 30 sub-tasks and each one has 12000 instructions.
3. 60 sub-tasks and each one has 6000 instructions.

Moreover, these users have same sensitivity to the monetary cost and execution time of the overall task, so the weight factors of the fitness function are set to \( \alpha = 0.5 \) and \( \beta = 0.5 \). Then applying the scenarios to configuration caseA, caseB, caseC to execute task scheduling to get the overall completion time and estimated cost of cloud tasks. Each case is executed 20 times and we take the average value eventually, analyze the best running configuration. The results are showing in Table 6.4:

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Task Nums</th>
<th>Task Length</th>
<th>VMs Configuration</th>
<th>Cost</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td>1500</td>
<td>A</td>
<td>192.14</td>
<td>13.07</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B</td>
<td>195.11</td>
<td>14.16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>198.63</td>
<td>17.20</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>750</td>
<td>A</td>
<td>195.33</td>
<td>13.05</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B</td>
<td>197.15</td>
<td>13.75</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>203.36</td>
<td>15.85</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>375</td>
<td>A</td>
<td>207.43</td>
<td>10.67</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B</td>
<td>203.88</td>
<td>13.97</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>218.10</td>
<td>14.65</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>3000</td>
<td>A</td>
<td>378.34</td>
<td>25.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B</td>
<td>381.73</td>
<td>28.70</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>385.05</td>
<td>32.80</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1500</td>
<td>A</td>
<td>382.19</td>
<td>24.17</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B</td>
<td>385.02</td>
<td>27.70</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>390.33</td>
<td>32.65</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>750</td>
<td>A</td>
<td>394.62</td>
<td>20.69</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B</td>
<td>391.38</td>
<td>25.82</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>403.42</td>
<td>29.95</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>6000</td>
<td>A</td>
<td>752.50</td>
<td>55.90</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B</td>
<td>755.75</td>
<td>56.20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>758.44</td>
<td>68.20</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>3000</td>
<td>A</td>
<td>756.68</td>
<td>49.90</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B</td>
<td>758.88</td>
<td>54.40</td>
</tr>
<tr>
<td>Scenarios</td>
<td>Task Nums</td>
<td>Task Length</td>
<td>VMs Configuration</td>
<td>Cost</td>
<td>Time</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
<td>-------------</td>
<td>-------------------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>4</td>
<td>60</td>
<td>1500</td>
<td>C</td>
<td>763.28</td>
<td>64.30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A</td>
<td>769.22</td>
<td>42.40</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B</td>
<td>765.56</td>
<td>51.47</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>779.56</td>
<td>53.95</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>12000</td>
<td>A</td>
<td>1498.84</td>
<td>108.10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B</td>
<td>1503.08</td>
<td>116.20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>1505.66</td>
<td>145.30</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>6000</td>
<td>A</td>
<td>1503.90</td>
<td>94.60</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B</td>
<td>1505.48</td>
<td>112.90</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>1510.72</td>
<td>133.30</td>
</tr>
<tr>
<td>4</td>
<td>60</td>
<td>3000</td>
<td>A</td>
<td>1514.13</td>
<td>86.65</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B</td>
<td>1513.62</td>
<td>106.90</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>1525.90</td>
<td>113.20</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>24000</td>
<td>A</td>
<td>2993.17</td>
<td>225.70</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B</td>
<td>2996.31</td>
<td>227.10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>2999.55</td>
<td>267.70</td>
</tr>
<tr>
<td>5</td>
<td>30</td>
<td>12000</td>
<td>A</td>
<td>2997.79</td>
<td>183.70</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B</td>
<td>3001.13</td>
<td>216.10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>3004.28</td>
<td>256.90</td>
</tr>
<tr>
<td>5</td>
<td>60</td>
<td>6000</td>
<td>A</td>
<td>3009.56</td>
<td>172.90</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B</td>
<td>3007.09</td>
<td>214.30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>3017.81</td>
<td>225.70</td>
</tr>
</tbody>
</table>

Table 6.4: Performance Comparison in Heterogeneous Cloud Environment

Comparing the optimization of cloud task execution time, figures are shown in Figure 6.1 to 6.3. Cloud tasks in five scenarios are divided into 15, 30 and 60 sub-tasks in turn. For instance, Cloud tasks with a total size of 180,000 are averagely assigned to 15 subtasks, and each subtask is 12000. Then sub-tasks are executed on resources configuration A, B and C, separately. We can see that the task execution time of 60 sub-tasks is the shortest compared with 30 sub-tasks and 15 sub-tasks, that is, the number of sub-tasks is inversely proportional to the total task execution time. The more sub-tasks are, the shorter the total task execution time is when the total task size is fixed. It is also true for resources configuration B and C,
but configuration B is not as obvious as A and C.

Furthermore, we compare the average time for executing 15, 30, 60 sub-tasks of each scenario by using 3 different resource configurations namely Case A, Case B, Case C. The result is displaying in Figure 6.4. We can clearly see that Configuration A shows the best performance and Case C gets the worst result among them. With the increasing of the size of total tasks, the performance difference of these 3 different configurations is becoming bigger. Here, the performance denotes the length of task execution time.

In terms of monetary cost performance, we compare its performance optimization in the same way that we compare time performance. We can see that the cost and execution time are inversely proportional. For example, the data is shown in Figure 6.5 to 6.7. In fact, the cost increases with the decrease of execution time, but the change is not dramatically. In other words, when execution time is reduced but the cost barely changed. It turns out that computing resources (VMs) perform better when total tasks of fixed size are allocated to a larger number of subtasks than when allocated to fewer subtasks, but each task carries more instructions. The data also show that VMs with configuration A have always best performance comparing to other two configurations.

In summary, according to the result above, we can say that the best performance are achieved with the more numbers of sub-tasks comparing
Figure 6.2: Time Performance Comparison B

Figure 6.3: Time Performance Comparison C
Figure 6.4: Comprehensive Time Performance Comparison

Figure 6.5: Cost Performance Comparison A
Figure 6.6: Cost Performance Comparison B

Figure 6.7: Cost Performance Comparison C
of more number of instructions contained by each sub-task. For example, dividing the total cloud task into 60 sub-tasks with 6,000 instructions per sub-task and assigning them to resources of configuration A, the task execution time is shorter than that of 15 sub-tasks with 24,000 instructions per sub-task. While the cost of it is a little bit higher than 15 sub-tasks but not that significant difference as the execution time changes. In addition, in general, assigning cloud tasks according to configuration A always yields the shortest task execution time and reasonable monetary cost compared to the other two configurations.
Chapter 7

Conclusion and Future Work

In this thesis, we proposed to use GA to solve the optimization problem of cloud task scheduling and heterogeneous resource allocation. In our algorithm, we proposed a fitness function which directly affects the evolutionary direction of population by setting the a time and a cost weight factor. Users can choose the preference solution by adjusting these factors namely solutions of task scheduling are time-optimal, cost-optimal or time-cost-comprehensive. In addition, we introduced an individual similarity checking function which selects chromosomes with small individual similarity according to the function when generating the initial population. In this way, we can ensure that there is a big difference among individuals in the population. When the population size is large, it can be distributed in the solution space in a large range and evenly, thus reducing the probability of local optimization and improving the ability of global search. Then we validated the effectiveness and high performance of GA algorithm in cloud task scheduling comparing to the results of EX. The experimental results showed that the GA can achieve reasonable result of task scheduling in cloud computing and produce optimum task scheduling solution. Finally, we created few heterogenous cloud task scenarios and heterogenous resources (VMs) configurations to exploit the implemented algorithm. We summarized how to select resource allocation according to the characteristics of cloud tasks and how to schedule cloud tasks can make the algorithm achieve better performance. At the same time, the algorithm also has many shortcomings, such as, its performance is not ideal or even worse than the EX for small numbers of cloud task scheduling optimization issue. On the other hand, according to experimental analysis, we can also see that the solution obtained by GA is not always the optimal
solution but is infinitely close to the optimal solution. At the later stage, its convergence speed gradually also slows down as a known disadvantage of GA, which might result in a larger error in the final solution. However, this error can be improved by optimizing the size of the initial population, increasing the number of iterations of the whole population.

On the other hand, by using the proposed algorithm to solve optimal task scheduling issue in heterogeneous cloud environment, which belong to the intractable combinatorial optimization class, it is possible to claim the contribution in this thesis as can be used to solve some other similar combinatorial optimizations problems, which are NP-hard. The research done in this thesis showed that GA can be used for efficient optimal allocation of limited resources, and other complex and conflicting situations, which are related to combinatorial optimization genre, and hence, it is possible to solve real-life large scale problems. For example, GA applied in the military case shows that GA can support military and other defence organizations for optimal assignments of soldiers to operations, rescheduling resources, and optimal planning of military or other defence activities.

Due to broad scope and multiple aspects of task scheduling optimization problem, it is not possible to answer each and every aspect of the considered problems in this thesis. These are important problems to investigate, and this section provides possible future directions for the research.

For future work, it is worth to investigate the effect of other parameters such as VMs, datacenters, memory, bandwidth for network and storage in cloud environments. For more details on that, the work also can be extended to more than one data center in a wilder heterogeneous environment. Furthermore, the distribution of the workflow application can be extended into two levels: when workflow tasks reach the service broker and when the workflow tasks are distributed to the available VMs of each Data Center (DC) based on the size of the tasks and the speed of each VM. The justification can be verified over real-time cloud or even in a real physical environment. In addition, the work can be improved through using dynamic workflow that allows more flexibility for the users to change the characteristics of the workflow tasks during the runtime.

Furthermore, another direction of the future work could be optimizing the way to solve optimization problem. For each of the optimization problems, considered in this thesis, the result is a set of Pareto-optimal solutions, which is just one aspect of multi-objective optimization. The obtained
Pareto-optimal sets of the large scale assignment, planning and rescheduling problems usually consist of many solutions, which make it harder for decision maker to identify the best solution for implementation. Ranking the Pareto-optimal set is another interesting aspect of the problem. There can be situations when GAs can take longer to solve large-sized instances of the considered problems effectively. In order to find optimal solutions quickly, parallel GAs can be developed and applied. In order to suggest best algorithm for the problems, the results of GAs can be compared with other search techniques like local search, constraint programming, simulated annealing, tabu search, particle swarm optimization and combinations of these search heuristics.
Bibliography


