A Fund-Constrained Investment Scheme for Profit Maximization in Cloud Computing

Kenli Li, *Member, IEEE*, Jing Mei, and Keqin Li, *Fellow, IEEE*

Abstract—Cloud computing is becoming more and more popular and has received considerable attention recently. As a new kind of Information Technology (IT) commercial model, understanding the economics of cloud computing becomes critically important. From the cloud service providers' perspective, profit maximization is the top issue for them. Because a multi-server system is devoted to serving one type of service requests and application, service providers should build multiple multi-server systems to satisfy the market requirements of different application domains. Because available funding for a service provider is generally limited, it cannot afford to invest in all application domains. Hence, how to select appropriate application domains for investment and allocate funding such that the total profit is maximized are important issues for service providers. To address this problem, a fund-constrained profit maximization model is proposed. However, the exact solution of this optimization model is very difficult to formulate due to its complexity. Hence, this paper presents a heuristic strategy to search for a high quality solution. In our strategy, the optimization problem is solved in four stages, and the solution is optimized gradually. Through the proposed heuristic investment strategy, an appropriate investment scheme can be developed that synthesizes the market requirement, the fund constraint, the service level agreement, and so forth. A series of numerical calculations is executed to assess the performance of the proposed strategy. Then, six other investment strategies are compared to our strategy. Our results show that the investment scheme designed using our strategy can produce much more profit than these six other strategies.

Index Terms—Cloud computing, deadline, fund allocation, multi-server system, profit maximization, queuing model, service-level agreement, waiting time.

1 Introduction

Cloud computing is becoming more and more popular and has received considerable attention recently. A new kind of IT commercial model, cloud computing centrally manages the resources and offers resources and services over the Internet to customers on demand [1].This can reduce the requirement for large capital outlays for the hardware to deploy service and the human expense to operate it [2]. Of course, the services provisioned by cloud computing are not free. The cloud service providers support the operation of cloud computing and earn profits by charging the customers who enjoy the services according to various pricing models [3, 4, 5].

In a cloud computing environment, there are always three tiers of participants: the infrastructure providers, the services providers, and the customers (see Fig. 1 and its further elaboration in Section 3.1). The infrastructure providers maintain the basic hardware and software facilities. The service providers rent resources from infrastructure providers and provide services to customers. The customers submit their service requests to the service providers and are charged based on the quantity and the quality of the provided services [6]. In this paper, we focus on the research of service providers. For most businesses, profit is the most important concern. Hence, how to maximize profit is the top issue for service providers. Profit maximization is an optimization problem that is a classical issue in cluster computing and grid computing [7, 8, 9, 10, 11]. With the development of cloud computing, understanding the economics of cloud computing becomes critically important. In [12, 13], cloud computing economics is introduced as a significant, new research topic in computer science.

There are many factors that affect the profit of the service providers such as market demand, capital input, configuration of the cloud service platform, pricing model, and so forth. For an application domain with a known market demand, the cloud service system must be configured appropriately to achieve profit maximization. If the capacity of the service system is configured to satisfy the peak demand, over-provisioning can occur because the available resources may exceed the actual demand most of the time; hence, a significant quantity of resources would be wasted. If the service system is configured with a lower capacity, under-provisioning can occur as the available resources are unable to fully meet the fluctuating demand, which also leads to profit loss. It is important for the service providers to trade off between the two extremes to maximize the profit. To achieve this goal, Cao *et al.* [1] proposed an optimal configuration strategy for the service system. However, the authors focused only on the profit maximization problem of a single domain and did not consider the funding constraint.

In this paper, we study a different issue from that discussed in the existing researches. In practice, many service providers want to configure their own cloud serving platforms by renting resources from infrastructure providers, but their available funding is limited. There are so many ser-

- *Jing Mei is with the College of Mathematics and Computer Science, Hunan Normal University in Changsha, Hunan, China, 410082. E-mail: jingmei1988@163.com*
- *Ken li Li and Keqin Li are with the College of Information Science and Engineering, Hunan University, and National Supercomputing Center in Changsha, Hunan, China, 410082. E-mail: lkl@hnu.edu.cn, lik@newpaltz.edu.*
- *Keqin Li is also with the Department of Computer Science, State University of New York, New Paltz, New York 12561, USA.*

*Manuscript received ****, 2015; revised ****, 2015.*

vice domains from the end-users that the available funding is not sufficient to support all serving domains. Hence, how to utilize the limited funding to obtain a maximum profit is an important and interesting problem. To serve diverse service requests from different application domains, service providers should build multiple service sub-platforms such that different types of server requests can be sent to different sub-platforms. Configuring a service sub-platform for an application domain incurs upfront costs, such as the renting of resources and cost of electricity. The electricity is a specific commodity, similar to natural gas, which can be used before paid for or be paid for before use. In this paper, we consider the electricity as an upfront cost. Since configuring of cloud service platforms incurs upfront costs and the available funding of service providers may be constrained, how to select the best application domains for investment and how to allocate the limited funding are two key problems. If the funding allocated to an application domain increases, then funding allocated to another domain will decrease. In addition, the amount of funding allocated to an application domain affects the profit obtained in this domain, and then affects the total profit of the cloud service providers. Hence, funding allocation is an important concern of cloud service providers. This paper addresses this problem and seeks an optimal funding allocation scheme as well as the optimal configuration scheme for each application domain.

When considering multiple multi-server systems and funding constraint, profit maximization becomes much more complicated. In general, if the available funding is sufficient, investing in all the application domains with the optimal configurations can achieve the maximal profit. However, if the budgets of the service providers are limited, they cannot afford to invest all application domains. So, an appropriate investment scheme should be found such that the total profit is maximized. This problem is complicated and consists of three sub problems: the first relates to the nature of the invested application domains; the second is the funding level allocated to each domain; the last is how to configure the multi-server system with the allocated funding. In this paper, we first build a fund-constrained profit maximization model for this problem. According to the analysis, the problem cannot be solved using techniques in optimization theory due to the complexity of the model. Hence, in this paper, a heuristic algorithm is developed to find an effective asymptotic solution. This heuristic algorithm is applied in three phases. First, an initial investment scheme is determined. Second, slight adjustment is done to find a better scheme with more total profit. Third, the remaining funding is allocated to further increase the profit. Throughout the whole process, the 0-1 knapsack problem is an important sub problem which should be solved many times. It is a classical NP-hard problem [14]. The methodology contributions of this work include:

- Define a fund-constrained profit maximization problem from the perspective of service providers.
- Analyze the revenues and costs of a service provider and build a fund-constrained profit maximization model.
- Develop a heuristic algorithm to solve the optimization problem, which can determine the invested application domains, the funding allocated to each application domain, and the configuration of each multi-server system in a flexible way.
- Conduct a series of numerical calculations to demonstrate the performance of our strategy, and compare our strategy with six other strategies to verify the superiority of our strategy.

The rest of the paper is organized as follows. Section 2 reviews the related work on profit maximization in cloud computing. Section 3 presents the models used in this paper, including the three-tier cloud computing model, the multiple multi-server systems model, the revenue and cost models. The problem is also described in detail in the section. Section 4 proposes our heuristic algorithm to find the sub-optimal investment scheme. Section 5 analyzes the performance of our investment strategy and demonstrates the performance of the proposed scheme through comparison with six other investment strategies. Section 6 presents the conclusions of this work.

2 Related Work

In this section, we review recent works relevant to the economic problem in cloud computing. As a new commercial model, profit is one of the most important issues in cloud computing, especially for the cloud service providers. The profit obtained by the service providers is determined by two parts: the revenues and the costs. Hence, increasing revenues and reducing costs are two ways of increasing profit. Because the costs of service providers consist of the resource rental and the electricity cost, there are two ways to reduce the costs: improve the system utilization to reduce the number of servers to be rented and/or reduce the energy consumption to reduce the electricity cost.

In Hu *et al.* [15], an autonomic resource management algorithm is proposed based on the response time distribution, which is used to select an appropriate resource allocation strategy between Shared Allocation (SA) and Dedicated Allocation (DA) which requires the smallest number of servers while meeting the Service-Level Agreements (SLAs). Mazzucco *et al.* [16] addressed the problem of maximizing the profit of cloud providers by trimming down their electricity cost. The idea is to improve server utilization based on the dynamical estimation of user demand and the system behaviors. Beloglazov *et al.* [17] introduced a resource provisioning and allocation algorithm which not only achieves energy efficiency but also satisfies the negotiated SLAs. Its basic idea is to reduce energy consumption by reducing the number of hosts. Cao *et al.* [18] studied the request dispatching problem of multiple multi-server systems with the objectives of energy optimization and load balancing. Two load distribution methods are presented based on power constraints and performance constraints, respectively. The above research aims at increasing profit but not achieving profit maximization.

There is some research focusing on the profit maximization problem for service providers. Chaisiri *et al.* [19] took into consideration the uncertainty of customer demand, and proposed a stochastic programming model with two-stage recourse to solve the profit maximization problem for the service providers. Cao *et al.* [1] proposed an optimal multi-server configuration strategy. Through the optimal strategy, the optimal configuration of a multi-server system, i.e., the server size and the server speed can be determined such that the profit of a multi-server system is maximized. However, these papers look at solving the profit maximization problem of single multi-server system and do not consider the fund constraint.

Some papers consider the profit issue in different cloud computing environments. For example, Liu *et al.* [20] considered a cloud service provider operating geographically distributed data centers in a multi-electricity-market environment, and proposed an energy-efficient, profit- and cost-aware request dispatch, and resource allocation algorithm to maximize a service provider's net profit. In this work, service requests are not distinguished and they can be dispatched to any data center.

Due to the randomness of task arrivals, service providers always face two situations: customer loss or resource waste. If the service provider deploys a cloud computing environment with high capacity to reduce the loss of customers, a great quantity of resource would be wasted; and conversely, if the service provider wants to increase the utilization of resource, many customers would be lost. Both of the situations reduce the profits of the service providers. To avoid the contradiction, the cloud federation strategy developed which allows server providers, mutually collaborating, to share their respective resources to fulfill each one's demand. Many researches adopted cloud federation to improve the profit [21, 22, 23, 24, 25, 26, 27, 28]. In [21], an analytical model that characterizes cloud federation is presented, which can be used to derive a provider's decisions about resource outsourcing, insourcing, and node shutdown. Focusing on the issue that resource sharing between multiple tenants leads to the decrease of the quality of service (Qos), Lee *et al.* [29] proposed a scheduling algorithm which can maximize the profit on the premise of guaranteeing the quality requirements of the service providers. However, the services discussed in the paper are interdependent - which is different from our paper.

Many researches discussed the economical problem from the perspective of the cloud customers. In the paper by [30], it is assumed that cloud providers offer cloud customers two provisioning plans for computing resources with different prices, namely reservation and on-demand plans. Based on the analysis of the two kinds of plans, a resource provisioning strategy is designed for cloud customers such that their total payment is minimized. In [31], many cloud service providers offer service with different QoS and price points. This paper develops a cloud resource procurement approach to automatically select appropriate cloud providers according to the requirements of the cloud customers.

In this paper, we study a profit maximization problem for the cloud server providers with multiple multi-server systems when the available funding is limited. To the best of our knowledge, such a fund-constrained profit maximization problem has not been studied in the existing literature.

Our focus is on finding an optimal investment portfolio from multiple application domains, and an optimal fund allocation and multi-server configuration scheme. This problem looks like the knapsack problem; however, it is much different. First, the weights and the values of all objectives are known in the knapsack problem, while the investment and the profit of each application domain in this paper are unknown. Second, the ratio of the value and the weight of

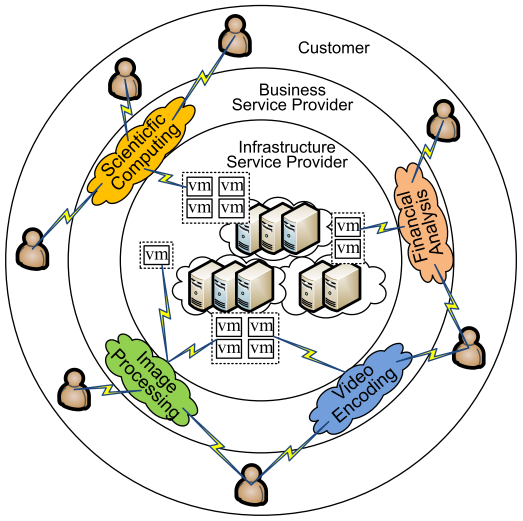


Fig. 1: The Three-Tier Cloud Structure

an object is fixed while the profit of an application domain is not proportional to the investment. When the fund invested in an application domain changes, its profit margin will change correspondingly - which might be positive, zero, or even negative. Due to so many differences, solving our problem is much more complicated than solving the knapsack problem and a precise optimal solution cannot be found. Hence, we have to design a heuristic algorithm to solve the problem in our paper. There are many existing works which focus on the variations of the knapsack problem, for example, multi-objective knapsack problem [32], multi-dimensional knapsack problem [33], and so forth. However, their problems are different from ours, so their solutions do not apply to solving our problem.

3 The Models

In this section, we first describe the three-tier cloud computing structure. Then, we introduce the related models used in this paper, including a multi-server queuing model, a revenue model, and a cost model.

3.1 Cloud Computing Structure

The cloud structure (see Fig. 1) consists of three typical parts, i.e., infrastructure providers, service providers, and customers. Here, we focus on the profit maximization problem of the service providers. This three-tier structure is used commonly in current cloud computing environments.

In the three-tier structure, an infrastructure provider owns and operates a cloud computing system which consists of a set of physical resources (server computers). We assume that the resources are homogeneous in terms of their computing capacity and capability. Being deployed for different applications, they can execute different service requests from different applications. Moreover, the resources are Dynamic Voltage Scaling (DVS)-enabled and can be run with different speeds.

service providers rent resources from infrastructure providers and provide services for customers. To serve different kinds of application requests, a cloud computing platform consisting of multiple multi-server systems is configured. The configuration of each multi-server system varies - including the server size and the server speed - and is determined by the market requirement of the objective application. In a multi-server system, the servers are homogenous and execute at the same speed.

A customer submits a service request to a service provider which delivers services on demand. The customer receives the desired result from the service provider under a specified Service-Level Agreement and pays for the service based on the amount of the service and the quality of the service.

3.2 Modeling Multiple Multi-server Systems

Cloud service providers configure cloud computing platforms as multiple multi-server systems and each multi-server system is deployed with special software and is devoted to serve one type of service requests and applications. A cloud computing platform consists of n heterogenous multi-server systems S_1, S_2, ..., S_n with sizes m_1, m_2, ..., m_n and speeds s_1, s_2, ..., s_n is shown in Fig. 2. Assume that a multi-server system S_i has m_i identical servers with speed s_i and it serves specific service requests A_i. Each multi-server system can be treated as an *M/M/m* queuing system which is elaborated as follows.

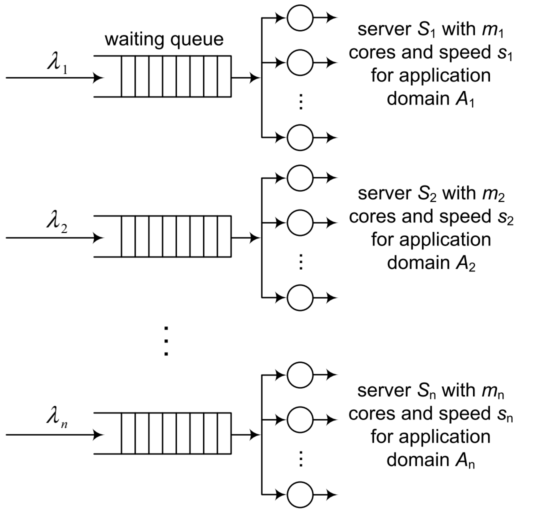

waiting queue
λ_1
server S_1 with m_1 cores and speed s_1 for application domain A_1
λ_2
server S_2 with m_2 cores and speed s_2 for application domain A_2
λ_n
server S_n with m_n cores and speed s_n for application domain A_n

Fig. 2: A Group of n Heterogenous Multi-server Systems

For each type of application domain A_i, there is a Poisson stream with arrival rate λ_i (measured by the number of requests per unit of time), i.e., the inter arrival times are independent and identically distributed (i.i.d.) exponential random variables with mean $1/\lambda_i$. Each multi-server system maintains an independent waiting queue with infinite capacity. When the incoming service requests cannot be processed immediately after they arrive, they are placed in the queue until they can be handled by any available server of system S_i. The first-come-first-served (FCFS) queuing discipline is adopted. The task execution requirements, measured by the number of instructions, are i.i.d exponential random variables r_i with mean $\overline{r_i}$. The m_i servers of system S_i have identical speed s_i. Therefore, the execution times of tasks on the multi-server system are also i.i.d. exponential random variables $t_i = r_i/s_i$ with mean $\overline{t_i} = \overline{r_i}/s_i$. The average service rate of each system is calculated as $\mu_i = 1/\overline{t_i} = s_i/\overline{r_i}$, and the server utilization is defined as $\rho_i = \lambda_i/m_i\mu_i = \lambda_i/m_i \times \overline{r_i}/s_i$. Let π_k^i be the probability that there are k service requests (waiting or being processed) in the *M/M/m* queuing system of S_i. Then, we have

$$\pi_k^i = \begin{cases} \pi_0^i \dfrac{(m_i\rho_i)^k}{k!}, \ k \le m_i; \\ \pi_0^i \dfrac{m_i^{m_i}\rho_i^k}{k!}, \ k > m_i, \end{cases} \tag{1}$$

where

$$\pi_0^i = \left(\sum_{k=0}^{m_i-1} \frac{(m_i\rho_i)^k}{k!} + \frac{(m_i\rho_i)^{m_i}}{m_i!} \frac{1}{1-\rho_i} \right)^{-1},$$

and Eq. (1) is right only when $0 < \rho_i < 1$.

For service providers, how to configure their cloud service platforms is an important issue because it greatly affects profits. Configuration includes two aspects. The first is determining in which application domains to invest and how many multi-server systems to construct. The second is determining how many servers need to be rented for each multi-server system, how fast the servers need to execute, and how much funding is allocated for each multi-server system. When taking into consideration funding constraints, the configuration problem for profit maximization becomes more complicated. In this paper, we try to solve this optimization problem. Since the profit is determined by the revenues and the costs, we present the revenue model and cost model in the following.

3.3 Cost Modeling

The costs to service providers consist of two major parts, i.e., the cost of infrastructure renting and the utility cost of energy consumption. Cloud service providers rent servers from infrastructure providers and pay them the corresponding rents. Assuming that the rental price of one server for a unit of time is β, the server rental price to build a multi-server system S_i with m_i servers is $m_i\beta$.

The cost of energy consumed per unit of time is determined by the price of electricity and the power consumed. Power dissipation and circuit delay in digital CMOS circuits can be accurately modeled by simple equations, even for complex microprocessors circuits, as described by Cao *et al.* [1, 18]. The power consumption of CMOS circuits consists of dynamic power, short-circuit power, and leakage power [34]. Dynamic power consumption is the dominant component in a well-designed circuit. In this paper, we adopt the following dynamic power model, which is adopted by many researchers [1, 34, 35]:

$$P_d = N_{sw}C_LV^2f. \tag{2}$$

Here, N_{sw} is the average gate switching factor at each clock cycle, C_L is the loading capacitance, V is the supply voltage, and f is the clock frequency [35]. In the ideal case, the relationship between the clock frequency f and the supply voltage V is $V \propto f^\phi$ for some constant $0 < \phi \le 1$ [18, 36]. The server execution speed s is linearly proportional to the clock frequency f, namely, $s \propto f$. Hence, the power consumption is $P_d \propto N_{sw}C_Ls^{2\phi+1}$. For ease of discussion, we assume that $P_d = bN_{sw}C_Ls^{2\phi+1} = \xi s^\alpha$ where $\xi = bN_{sw}C_L$ and $\alpha = 2\phi + 1$. In this paper, we set $N_{sw}C_L = 7.0$,

$b = 1.3456$, and $\phi = 0.5$. Hence, $\alpha = 2.0$ and $\xi = 9.4192$. The value of power consumption calculated by $P_d = \xi s^{\alpha}$ is close to the value of the Intel Pentium M processor [37]. It is reasonable that a server still consumes some amount of static power [38], denoted as P^*, when it is idle. For a fully utilized server, the average amount of energy consumption per unit of time is $P = \xi s^{\alpha} + P^*$.

However, the servers are not always fully utilized. For the multi-server system S_i with speed s_i, its utilization is $\rho_i = \lambda_i/m_i \times \overline{r_i}/s_i$. Hence, the dynamic energy consumption of a server in one unit of time is $\rho_i \xi s_i^{\alpha}$. Assuming that the price of energy is δ per Watt, then the cost of energy consumed by a server of system S_i per unit of time is $\delta(\rho_i \xi s_i^{\alpha} + P^*)$. Based on the analysis above, the cost of system S_i with m_i servers and a speed s_i per unit of time is $C_i = m_i(\beta + \delta(\rho_i \xi s_i^{\alpha} + P^*))$.

3.4 Revenue Modeling

Because of limited computing capacity, the service requests that cannot be handled immediately after entering the systems must wait in their respective queues until any server of the corresponding system is available. To guarantee the Quality of Service (QoS), there is a Service-Level Agreement (SLA) negotiated between the customers and the service provider. The SLA clearly outlines the QoS (measured by the waiting time) and the corresponding charge. We define the service charge function for a service request with execution requirement r and waiting time ω in Eq. (3),

$$\mathcal{E}(r,\omega) = \begin{cases} a_i r, 0 \le \omega \le d; \\ 0, \omega > d, \end{cases} \tag{3}$$

where a_i is a constant, which indicates the price of application domain A_i when serving requests of one billion instructions and d indicates the maximum waiting time that a service request can tolerate. According to the SLA, if the requests are handled before deadline d, the customers pay ar to the service provider. Otherwise, the provider serves the requests for free as a penalty. In this paper, we assume that the service provider will not change the pricing scheme once it is determined.

The following theorem gives the expected charge to a service request of A_i.

Theorem 3.1. The expected charge to a service request of A_i is

$$\mathcal{E}_i = a_i \overline{r_i}\left(1 - \Pi_q^i e^{-(1-\rho_i)m_i\mu_i d}\right),$$

where $\Pi_q^i = \pi_{m_i}^i/(1-\rho_i)$.

Proof 3.1. The proof is given in the supplementary file.

Hence, the total revenue obtained by serving service requests of A_i during unit of time can be calculated as:

$$R_i = \lambda_i \mathcal{E}_i = \lambda_i a_i \overline{r_i}\left(1 - \Pi_q^i e^{-(1-\rho_i)m_i\mu_i d}\right). \tag{4}$$

3.5 Problem Description

Cloud service providers rent servers from the infrastructure providers to construct multiple multi-server systems, and provide service to customers with different requests. For cloud service providers with limited budgets, how to invest the funds such that the profit is maximized is a critical issue.

Our *fund-constrained profit optimization* problem for multiple heterogenous multi-server systems can be specified as follows: given the budget per unit of time F, the task arrival rates of n kinds of different applications $\lambda_1, \lambda_2, ..., \lambda_n$, the average task execution requirement $\overline{r_1}, \overline{r_2}, ..., \overline{r_n}$, and the base power supply P^*, find the funds $C_1, C_2, ..., C_n$ invested to different multi-server systems, and their configurations, including the numbers of the servers $m_1, m_2, ..., m_n$ and the server speeds $s_1, s_2, ..., s_n$, such that the profit is maximized, subject to the constraint $\sum_{i=1}^{n} C_i \le F$, where $C_i = m_i(\beta + \delta(\rho_i \xi s_i^{\alpha} + P^*))$ and $0 < \rho_i < 1$.

Let x_i denote the 0-1 variable indicating whether the service provider invests in application domain A_i. If so, $x_i = 1$; otherwise, $x_i = 0$. Let C_i be the fund allocated to the ith application domain A_i and R_i be the revenue from serving A_i. The optimization problem can be formulated as follows, i.e., maximizing

$$\begin{aligned} \mathit{Pro} &= \sum_{i=1}^{n}(R_i - C_i)x_i \\ &= \sum_{i=1}^{n} \lambda_i a_i \overline{r_i}\left(1 - \Pi_q^i e^{-(1-\rho_i)m_i\mu_i d}\right) x_i \\ &\quad - \sum_{i=1}^{n}\left(m_i(\beta + \delta P^*) + \delta\xi\lambda_i \overline{r_i} s_i^{\alpha-1}\right) x_i, \end{aligned} \tag{5}$$

subject to

$$\begin{gathered} \sum_{i=1}^{n}\left(m_i(\beta + \delta P^*) + \delta\xi\lambda_i \overline{r_i} s_i^{\alpha-1}\right) x_i \le F, \\ 0 < \rho_i < 1, \forall i = 1, 2, ..., n. \end{gathered} \tag{6}$$

This optimization problem has $3n$ variables, which are $m_1, ..., m_n$, $s_1, ..., s_n$, $x_1, ..., x_n$, and it is a combination of discrete optimization and continuous optimization problems. Hence, there is absolutely no closed-form solution. This problem seems like a knapsack problem, but it is much more complicated than knapsack problems. In a traditional knapsack problem, the weight and the value of all objects are known and the objects are divisible. In addition, the value of an object after divided is proportional to its weight. However, in this study, the fund allocating problem cannot be classified as this kind of knapsack problem because the profit obtained in each domain is not proportional to the fund invested to that domain. Similarly, the fund allocating problem does not belong to 0-1 knapsack problem because each domain can be invested with arbitrary fund. To solve the optimization problem, this paper contributes to develop a heuristic algorithm to solve the problem by translating it into a series of 0-1 knapsack problems.

In Table 1, we summarize all the notations used in the paper to improve the readability.

4 A Heuristic Algorithm

In this section, a heuristic algorithm is developed to solve the *fund-constrained profit optimization* problem. This algorithm consists of four phases. First, the relationship between the investment amount and the optimal profit is analyzed for each multi-server system. Given an arbitrary invested fund, the partial derivative is adopted to get the optimal multi-server configuration and the optimal profit.

TABLE 1: Notations used in this paper.

Notation	Description
A_i	the ith application domain
S_i	the multiserver system configured for application A_i
m_i	the server size of system S_i
s_i	the server speed of system S_i
λ_i	task arrival rate of A_i
r_i	task execution requirements with mean $\overline{r_i}$
μ_i	average service rate of system S_i
ρ_i	the server utilization of S_i
π_k^i	the probability that k requests are in the system S_i
F	the total budget
C_i	the investment amount allocated to system S_i
ω	the waiting time of requests
a_i	the price per one billion instructions of application domain A_i
d	the maximum waiting time of requests
$\mathcal{E}(r,\omega)$	the service charge of a request with r billion instructions and waiting time ω
$\mathcal{E}_i$	the expected charge to a service request of A_i
Pro_i	the profit of S_i
Pro	the total profit
β	the rental price of single server
δ	the electricity price

Second, a simple greedy algorithm attempting to maximize the profit is designed to determine the initial investment scheme. Third, we adjust the number of invested application domains, the investment amount allocated to each domain, and the configuration of each multi-server system. Last, we allocate the remaining fund to increase profit further. In the following subsections, the four phases are presented in detail.

4.1 Profit Maximization of Single Domain

To analyze the relationship between the investment amount and the optimal profit, we build a profit maximization model aiming at the multi-server system S_i. The profit of system S_i, denoted as Pro_i, is affected mainly by the investment amount C_i and the server configuration (the server size m_i and the speed s_i), which is calculated as:

$$Pro_i = \lambda_i a_i \overline{r_i}\left(1-\Pi_q^i e^{-(1-\rho_i)m_i\mu_i d}\right) - C_i, \tag{7}$$

where

$$C_i = m_i(\beta + \delta P^*) + \delta\xi\lambda_i\overline{r_i}s_i^{\alpha-1}. \tag{8}$$

In order to simplify the following deduction, we use the following closed-form approximation $\sum_{k=0}^{m_i-1}\frac{(m_i\rho_i)^k}{k!} \approx e^{m_i\rho_i}$ and $m_i! \approx \sqrt{2\pi m_i}(\frac{m_i}{e})^{m_i}$ [1, 39]. Therefore, we get the following closed-form expression of Π_q^i:

$$\Pi_q^i = \left(\sqrt{2\pi m_i}(1-\rho_i)\left(\frac{e^{\rho_i}}{e\rho_i}\right)^{m_i} + 1\right)^{-1}.$$

The profit is represented as:

$$Pro_i = \lambda_i a_i\overline{r_i} - \frac{\lambda_i a_i \overline{r_i} e^{-(1-\rho_i)m_i\mu_i d}}{\sqrt{2\pi m_i}(1-\rho_i)\left(\frac{e^{\rho_i}}{e\rho_i}\right)^{m_i} + 1} - C_i. \tag{9}$$

According to Eq. (8), we know that the speed s_i can be determined by the investment amount C_i and the server size m_i, which is formulated as

$$s_i = \sqrt[\alpha-1]{\frac{C_i - m_i(\beta+\delta P^*)}{\delta\xi\lambda_i\overline{r_i}}}. \tag{10}$$

Substituting Eq. (10) to Eq. (9), the profit can be formulated as a function of C_i and m_i as:

$$Pro_i = \lambda_i a_i \overline{r_i} - \frac{\lambda_i a_i \overline{r_i} D_1}{\sqrt{2\pi m_i} D_2 D_3 + 1} - C_i, \tag{11}$$

where

$$D_1 = e^{\left(\lambda_i - \sqrt[\alpha-1]{\frac{C_i - m_i(\beta+\delta P^*)}{\delta\xi\lambda_i\overline{r_i}}}\frac{m_i}{\overline{r_i}}\right)d},$$

$$D_2 = 1 - \frac{\lambda_i\overline{r_i}}{\sqrt[\alpha-1]{\frac{C_i - m_i(\beta+\delta P^*)}{\delta\xi\lambda_i\overline{r_i}}}m_i},$$

$$D_3 = \left(\frac{e^{\frac{\lambda_i\overline{r_i}\sqrt[\alpha-1]{\delta\xi\lambda_i\overline{r_i}}}{\sqrt[\alpha-1]{C_i - m_i(\beta+\delta P^*)}m_i}}}{e\frac{\lambda_i\overline{r_i}\sqrt[\alpha-1]{\delta\xi\lambda_i\overline{r_i}}}{\sqrt[\alpha-1]{C_i - m_i(\beta+\delta P^*)}m_i}}\right)^{m_i}.$$

4.1.1 Optimal Size of Single Domain

Given α, β, γ, ξ, P^*, λ_i, $\overline{r_i}$, and the investment amount C_i, the problem is to find m_i such that the profit Pro_i is maximized. Because $0 < \rho_i < 1$, the changing interval of m_i must satisfy the following equation:

$$\begin{cases} m_i > 0; \\ s_i = \sqrt[\alpha-1]{\dfrac{C_i - m_i(\beta+\delta P^*)}{\delta\xi\lambda_i\overline{r_i}}} > 0; \\ \rho_i = \dfrac{\lambda_i\overline{r_i}}{m_i\sqrt[\alpha-1]{\frac{C_i - m_i(\beta+\delta P^*)}{\delta\xi\lambda_i\overline{r_i}}}} < 1. \end{cases} \tag{12}$$

Simplifying Eq. (12), we can get the equation of m_i as follows,

$$\begin{cases} 0 < m_i < C_i/(\beta + \delta P^*); & \text{(13a)} \\ m_i^{\alpha}(\beta+\delta P^*) - m_i^{\alpha-1}C_i + \delta\xi(\lambda_i\overline{r_i})^{\alpha} < 0. & \text{(13b)} \end{cases}$$

In order to solve Eq. (13b), we use Matlab to solve equation

$$\Delta = m_i^{\alpha}(\beta+\delta P^*) - m_i^{\alpha-1}C_i + \delta\xi(\lambda_i\overline{r_i})^{\alpha} = 0,$$

and then find the solution interval $[m_{min}, m_{max}]$ such that $\Delta'(m_{min}) < 0$ and $\Delta'(m_{max}) > 0$, where

$$\Delta'(m_i) = \alpha(\beta+\delta P^*)m_i^{\alpha-1} - C_i(\alpha-1)m_i^{\alpha-2}.$$

Combined with Eq. (13a), the effective interval of m is $[\max\{0, m_{min}\}, \min\{m_{max}, C_i/(\beta+\delta P^*)\}]$. For example, let $\beta = 1.5$, $\delta = 0.1$, $P^* = 2$, $C_i = 26.5$, $\lambda_i = 10$, $\overline{r_i} = 1$, $a_i = 20$, and $d = 5$, the solution intervals of Eq. (13a) and Eq. (13b) are [0,15.5852] and [5.4830,10.1052], respectively. Then, the effective interval of m_i is [5.4830,10.1052].

According to our calculation using Eq. (11), we observe that the profit is a convex function of m_i within the effective interval, which is shown in Fig. 3. Therefore, there must be an optimal m_i such that the profit is maximized.

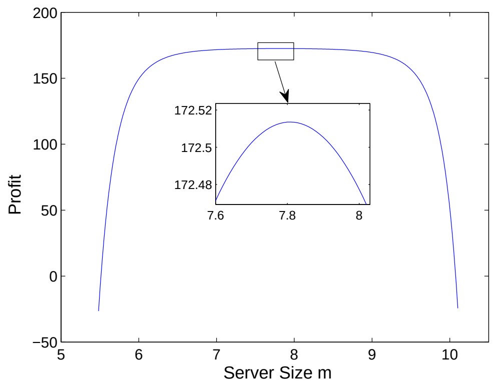


Fig. 3: The profit versus the server size under given budget.

To maximize Pro_i, we need to find m_i such that

$$\begin{aligned}\frac{\partial Pro_i}{\partial m_i} =& \frac{\sqrt{2\pi m_i}\lambda_i a_i \overline{r_i} D_1 \left(\frac{\partial D_2}{\partial m_i} D_3 + \frac{\partial D_3}{\partial m_i} D_2\right)}{(\sqrt{2\pi m_i} D_2 D_3 + 1)^2} \\ &- \frac{\lambda_i a_i \overline{r_i}\frac{\partial D_1}{\partial m_i}}{\sqrt{2\pi m_i} D_2 D_3 + 1} \\ =&0,\end{aligned} \tag{14}$$

where

$$\frac{\partial D_1}{\partial m_i} = \frac{dD_1}{\overline{r_i}} \sqrt[\alpha-1]{\frac{C_i - m_i(\beta+\delta P^*)}{\delta\xi\lambda_i\overline{r_i}}} \left(\frac{m_i(\beta+\delta P^*)}{(\alpha-1)(C_i - m_i(\beta+\delta P^*))} - 1\right),$$

$$\frac{\partial D_2}{\partial m_i} = \frac{\lambda_i\overline{r_i}}{m_i^2} \sqrt[\alpha-1]{\frac{\delta\xi\lambda_i\overline{r_i}}{C_i - m_i(\beta+\delta P^*)}} \left(1 - \frac{m_i(\beta+\delta P^*)}{(\alpha-1)(C_i - m_i(\beta+\delta P^*))}\right),$$

$$\begin{aligned}\frac{\partial D_3}{\partial m_i} =& \left(\frac{\beta+\delta P^*}{(\alpha-1)(C_i - m_i(\beta+\delta P^*))}\left(\lambda_i\overline{r_i} \sqrt[\alpha-1]{\frac{\lambda_i\overline{r_i}\delta\xi}{C_i - m_i(\beta+\delta P^*)}}\right.\right. \\ &\left.\left. - m_i\right) - \ln\left(\frac{(\lambda_i\overline{r_i})\sqrt[\alpha-1]{\lambda_i\overline{r_i}\delta\xi}}{\sqrt[\alpha-1]{C_i - m_i(\beta+\delta P^*)}m_i}\right)\right) D_3.\end{aligned}$$

It is apparent that there is no closed-form solution to m_i, so we use the standard bisection method to find m_i numerically [40]. The algorithm is shown as Alg. 1, and the decreasing interval $[lm, um]$ is set as $[\max\{0, m_{min}\}, \min\{m_{max}, C_i/(\beta+\delta P^*)\}]$. It should be noticed that the solution of Eq. (13) might be null, which means that the budget is too low to configure enough capacity for market demand. In this case, the profit must be negative, and it is unnecessary to find the optimal configuration.

4.1.2 Optimal Fund of Single Domain

Given α, β, γ, ξ, P^*, λ_i, $\overline{r_i}$, and m_i, the problem is to find C_i such that the profit Pro_i is maximized.

Given m_i, the effective interval of C_i must satisfy the following equation:

$$\begin{cases} C_i > 0; \\ s_i = \sqrt[\alpha-1]{\dfrac{C_i - m_i(\beta+\delta P^*)}{\delta\xi\lambda_i\overline{r_i}}} > 0; \\ \rho_i = \dfrac{\lambda_i\overline{r_i}}{m_i\sqrt[\alpha-1]{\frac{C_i - m_i(\beta+\delta P^*)}{\delta\xi\lambda_i\overline{r_i}}}} < 1. \end{cases} \tag{15}$$

Algorithm 1 Optimal Server Size under Given Budget

Input: C_i, λ_i, $\overline{r_i}$.
Output: m_i, Pro_i.

```
1: Find the decreasing interval $[lm, um]$ of $\partial Pro_i/\partial m_i$ such
   that $\partial Pro_i(lm)/\partial m_i > 0$ and $Pro_i(um)/\partial m_i < 0$;
2: while $um - lm > \varepsilon$ do
3:     $mid \leftarrow (lm + um)/2$;
4:     if $\partial Pro_i(mid)/\partial m_i > 0$ then
5:         $lm \leftarrow mid$;
6:     else
7:         if $\partial Pro_i(mid)/\partial m_i < 0$ then
8:             $um \leftarrow mid$;
9:         end if
10:    else
11:        $m_i \leftarrow mid$;
12:        break;
13:    end if
14: end while
15: $m_i \leftarrow (lm + um)/2$;
16: $Pro_i \leftarrow$ Calculated using Eq. (11).
```

Simplifying Eq. (15), we can get the following inequality of C_i:

$$\begin{cases} C_i > 0; & (16a) \\ C_i > m_i(\beta+\delta P^*); & (16b) \\ C_i > m_i(\beta+\delta P^*) + \dfrac{\delta\xi(\lambda_i\overline{r_i})^\alpha}{m_i{}^{\alpha-1}}. & (16c) \end{cases}$$

Through Eq. (16), the effective interval of C_i can be solved as $[m_i(\beta+\delta P^*)+\frac{\delta\xi(\lambda_i\overline{r_i})^\alpha}{m_i{}^{\alpha-1}}, +\infty]$. Let $m_i = 8$, and other parameters are the same as those in Fig. 3. The effective interval of C_i is [25.3740, $+\infty$], and the changing trend of profit with increasing C_i is shown in Fig. 4. It is obvious that the optimal profit can be obtained in the extremal point.

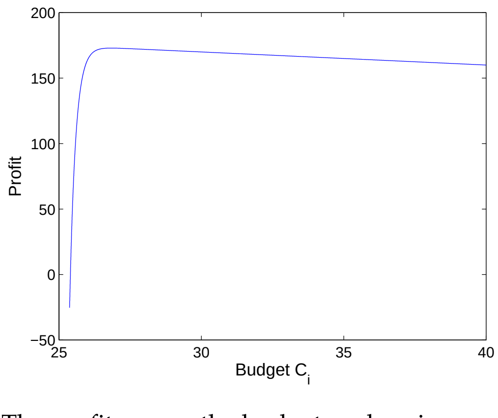


Fig. 4: The profit versus the budget under given server size.

To maximize Pro_i under given m_i, we need to find C_i such that $\partial Pro_i/\partial C_i = 0$, where $\partial Pro_i/\partial C_i$ is derived as follows:

$$\begin{aligned}\frac{\partial Pro_i}{\partial C_i} =& \frac{\sqrt{2\pi m_i}\lambda_i a_i \overline{r_i} D_1 \left(\frac{\partial D_2}{\partial C_i} D_3 + \frac{\partial D_3}{\partial C_i} D_2\right)}{(\sqrt{2\pi m_i} D_2 D_3 + 1)^2} \\ &- \frac{\lambda_i a_i \overline{r_i}\frac{\partial D_1}{\partial C_i}}{\sqrt{2\pi m_i} D_2 D_3 + 1} - 1 \\ =&0,\end{aligned} \tag{17}$$

where

$$\frac{\partial D_1}{\partial C_i}=\frac{-m_i d D_1}{\overline{r_i}(\alpha-1)(C_i-m_i(\beta+\delta P^*))}\sqrt[\alpha-1]{\frac{C_i-m_i(\beta+\delta P^*)}{\delta\xi\lambda_i\overline{r_i}}},$$

$$\frac{\partial D_2}{\partial C_i}=\frac{\lambda_i\overline{r_i}}{(\alpha-1)m_i(C_i-m_i(\beta+\delta P^*))}\sqrt[\alpha-1]{\frac{\delta\xi\lambda_i\overline{r_i}}{C_i-m_i(\beta+\delta P^*)}},$$

$$\frac{\partial D_3}{\partial C_i}=\frac{D_3}{(\alpha-1)(C_i-m_i(\beta+\delta P^*))}\Bigg(m_i-\lambda_i\overline{r_i}\sqrt[\alpha-1]{\frac{\delta\xi\lambda_i\overline{r_i}}{C_i-m_i(\beta+\delta P^*)}}\Bigg).$$

We can use the bisection method to find the optimal C_i, too.

4.1.3 Optimal Fund and Size of Single Domain

To find the optimal C_i^* and m_i^*, we solve Eq. (14) and Eq. (17) by a nested bisection search procedure which is used in [1]. The algorithm is given as Alg. 2.

Algorithm 2 Optimal Budget of Single Domain

```
Input: $\lambda_i$, $\overline{r_i}$.
Output: $C_i^*$, $m_i^*$, $Pro_i^*$.
 1: Observe and find the interval of budget $[lC, uC]$ in which
    the optimal profit is obtained;
 2: while $uC - lC > \varepsilon$ do
 3:     $C_{mid} = (lC + uC)/2$;
 4:     Call Alg. 1 to calculate the optimal $m_{opt}$ under budget
        $C_{mid}$;
 5:     if $\partial Pro_i(C_{mid}, m_{opt})/\partial C_i > 0$ then
 6:         $lC \leftarrow C_{mid}$;
 7:     else
 8:         $uC \leftarrow C_{mid}$;
 9:     end if
10: end while
11: $C_i^* = (lC + uC)/2$;
12: Call Alg. 1 to calculate the optimal $m_i^*$ under budget $C_i^*$;
13: $Pro_i^* \leftarrow$ Calculated using Eq. (11).
```

In Tab. 2, we demonstrate the optimal profit and the optimal configuration of different market demand. Here, assuming that $a_i = 20$ cents per one billion instructions, $d = 10$ seconds, $\beta = 1.5$ cents per second, $\delta = 0.1$ cents per Watt, and $P^* = 2$ Watts. For $\lambda = 5, 10, 15, 20, 25$, and $\overline{r} = 1$, we display the optimal profit Pro^* and the corresponding investment and configuration.

TABLE 2: The optimal investment, configuration, and profit versus λ.

λ	m^*	s^*	C^*	Pro^*	ROI
5	4.1271	1.4770	13.9721	85.8178	6.1421
10	7.9081	1.4214	26.8324	172.9453	6.4454
15	11.6642	1.3998	39.6066	260.1655	6.5687
20	15.4102	1.3880	52.3454	347.4234	6.6371
25	19.1505	1.3805	65.0648	434.7017	6.6811

4.2 Initial Investment Scheme

After the optimal investment amount C_i^* and the server configuration for each multi-server system S_i are calculated, the following work is to select multiple initial investment domains. Once the investment and the corresponding profit of all domains are known, the domain selecting problem is actually a 0-1 knapsack problem which can be solved by any existing algorithm. In this paper, we choose two existing methods – the greedy strategy and dynamic programming – to solve the 0-1 knapsack problem, respectively. Greedy strategy mostly fails to find the global optimal solution, but it is quick to think up and has low time complexity. In contrast, dynamic programming can achieve the global optimal solution but its time complexity is $O(2^n)$. The algorithm determining the initial investment scheme is given as Alg. 3 and the methods solving 0-1 knapsack problem are shown as Alg. 4 and Alg. 5.

Algorithm 3 Initial Investment Scheme

```
Input: $F$, $n$, $Pro_{1:n}^*$, $C_{1:n}^*$, $m_{1:n}^*$.  // $m_{1:n}^* \Leftrightarrow \{m_1^*, m_2^*, ..., m_n^*\}$
Output: The initial investment and configuration scheme, in-
    cluding $x_{1:n}$, $C_{1:n}$, $m_{1:n}$, $Pro_{1:n}$, totalFund, and totalPro.
1: $[x_{1:n}, totalPro] \leftarrow knapsack\_problem(Pro_{1:n}^*, C_{1:n}^*, F)$;
2: for $i = 1 : n$ do
3:     $C_i = x_i \cdot C_i^*$, $Pro_i = x_i \cdot Pro_i^*$, $m_i = x_i \cdot m_i^*$;
4: end for
5: $totalFund = \sum_{i=1}^n C_i$.
```

Algorithm 4 *knapsack_problem* (Greedy Strategy)

```
Input: $Pro_{1:n}$, $C_{1:n}$, $F$.
Output: $x_{1:n}$, totalPro.
 1: Initialize $x_i = 0$ for $\forall i$;
 2: $totalFund = 0, totalPro = 0$;
 3: Rank the $n$ application domains in the non-increasing order
    of the Pro values;
 4: Set $i = 1$;
 5: while $i \leq n$ do
 6:     curr $\leftarrow$ the index of the application domain with maximal
        Pro;
 7:     if $totalFund + C_{curr} \leq F$ then
 8:         $x_{curr} \leftarrow 1$;
 9:         $totalPro \leftarrow totalPro + Pro_{curr}$;
10:         $totalFund \leftarrow totalFund + C_{curr}$;
11:     end if
12:     Delete the curr-th application;
13:     $i++$;
14: end while
```

In Alg. 4, greedy strategy is adopted to solve 0-1 knapsack problem. We first rank the applications in the non-increasing order of the optimal profit values *Pro** (line 3). Then, the application domain with the maximal optimal profit is considered (line 6). If the remaining funding can afford the cost, invest it (lines 8-9); otherwise, consider the next application domain with a smaller *Pro**.

In Alg. 5, dynamic programming is adopted to solve the 0-1 knapsack problem. The optimal solution can be calculated recursively by:

$$f(i,F)=\begin{cases}\max\{f(i+1,F), f(i+1,F-C_i)+Pro_i\}, & F\geq C_i;\\ f(i+1,F), & F\leq C_i;\end{cases}\tag{18}$$

and

$$f(n,F)=\begin{cases}Pro_n, & F\geq C_n;\\ 0, & F\leq C_n;\end{cases}\tag{19}$$

Algorithm 5 *knapsack_problem* (Dynamic Programming)

```
Input: $C_{i:n}$, $Pro_{i:n}$, $F$.
Output: $x_{i:n}$, $totalPro$.
1: if $i < n$ then
2:   if $F \geq C_i$ then
3:     $[x_{i:n}, totalPro] = \max\{knapsack\_problem(C_{i+1:n}, Pro_{i+1:n},$
4:         $F), knapsack\_problem(C_{i+1:n}, Pro_{i+1:n}, F-C_i)$
5:         $+Pro_i\}$;
6:   else
7:     $[x_{i:n}, totalPro] = knapsack\_problem(C_{i+1:n}, Pro_{i+1:n}, F)$;
8:   end if
9: else
10:   if $i == n$ then
11:     if $F \geq C_n$ then
12:       $totalPro = Pro_n$, $x_n = 1$;
13:     else
14:       $totalPro = 0$, $x_n = 0$;
15:     end if
16:   end if
17: end if
```

Algorithm 6 Adjustment of Investment Domains

```
Input: $F$, $n$, $C^*_{1:n}$, $\lambda_{1:n}$, $\overline{r_{1:n}}$, $totalFund$, and $totalPro$.
Output: The investment and configuration scheme after adjustment, including $x_{1:n}$, $C_{1:n}$, $m_{1:n}$, $Pro_{1:n}$, $totalFund$, and $totalPro$.
1: Set $Step$ and $\aleph$;
2: for $i = 1 : \aleph$ do
3:   for $j = 1 : n$ do
4:     $C_j^{'} = C_j^* - C_j^* \cdot Step \cdot i$;
5:     Call Alg. 1 with parameters $(C_j^{'}, \lambda_j, \overline{r_j})$ to get $m_j^{'}$ and $Pro_j^{'}$;
6:   end for
7:   $[x_{1:n}^{'}, totalPro^{'}] \leftarrow knapsack\_problem(F, Pro_{1:n}^{'}, C_{1:n}^{'})$;
8:   if $totalPro < totalPro^{'}$ then
9:     $totalPro \leftarrow totalPro^{'}$;
10:     $x_i \leftarrow x_i^{'}$, $C_i \leftarrow x_i \cdot C_i^{'}$, $m_i \leftarrow x_i \cdot m_i^{'}$, $Pro_i \leftarrow x_i \cdot Pro_i^{'}$ for $\forall i$;
11:     $totalFund = \sum_{i=1}^{n} C_i$.
12:   end if
13: end for
```

where $f(i, F)$ denotes the maximal profit when the total funding is F and the selectable domains are from i to n.

After the first phase, the generated initial investment scheme is an available and good solution.

4.3 Adjustment of Investment Domains

After the initial investment scheme is determined by Alg. 3, we further find if there is a scheme which produces more profit than the initial one. In the initial investment scheme, each application domain is invested with enough funding to guarantee its maximal profit. Obviously, there is a lot of funding left but not enough to invest in another application domain. So, can we spare a little funding from the selected domains and exploit another investment domain with the remaining funding? Doing so can probably increase the total profit. Based on the idea, we try to find a better investment scheme by slightly adjusting the investment in each domain in the second phase. The process is described as Alg. 6.

In line 1 of Alg. 6, two important parameters *Step* and $\aleph$ are set. *Step* represents the ratio of the reduced funding in each loop and the optimal investment, and $\aleph$ is the number of loops which is related with *Step*. The smaller the *Step* is, the greater the $\aleph$ should be set; hence, more accurate solutions can be obtained. In each loop, the funding invested to each multi-server system is reduced (line 4) and the optimal profit is calculated based on the reduced investment (line 5). After the optimal profits of all systems are calculated under the reduced investments, Alg. 3 is called to select the invested applications, and the total profit of the new investment scheme is calculated (line 7). Then, the profit of the initial investment scheme is compared with that of the new investment scheme after adjustment (line 8). If the latter is greater than the former, replace the initial investment scheme with the new one. Other parameters such as the profit and the server size are updated correspondingly (lines 9-11).

In this phase, the funding invested to each domain is gradually reduced on the basis of the optimal investment. Hence, the searching direction is certain and single, which makes searching much easier. Moreover, the searching times and the accuracy can be controlled by parameter *Step* and $\aleph$, hence the algorithm can be finished quickly whether a better solution can be found.

4.4 Final Fund Allocation

After the third phase in Alg. 6, the final invested applications are determined. However, there is still some unallocated funding. The following phase is to allocate the remaining funding to increase profit further, which is shown as Alg. 7.

First, the n application domains are ranked in the non-increasing order of C_i values (line 1). That means that the remaining funding is first invested in the application domain with larger C_i value. The reason is explained as follows: According to Alg. 6, we know that the application domain with the maximal C_i reduces the most investment amount, hence, leads to the greatest loss of profit among all fields. Investing funding in this application domain can obtain the maximal profit growth. Of course, the funding allocated to an application domain cannot exceed its optimal investment; otherwise, the profit would be decreased. If the remaining funding can be invested to the current application domain and does not exceed its optimal investment, allocate it all to the application domain and update its configuration parameters (lines 7-11). If the remaining funding exceeds the investment demand of the current application domain, allocate the required fund to it and turn to next application domain (lines 13-18).

5 Performance Analysis and Comparison

In this section, a series of numerical calculations are conducted to verify the performance of our algorithm.

5.1 Performance Analysis

5.1.1 *Single Application Domain*

Given an application domain, its profit is affected by task arrival rate and investment. In our first group of calculations, we aim at observing the changing trend of profit of a single application domain with an increasing level of investment under different task arrival rates. The parameters are the same as those used in Table 2. For an application domain

Algorithm 7 Final Fund Allocation

```
Input: $F$, $n$, $C^*_{1:n}$, $\lambda_{1:n}$, $\overline{r_{1:n}}$, $x_{1:n}$, $C_{1:n}$, $m_{1:n}$, $Pro_{1:n}$, $totalFund$, and $totalPro$.
Output: The investment and configuration scheme after balance allocation, including $x_{1:n}$, $C_{1:n}$, $m_{1:n}$, $Pro_{1:n}$, $totalFund$, and $totalPro$.
1:  Rank the $n$ application domains in the non-increasing order of the $C_i$ values;
2:  $leftFund \leftarrow F - totalFund$;
3:  for $i = 1 : n$ do
4:      $curr \leftarrow$ the index of the application domain with maximal $C_i$;
5:      if $x_{curr} == 1$ then
6:          if $leftFund \le C^*_i - C_i$ then
7:              $totalFund \leftarrow F$;
8:              $totalPro \leftarrow totalPro - Pro_{curr}$;
9:              $C_{curr} \leftarrow C_{curr} + leftFund$;
10:             Call Alg. 1 with parameters $(C_{curr}, \lambda_{curr}, \overline{r_{curr}})$ to get $m_{curr}$ and $Pro_{curr}$;
11:             $totalPro \leftarrow totalPro + Pro_{curr}$;
12:         else
13:             $totalFund \leftarrow totalFund - C_{curr}$;
14:             $totalPro \leftarrow totalPro - Pro_{curr}$;
15:             $C_{curr} \leftarrow C^*_{curr}$, $m_{curr} \leftarrow m^*_{curr}$, $Pro_{curr} \leftarrow Pro^*_{curr}$;
16:             $totalFund \leftarrow totalFund + C_{curr}$;
17:             $totalPro \leftarrow totalPro + Pro_{curr}$;
18:             $leftFund \leftarrow F - totalFund$;
19:         end if
20:     end if
21: end for
```

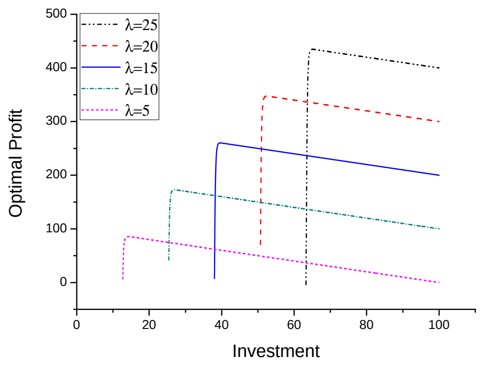


Fig. 5: Optimal profit versus total investment.

with $\overline{r} = 1$ and λ = 5, 10, 15, 20, 25, we display the maximal profit for different investment in Fig. 5. From the figure we can see that the maximal profit obtained by a multi-server system reaches the peak at a certain investment, and then decreases with increasing investment. That is because marginal profit is decreasing with the increasing investment and even becomes a negative value, which leads to the decrease of total profit. Moreover, investing in an application domain with greater market requirements can produce more profit than investing in the small ones. When an investor has sufficient funding and only invests in one domain, selecting an application domain with the highest λ will lead to the greatest profit. Of course, the investment is not *the more the better*. There exists a best investment which leads to the optimal profit, and more or less investment would lead to a loss of profit.

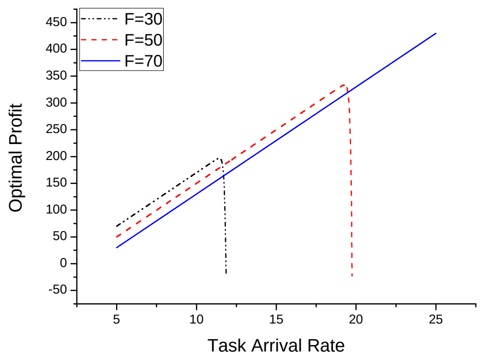


Fig. 6: Optimal profit versus λ.

In the second group of calculations, we observe how the profit changes with the increasing workload under the given investment. For the given investment 30, 50, 70, the maximal profit for different λ is shown in Fig. 6. The figure shows that the profit is increasing with the increasing λ at the beginning and reaches the peak at a point. After the peak, the increase of task arrival rate leads to the sharp decrease of profit, because the maximal capacity of the system under the given fund is limited and the waiting time of a great amount of tasks exceeds the deadline. Another phenomenon we can find from the figure is that the profit of $F = 30$ is greater than that of $F = 50$ and 70 when λ is small, which reinforces the trend observed in Fig. 5. Hence, we can conclude that for an application domain with a given task arrival rate, adding investment can increase the profit - but not always. Hence, when an investor has limited funding and only invests one domain, the application domain should be selected according to the available funds. The λ value is not the more the better. A higher λ would lead to less profit, even negative profit.

5.1.2 Multiple Application Domains

Since this paper seeks to find an optimal investment portfolio from a variety of application domains to achieve profit maximization, we will verify the performance of the proposed fund allocation scheme for multiple application domains. First, given the total number of the application domains and the total investment, we observe the changing trend along with the increasing average workload. Here, the number of the application domains n is set as 5, 10, and 15. The task arrival rates are set as 8 to 12 in step of 1 for the 5-application-domain set, 5.5 to 14.5 in step of 1 for the 10-application-domain set, and 3 to 17 in step of 1 for the 15-application-domain set. The $\overline{r_i}$ is set as 1 and the total investment is increasing from 100 to 300.

In Fig. 7(a), we give the statistics on how many application domains are invested with the increasing investment when n is equal to 5, 10, and 15, respectively. It is apparent that the number of invested application domains shows a ladder like increase. In other words, when the investment amount is in a small interval, the number of invested domains does not change.

In Fig. 7(b), the actual consumed funding is presented. From the figure, we can see that our investment strategy does not require spending all the available funds. It designs the optimal investment scheme and calculates the corresponding budget according to different market demands. When $n = 5$, the optimal profit is achieved when the total investment is 134.0808, in which investment is made in all the five application domains and each one can achieve its maximal profit. Hence, when the available investment is further increased, the real consumed investment will not be changed by our proposed investment strategy. Moreover, even if the available investment is less than the optimal investment, for example 125 for $n = 5$, there is still a portion of surplus funds. The reasons are explained as follows. Assuming that k $(k < n)$ application domains are selected as the invested domains and the optimal funds are allocated to them such that each one can achieve the maximal profit. There are some funds unallocated because they are not enough to invest in one more application domain. And if allocating them to those invested application domains, the total profit is reduced. Hence, the best choice is to leave this part of the funds unused. In addition, the more the application domains there are, the closer the available investment and the real investment are. That is because more combinations provide more flexibility when determining the investment scheme.

Fig. 7(c) shows the changing trend of the optimal profit that a service provider can achieve with the increasing available funds. It is shown that when the investment reaches a critical point, any further increase of the investment will not affect the total profit, because the extra funds are unused by our investment strategy. Overall speaking, the changing trends of the profit are similar to that of the consumed funds.

5.2 Performance Comparison

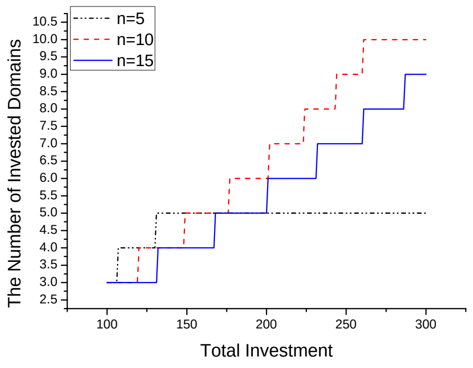
5.2.1 Inter Comparison

In order to verify the performance of the investment schemes generated by our algorithm, we compare our algorithm with six other investment strategies which are introduced as follows:

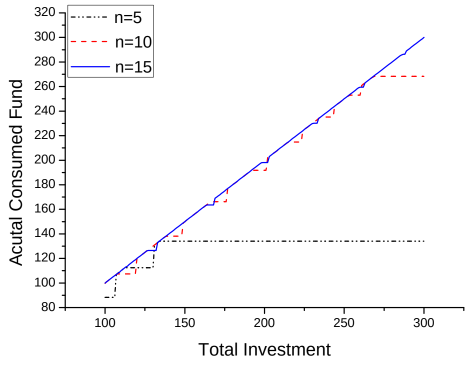
- **Application selection strategy.** Rank the application domains in the non-increasing order of $\lambda_i \overline{r_i}$ values, and select the first $n/3$, $n/2$, or n application domains in which to invest.
- **Fund allocation strategy.** Allocate the funds to the selected application domains equally or in proportional to the $\lambda_i \overline{r_i}$ value. Equal allocation and proportional allocation are two common methods which are adopted in many aspects and are quick to think up.

Combining the above strategies, we produce six investment strategies, naming, $Equ1/3$, $Equ1/2$, $Equ1/1$, $Pro1/3$, $Pro1/2$, and $Pro1/1$, respectively. For example, $Equ1/3$ means selecting the first $n/3$ domains and allocating the funds equally. Because two different methods are adopted in our algorithm to solve the 0-1 knapsack problem, we mark them as Greedy algorithm with Adjustment (*Greedy_A*) and Dynamic Programming with Adjustment (*DP_A*), respectively.

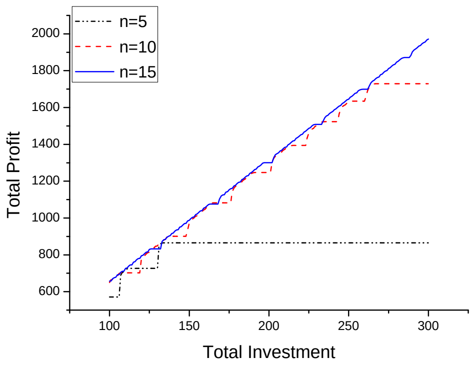
Fig. 8 presents the profit comparison of the eight investment schemes with the increasing investment. In the calculation, the number of application domains is set as 6, the task arrival rates are set from 20 to 25 in step of 1, and the task requirements are $\overline{r_i} = 1$. The total investment fund is increasing from 100 to 400.



(a)



(b)



(c)

Fig. 7: Optimal profit and the number of invested domains versus total investment.

From the figure, it is easy to conclude that the investment schemes generated adopting our investment strategy are better than the compared schemes, because their profits are always greater than the other six schemes. That is because the other six investment strategies invest in a fixed number of application domains, while our investment strategy determines the number of invested domains adaptively according to different factors such as the available funds, the number of application domains, the task arrival rate of each application, and so on. Hence, our strategy can generate more profit than the other six strategies.

The curves of *Greedy_A* and *DP_A* show the staircase-like increasing trends with the increase of available funds

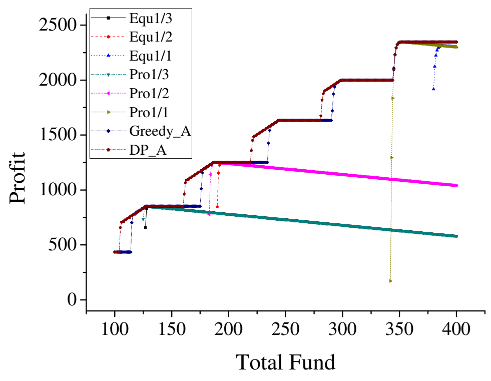


Fig. 8: Optimal profit versus total investment.

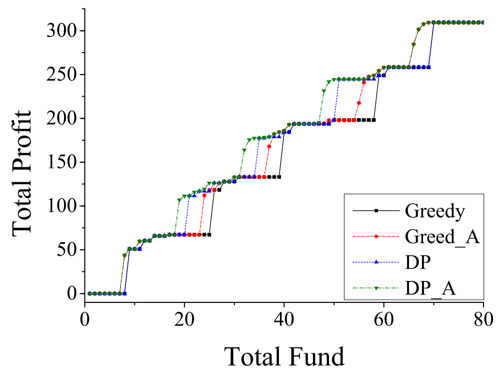
and both of them reach a steady state after surpassing a boundary. However, for the other six strategies, their profits increase first and then decrease with the increasing available funds. That is because the six strategies allocate all of the available funds to the selected application domains but do not consider if they need so much funding. Once the investment exceeds the optimal demand, increasing investment will not generate more profit but will have the opposite impact.

Moreover, our investment strategy has another advantage compared with the six other strategies. Given an arbitrary number of application domains, our strategy can determine the number of invested domains flexibly according to the available funds while the other six cannot. Which application domains are invested in is determined for the six strategies once the number of application domains and their task arrival rates are given. If the available funding is small, the funds allocated to each application domain might be not enough to generate profit; hence, the total profit would be negative.

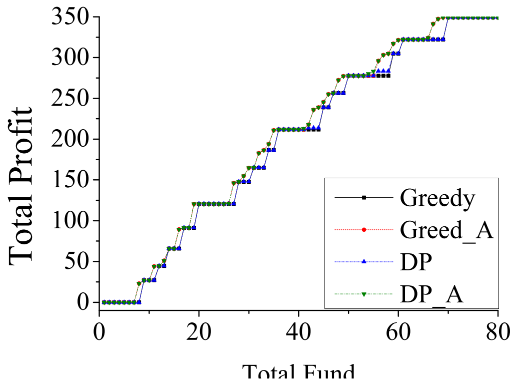
5.2.2 Intra Comparison

Our algorithm generates a better investment scheme by slight adjustment based on an initial investment scheme. In the following, we compare the profit of the final schemes with the initial schemes to test how much profit can increase after adjustment. Similarly, we label the two final schemes as *Greedy_A* and *DP_A*, and the initial schemes *Greedy* and *DP*.

In this group of calculations, we set the number of application domains as 5, the λ_i values are set as 3, 4, 5, 6, and 7 and the price a_i values are set as 20, 18, 16, 14, and 12, respectively. The total funding is changing from 1 to 80. Fig. 9(a) gives the results and shows that our strategies with adjustment *DP_A* and *Greedy_A* can produce more profit compared with the corresponding strategies without adjustment *DP* and *Greedy* in more than 36.29 and 30 percent of cases, respectively, and the profit gap between them changes periodically. For example, the profit gap between *DP* and *DP_A* is zero when the total investment is between 42 and 46, and then it becomes very large when the total investment is between 47 and 50. When the total investment increases further, the profit gap becomes small to zero again at 51.



(a)



(b)

Fig. 9: The profit: the initial investment vs. the final investment.

The reasons are explained as follows: When the total investment is between 42 and 46, a better investment scheme cannot be found by adjustment. When the total investment is greater than 47 but less than 50, one more application domain can be invested after adjustment. Hence, the profit of *DP_A* is much greater than *DP*. When the total investment is increasing further, *DP* and *DP_A* can invest the same number of application domains but the invested domains are different from each other, hence, *DP_A* produces only slightly more profit than *DP*. The changing trend of *Greedy* and *Greedy_A* is similar to that of *DP* and *DP_A*. We can observe from the figure that our algorithm can produce more profit when combined with dynamic programming than greedy strategy.

In the second group of calculations, the other parameters are similar with those in Fig. 9(a) except the changing trend of the task arrival rates and they are 7, 6, 5, 4, 3, respectively. The results are given in Fig. 9(b) which shows that *DP_A* and *Greedy_A* can achieve more profit than *DP* and *Greedy* in 30 percent of cases. Comparing the two figures, it can be found that when the changing trend of the task arrival rates is similar with that of the serving prices, the profit gain adopting our approach is much lower than the opposite situation. For example, when the total fund is 34, *DP_A* achieves 33.17% profit gain compared with *DP* in Fig. 9(a), while only 13.12% in Fig. 9(b). In addition, Fig. 9(b) shows that the approach adopting *DP* does not perform better than that adopting *Greedy*. Hence, the profit gain of our approach is affected by the combination of the λ value and the a value. In overall, adopting our approach can always obtain more profit.

5.3 Quality of Solutions

Our algorithm is heuristic which can only generate approximate solutions, hence, it is necessary to know how well the solutions approximate the global optimal solution. However, due to the complexity of our problem, we cannot get the global optimal solution, so the error between the optimal solution and the heuristic solutions cannot be calculated. However, when the server size m and the server speed s are discrete variables, the global optimal solutions can be solved by a brutal force searching approach. Hence, we compare our solutions with the discrete optimal solution. In the comparison, the server size m is an integer whose value range is from 1 to 20, and the server speed s is a discrete variable whose value range is from 0.2 to 2.0 in step of 0.2. Hence, we need to do some modification on the solutions obtained by our algorithm. The process is given as follows. Assumed that the optimal investment of server S_i is C_i and the corresponding optimal size and speed are m_i and s_i. Comparing the four closest configurations around (m_i, s_i), the configuration with the most profit is adopted as the practical configuration of server S_i if the consumed funds do not exceed the available funds.

In the group of comparison, 5 application domains are given and their arrival rates are set as 8, 9, 10, 11, and 12, respectively. The $\bar{r}_i$ values are set as 1. Table 3 shows the comparison of investment amount, server configuration, and profit of each domain when the available investment is 107. The results show that the profit obtained by adopting our strategy is close to the global optimal profit when the server size and server speed are discrete variables, and the solution adopting dynamic programming is much closer than that adopting greedy strategy. For example, the profit of *Greedy_A* is 670.9269, which is 2.83% less than the global optimal profit, but the profit of *DP_A* is 689.4496, which is only 0.15% less than the global optimal profit. Hence, our algorithm is a good heuristic algorithm.

For an investor who has a limited fund, he is more inclined to select and invest application domains with higher λ first because it seems to be able to make more profit. However, according to our calculation results, that is not always true. If a service provider wants to achieve the most profit under a limited fund, it is better to adopt the solution proposed in this paper, which achieves a better solution because the solution is gradually improved based on the initial solution by adjusting fund allocated to different domains.

6 Conclusions

In this paper, we have studied the fund allocation and profit maximization problem for the service providers with fund constraint. A fund constrained profit maximization model is formulated. According to the analysis, the optimization problem is too complexity to get a global optimal solution because it consists of three subproblems which are the selection of invested application domains, fund allocation, and multiserver configuration. Hence, we design a heuristic algorithm which solve the problem in three steps. An initial investment scheme is find firstly, and then adjustment is done to get a better investment portfolio. Lastly, the left fund is allocated further to get the final investment scheme. The performance evaluation of our algorithms has been performed by a series of simulations, and six other investment strategies are adopted to compare with our strategy. From the comparison results, our strategy can design the best investment scheme with the maximal profit among all the strategies.

Acknowledgments

The authors thank the anonymous reviewers for their valuable comments and suggestions. The research was partially funded by the Key Program of National Natural Science Foundation of China (Grant Nos. 61133005, 61432005), the National Natural Science Foundation of China (Grant Nos. 61370095, 61472124), National High-tech R&D Program of China (2015AA015303).

References

[1] J. Cao, K. Hwang, K. Li, and A. Y. Zomaya, “Optimal multiserver configuration for profit maximization in cloud computing,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1087–1096, 2013.

[2] P. Mell and T. Grance, “The NIST definition of cloud computing,” *National Institute of Standards and Technology*, vol. 53, no. 6, p. 50, 2009.

[3] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, “Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility,” *Future Generation computer systems*, vol. 25, no. 6, pp. 599–616, 2009.

[4] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica, “Above the clouds: A berkeley view of cloud computing,” *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, vol. 28, p. 13, 2009.

[5] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, “A break in the clouds: towards a cloud definition,” *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 50–55, 2008.

[6] J. Chen, C. Wang, B. B. Zhou, L. Sun, Y. C. Lee, and A. Y. Zomaya, “Tradeoffs between profit and customer satisfaction for service provisioning in the cloud,” in *Proceedings of the 20th international symposium on High performance distributed computing*. ACM, 2011, pp. 229–238.

[7] J. Xu, A. Lam, and V. Li, “Chemical reaction optimization for task scheduling in grid computing,” *IEEE Trans. Parallel and Distributed Systems*, vol. 22, no. 10, pp. 1624–1631, Oct 2011.

[8] J. Yang, H. Xu, L. Pan, P. Jia, F. Long, and M. Jie, “Task scheduling using bayesian optimization algorithm for heterogeneous computing environments,” *Applied Soft Computing*, vol. 11, no. 4, pp. 3297 – 3310, 2011.

[9] K. Li, “Optimal load distribution in nondedicated heterogeneous cluster and grid computing environments,” *Journal of Systems Architecture*, vol. 54, no. 1 - 2, pp. 111 – 123, 2008.

[10] Q.-M. Kang, H. He, H.-M. Song, and R. Deng, “Task allocation for maximizing reliability of distributed computing systems using honeybee mating optimization,” *Journal of Systems and Software*, vol. 83, no. 11, pp. 2165 – 2174, 2010.

[11] S. Kumar, K. Dutta, and V. Mookerjee, “Maximizing business value by optimal assignment of jobs to resources in grid computing,” *European Journal of Operational Research*, vol. 194, no. 3, pp. 856 – 872, 2009.

TABLE 3: Quality of solutions (single application domain).

λ	Greedy_A				DP_A				Brute Force			
	m_i	s_i	C_i	Pro_i	m_i	s_i	C_i	Pro_i	m_i	s_i	C_i	Pro_i
8	7	1.2	20.9424	122.6334	6	1.5	21.5030	137.8665	6	1.5	21.5030	137.8665
9	-	-	-	-	7	1.4	23.7682	154.1115	7	1.5	24.6159	155.3349
10	7	1.5	26.0288	161.5190	-	-	-	-	-	-	-	-
11	9	1.3	28.7695	186.6725	9	1.4	29.8056	190.1581	8	1.5	29.1417	189.9367
12	9	1.4	31.1243	200.101	10	1.3	31.6940	207.3134	10	1.3	31.6940	207.3134
Total Fund	106.8649				106.7708				106.9546			
Total Profit	670.9269				689.4496				690.4516			

[12] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[13] V. G. Dłaz, *Handbook of Research on Innovations in Systems and Software Engineering*. IGI Global, 2014.

[14] http://en.wikipedia.org/wiki/Knapsack_problem, 2014.

[15] Y. Hu, J. Wong, G. Iszlai, and M. Litoiu, "Resource provisioning for cloud computing," in *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research*. IBM Corp., 2009, pp. 101–111.

[16] M. Mazzucco, D. Dyachuk, and R. Deters, "Maximizing cloud providers' revenues via energy aware allocation policies," in *2010 IEEE 3rd International Conference on Cloud Computing,*. IEEE, 2010, pp. 131–138.

[17] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755–768, 2012.

[18] J. Cao, K. Li, and I. Stojmenovic, "Optimal power allocation and load distribution for multiple heterogeneous multicore server processors across clouds and data centers," *IEEE Trans. Computers*, vol. 63, no. 1, pp. 45–58, 2014.

[19] S. Chaisiri, B.-S. Lee, and D. Niyato, "Profit maximization model for cloud provider based on windows azure platform," in *2012 9th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON),*, May 2012, pp. 1–4.

[20] S. Liu, S. Ren, G. Quan, M. Zhao, and S. Ren, "Profit aware load balancing for distributed cloud data centers," in *2013 IEEE 27th International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE, 2013, pp. 611–622.

[21] I. Goiri, J. Guitart, and J. Torres, "Characterizing cloud federation for enhancing providers' profit," in *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*. IEEE, 2010, pp. 123–130.

[22] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Cáceres *et al.*, "The reservoir model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 53, no. 4, pp. 4–1, 2009.

[23] R. Buyya, R. Ranjan, and R. N. Calheiros, "Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services," in *Algorithms and architectures for parallel processing*. Springer, 2010, pp. 13–31.

[24] A. N. Toosi, R. N. Calheiros, R. K. Thulasiram, and R. Buyya, "Resource provisioning policies to increase iaas provider's profit in a federated cloud environment," in *2011 IEEE 13th International Conference on High Performance Computing and Communications (HPCC)*. IEEE, 2011, pp. 279–287.

[25] T. Kurze, M. Klems, D. Bermbach, A. Lenk, S. Tai, and M. Kunze, "Cloud federation," in *The Second International Conference on Cloud Computing, GRIDs, and Virtualization*, 2011, pp. 32–38.

[26] A. Celesti, F. Tusa, M. Villari, and A. Puliafito, "Three-phase cross-cloud federation model: The cloud sso authentication," in *2010 Second International Conference on Advances in Future Internet (AFIN)*. IEEE, 2010, pp. 94–101.

[27] X. Yang, B. Nasser, M. Surridge, and S. Middleton, "A business-oriented cloud federation model for real-time applications," *Future Generation Computer Systems*, vol. 28, no. 8, pp. 1158–1167, 2012.

[28] M. M. Hassan, B. Song, and E.-N. Huh, "Distributed resource allocation games in horizontal dynamic cloud federation platform," in *2011 IEEE 13th International Conference on High Performance Computing and Communications (HPCC)*. IEEE, 2011, pp. 822–827.

[29] Y. C. Lee, C. Wang, A. Y. Zomaya, and B. B. Zhou, "Profit-driven scheduling for cloud services with data access awareness," *Journal of Parallel and Distributed Computing*, vol. 72, no. 4, pp. 591–602, 2012.

[30] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimization of resource provisioning cost in cloud computing," *IEEE Trans. Services Computing*, vol. 5, no. 2, pp. 164–177, 2012.

[31] A. Prasad and S. Rao, "A mechanism design approach to resource procurement in cloud computing," 2013.

[32] T.-J. Chang, N. Meade, J. Beasley, and Y. Sharaiha, "Heuristics for cardinality constrained portfolio optimisation," *Computers & Operations Research*, vol. 27, no. 13, pp. 1271 – 1302, 2000.

[33] P. Chu and J. Beasley, "A genetic algorithm for the multidimensional knapsack problem," *Journal of Heuristics*, vol. 4, no. 1, pp. 63–86, 1998.

[34] J. Mei, K. Li, J. Hu, S. Yin, and E. H-M Sha, "Energy-aware preemptive scheduling algorithm for sporadic tasks on dvs platform," *Microprocessors and Microsystems*, vol. 37, no. 1, pp. 99–112, 2013.

[35] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power cmos digital design," *IEICE Transactions on Electronics*, vol. 75, no. 4, pp. 371–382, 1992.

[36] B. Zhai, D. Blaauw, D. Sylvester, and K. Flautner, "Theoretical and practical limits of dynamic voltage scaling," in *Proceedings of the 41st annual Design Automation Conference*. ACM, 2004, pp. 868–873.

[37] "Enhanced intel speedstep technology for the intel pentium m processor," *White Paper, Intel*, Mar 2004.

[38] P. de Langen and B. Juurlink, "Leakage-aware multiprocessor scheduling," *Journal of Signal Processing Systems*, vol. 57, no. 1, pp. 73–88, 2009.

[39] K. Li, "Optimal configuration of a multicore server processor for managing the power and performance tradeoff," *The Journal of Supercomputing*, vol. 61, no. 1, pp. 189–214, 2012.

[40] A. Eiger, K. Sikorski, and F. Stenger, "A bisection method for systems of nonlinear equations," *ACM Trans. Mathematical Software (TOMS)*, vol. 10, no. 4, pp. 367–377, 1984.

[41] H. Goldberg and T. Senator, "The fincen ai system: Finding financial crimes in a large database of cash transactions," in *Agent Technology*, N. Jennings and M. Wooldridge, Eds. Springer Berlin Heidelberg, 1998, pp. 283–302.

[42] N. K. Boots and H. Tijms, "An M/M/c queue with impatient customers," *Top*, vol. 7, no. 2, pp. 213–220, 1999.