2019 Master's Thesis

Real-time Load Balancing based on Stackelberg Game and Reinforcement Learning in Cloudlet Network

A Thesis Submitted to the Department of Computer Science and Communications Engineering, the Graduate School of Fundamental Science and Engineering of Waseda University in Partial Fulfillment of the Requirements for the Degree of Master of Engineering

Zhiqiang Gu
5118F036-0

Supervisor: Prof. Yoshiaki Fukazawa
Research Guidance: Research on Software Development Engineering

Fukazawa Laboratory

Submission Date: January 29, 2020

Abstract

In mobile cloud computing, due to the mobility of users and uncertainty of task type, the load on each cloudlet is changing over time. And because of the limited resources on the cloudlets, when a cloudlet accepts too many user tasks in a short time, the tasks that need to be executed may exceed the load of the cloudlet. At this time, some tasks may not get the execution resources and need to wait for other tasks before being executed. If these tasks wait too long, it is meaningless for the users to offload the tasks to the cloudlet. If the cloudlets are connected with each other to form a cloudlet network, based on geographical location and each cloudlet can communicate and transfer data with each other. Then the cloudlets with a high load can transfer excessive tasks to the cloudlets with a low load. In this paper, we propose a load balancing framework in cloudlet network based on the Stackelberg game to solve the problem of load unbalancing. The multi-leader multi-follower Stackelberg game and reinforcement learning are combined to get an optimal task transferring strategy for overall cloudlets in the incomplete information and noncooperative game. Experimental results show that our proposed method can approach the optimal solution with fewer iterations. Compared with the agent-based centralized management method, our proposed method can find a good solution in a shorter time. In addition, the framework we proposed has better scalability and can meet the different requirements of different tasks.

Keywords: Mobile cloud computing, Load balancing, Stackelberg game, Reinforcement learning, Cloudlet

Content

List of Figures

List of Tables

Chapter 1

Introduction

With the advent of the mobile Internet era, mobile devices are exploding every year. From autonomous cars to humanoid robots and from intelligent personal assistants to smart home devices, the world around us is undergoing a fundamental change, transforming the way we live, work, and play [9]. It is estimated that by 2025, there will be 34.2 billion network connected devices worldwide, of which 21.5 billion will be IoT devices, as shown in Figure 1.1 [10]. The proportion of IoT devices is increasing year by year.

With the rapid emergence and sharp growth of Internet of Things (IoT), enormous computation and communication requirements are induced, exceeding the capacity of current data centers and mobile networks, as shown in Figure 1.2. IoT has lots of features. Such as mobile, numerous, power limited, low computational capability and so on.

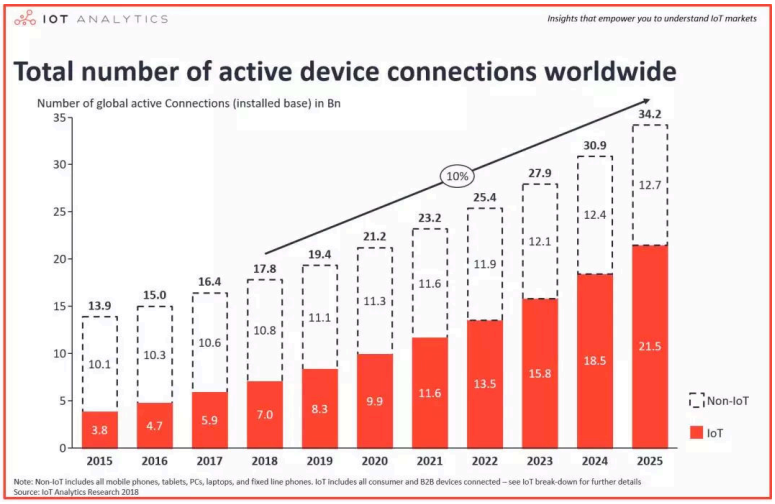

Figure 1.1. Total number of active device connection worldwide

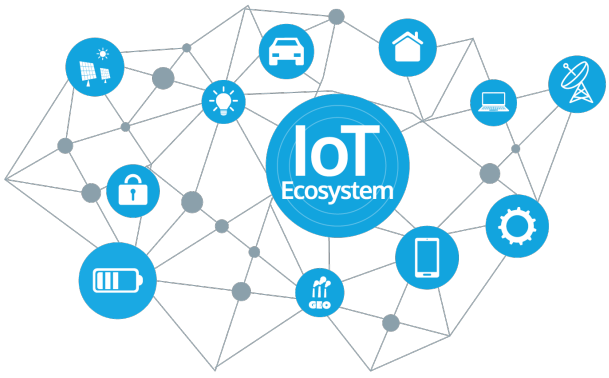

Figure 1.2. Internet of Things (IoT)

With the increasing number of mobile devices, the demand for various services is increasing, such as face recognition and speech recognition software and online games [1]. However, while the resource demands of newly developed applications continue growing, the computing capacity of mobile devices remains limited due to their portable sizes and small battery capacity. Cloud computing is envisioned as a promising method to address such a challenge.

A traditional solution to overcome the resource poverty of mobile devices is to offload the complex tasks to the clouds with rich computing resource. A mobile device can reduce its workload and prolong its battery life by offloading its computation-intensive tasks to a remote cloud for execution [11], [12], as shown in Figure 1.3.

Figure 1.3. Cloud Computing System

Many cloud service providers like Amazon, Google and Microsoft have built many cloud computing centers and provided the users with various services for variety of computational tasks. The huge storage and computational capability at the cloud servers of cloud computing centers enables them to quickly execute the computational tasks offloaded by the mobile devices. Many of these tasks if performed locally on the mobile devices may take hours to be executed, which not only occupies local computational resources, but also decreases the mobile devices' battery life.

With the advent of the mobile Internet era and the increase in mobile devices, there is more and more researches [17]-[21] on mobile cloud computing. In addition to the lack of computing resources, users are constantly moving. Because the mobile device is connected to the cloud through a wireless network, this connection is not stable and will be disconnected at any time due to the user's movement, resulting in interruption of data transmission and getting no results. So user mobility is also an important issue in mobile cloud computing.

In order to solve the problems of mobility and resource limitation, researches on resource allocation in mobile cloud computing are becoming more and more [13] – [16], [18]. Mobile cloud computing is that mobile devices take use of the resources of remote cloud to execute complex tasks in order to have a shorter

execution time and decrease the energy consumption of mobile devices. Because there is a long distance between the mobile device and the remote cloud, the user may have a high communication latency. Cloudlet is a solution to overcome the problem of high communication latency between remote cloud and mobile devices, which is deployed in different areas and close to users.

In recent years, researches on cloudlets are becoming more and more [13]-[16]. Cloudlet is a small cloud distributed in different areas and is much closer to users compared to remote cloud. Each cloudlet covers one area, provides relatively powerful computing power to perform tasks or provide some services to users. However, the cloudlet has limited coverage range, because the user communicates to cloudlet with the wireless network, as shown in Figure 1.4.

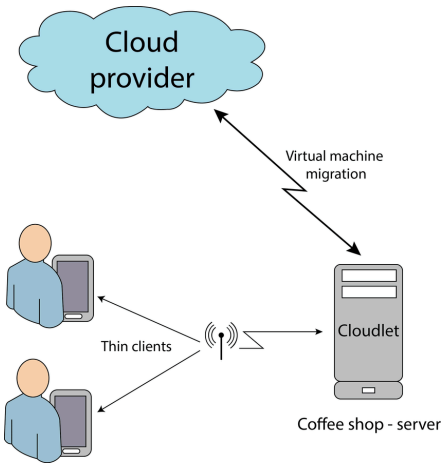

Figure 1.4. Cloudlet-based Mobile Cloud Computing

Cloudlet is a reliable solution to solve the problems of resource limitations of mobile devices and high communication latency [18]. A cloudlet is a trusted, resource-rich, Internet-connected computer or a cluster of computers, which can be utilized by mobile devices via a high-speed wireless local area network (WLAN) [2].

Here are two examples to show the broad promising application of cloudlet. Face recognition and real-time video decoding are both widely used complex tasks in our smart phones. Executing these tasks in cloudlets instead of mobile devices can have less execution time and less power consumption, as shown in Figure 1.5. And executing real-time video decoding task in cloudlets instead of remote cloud can have less communication time and provide the users with real-time gaming experience, as shown in Figure 1.6.

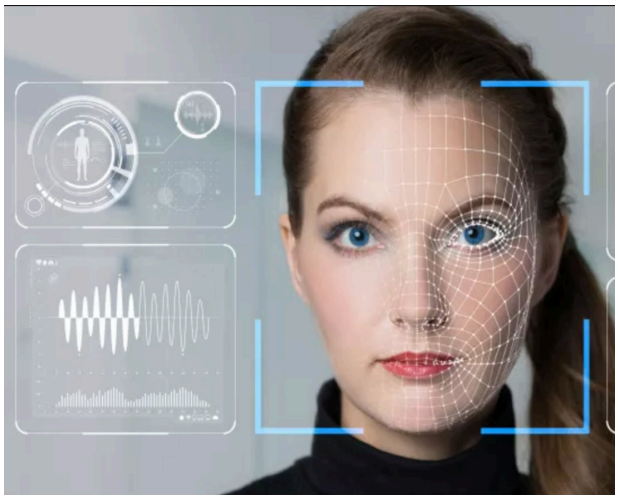

Figure 1.5. Face recognition application

Figure 1.6. Real-time online video game

Due to the mobility of users and uncertainty of task type, the load on each cloudlet is changing over time. And because of the limited resources on the cloudlets, when a cloudlet accepts too many user tasks in a short time, the tasks that need to be executed may exceed the load of the cloudlet. At this time, some tasks may not get the execution resource and need to be waited for other tasks before being executed. If these tasks wait too long, it is meaningless for the users to offload the tasks to the cloudlet. The tasks executed locally may have less execution time (includes waiting time).

There are some researches about the load balancing in cloudlet network [24] – [28]. They assume that cloudlets are connected to each other and can communicate and transfer data. They use a central cloudlet manager to gather information of cloudlets and tasks and based on the gathered information to calculate the optimal task transferring strategy with the minimal task response time. However, the running time of their methods will increase dramatically as the number of cloudlets increases. We think that the objective of minimal task response time is not sufficient for various types of tasks.

We assume that cloudlets are interconnected based on geographic location to form a big cloudlet network, as shown in Figure 4.1. The cloudlets with a high load can transfer excessive tasks to the cloudlets with a low load. The former cloudlet is defined as task server, the latter is defined as resource server.

Considering the resource scheduling from the perspective of the supply and demand relationship between the resource server and the task server can improve the resource utilization of the overall cloudlet network and decrease the task execution time and computing cost, and ultimately improve the user's task offloading experience.

In this paper, in order to avoid using the central cloudlet manager and decrease the communication volume and algorithm running time, we will use the incomplete information and noncooperative game theory to solve the problem of load unbalancing in cloudlet network. Stackelberg game is one of the most famous and useful game theories. In our scenario, the resource server and task server are modeled as a multi-leader and multi-follower Stackelberg game in cloudlet network, and then the existence of the task server Nash equilibrium point under the condition of resource server price determination is proved. Finally, the optimal pricing of the resource server and the optimal task transferring strategy of task server are solved by the proposed method.

The contributions of this paper are described as follows:

- A task load balancing framework based on the Stackelberg Game in cloudlet network is proposed to solve the problem of load imbalance. The load between the idle cloudlets and the high-load cloudlets is transferred to solve the problem of task queuing caused by excessive load on the cloudlets.
- The multi-leader multi-follower Stackelberg game and reinforcement learning are combined to get an optimal task transferring strategy for overall cloudlets in the incomplete information and noncooperative game. The players cannot know the transfer strategies of all other players, and it costs some time and energies to get that information. In our method, players adjust the selection probability only based on their own rewards.
- Based on Stackelberg game, the followers learn the reward each time through reinforcement learning and changes the selection probability of the leaders.
- In calculating the reward of a task, the type of the task is taken into account, such as time-sensitive and computation-sensitive. The tasks that are different in types may have different time sensitivity and objective. Compared with computation-sensitive tasks, time-sensitive tasks are more concerned about the execution time. In contrast, computation-sensitive tasks are more concerned about the fee paid to the cloudlets for task execution.

Chapter 2

Related work

In this chapter, we will introduce some researches in mobile cloud computing (MCC) and some researches about resource allocation in Cloudlet. We will also introduce some researches about Stackelberg game and reinforcement learning, which will be used in our proposed method.

2.1 Mobile Cloud Computing

In [19], Z. Kuang et al, propose an agent-based MCC framework to enable the device to receive offloading results faster by making offloading decision on the agent. Moreover, to get an offloading strategy, they formulate the problem of maximizing energy savings among multiple users under the completion time and bandwidth constraints, and they propose a Dynamic Programming After Filtering algorithm to solve the optimization problem. However, this paper only considers the remote cloud resource, so the users may have a higher communication delay.

In [18], W. Li et al, provide an overview of the mechanisms and open issues for mobility-augmented service provisioning in MCC. They introduce three key mechanisms with respect to mobility augmentation, heterogeneous network convergence and mobile service provisioning. Moreover, they discuss the open challenges to reveal the future direction of MCC. This paper explores a key factor in mobile cloud computing: mobility, and points out some challenges and issues that can guide some research directions. Although this paper presents some feasible solutions to mobility, they do not conduct the experiment to verify their proposal.

In [17], by using stochastic geometry, H. Lee et al, analyze the outage probability of task offloading in the MCC system with only remote cloud servers and that in the heterogeneous MCC with both remote cloud servers and cloudlets. The analysis provides useful information, i.e., how the varying system parameters affect the outage probability. They show that the use of cloudlets is a promising solution to overcome this limitation. However, a tradeoff exists in using cloudlets due to their deployment and operation costs. Thus, to address this tradeoff, they also study the optimal cloudlet deployment to maximize the cloud service provider’s profit while guaranteeing maximum outage probability requirements. This paper proves the usefulness of cloudlet and studies how to deploy the cloudlets in a region.

In [20], S. Ahn et al, newly model computation offloading competition when multiple clients compete with each other so as to reduce energy cost and

improve computational performance. They design an energy-oriented task scheduling scheme, which aims to maximize the welfare of clients in terms of energy efficiency. Under this proposed job scheduling, as a joint consideration of the destination and client sides, competition behavior among multiple clients for optimal computation offloading is modeled and analyzed as a non-cooperative game by considering a trade-off between different types of destinations. Based on this game-theoretical analysis, they propose a novel energy-oriented weight assignment scheme in the mobile terminal side to maximize mobile terminal energy efficiency.

In [21], C. Tang et al, model the task scheduling problem at the end-user mobile device as an energy consumption optimization problem, while taking into account task dependency, data transmission and other constraint conditions such as task deadline and cost. They further present several heuristic algorithms to solve it. A series of simulation experiments are conducted to evaluate the performance of the algorithms and the results show that their proposed algorithms outperform the state-of-the-art algorithms in energy efficiency as well as response time.

In [29], H. Cao et al, investigate the problem of multiuser computation offloading for cloudlet-based mobile cloud computing in a multichannel wireless contention environment. The studied system is fully distributed so that each mobile device user can make the offloading decisions based only on its individual information, and without information exchange. They first formulate this multiuser computation offloading decision making problem as a noncooperative game. After analyzing the structural property of the formulated game, they show that it is an exact potential game, and has at least one pure-strategy Nash equilibrium point (NEP). To achieve the NEPs in a fully distributed environment, they propose a fully distributed computation offloading (FDCO) algorithm based on machine learning technology. They only emphasize the performance and effectiveness of their method, do not mention that the strategies they get are good or bad.

2.2 Resource Allocation in Cloudlet

In [13], X. Ma et al, research how to optimize the processing delay and energy consumption of computing tasks. This paper proves that there is a Nash equilibrium point, that is, all tasks are offloaded to the most reasonable cloudlets, and the processing delay and energy consumption of all tasks are minimal, and therefore no task needs to be moved. This paper does not take into account the user's mobility, that is, when a user moves from the coverage area of one cloudlet to the coverage area of another cloudlet.

In [14], Md. Whaiduzzaman et al, point that the limited resources of cloudlets can become heavily loaded during peak utilization, so available computational capacity decreases per user and at times mobile devices find no

execution time benefit for using the cloudlet. They propose a Mobile Device based Cloudlet Resource Enhancement (MobiCoRE) method, which can ensure that the mobile devices always have time benefit for its tasks.

In [15], M. Jia et al, study cloudlet placement and mobile user allocation to the cloudlets in a wireless metropolitan area network (WMAN). They devise an algorithm for the problem, which enables the placement of the cloudlets at user dense regions of the WMAN, and assigns mobile users to the placed cloudlets while balancing their workload. They also conduct experiments through simulation. The simulation results indicate that the performance of the proposed algorithm is very promising.

In [16], S. Rashidi et al, point that to accommodate to exponential growth of requests, user requests should be distributed to different cloudlets according to the latest network and server status. Therefore, finding the best place to offload is vital to both functionality and performance of the system. They propose an adaptive neuro-fuzzy inference system to achieve the accurate and timely parameters of network and servers' status effectively.

2.3 Load Balancing in Cloudlet Network

The load balancing problem occurs in task offloading where a power-limited cloudlet needs help from other ones [24]. [25] points out in HetNet, User devices tend to connect to the high-power cloudlet, leading to the overloading issue.

In [26], D. Yao et al, establish a five-tuple characterized task model to capture the response time of offloaded tasks and formulate the task allocation problem as an integer linear problem (ILP) under certain conditions. Furthermore, they propose a two-step appointment-driven strategy to solve this problem with minimal task response time. Specifically, a modified genetic algorithm (GA) is adopted to coordinate the load of the nodes.

In [27], V. Chamola et al, consider a network of connected cloudlets which provide service to the mobile devices in a given area. They address the issue of task assignment in such a scenario (i.e. which cloudlet serves which mobile device) aimed towards improving the quality of service experienced by the mobile devices in terms of minimizing the latency. They use a central cloudlet manager collect information and make load balancing strategy for cloudlet network, which may have a large communication volume for the manager and cost a large amount of time to get the strategy, when the number of cloudlets is huge.

In [28], M. Jia et al, investigate how to balance the workload between multiple cloudlets in a network to optimize mobile application performance. They first introduce a system model to capture the response times of offloaded tasks, and formulate a novel optimization problem, that is to find an optimal redirection of tasks between cloudlets such that the maximum of the average response time of tasks at cloudlets is minimized. They then propose a fast,

scalable algorithm for the problem. In this paper, they minimize the average response time, which is important for time-sensitive tasks. However, for computation-sensitive tasks, they care more about the fee paid for cloudlet provider. It is not sufficient to pursue only the minimum response time.

2.4 Resource Allocation with Stackelberg Game

Stackelberg games [8] consist of a hierarchical structure with a leader and followers. In a Stackelberg game, the leader first sets a price which is charged to the followers, then the followers react to the charged price and compete with each other. The entire Stackelberg game process consists of two phases: the leader decision phase and the follower decision phase and three elements: player, strategy and payoff.

In [22], B. Liu et al, research the resource allocation problem in threat defense for the resource-constrained IoT system, and propose a Stackelberg dynamic game model to get the optimal allocated resources for both the defender and attackers. The proposed Stackelberg dynamic game model is composed by one defender and many attackers. Given the objective functions of the defender and attackers, they analyze both the open-loop Nash equilibrium and feedback Nash equilibrium for the defender and attackers. Then both the defender and attackers can control their available resources based on the Nash equilibrium solutions of the dynamic game. Numerical simulation results show that correctness and effeteness of the proposed model. The Stackelberg game used by them is a tradition and simple game with a leader and many followers.

In [3], the purpose of this paper is to improve the efficiency of resource allocation. J. Wang et al, can get the resource allocation quickly with multi-leader multi-follower Stackelberg game and Q-learning. However, in the scenario of this paper, the reward of a player is not affected by the other player. Because there is no resource competition, a player's strategy will not be affected by other players' strategies. As long as they find the best strategy for each player, they can find the best strategy for the overall players. Because a player's behavior can correspond to a stable reward, this paper can use Q-learning to record behavior and state.

In [4], due to limited computing resource, S. Guo et al, research how to improve efficiency of resource allocation is a challenge. In this paper, they propose a hierarchical architecture in Smart Home with mobile edge computing, providing low-latency services and promoting edge process for smart devices. Based on that, a Stackelberg Game is designed in order to allocate computing resource to devices efficiently. Then, one-to-many matching is established to handle resource allocation problems. The game in this paper is an original Stackelberg game, which consists of one leader, many followers. A leader can easily know the strategies and rewards of all followers, making it easy to get the optimal strategy. Although there is resource competition, each time the same devices compete for the same resource. In our scenario, each time different

players compete for different resources.

In [5], considering the competing characteristics of multi-tenant environments in cloud computing, W. Wei et al, propose a cloud resource allocation model based on an imperfect information Stackelberg game (CSAM-IISG) using a hidden Markov model (HMM) in a cloud computing environment. Based on the unit prices of different types of resources, a resource allocation model is proposed to guarantee optimal gains for the infrastructure supplier. The players use other competitors' historical information to predict their bids, so the players need to communicate and transfer information with other. Although players only need to know the information of the surrounding competitors, it takes some time and energies to collect information and calculate bids. The objective of this paper is maximizing the profits of service providers (leader), which may be unfair to the followers.

2.5 Reinforcement Learning

Reinforcement learning (RL) [23] is an area of machine learning concerned with how software agents ought to take actions in an environment in order to maximize some notion of cumulative reward. Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning.

The paper [6] presents a method to achieve the service discovery process using the principles of multilevel matching based on multi-level specifications and customization based on reinforcement learning techniques. In this method, services are selected dynamically using an on-line performance-based reinforcement feedback.

One of the earliest models of reinforcement learning is called a learning automaton [7] where the agent attempts to learn the optimal action from a finite set using reward/penalty reinforcement from a stationary teacher/environment with unknown reward probabilities. The learning problem is formulated as updating the agent's action probabilities on the basis of trials consisting of an action performed and the reinforcement received [6].

A popular model-free algorithm is the so-called L_{RI} (Linear Reward-Inaction) algorithm described by

$$p_i(k+1) = p_i(k) + \alpha r(k)(1 - p_i(k))$$
$$p_j(k+1) = p_j(k) - \alpha r(k) p_j(k)$$

where $p_i(k)$ is the agent's probability of choosing action a_i at trial k, a_i is the action chosen at trial k, $r(k)$ is the reinforcement received (with $r(k) = 0$ signifying penalty, and $r(k) = 1$ signifying reward), and $\alpha > 0$ is the learning step-size [6].

Chapter 3

Problem Definition

3.1 Scenario Definition

In one area, there is a cloudlet network composed of many cloudlets. Users are using mobile devices and are constantly moving. Users are sending tasks to nearby cloudlets for cloud computing at all times. The types of user offloaded tasks are various, there are time-sensitive tasks and computation-sensitive tasks. Due to the uncertainty of user mobility and task types, the load on each cloudlet is constantly changing. If a cloudlet suddenly receives many users' task offloading requests, then these tasks may exceed the load capacity of the cloudlet, and the excess tasks will not be executed, or wait for other tasks to complete before being executed. And at the same time, some cloudlets may only accept fewer task execution requests, then there will be excessive computing resources on these cloudlets that are not being used. Because these cloudlets are interconnected, if tasks on heavily loaded cloudlets are transferred to lightly loaded cloudlets for execution, the resources of the cloudlet network will be fully utilized, and the response time of these tasks will be reduced.

In our scenario, if we use an agent to manage and collect the information of all cloudlets, it will generate lots of communication data and calculation, which will result in high latency. And there are lot of researches using a central cloudlet manager to solve the problem of load unbalancing in cloudlet network [26], [27], [28]. For the central cloudlet manager, if there are a large amount of cloudlets in the network, it will have a lot of drawbacks, such as large communication volume, large algorithm running time, poor scalability and so on.

So in our scenario, we don't want to use the central method to solve this problem. For less communication volume, we set the cloudlet can only communicate with the surrounding and connected cloudlets. So for one cloudlet, the communication volume is related to the connection number of cloudlets. Because the cloudlets compete computing resources with others and don't know all the information of others, so it's an incomplete information game. We will use the Stackelberg game to solve the problem of load unbalancing.

We define that a cloudlet that will have excessive computing resources and be able to accept tasks from other cloudlets is called resource server. A cloudlet that have more tasks than its own load and needs to reduce its tasks is called task server.

In an incomplete information Stackelberg game, a task server cannot know the decisions and rewards of all task servers. Since the resources on resource server are limited and the task server doesn't know the others' strategies, if many task servers select the same resource server, which causes the tasks in the resource server to exceed its own load and some tasks will not be able to obtain computing resources, resulting in calculation failure.

In our scenario, for a task server, it cannot definitely know which resource server is best. Because of the competition of resources and limited resources, the cloudlets' strategies will affect each other. So, the task is distributed to the same resource server and may get different rewards. Our idea is that each task server requests resources from surrounding resource servers multiple times and calculate the reward. If the reward of a resource server is the largest and the same reward can always be obtained, it means that the reward is the best reward and the resource server is the optimal task transferring destination. Our objective is that how to get the best rewards and find the optimal resource servers for all task servers.

3.2 Problems

Here we define two problems in our scenario:

1. How to quickly find a good strategy of all cloudlets to solve the problem of load unbalancing in a cloudlet network, without using an agent in an incomplete information Stackelberg game?

We assume that each task server can not cooperate, know each other's strategy and communicate with other task servers, but can change its own strategy according to rewards from each resource server. The reward of a task server will be affected by other task servers' strategies, so the reward of a task server from the same resource server is changing. For different type of tasks, their requirements are different. In exiting researches [26], [27], [28], they have a unique objective for all tasks, such as minimal task response time. We want to propose a method to meet the different requirements of different types of tasks.

2. How to go through fewer iterations and strategy updates, all task servers tend to adopt a stable strategy, and obtain the same reward?

In Stackelberg game, we need some iterations and strategy updates to get a good task transferring strategy. If after multiple iterations, each task server tends to choose a stable strategy, then this strategy may be the best strategy for all task servers.

Chapter 4

System Model

In this chapter, we will introduce the cloudlet network model and system modeling with Stackelberg game and definition in cloudlet network.

4.1 Cloudlet Network

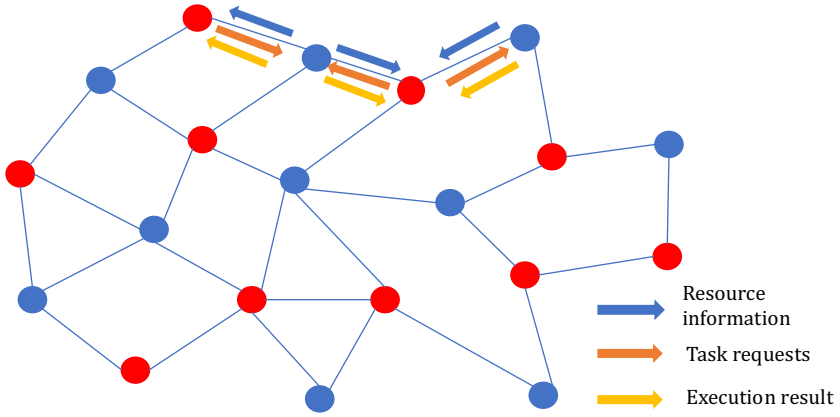

Figure 4.1. Cloudlet Network

We assume that the cloudlets are connected with each other and can communicate and transfer data with each other with dedicated high-speed wired lines as Figure 4.1 shows. The cloudlets are deployed in different places. Each cloudlet covers one area and provides task offloading services for the users in this area. In Figure 4.1, the blue nodes represent the low-load cloudlets and are called as resource servers. The red nodes represent the high-load cloudlets and are called as task servers. Colored arrows indicate data transfer direction. The blue arrows represent that resource servers distribute the resource information to the surrounding task servers. The orange arrows represent that task servers send the task execution requests to the surrounding resource servers. The yellow arrows represent that resource servers send the task execution results to the surrounding task servers.

4.2 System Modeling with Stackelberg Game

Stackelberg games consist of a hierarchical structure with a leader and followers. In a Stackelberg game, the leader first sets a price which is charged

to the followers, then the followers react to the charged price and compete with each other. In a Stackelberg game, one player (the "leader") moves first, and all other players (the "followers") move after him. Figure 4.2 is the system model of Stackelberg game.

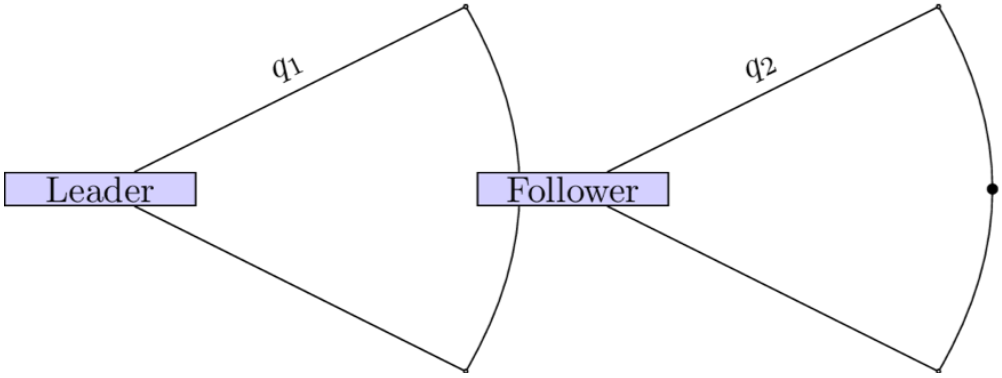

Figure 4.2. System model of Stackelberg game

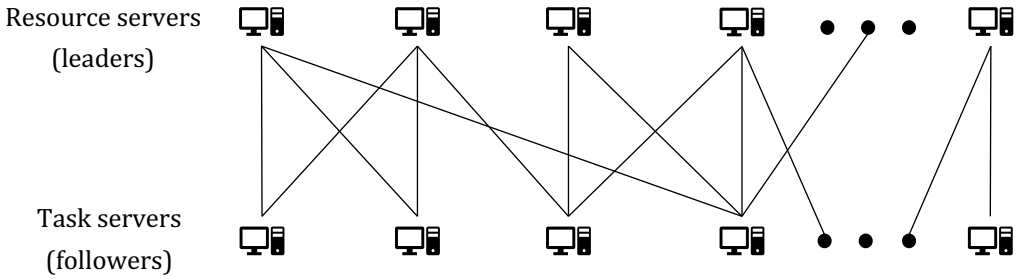

Figure 4.3. System model with Stackelberg Game in cloudlet network

Figure 4.3 is the system model with Stackelberg game in cloudlet network we proposed. When many task servers distribute tasks to the same resource server, each task server will compete the bandwidth and computing resources of the resource server, thus creating a competitive relationship. At the same time, the allocation strategy of each task server will indirectly restrict the strategies of other task servers by affecting the pricing strategies of the resource servers.

Therefore, there is a noncooperative game relationship between task servers and resource servers.

Nash equilibrium is a solution to the noncooperative game and is a stable state of countermeasures between the various players. Assuming that each player can make rational decisions, when the game reaches the Nash equilibrium, the utility of each player is maximized, and each player participating in the game cannot change the strategy to obtain more profits. However, the optimal solution does not necessarily exist. Some noncooperative game models do not have an optimal solution, and some models may have multiple optimal solutions. In our scenario, we want to get a good solution quickly instead of getting the optimal solution slowly. If it takes a lot of time to get the optimal solution, it's meaningless for the time-sensitive tasks to be transferred to other cloudlets. It may take less time to execute these

tasks locally.

4.3 Stackelberg Game Definition in Cloudlet Network

The entire Stackelberg game process consists of two phases: the leader decision phase and the follower decision phase and three elements: player, strategy and payoff. We define the player, strategy and payoff of Stackelberg game in our scenario as follows:

1. Two phases in Stackelberg game

Leader decision phase: resource servers (leaders) first determine the unit price of their own resources, and then send information such as price, resource number and computing power to the surrounding task servers. The resource server will adjust the price strategy based on the fee paid by the task servers.

Follower decision phase: task servers (followers) distribute their tasks to surrounding resource servers and calculate the rewards of the tasks based on the performance, fee paid for resource servers and task execution time. In our scenario, the task server will choose the resource server with the largest reward.

2. Payoff

Payoff is defined as the reward obtained by the player in Stackelberg game. The resource servers' payoff is defined as $G_i(x, y)$ and the task servers' payoff is defined as $F_j(x, y)$, $i \in M$, $j \in N$, $x \in X$, $y \in Y$.

3. Strategy

The strategy of task server is distributing the tasks to ideal cloudlets. For the resource server, the strategy is to set the unit price for task execution. The task servers' strategies combination is defined as X = $\{x_1, x_2, \ldots, x_m\}$, and the set is represented as X. The resource servers' strategies combination is defined as y = $\{y_1, y_2, \ldots, y_n\}$, and the set is represented as Y.

The price strategy of resource server j: p_j. The price strategies of all resource server:

$$P = \{p_1, p_2, \ldots, p_m\}$$

Define $q_i = (q_{ij}, q_{-ij})$ as resource requirement vector for task server *i*. q_{ij} represents the computing and network resources policy requested by task server *i* to resource server *j*. q_{-ij} represents the resource policies of task server *i* except the resource server *j*.

$Q = \{q_1, q_2, \ldots, q_n\}$ represents a combination of resource requirements for all task servers in the cloudlet network.

4. Player

Define the resource servers as leaders. In the entire task transferring system, the resource server is the occupant of the resource, and its decision is a prerequisite for the decision of the task servers, and its pricing will affect the task transferring of the task servers.

Define the task servers as followers. According to the difference between the resource servers' prices and execution capabilities, followers aim to optimize their own utility and adjust the task transferring strategies.

Define *m* leaders and *n* followers, represented by two sets M = {1, 2, ..., *m*} and N = {1, 2, ..., *n*}, respectively.

We define that the utility function of the task server consists of three parts: payoff, cost, and task execution time. Utility function of task server *i* is defined as:

$$F_i(\mathrm{p},\ q_i,\ q_{-i}) = U_i(\sum_{j=1}^{m} q_{ij}) \ - \ \sum_{j=1}^{m} P_j(p_j,\ q_{ij}) \ - \ \sum_{j=1}^{m} T_j(q_{ij})$$

Function U_i is the payoff function with the resource vector q_i. The payoff of the task server is related to the resources obtained. $P_j(p_j,\ q_{ij})$ is the fee paid by the task server *i* to the resource server *j*. $T_j(q_{ij})$ is the tasks execution time of task server *i*. The specific function we define is as follows:

$$\mathrm{U(x)} = \alpha \log(1 + x)$$

α varies from user to user, it can be understood as user satisfaction with resources.

$$P_j(p_j,\ q_{ij}) \ = \ p_j * q_{ij},\ p_j \ = (p_c,\ p_b)_j,$$

$$T_j(q_{ij}) \ = \ \beta \frac{e_i}{q_{ij}},$$

$e_i \ = (e_c,\ e_s)_i$. e_c is the computation instructions of the task *i*. e_s is the data size of task *i*. β is related to the task type. For delay sensitive tasks, the β value is large. For delay insensitive tasks, the β value is small.

Payoff of resource server *j* is the fee of selling the resources to the task servers.

Utility function of resource server *j* is defined as follows:

$$G_j(p_j, \mathrm{q}) = \ \sum_{i=1}^{n} q_{ij} * \ p_j$$

The pricing strategy of the resource server is defined:

- Related to its own resources
- Related to the load status of the past period of time
- When the number of requested resources exceeds its own resources, the price will be set high to reduce resource requests from task servers.

4.4 Nash equilibrium of noncooperative game

We suppose that in the case that the leaders have made decisions, each follower of the participating game performs a noncooperative game [30] under the decision to maximize his own profit, then the best strategy of the follower under the leader strategy can be defined as:

$$N(x) = \{y^* = (y_1^*, y_2^*, \ldots, y_j^*, \ldots, y_n^*): F_j(x, y_j^*, y_{-j}^*) \geq F_j(x, y_j, y_{-j}^*)\}$$

Where y_j^* represents the best strategy for follower *j*; y_{-j}^* represents the best set of strategies for followers other than *j*.

It can be seen from the above formula that in the case that the leaders' strategies have been determined, each follower has an optimal strategy and will not obtain greater profits by adjusting his own strategy.

When all followers' strategies satisfy the above formula, the optimal strategy space y^* is called a Nash equilibrium point of the noncooperative game.

4.5 Game Equilibrium Problem Definition

Define the Stackelberg game as

$$V = \{x, y, G_i(x, y), F_j(x, y)\}, i \in M, j \in N, x \in X, y \in Y$$

The equilibrium problem can be defined as:

Existence of $(x^*, y^*) \in X \times Y$ makes $G_i(x_i^*, x_{-i}^*, y^*) \geq G_i(x_i, x_{-i}^*, y^*)$ true, $i \in M$, $y^* \in N(x)$.

If any strategy combination $(x^*, y^*) \in X \times Y$ satisfies the above formula, then (x^*, y^*) is called the subgame perfect Nash equilibrium point of the Stackelberg game.

Next is the proof of the existence of Nash equilibrium in the utility function of task server.

$$F_i(p, q_i, q_{-i}) = U_i\left(\sum_{j=1}^{m} q_{ij}\right) - \sum_{j=1}^{m} P_j(p_j, q_{ij}) - \sum_{j=1}^{m} T_j(q_{ij})$$

$$= \alpha_i \log\left(1 + \sum_{j=1}^{m} q_{ij}\right) - \sum_{j=1}^{m} p_j * q_{ij} - \sum_{j=1}^{m} \beta_i \frac{e_i}{q_{ij}}$$

First derivation of F_i:

$$\frac{\partial F_i(p, q_i, q_{-i})}{\partial q_{ij}} = \frac{\alpha_i}{1+\sum_{j=1}^{m} q_{ij}} - p_j + \sum_{j=1}^{m} \beta_i \frac{e_i}{{q_{ij}}^2}$$

Second derivation of F_i:

$$\frac{\partial {F_i}^2(p, q_i, q_{-i})}{\partial {q_{ij}}^2} = -\frac{\alpha_i}{\left(1+\sum_{j=1}^{m} q_{ij}\right)2} - \sum_{j=1}^{m} \beta_i \frac{2*e_i}{{q_{ij}}^3}$$

Because $\alpha_i > 0$, $\beta_i > 0$, $\frac{\partial {F_i}^2(p, q_i, q_{-i})}{\partial {q_{ij}}^2} < 0$ is always true.

The utility function is a strict concave function that ensures the existence of a Nash equilibrium. Figure 4.4 is the structure example of existence of Nash equilibrium in our proposed method.

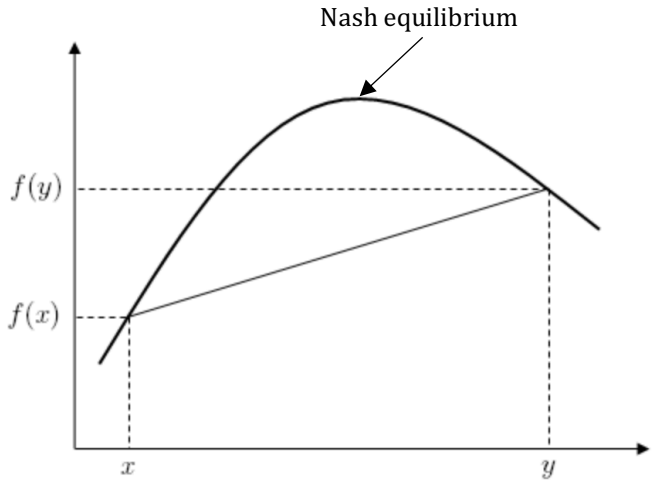

Figure 4.4. Example of existence of Nash equilibrium in our proposed method

Chapter 5

Proposal

In this chapter, we will introduce the optimization problem in our scenario and our proposed method, which combines Stackelberg game and reinforcement learning.

5.1 Optimization problem

In an incomplete information game, the task servers and resource servers don’t know the strategies of others and only can communicate with surrounding cloudlets. The optimization problem in our scenario is that how to record the impact of other players' strategies on a player's own rewards and make each player's next decision more reasonable and accurate based on each reward after each iteration.

The probability of selecting a resource server for a task server that was previously thought is 0 or 1, which means unselected or selected. For a task server, after selecting a resource server, the next choice of which resource server is uncertain, may be randomly selected in the traditional Stackelberg game model.

With reinforcement learning, we represent the behavior of the resource server selection by probability, that is, the task has such a probability to choose the resource server. In this way, each time the probability of each resource server is updated according to the obtained rewards.

We use reinforcement learning to determine which resource server the task server should choose. That is, when the task servers are going to distribute the tasks to the ideal resource servers, they just choose the resource server with the biggest selection probability.

We model the resource server and task server into a multi-leader and multi-follower Stackelberg game and build their respective utility functions. We combine Stackelberg game and reinforcement learning to find the optimal task distributing strategy quickly and efficiently.

If a task server can get a relatively large reward from the resource with the biggest selection probability multiple times, it means that the reward is stable. If the behaviors and rewards of the all task servers tend to be stable, it means we find the Nash equilibrium of Stackelberg game, that is, the optimal task transferring strategy for all task servers.

5.2 Stackelberg Game + Reinforcement Learning

For a task server, there are some resource servers to choose. Task server can calculate the rewards of each task based on the performance and resources of the resource servers. Then, based on the rewards, the selection probability of each resource server is updated.

For resource server *j*, the reward of task *i* will update the probability that task *i* chooses cloudlet *j*, and it will also update the selection probability of other resource servers. If a task gets a better reward, it will increase the probability of selecting this resource server, and it will also reduce the probability of selecting other resource servers. If a task gets a poor reward, it will reduce the probability of selecting this resource server, and it will increase the probability of selecting other resource servers.

In order to achieve this goal, we refer the Linear Reward-Inaction algorithm [6] and propose our reinforcement learning method.

Task server probability function is defined with reinforcement learning as:

$$p_i(k+1) = p_i(k) + \alpha(r(k) - r)(1 - p_i(k))$$

$$p_j(k+1) = p_j(k) - \alpha(r(k) - r)p_j(k) \ (j \epsilon [1, \mathrm{M}] \text{ except } i)$$

k is iteration time of task server. r represents the average reward in the past period of time.

When a task server takes a behavior, gets a reward $r(k)$. If $r(k)$ is greater than r, then the selection probability of this resource server will increase, and the selection probability of other resource servers will decrease. If $r(k)$ is less than r, then the selection probability of this resource server will decrease, and the selection probability of other resource servers will increase. When the task server takes an action and gets the same rewards for a period of time, then $r(k) - r$ is equal to 0, and the selection probability of each resource server does not change. It means that the task transferring strategy and reward of the task server will tend to be stable.

Resource server price function is defined with reinforcement learning as:

$$q_i(t+1) = q_i(t) + v_i(s(\mathrm{t}) - s)$$

t is the iteration time of resource server. s represents the average total reward of the tasks in this resource server in the past period of time. $s(t)$ represents the total reward of the tasks at time t. $q_i(t)$ is the resource server's price at time t. $v_i > 0$ is the learning step-size.

5.3 Stackelberg game and reinforcement learning (SGRL) algorithm

The pseudo code of Stackelberg game and reinforcement learning (SGRL) algorithm is shown in Algorithm 1.

Algorithm 1 Stackelberg game and reinforcement learning (SGRL) algorithm

Input:
Cloudlets information in cloudlet network
Output:
Task transferring strategies of task servers
Initialization
Each resource server initializes the price strategy based on its resources
for each iteration t **do**
  resource server *j* distributes its information of price, performance to the surrounding task server *i*
  Task server *i* receives the information of surrounding resource server *j*
  Task server *i* initializes the selection probability *p*
  for each iteration k **do**
    Task server *i* selects the resource server *j* with the max *p*
    Resource server *j* receives the task execution requests from surrounding task server *i*
    Resource server *j* decides which task is executed based on the resources it has and sends the execution decision to the task server *i*
    if the tasks are accepted
      The tasks are executed in resource server *j*
    else
      The tasks are executed locally in task server *i*
    end if
    Task server *i* calculates the rewards of resource servers *j*
    Task server *i* updates the selection probability *p*
  end for
end for

5.4 Iterative algorithm running time series

In the process of the task servers reaching the Nash equilibrium, the price of the resource server must remain unchanged, and the resource servers need to observe and wait for the task transferring strategies of the task servers to stabilize. This waiting time is one iteration time slice Δt of the resource

servers. The iterative process of the price change includes multiple time slices Δt, and each time slice Δt includes multiple iteration time slices $\Delta \tau$ of task servers. The algorithm running time series is shown in the Figure 5.1.

Through multiple iteration time slices $\Delta \tau$, the task transferring strategy of task servers is gradually optimized. Through multiple iteration times Δt, the price strategy of the resource server is gradually optimized.

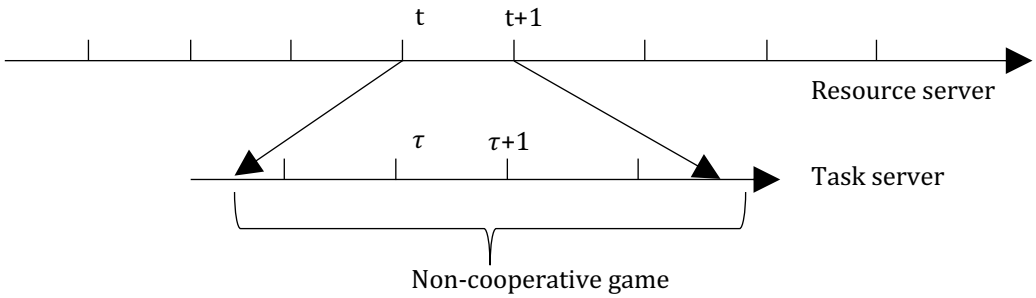

Figure 5.1. Iterative algorithm running time series

Chapter 6

Simulation Results

6.1 Simulation Definition

In order to prove that our proposed method has much less algorithm running time and better scalability than the method in [27], we make the following simulation definition.

We construct a cloudlet network consisting of 2000 cloudlets. Each cloudlet has a 50% probability of becoming a resource server or task server. Each cloudlet is connected to nearby cloudlets based on its geographic location. The number of each cloudlet connected to other cloudlets is between 2 and 10. The computing performance varies among cloudlets. The instruction per second of cloudlet's processor speed varies from 500 to 2000. The number of tasks that exceed the load on each task server is between 3 and 8. Tasks are divided into 4 types. There are simple tasks with a small amount of data and a small amount of calculation. There are also complex tasks with a large amount of data and a large amount of calculation. The tasks are defined in Table 1.

The simulation is performed on a macOS Catalina machine with an Intel Core i7 3760h 2.3 GHz CPU, 8 GB RAM.

Task type	Data size	Instruction size
0	1~100	1~100
1	100~1000	100~1000
2	1000~5000	1000~5000
3	5000~10000	5000~10000

Table 6.1. Task definition

There are 5 to 10 excessive VMs on each resource server. If the resource server runs out of all VMs, it will no longer be able to accept tasks, unless there are tasks that have been executed and free up some computing resources.

If all the VMs on the resource server connected to the task server are used up, and there is no place to transfer the extra tasks on the task server, then the task will wait for the local task to finish before executing it locally. Tasks executed locally have a waiting time.

6.2 Comparison methods

Agent-based central management method

In this method, there is an agent to collect all cloudlet information, including task server task information, resource server computing power, and number of VMs. In order to obtain the minimum execution time of all tasks, the simplest method is transferring the tasks with a large amount of calculation to the resource servers with a strong computing capability for execution. Tasks with a small amount of computation are transferred to the resource servers with a weak computation capability for execution. If the number of tasks received by the resource server exceeds the number of its own VMs, additional tasks with a small amount of calculation will be executed locally on the task server. The execution time of the overall tasks will be minimal.

6.3 Task execution time comparison

We randomly generate a cloudlet network 50 times. We use our proposed Stackelberg game and reinforcement learning (SGRL) method and agent-based method to get the task transferring strategy with minimum task execution time, and record the sum of the execution time of all tasks. The results are shown in Figure 6.1, the smaller the value, the better. From the Figure 6.1, we can see that the execution time of all tasks obtained by SGRL is longer than that of the comparison method. Figure 6.2 shows the gap ratio of task execution time. The smaller the value, the better. The average task execution time gap ratio is 181.29%.

The agent-based method knows all the information of all tasks and cloudlets, so it can get the task transferring strategy with the minimum task execution time. Because our proposed method SGRL is a game with incomplete information, our results will only be infinitely close to the results of the agent-based method.

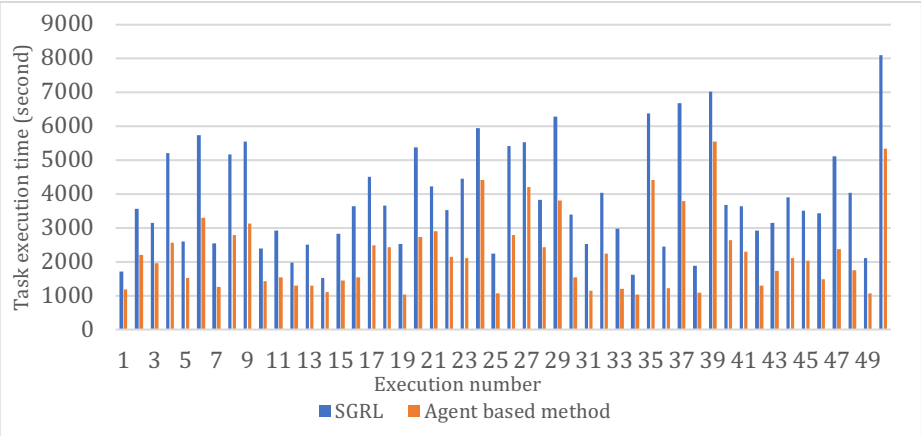

Figure 6.1. Task execution time comparison

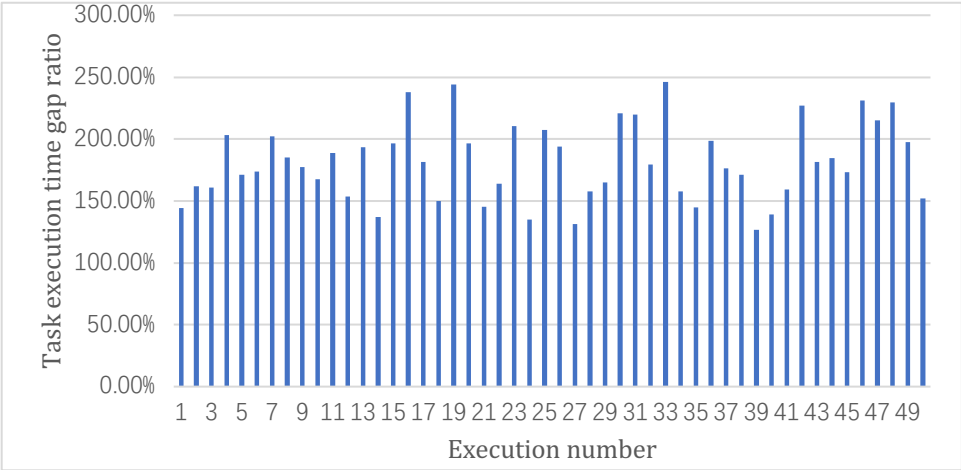

Figure 6.2. Task execution time gap ratio

6.4 Algorithm running time comparison

In SGRL method, we calculate the average time Δt spent by an iteration of a resource server and the average time $\Delta\tau$ spent by an iteration of a task server. $\Delta t = \Delta\tau * r_2$, r_2 is the iteration times of a task server. The overall iteration time is $\Delta t * r_1$, r_1 is the iteration times of a resource server.

Because the leaders and followers in the Stackelberg game adjust their strategies in their respective iteration intervals at the same time, each iteration time of the leader and follower is very short. The final running time is $\Delta\tau * r_2 * r_1$.

The Figure 6.3 is a typical task execution time trend graph from our experimental results.

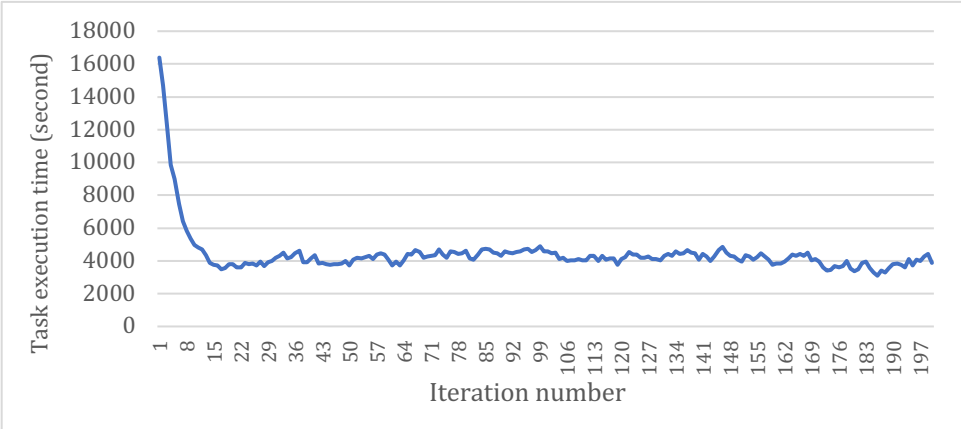

Figure 6.3. A typical task execution time trend graph

The experimental results show that the average number of iterations to obtain a solution is around 20. So the average running time to obtain a solution

is about $20\Delta\tau$.

When the density of the cloudlet network is 10, that is, when one cloudlet is connected to 10 cloudlets on average, the average running time $\Delta\tau$ for one iteration is 0.000548s. The average running time of getting a better solution is 0.01096s.

When the number of cloudlets with the same density is 2000, the average running time of the agent-based centralized management method is 7.159969s. The algorithm running time comparison between SGRL and agent-based method is shown in Figure 6.4. And the execution time of agent-based method will increase exponentially with the number of cloudlets, as shown in Figure 6.5. The running time of SGRL method does not increase with the number of cloudlets increasing.

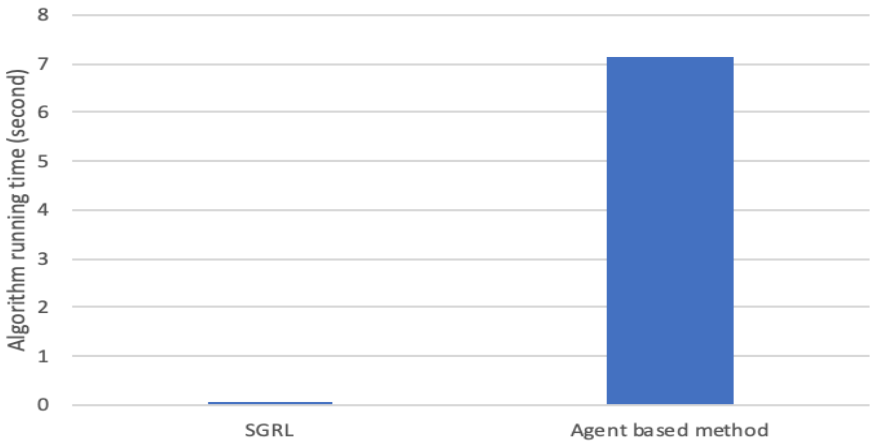

Figure 6.4. Algorithm running time comparison

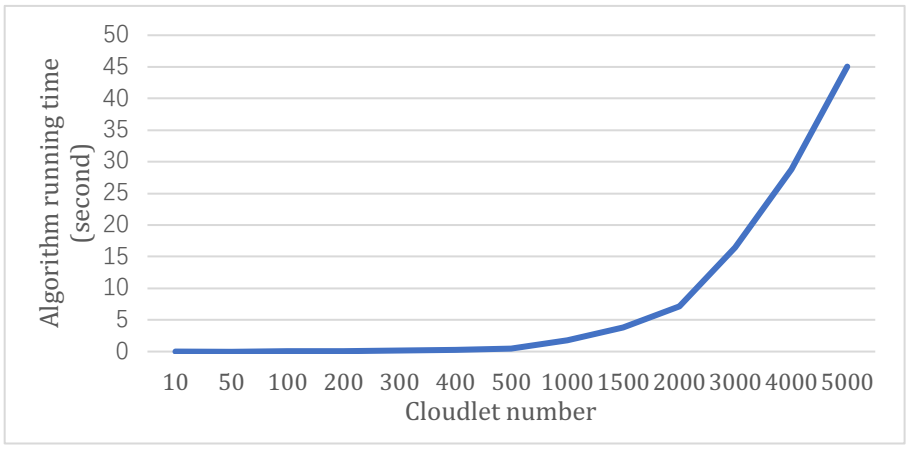

Figure 6.5. Algorithm running time trend graph of agent-based method

Chapter 7

Conclusion and Future Work

Experimental results show that our proposed method can approach a good solution with fewer iterations (about 20). Compared with the agent-based central management method, our proposed method can find a good solution much more quickly. Therefore, our proposed framework can get a task transferring strategy in near real time, so as to provide the users with a good task offloading experience.

Our method can meet the different requirements of different tasks, because each player in the Stackelberg game can define its own utility function. Such as execution time, price, performance of cloudlet and so on. However, with agent-based central management method, we cannot specify different utility functions for different tasks, we must use a unique objective function, such as the least execution time or the least money spent.

The framework we proposed has better scalability. Because in our proposed framework, each cloudlet only needs to communicate with surrounding cloudlets that can provide computing resources, the communication volume is very small. And the communication volume and running time are only related to the density of the cloudlet network and will not increase with the number of cloudlets increasing. However, the running time of the agent-based centralized management method will increase exponentially with the number of cloudlets increasing.

In the future work, we will narrow the gap between the optimal solution of agent-based method and the solution of the SGRL method. Although it is almost impossible to get the optimal solution in an incomplete information game, we will optimize our method to make the solution of the task transfer strategy closer to the optimal solution and strive to keep the gap within 10%. The scenario we originally envisioned was that after multiple iterations, all players will gradually choose a fixed target, so that on the basis of not obtaining all the information, we will get a solution that all players are satisfied with. From Figure 6.3, the strategy chosen by each player will change slightly, so that the result of the objective function fluctuates. In the future work, we will research how to improve the Stackelberg game and reinforcement learning method to avoid volatility and get a stable solution.

Chapter 8

References

[1] Z. Pang, L. Sun, Z. Wang, E. Tian and S. Yang, "A Survey of Cloudlet based Mobile Computing", 2015 International Conference on Cloud Computing and Big Data, IEEE.
[2] Z. Gu, R. Takahashi, and Y. Fukazawa, "Real-time Resources Allocation Framework for Multi-Task Offloading in Mobile Cloud Computing," 2019 CITS, IEEE.
[3] J. Wang, Y. Sun, B. Wang, B. Wang, A. Wang, S. Li, and Z. Sun, "Resource Allocation for D2D Video Multicast Using Multi-Leader Multi-Follower Stackelberg Game With Personalized Incentives," Special Section on Mobile Multimedia: methodology and applications, 2019.
[4] S. Guo, X. Hu, G. Dong, W. Li, and X. Qiu, "Mobile edge computing resource allocation: A joint Stackelberg game and matching strategy," Emerging Technologies of 5G IoT for Future Smart Cities - Research Article, 2019.
[5] W. Wei, X. Fan, H, Song, X. Fan, and J. Yang, "Imperfect Information Dynamic Stackelberg Game Based Resource Allocation Using Hidden Markov for Cloud Computing," IEEE Transactions on services computing, VOL. 11, NO. 1, January/February 2018.
[6] RR. Raje, S. Mukhopadhyay, S. Phatak, R. Shastri, and LS. Gallege, "Software Service Selection by Multi-level Matching and Reinforcement Learning," BIONETICS 2010, LNICST 87, pp. 310–324, 2012.
[7] Narendra, K.S., Thathachar, M.A.L.: Learning Automata: An Introduction. Prentice-Hall (1989).
[8] (2020, Janu.) Stackelberg competition. [Online]. Available: https://en.wikipedia.org/wiki/Stackelberg_competition.
[9] David Reinsel, John Gantz, John Rydning, "Data Age 2025: The Evolution of Data to Life-Critical Don't Focus on Big Data; Focus on the Data That's Big", April 2017.
[10] Knud Lasse Lueth, "State of the IoT & Short term outlook 2018", IoT Analytics.
[11] E. Cuervo, A. Balasubramanian, D.-K. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. "Maui: making smartphones last longer with code offload", Proceedings of 8th International Conference on Mobile Systems, Applications, and Services, pp. 49–62, ACM, 2010.
[12] Q. Xia, W. Liang, and W. Xu. "Throughput maximization for online request admissions in mobile cloudlets", Proceedings of 38th Conference on Local Computer Networks, pp. 589–596, IEEE, 2013.
[13] X. Ma, C. Lin, H. Zhang and J. Liu, "Energy-Aware Computation Offloading

of IoT Sensors in Cloudlet-Based Mobile Edge Computing," Sensors 2018, 18, 1945; doi:10.3390/s18061945.

[14] Md. Whaiduzzaman, A. Naveed and A. Gani, "MobiCoRE: Mobile device based cloudlet resource enhancement for optimal task response," IEEE transcations on services computing, vol. 11, NO. 1, january/february 2018.

[15] S. Rashidi and S. Sharifian, "Cloudlet dynamic server selection policy for mobile task off-loading in mobile cloud computing using soft computing techniques," J Supercomput (2017).

[16] M. Jia, J. Cao and W. Liang, "Optimal Cloudlet Placement and User to Cloudlet Allocation in Wireless Metropolitan Area Networks", IEEE Transactions on Cloud Computing, vol. 5, no. 4, October - December 2017.

[17] H. Lee and J. Lee, "Task Offloading in Heterogeneous Mobile Cloud Computing: Modeling, Analysis, and Cloudlet Deployment", 2018 IEEE Access.

[18] W. Li, Y. Zhao, S. Lu and D. Chen, "Mechanisms and Challenges on Mobility-Augmented Service Provisioning for Mobile Cloud Computing", IEEE Communications Magazine, March 2015.

[19] Z. Kuang, S. Guo, J. Liu and Y. Yang, "A quick-response framework for multi-user computation offloading in mobile cloud computing", Future Generation Computer Systems (2017), https://doi.org/10.1016/j.future.2017.10.034.

[20] S. Ahn, J. Lee, S. Park, S. H. S. Newaz and J. K. Choi, "Competitive Partial Computation Offloading for Maximizing Energy Efficiency in Mobile Cloud Computing", Special section on emerging trends, issues, and challenges in energy-efficient cloud computing, 2017.

[21] C. Tang, M, Hao, X. Wei and W. Chen, "Energy-aware task scheduling in mobile cloud computing", Distributed and Parallel Databases (2018), https://doi.org/10.1007/s10619-018-7231-7.

[22] B. Liu, H. Xu and X. Zhou, "Stackelberg Dynamic Game-Based Resource Allocation in Threat Defense for Internet of Things", Sensors 2018, 18, 4074; doi:10.3390/s18114074.

[23] (2020, Janu.) Reinforcement learning. [Online]. Available: https://en.wikipedia.org/wiki/Reinforcement_learning.

[24] J. Chen, Y. Wu, L. P. Qian, H. Peng, and H. Zhou, "Energy-efficient content distribution via mobile users cooperations in cellular networks," Peer-to-Peer Networking and Applications, pp. 1–15, 2016.

[25] J. G. Andrews, S. Singh, Q. Ye, X. Lin, and H. S. Dhillon, "An overview of load balancing in hetnets: Old myths and open problems," IEEE Wireless Communications, vol. 21, no. 2, pp. 18–25, 2014.

[26] D. Yao, L. Gui, F. Hou, F. Sun, D. Mo and H. Shan, "Load Balancing Oriented Computation Offloading in Mobile Cloudlet", IEEE, 2017.

[27] V. Chamola, C. K. Tham and G. S. S. Chalapathi, "Latency Aware Mobile Task Assignment and Load Balancing for Edge Cloudlets", The First International Workshop on Smart Edge Computing and Networking 2017.

[28] M. Jia, W. Liang, Z. Xu and M. Huang, "Cloudlet Load Balancing in Wireless Metropolitan Area Networks", IEEE INFOCOM 2016 - The 35th Annual IEEE

International Conference on Computer Communications.
[29] H. Cao and J. Cai, "Distributed Multiuser Computation Offloading for Cloudlet-Based Mobile Cloud Computing: A Game-Theoretic Machine Learning Approach", IEEE Transactions on vehicular technology, VOL. 67, NO. 1, January 2018.
[30] Zhang Lei, Zhou Jinhe, Zhang Yuan. Cache Resource Allocation Algorithm in Cloud CDN Based on Stackelberg Game[J]. Computer Engineering, 2017, 43(8):15-20, 25.

Acknowledgement

I would like to thank my supervisor, Professor Yoshiaki Fukazawa and Teaching assistant Ryuichi Takahashi. They gave me a lot of help and suggestions during my research. It was because of their help that I was able to complete this thesis. Thank you very much.