A Queue Waiting Cost-Aware Control Model for Large Scale Heterogeneous Cloud Datacenter

Weihua Bai, Jiaxian Zhu, Shaowei Huang, and Huibing Zhang

Abstract—Optimal load distribution and speed scaling in a heterogeneous data center that takes account of the tradeoff between the maintenance and operation costs and system performance are crucial issues of cloud computing. Due to the changing states of cloud centers and diversity of the task arrival rate, a static control model is infeasible for cloud computing. In this article, we aim to provide a novel multi-server control model with dynamic feedback to acquire dynamic states of the cloud system, and with queue waiting cost-awareness to optimize the queue wait time and load distribution in task assignment and server configuration management. Using the technology of speed scaling, each server in a data center is configured as an $M/M/1$ queue system with variable service rate, and the service rate is a function of the length of the task queue. We formulate two optimization problems, the optimal load distribution problem and the optimal service rate controlling problem, and provide algorithms to solve these problems, facilitating load distributions and service rate adjustments. We also present numerical simulations to validate our model. The results show our model to be efficient in multi-server dynamic configurations and task assignments according to the feedback information for the tradeoff between the system cost and performance.

Index Terms—Cloud computing, convex optimization, load distribution, multi-server control, queuing model, queue waiting time

1 Introduction

1.1 Motivation

Cloud computing is a popular paradigm for providing services to users via three fundamental models: infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS) [1]. A cloud computing provider builds data center infrastructure: a large-scale, heterogeneous array of servers that differ in terms of throughput and load capacity [2], [3]. It manages computing resources (e.g., CPU, memory, network and storage) via resources pooling and serves multiple consumers using a multi-tenant model [4]. With the increasing complexity and diversity of user requirements for Big Data processing services, the efficient allocation of computing resources is the most important problem facing modern and future data centers. Combined with multi-server dynamic configuration [5], especially by using the technology of speed scaling [6], optimal load distribution over large numbers of servers brings optimal aggregate system performance [7], while ensuring efficient use of computing resources. The server's computing ability, the service rate and the resources are often treated as the static information of a server. However, this information is constantly changing with the state of the server. A few widely used representative load distribution and balancing algorithms only use such static information as the basic input to dispatch tasks to servers in a static manner, regardless of the cost consumption in the computing node.

As we showed in Section 2, many existing studies are discussing how to optimize load distribution to improve the performance of mean response time. However, the methods discussed have not considered the following important issues:

- Impact on system cost and performance due to queue waiting time. The number of tasks waiting to be executed is a major factor affecting system throughput.
- Dynamic alteration of the service rate of individual servers.
- The consumption cost of resource pooling is affected by the heterogeneous architectures.

A server can be configured dynamically as a variable service rate queuing system with infinite capacity ($M/M/1/\infty$), via dynamic voltage and frequency scaling (DVFS, equivalently, dynamic voltage scaling, dynamic frequency scaling, dynamic speed scaling) [8]. The service rate changes as a function of the number of tasks in the queue. An efficient server control model should consider the tradeoff between performance and the scaling of consumption, including energy costs, the cost of frequency scaling per unit time, and queuing delay costs.

By analyzing the real workload trace, we reach two conclusions:

- The response time and execution time ratio of a task is higher than 2.89 on average [9]. Since the response time is composed of the waiting time and the execution time, it means that the occupation ratio of the waiting time is more than the execution time in the task lifecycle.
- The requests for memory resources are more intensive than for the cpu resources [10], and tasks accumulated

- *W. Bai, J. Zhu, and S. Huang are with the School of Computer Science, Zhaoqing University, Zhaoqing 526061, China. E-mail: bandwerbai@gmail.com, zhujiaxian@zqu.edu.cn, shw.huang@qq.com.*
- *H. Zhang is with the Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin 541004, China. E-mail: zhanghuibing@guet.edu.cn.*

Manuscript received 7 Aug. 2019; revised 9 Mar. 2020; accepted 23 Apr. 2020. Date of publication 27 Apr. 2020; date of current version 7 June 2022. (Corresponding author: Huibing Zhang.) Recommended for acceptance by C. Hsu. Digital Object Identifier no. 10.1109/TCC.2020.2990982

in a computing node will cause extra memory resources consumption.

The task queue length and the task waiting time are often used as the criterion to evaluate the accumulation status of tasks in a computing node. In the heterogeneous cloud datacenters, the specific situation of each computing node is different to others. A computing node should be reconfigured independently and dynamically according to the appropriate task intensity for better performance or cost saving. By considering the tradeoff between the cost saving and the performance of the system, we present a queue waiting cost-aware control model with two execution and optimization stages.

- Stage 1: The main controller dispatches tasks to every computing node according to the cost consumption, the mean service rate and the length of the task waiting queue in each node.
- Stage2: Each computing node will adjust its service rate according to the length of its task waiting queue and then give a feedback to the main controller (the dispatcher server). This stage is also work for the stage 1 for better cost saving and performance ratio.

These two stages compose the whole optimized process, and they work cyclically and dynamically in the multi-server control model.

1.2 Our Contributions

In this paper, we formulate and provide algorithms to solve the optimal load distribution problem and the optimal controlling service rate problem. Our model considers the optimal load distribution based on queue waiting time and the optimal control of multi-servers by service rate adjustment. Our contributions in this paper can be summarized as follows:

1) Queue waiting cost is considered an objective cost function for the optimal dispatch of tasks. We present an efficient and novel load distribution strategy based on treating every execution server as an $M/M/1/\infty$ queuing system with variable service rate.
2) Using the technology of speed scaling, each execution server is configured with sequential service rates for dynamic configuration. The tradeoff between performance and scaled consumption is taken into account by a service rate control algorithm, which sets the service rate to be proportional to the mean queuing length. Every execution server dynamically returns this service rate information to the dispatcher, where such information includes parameters of the optimization load distribution strategy.
3) We conduct extensive numerical simulation to evaluate and validate our model. The simulation results indicate that the proposed load distribution optimization strategy can optimize task dispatch to achieve a minimal mean queue waiting cost in each execution server.

The remainder of this paper is organized as follows: In Section 2, we briefly review some related research. A dynamic feedback and queue waiting cost-aware optimal load distribution and multi-server control model for large-scale cloud data center is presented in Section 3. The optimal load distribution and balancing strategy, which is related to the service rate, and the optimal control for servers, which treats each server as a variable service rate queuing system and considers the tradeoff between the performance and scaling cost, are introduced in Section 4. In Section 5, we present numerical simulations and the results of performance metrics to evaluate and validate the analysis method and the applicability of the optimal multi-server control model. Finally, conclusions and directions for future work are presented in Section 6.

2 Related Research

Load distribution and multi-server control that provides balanced operations and reduced maintenance costs is an essential research issue regarding cloud computing environments.

In [11], the authors characterized distributed computer systems as queuing models, formulated the problem of load balancing, and presented effective static and dynamic optimization strategies. A distributed computer system that consists of heterogeneous servers connected by networks was modeled as a queuing network and efficient strategies for static load balancing optimization were presented in [12]. In [13], the authors used a mathematical model, a nonlinear optimization problem, to analyze system performance and to optimize both the workload allocation scheme and the task dispatching strategy in a network of heterogeneous computers. In particular, many algorithms were proposed for optimal load balancing in cloud computing environments [14], [15]. Game theory was applied to the load balancing strategy in cloud environments [16] and in distributed systems [17]. Based on improvement of the Round Robin algorithm, many strategies for load balancing were suggested by Cardellini *et al.* [13], [15], [18].

The speed scaling strategy for optimizing energy costs is an important research topic. In [19], the authors studied the interaction between static speed scaling and load balancing. Based on speed scaling, optimal power allocation and performance in data centers were discussed in [8] and [20], [21]. Specifically, the optimal power allocation problem was formulated and solved in [8] and [22]. The model for trading off the mean energy consumption and the mean response time was presented in [23], while the model for trading off the energy cost and delay in both networking and processing components of base stations was presented in [24].

Queueing theory can be used to model and analyze the architecture of a large-scale data center composed of heterogeneous servers [8], [13], [23], [24]. In [25], [26], the authors discussed dynamic control of the service rate in a queuing system as a means of minimizing the average holding cost. A queuing system with variable service rate was extensively studied in [27], [28], and the results are useful to the optimization problem in speed scaling; this can be realized by an adjustable service rate for data center servers.

Studies have shown that servers (includes processor, memory, and storage systems) and networks (includes links, transit, and equipment) are a great source of data center cost, with consumption cost being approximately 60 percent of that total cost, while the power consumption cost is approximately 15 percent [29]. Dynamically allocated virtual machines (VMs) and cost-minimizing task

scheduling were presented in [30]. Mistral (a controller framework) optimizes power consumption, performance benefits, and the transient costs [31]. The algorithms were presented to solve the resource allocation problem for optimizing the system cost in [32] and the network aspect of the load distribution (both inter and intra data center) in [33], [34]. A strategy-based dynamic server provisioning to minimize margin costs and true costs in an Infrastructure-as-a-Service (IaaS) cloud was presented in [35]. However, most of the issues do not consider the holding cost (or congestion cost, this being proportional to the queue length), which is caused by increases in VM instances, memory size, data storage, network links, etc.

3 Dynamic Control Model

In this section, we present a dynamic feedback and queue waiting cost-aware optimal load distribution and multi-server control model, consisting of a dynamic feedback controller (DFC, which works as a queuing waiting cost-aware model) and a service rate controller (SRC, which works as a server control model). We formulate the problem of optimal load distribution with a mathematical model based on convex optimization theory and queueing theory, and control each server with optimal dynamic provisioning by treating each server as an $M/M/1/\infty$ queuing system with a variable service rate.

3.1 Dynamic Feedback Controller

Assume that a large-scale cloud computing data center has a group of n heterogeneous servers $S_1, S_2, \ldots, S_n$, with different service rates. As we know, servers having different configurations will incur different costs (e.g., different price of a VM, different memory consumptions, different data storage cost, and different network bandwidth cost). When the queue piles up with tasks waiting for execution, the congestion cost is greatly increased (e.g., the utilization and occupied cost of computing resource pooling, the cost of network links, and the cost of fault-tolerant recovery) in the system. Each server has its own cost coefficient, which can be set or calculated using historical information respecting each server in the data center, to be taken into account for considering the queue waiting time cost. In the waiting cost-aware model, each server in the data center is treated as an $M/M/1$ queuing system and the waiting cost function is set as the objective function to solve.

Let $C_i (1 \le i \le n, C_i \in (0,1])$ denote the cost coefficient of server S_i in a data center, λ the task arrival rate (this is a Poisson stream as measured by the number of tasks per second), and $\overline{\mu_c^i} (1 \le i \le n)$ the current mean service rate of server S_i. The tasks assigned to server S_i by the dispatcher is a Poisson stream with arrival rate $\lambda_i (1 \le i \le n)$ and $\lambda = \lambda_1 + \lambda_2 + \ldots + \lambda_n = \sum_{i=1}^{n} \lambda_i$, as calculated by the load distribution and balancing algorithm in the DFC. In the DFC, we treat each server in the data center as an $M/M/1$ queuing system and use the current service rate of server S_i, $\overline{\mu_c^i}$, to obtain the mean waiting time of S_i, W_q^i, as

$$W_q^i = \frac{\lambda_i}{\overline{\mu_c^i}\left(\overline{\mu_c^i} - \lambda_i\right)}, i \in \{1, 2, \ldots, n\} \text{and } \lambda_i < \overline{\mu_c^i}. \tag{1}$$

Accordingly, the waiting cost of each computing node can be calculated as $\frac{\lambda_i}{\lambda}(W_q^i + C_i\overline{\mu_c^i})$. The mean waiting time cost $\overline{C_w}$ of the data center with large-scale n heterogeneous servers can be formulated as

$$\overline{C_w} = \sum_{i=1}^{n} \frac{\lambda_i}{\lambda}\left(W_q^i + C_i\overline{\mu_c^i}\right) = \sum_{i=1}^{n}\left(\frac{\lambda_i^2}{\overline{\mu_c^i}\left(\overline{\mu_c^i} - \lambda_i\right)\lambda} + \frac{\lambda_i}{\lambda} C_i\overline{\mu_c^i}\right), \quad i \in \{1, 2, \ldots, n\} \text{and } \lambda_i < \overline{\mu_c^i}. \tag{2}$$

The DFC works with the dispatcher. The aim of the queue waiting cost-aware model is to split the task stream into n sub-streams (assigning the ith sub-stream λ_i to server S_i) with the target being queue waiting time minimization $\overline{C_w}$.

3.2 Service Rate Controller

Using the technology of speed scaling, let M_i $(1 \le i \le n)$ denote the number of variable service rates of server S_i (each server is treated as an $M/M/1/\infty$ queuing system with variable service rates). Assume that μ_j^i $(1 \le i \le n, 1 \le j \le M_i$, and $\mu_{j-1}^i \le \mu_j^i \le \mu_{j+1}^i)$ denotes the jth service rate of server S_i and L_q^i the mean queue length of server S_i (measured by the number of tasks waiting in the execution queue). The SRC will adjust the service rate according to the L_q^i. It uses a control variable k_i ($k_i \in N_+$) to control the adjustment of the service rate of server S_i. The server's service rate will be adjusted up from μ_j^i to μ_{j+1}^i when $L_q^i \in [(j-1)k_i + 1, jk_i] \rightarrow L_q^i \in [jk_i + 1, (j+1)k_i]$, or down to μ_{j-1}^i when $L_q^i \in t[(j-1)k_i + 1, jk_i] \rightarrow L_q^i \in [(j-2)k_i + 1, (j-1)k_i]$. The current service rate of server S_i μ_c^i can be formulated as

$$\mu_c^i = \begin{cases} \mu_1^i, 0 \le L_q^i \le k_i, \mathrm{k_i} \in N_+; \\ \mu_j^i, (j-1)k_i + 1 \le L_q^i \le jk_i, 1 < j \le M_i, \mathrm{k_i} \in N_+; \\ \mu_{M_i}^i, M_i k_i < L_q^i. \end{cases} \tag{3}$$

In the adjustment proceeding, we consider three kinds of cost in server S_i:

1) the power consumption, which is proportional to the mean service rate;
2) the adjustment consumption, which is proportional to the mean times of adjustment per unit time;
3) the holding consumption, which is proportional to the mean queue length.

Let c_p^i, c_a^i, and c_h^i denote these three kinds of cost coefficients per unit time in server S_i. The mean total cost of server S_i is an objective function with the control variable k_i as its parameter. The formulation of $F_i(k_i)$ is

$$F_i(k_i) = c_p^i F_\mu^i(k_i) + c_a^i F_t^i(k_i) + c_h^i F_l^i(k_i). \tag{4}$$

In (4), $F_\mu^i(k_i)$ denotes the mean service rate function, $F_t^i(k_i)$ the mean number of adjustments per unit time, and $F_l^i(k_i)$ the mean queue length function. All of these are related to the control variable k_i.

Since the control variable k_i will impact $F_i(k_i)$ greatly, the aim of the SRC is to choose an optimal value of k_i to minimize the $F_i(k_i)$ by the queueing theory.

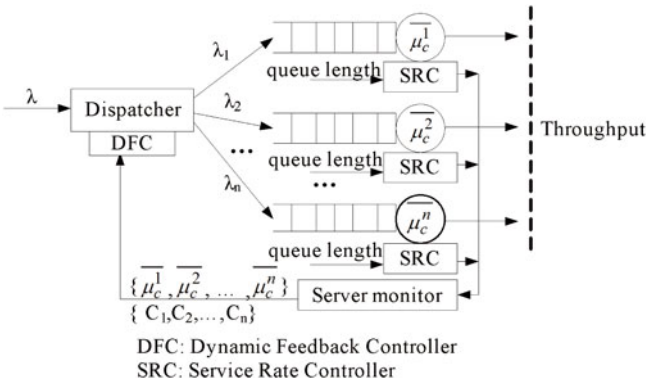

Fig. 1. Dynamic control model of the cloud computing data center.

3.3 Dynamic Control Model

The dynamic control model in a large-scale data center with n heterogeneous servers is shown in Fig. 1. There are three steps in the dynamic control model to optimally control the multi-server.

1) Load distribution: After gaining the parameters of $\overline{\mu_c^i}$, C_i, and λ from the server monitor dynamically, the DFC will control the dispatcher to assign a Poisson stream (tasks) λ_i to server S_i according to the load distribution and balancing algorithm to minimize the cost metric $\overline{C_w}$.
2) Optimal control for servers: First, the SRC of server S_i uses the optimal value of k_i, which is chosen by minimizing the mean total cost of server S_i, $F_i(k_i)$, to calculate the mean service rate of server S_i, $\overline{\mu_c^i}$, according to the task arrival rate λ_i. Then, the $\overline{\mu_c^i}$ will be used to control the server's service rate for optimal performance metrics (e.g., the mean queue waiting time, the mean queue length, and the mean response time). Finally, the SRC provides feedback information to the server monitor dynamically for optimal load distribution with queue waiting cost-awareness.
3) Dynamic feedback: the server monitor updates the information of the parameters ($\overline{\mu_c^i}$, C_i , n, λ_i) and then sends it to the DFC dynamically.

4 Optimal Control Model For Multi-Servers

In this section, we formulate the optimization problems and discuss the load distribution strategy in the DFC, dynamic provisioning for optimal control in the SRC, and the dynamic control strategy to solve these optimization problems.

4.1 Load Distribution Strategy

In cloud computing resource allocation and management, optimizing the queue waiting time can help providers reduce resource consumption and thereby improve the throughput of the system. Since the cost coefficient of each server is dependent on prior information, and since there is a tradeoff between the queue waiting time and holding cost of tasks in queue, the objective function in our optimal model is a sum of the mean system queue waiting time and the maintenance and operation costs. This optimal load distribution problem can be formulated as the queuing waiting cost minimization problem, which is stated as follows: to minimize the queue waiting cost by jointly finding the optimal dispatching strategy $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n)$, subject to the queuing stability constraint in each queuing system. Mathematically, the optimization problem is formulated as

$$\min_{\{\lambda,\overline{\mu_c^i},C_i\}} \left(F(\lambda) = \overline{C_w} = \sum_{i=1}^{n} \left(\frac{\lambda_i^2}{\overline{\mu_c^i}\left(\overline{\mu_c^i} - \lambda_i\right)\lambda} + \frac{\lambda_i}{\lambda} C_i \overline{\mu_c^i} \right) \right) \tag{5}$$

$$\text{s.t.} \sum_{i=1}^{n} \lambda_i = \lambda \tag{5-1}$$

$$\lambda_i \geq 0, \forall i = 1, \ldots, n \tag{5-2}$$

$$\overline{\mu_c^i} - \lambda_i > 0, \forall i = 1, \ldots, n. \tag{5-3}$$

The cost coefficient C_i can be set according to the state of server S_i. A resource allocation scheme with utilization threshold $[\Theta_l, \Theta_h]$ can control a computing node (server) to be added to a cloud system or be removed from it [36]. We set the service rate threshold $[\mu_{Min}^i, \mu_{Max}^i]$ for server S_i and set the minimum service rate μ_{Min}^i as the initial value of $\overline{\mu_c^i}$ in (5).

Lemma 1. *The optimization problem in (5) is a convex optimization problem.*

(To enhance readability, the proofs of all lemmas of the paper are given in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCC.2020.2990982.)

Lemma 2. *The solution of the optimization problem in (5) is given as follows:*

$$\lambda_i(\delta) = \begin{cases} \overline{\mu_c^i} - \overline{\mu_c^i}\left(\frac{1}{\sqrt{1 - C_i \overline{\mu_c^i}^2 + \delta \lambda \overline{\mu_c^i}}} \right), \delta > \frac{C_i \overline{\mu_c^i}^2}{\lambda \overline{\mu_c^i}} \\ 0, \delta \leq \frac{C_i \overline{\mu_c^i}^2}{\lambda \overline{\mu_c^i}}. \end{cases} \tag{6}$$

which is a strictly increasing function of δ.

Since the function $\lambda(\delta)$ is a strictly increasing function of δ, it is possible to use the binary search algorithm to find the value of δ based on the constraint of $\sum_{i=1}^{n} \lambda_i(\delta) - \lambda \leq \varepsilon$ in the domain $\delta \in (0, \infty)$. The load distribution strategy is to find the solution (δ, $\lambda_1, \lambda_2, \ldots, \lambda_n$). The strategy is named Algorithm *Calculate_lds* ($C_i[\,]$, $\overline{\mu_c^i}[\,]$, λ, n); the input is $C_i[\,]$, $\overline{\mu_c^i}[\,]$, λ, n; and the result is [δ, $\lambda_1, \lambda_2, \ldots, \lambda_n$].

The algorithm *Calculate_lds* uses (6) to solve the function of $\sum_{i=1}^{n} \lambda_i\left(\delta^{(1)}\right)$.

4.2 Dynamic Provisioning for Optimal Control

In our model, we configure each server with sequence service rates $\{\mu_1^i, \mu_2^i, \ldots, \mu_{M_i}^i\}$. By controlling the service rate, the queue length can be controlled, thus optimizing the mean queue waiting time. As discussed in Section 3.3, there are M_i different service rates in server S_i, and the variable k_i ($k_i \in N_+$) adjusts the service rate of server S_i. Here, we will discuss the functions of $F_\mu^i(k_i)$, $F_t^i(k_i)$, and $F_l^i(k_i)$ in (4).

4.2.1 Obtaining $F_\mu^i(k_i)$

Let $\rho_{ij} = \frac{\lambda_i}{\mu_j^i} < 1 (1 \leq i \leq n, 1 \leq j \leq M_i)$ denote the utilization of server S_i with the service rate μ_j^i, and $p(i, j, l)$ denote

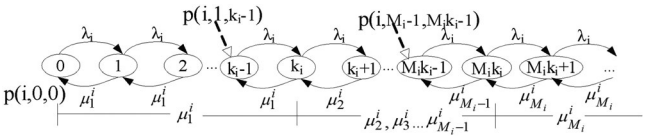

Fig. 2. State-transition-probability diagram—$M/M/1$ with variable service rate.

the percentage of time in the steady state that the queue system is in state l (the task queue length is equate to l) of service S_i with the service rate μ_j^i.

It is easy to prove that the $M/M/1/\infty$ queue with a variable service rate problem is a birth-death process with $\lambda_n{}' = \lambda_i$ and $\mu_n{}' = [\mu_1^i, \mu_2^i, \ldots, \mu_{M_i}^i]$.

The state-transition-probability diagram—$M/M/1/\infty$ with variable service rate—is shown in Fig. 2.

Lemma 3. *The mean service rate function of server S_i, $F_\mu^i(k_i) = \sum_{j=1}^{M_i} \mu_j^i p(i,j,\cdot)$, can be formulated as*

$$F_\mu^i(k_i) = \left(\mu_1^i + \sum_{j=1}^{M_i-1}\left(\mu_j^i \frac{\rho_{ij}\left(1-\rho_{ij}^{k_i}\right)}{1-\rho_{ij}}\left(\prod_{t=1}^{j-1}\rho_{it}^{k_i}\right)\right) + \mu_{M_i}^i \frac{\rho_{iM_i}}{1-\rho_{iM_i}}\left(\prod_{t=1}^{M_i-1}\rho_{it}^{k_i}\right)\right)p(i,0,0). \tag{7}$$

4.2.2 Obtaining $F_t^i(k_i)$

As shown in Fig. 2, when the steady state changes from p (i,j,jk_i) to p$(i,j+1,jk_{i+1})$, the service rate will change from μ_j^i to μ_{j+1}^i. The probability of the service rate changing equates to the steady state p$(i,j+1,jk_{i+1})$. Thus, using μ_{j+1}^i, the number of changes to the service rate per unit time is $\mu_{j+1}^i p(i,j+1,jk_i+1)$. In a server, the mean number of adjustments per unit time is twice the number of service rate changes, and can be formulated as

$$\begin{aligned} F_t^i(k_i) &= 2\sum_{j=0}^{M_i-1}\mu_{j+1}^i p(i,j+1,jk_i+1) \\ &= 2\sum_{j=1}^{M_i}\left(\lambda_i\left(\prod_{t=1}^{j-1}\rho_{it}^{k_i}\right)\right)p(i,0,0). \end{aligned} \tag{8}$$

4.2.3 Obtaining $F_l^i(k_i)$

From the state-transition-probability diagram shown in Fig. 2, each steady state denotes the queue length and the probability of the queue length at this moment. Thus, using (A.19) and (A.21), the mean queue length is formulated as

$$\begin{aligned} F_l^i(k_i) &= \sum_{l=0}^{\infty} lp(i,\cdot,l) \\ &= \left(\sum_{j=1}^{M_i}\sum_{l=(j-1)k_i+1}^{jk_i} l\rho_{ij}^{l-(j-1)k_i}\left(\prod_{t=1}^{j-1}\rho_{it}^{k_i}\right) + \sum_{l=M_ik_i+1}^{\infty} l\rho_{ij}^{l-M_ik_i}\left(\prod_{t=1}^{M_i}\rho_{it}^{k_i}\right)\right)p(i,0,0). \end{aligned} \tag{9}$$

Equation (9) can be rewritten as

$$F_l^i(k_i) = \left(\sum_{j=1}^{M_i}\sum_{l=(j-1)k_i+1}^{jk_i} l\rho_{ij}^{l-(j-1)k_i}\left(\prod_{t=1}^{j-1}\rho_{it}^{k_i}\right) + \frac{\rho_{iM_i}(1+M_ik_i-M_ik_i\rho_{iM_i})}{(1-\rho_{iM_i})^2}\left(\prod_{t=1}^{M_i}\rho_{it}^{k_i}\right)\right)p(i,0,0). \tag{10}$$

4.2.4 Obtaining k_i

k_i ($k_i \in N_+$) is the control variable of server S_i for controlling adjustment to the service rate. In order to obtain the k_i, it should be proved that for $F_i(k_i)$ there exists a minimum value $F_i(k_i{}') = \mathrm{C}_{f\min}$, for which the $k_i{}' \in (0,N)$ and the $k_i{}'$ only in $(0,N)$.

Lemma 4. *If $\lim_{k_i\to+\infty} F_i(k_i) = A$, $\forall\varepsilon > 0$, $\exists k_i = N$, when $k_i > N, |F_i(k_i) - A| < \varepsilon$ holds.*

Lemma 5. *Let $F_i(0) = B$, $\lim_{k_i\to+\infty} F_i(k_i) = A$, where there exists a minimum value $F_i(k_i{}') = C_{fmin} = min(F_i(k_i))$, $k_i{}' \in (0, N)$ and the $k_i{}'$ are only in $(0,N)$.*

From Fig. 2 and the actual physical meaning of k_i, we discuss the convexity of the function $F_i(k_i), k_i \in (0,N)$, and prove that it has the unique minimum value in $(0, \min(A,B))$.

Lemma 6. *$F_i(k_i)$ is a concave function in $(0,N)$, $\exists k_i{}' \in (0, N), F_i(k_i{}') = min(F_i(k_i))$.*

For example, we can obtain the diagram of the mean total cost function $F_i(k_i)$ of server S_i shown in (4).

Algorithm 1. *Calculate_ki*$(c_p^i, c_a^i, c_h^i, \mu^i[\,], M_i, \lambda_i)$.

Input: $(c_p^i, c_a^i, c_h^i, \mu^i[\,], M_i, \lambda_i)$
Output: k_i //the optimal control variable
1: $u\_val \longleftarrow call\_Ff(\infty)$ //calculate the function of $F_i(\infty) = c_p^i\mu_1^i + c_a^i 2\lambda_i\frac{\mu_1^i-\lambda_i}{\mu_1^i} + c_h^i\frac{\lambda_i}{\mu_1^i-\lambda_i}$
2: $k^{(u)} \longleftarrow 1$;
3: $\Delta 1 \longleftarrow 2$; //step1
4: $\Delta 2 \longleftarrow 0.618$; //step2
5: **Begin Do_While** //find the upper bound of k_i
6: $k^{(u)} \longleftarrow \Delta 1 * k^{(u)}$;
7: $e\_point1 \longleftarrow call\_Fi(k^{(u)})$;
8: $e\_point2 \longleftarrow call\_Fi(|\Delta 2 * k^{(u)}|) + k^{(u)})$;
9: **If** $(|e\_point1 - u\_val| \le \varepsilon$ && $|e\_point2 - u\_val| \le \varepsilon)$ //the two continue points tends to $F_i(\infty)$
10: **Break**;
11: **End If**;
12: **End Do_While**;
13: ends $\longleftarrow k^{(u)}$; // the end point of the searching
14: **While** $(call\_Fi(\lfloor \Delta 2 * k^{(u)} \rfloor) > call\_Fi(\lfloor \Delta 2 * k^{(u)} \rfloor - 1))$ // decreasing $k^{(u)} \ge 1$
15: $k^{(u)} \longleftarrow \lfloor \Delta 2 * k^{(u)} \rfloor$;
16: ends$\longleftarrow k^{(u)}$;
17: **End While**;
18: starts$\longleftarrow \lfloor \Delta 2 * k^{(u)} \rfloor$;
19: **For** (i=starts; $i <$ ends; $i++$) //find the optimal k_i
20: $k_i \longleftarrow findmin(call\_Fi(i))$;
21: **End For**;
22: **return** k_i;

Since the function of $F_i(k_i)$ is a concave function in $(0, N)$ (Lemma 6), we can use a line search algorithm to calculate the optimal control variable k_i within small loops. When the SRC in each server receives the λ_i, the SRC will use the Algorithm 1 $Calculate\_k_i$ to obtain k_i to control the speed scaling.

After obtaining the optimal k_i, the SRC will calculate the value of $F^i_\mu(k_i)$ and send it to the DFC.

4.3 Dynamic-Control-Strategy-Based Dynamic Feedback

The dynamic control strategy, which works between the DFC and the SRC, is used to adjust server provisioning via the information of the server's states. There are two states and M_i selection whose service rates should be adjusted by the DFC or the SRC, according to these parameters via dynamic feedback:

1) $\sum_{i=1}^{n} \mu^i_c \le \lambda$ and $\lambda < \sum_{i=1}^{n} \mu^i_{M_i}$

   If the condition $\sum_{i=1}^{n} \mu^i_c > \lambda$ cannot be satisfied for Algorithm $Calculate\_lds$ ($C_i[\,]$, $\overline{\mu^i_c}[\,]$, λ, n) to optimize the load distribution, DFC should actuate the SRC to raise the minimum service rate, which means selecting a greater service rate from the service rate sequence as the minimum service rate.

   The DFC will send the information to the related SRC to adjust server provisioning, and the SRC will update the service provisioning rate and send the value of $F^i_\mu(k_i)$ back to the DFC in the next time slice for load distribution.

2) $\lambda_i/\overline{\mu^i_c} < \Theta^i_l$ *or* $\lambda_i/\overline{\mu^i_c} > \Theta^i_h$

   The SRC should adjust the minimum service rate inverse to the slope of the task arrival rate (e.g., if $\lambda_i/\overline{\mu^i_c} < \Theta^i_l$, the range of the service rate will change from $\{\mu^i_t, \mu^i_{t+1}, \ldots, \mu^i_{M_i}\}$ to $\{\mu^i_{t-1}, \mu^i_t, \mu^i_{t+1}, \ldots, \mu^i_{M_i}\}$, where $\mu^i_t > \mu^i_{t-1}$ and $t > 1$). Let $[\Theta^i_l, \Theta^i_h]$ denote the utilization threshold of server S_i; it controls the adjustment of the service rate.

   The utilization threshold and the gap between the two neighboring service rates in the service rate sequence will impact service rate oscillation. If the gap between two neighboring service rates is great, oscillation of the service rate will be great. In practice, the oscillation should be avoided. Thus, we can set a switch threshold, which is a utilization threshold of the system that is defined as $\xi = \lambda / \sum_{i=1}^{n} \mu^i_{M_i}$. This switch threshold is used to control whether to use the next algorithm, $Adjust\_Rr(\lambda_i, \mu^i[\,], \Theta^i_l, \Theta^i_h, VM_i)$, to change the range of the service rate. Since load distribution and balancing is more important under high-load conditions than under low-load conditions, the switch threshold is set as $\xi = 0.7$ (the value of ξ can be adjusted according to the cloud environments).

   Since the volume of data exchanged between the DFC and the SRC is small, and since each algorithm computes in small steps, the communication and calculation time can be ignored in the dynamic control procedure. The utilization threshold guides SRC control of the service rate up or down in direct relation to the task arrival rate, which will avoid system oscillation. Conversely, if the task arrival rate oscillates quickly, the DFC will actuate the related servers, thereby reducing the oscillation rate.

3) M_i Selection

   The technology of dynamic voltage and frequency scaling (DVFS) [37], [38] can dynamically scale the server speed, the rate of which can be scaled in the range $[\mu_{min}, \mu_{max}]$. Server S_i can choose service rate from the set of $R_i = \{\mu^i_1, \mu^i_2, \ldots, \mu^i_{Mi}\}$ ($\mu^i_t \in [\mu^i_{min}, \mu^i_{max}], \mu^i_{M_i} = \mu^i_{max}, 1 \le t \le M_i,$), where M_i denotes the number of variable service rates of server S_i. M_i selection will impact the cost, level of sensitivity, and oscillation amplitude of the system.

   M_i and $M_i^{'}(M_i > M_i^{'})$ are different selections of the service rates of server S_i. Let $\Delta\mu = \frac{\mu_{max} - \mu_{min}}{M} = \mu_{t+1} - \mu_t, (1 \le t \le M)$. Since $M_i > M_i^{'}$, based on Algorithm 2, $Adjust\_M()$, and Algorithm 3, $Adjust\_Rr()$, we can obtain $\Delta\mu_i^{'} > \Delta\mu_i$, and the results are as follows:

   - Impact of cost: Since the service cost and the energy consumption are proportional to the service rate, when the DFC actuates the SRC to raise the minimum service rate under the condition of $\sum_{i=1}^{n} \mu^i_c \le \lambda$, the increase of system consumption with the setting of $M_i^{'}$ will be higher than that with the setting of $M_i (\Delta SC^{'} > \Delta SC)$.
   - Level of sensitivity and the oscillation amplitude: If the task arrival rate λ has changed, the DFC will actuate the SRC to select the minimum service rate in the list of R_i to meet the change of λ. The level of sensitivity is controlled by $\Delta\mu$ caused by $\Delta\lambda$, which means that the server with a setting of $M_i (M_i > M_i^{'})$ is more highly sensitive to the slightest change $\Delta\lambda$ than with the setting of $M_i^{'}$ and that the oscillation amplitude of the former is less than the latter.

Furthermore, the M_i selection can be adjusted by cloud providers according to the servers' status for reasonable cost and system stability.

Algorithm 2. $Adjust\_M(\lambda, \mu_c[], \mu\_Info[\,][\,], C_i[], VM[\,], n)$.

Input: λ,$\mu_c[\,]$,$\mu\_Info[\,]$, $C_i[\,]$;
Output: $\mu_r[\,]$, $M[\,]$;
1: **While** ($\sum \mu_c[\,] \le \lambda$ && $\lambda < \sum_{i=1}^{n} \overline{\mu^i_{M_i}}$)
2: **For** (each $\mu_c[\,]$)
3: $\mu_a[\,] \longleftarrow \mu\_Info[\,][q\,]$; //find the service rate of each server in the sequence $\mu\_Info[\,][q\,]$ which satisfies the condition: $\mu\_Info[\,][q-1\,] < \mu_c[\,] < \mu\_Info[\,][q\,]$
4: $\Delta V[\,] \longleftarrow C_i[\,] * (\mu_a[\,] - \mu_c[\,])$; //calculate the increasing
5: **End For**;
6: $t \longleftarrow index(min\{\Delta V[\,]\}\,)$;
7: $\mu_c[t] \longleftarrow \mu_a[t]$; //raise the minimum service rate of server S_t;
8: $VM[t] \longleftarrow VM[t] - 1$; //Adjust the range of the service rate of server S_t;
9: **End While**;
10: $\mu_r[\,] \longleftarrow \mu_c[\,]$;
11: $M[\,] \longleftarrow VM[\,]$;
12: **return** $\mu_r[\,], M[\,]$; //sent it to the SRC of the related server.

TABLE 1
Parameters Settings 1

$n = 3$ (number of generations in a data center)				
Λ [task arrival rate, $\lambda \in (0, 17)$]				
$[\Theta_l, \Theta_h]$:[0.60, 0.90] (utilization threshold)				
i	$\mu^i[\,]$	(c_p^i, c_a^i, c_h^i)	M_i	C_i
1	{1, 2, 3, 4}	(0.45, 0.25, 0.3)	4	0.4
2	{2, 2.5, 3, 3.5, 4, 5}	(0.6, 0.15, 0.25)	6	0.5
3	{3, 5, 7, 8}	(0.55, 0.25, 0.2)	4	0.6

Algorithm 3. $Adjust\_Rr\ (\lambda_i, \mu^i[\,], \Theta_l^i, \Theta_h^i, VM_i)$.

Input: $\lambda_i, \mu^i[\,], \Theta_l^i, \Theta_h^i, VM_i$
Output: VM_i
If $(\lambda_i/\overline{\mu_c^i} < \Theta_l^i$ && $VM_i > 1)$
  $VM_i \leftarrow VM_i{+}1$; // decrease $\left\{\mu^i_{M_i-VM_i+1}, \ldots, \mu^i_{M_i}\right\} \Rightarrow \left\{\mu^i_{M_i-VM_i}, \ldots, \mu^i_{M_i}\right\}$
Else if $(\lambda_i/\overline{\mu_c^i} > \Theta_h^i$ && $VM_i < M_i)$
  $VM_i \leftarrow VM_i{-}1$; //increase $\left\{\mu^i_{M_i-VM_i+1}, \ldots, \mu^i_{M_i}\right\} \Rightarrow \left\{\mu^i_{M_i-VM_i+2}, \ldots, \mu^i_{M_i}\right\}$
End if;
$t \longleftarrow index(min\{\Delta V[\,]\}\,)$;
$\overline{\mu_c^i} \leftarrow F_\mu^i(k_i)$; // sent $\overline{\mu_c^i}$ *and* VM_i to DFC
return VM_i; //adjust the service rate sequence.

5 Numerical Validation

In this section, we demonstrate some numerical examples to validate the model; all the parameters given are illustrative and can be changed to suit any given cloud computing environment.

A typical, modern heterogeneous data center has a large number of servers of different generations. In the model, the parameter n denotes the number of generations; it also can denote the number of servers in a data center.

In the simulation experiment, we assumed that $n = 3$ and the utilization threshold $[\Theta_l, \Theta_h]$=[0.60, 0.90], the parameter settings are shown in Table 1.

In the numerical examples, we consider some load distribution strategies and compare some performance metrics and costs among these strategies. The load distribution strategies are compared as follows:

1) Proportion load distribution: In this strategy, servers are not configured with a variable service rate; instead, the dispatcher assigns each server a load in proportion to the reciprocal of the service rate. This is a general strategy for load distribution in a data center. Let $L(P)$ denote this strategy, which can be formulated as

$$\lambda_i = \frac{\mu^i_{M_i}}{\sum_{i=1}^n \mu^i_{M_i}}, \lambda_i \le \lambda \text{ and } \lambda = \sum_{i=1}^n \lambda_i. \tag{11}$$

2) $L(Max)$ load distribution: In this strategy, each server is configured with the max service rate $\mu^i_{M_i}$ and uses Algorithm $Calculate\_lds\ (C_i[\,],\ \overline{\mu_c^i}[\,],\ \lambda,\ n)$ $Calculate\_lds$ to assign the load to each server. This strategy is without speed scaling and it can be formulated as

$$\lambda_i[\,] = Calculate\_lds\big(C_i[\,], \mu^i[\,], \lambda, n\big). \tag{12}$$

3) $L(LD)$ load distribution: In this strategy, using the technology of speed scaling, each server is configured with a variable service rate and the Algorithm $Calculate\_lds\ (C_i[\,],\ \overline{\mu_c^i}[\,],\ \lambda,\ n)$ $Calculate\_lds$ assigns the load to each server. Algorithm 2 $Adjust\_M$ in the DFC adjusts the minimum service rate in the sequence of the service rates.
4) $L(LDS_{(1)})$ load distribution: In this strategy, each server is configured with a variable service rate (speed scaling). The dispatcher uses the $Calculate\_lds\ (C_i[\,],\ \overline{\mu_c^i}[\,],\ \lambda,\ n)$ $Calculate\_lds$ to assign a load to each server. Using λ_i, the SRC uses Algorithm 1 and Algorithm 3 under the switch threshold $\xi = 1$ to calculate the optimal k_i and alter the service rate, and then sends the value of $F_\mu^i(k_i)$ to the DFC dynamically; the DFC calculates using Algorithm 2 to obtain all the parameters $(C_i[\,], \mu^i[\,], \lambda_i[\,], \mathrm{VM}[\,])$, and then sends it to the dispatcher for the next task assignment and to the SRC for service rate adjustment, respectively. $L(LDS_{(1)})$ works by this cooperative work and feedback mechanism between the DFC and the SRC.
5) $L(LDS_{(0.7)})$ load distribution: This strategy is similar to $L(LDS_{(1)})$ and the switch threshold $\xi = 0.7$.

5.1 Performance simulation

In this section, we present extensive simulation experiments using MultiRECloudSim [39] and Matlab to validate the effectiveness of the multi-server control model. The implementation details about MultiRECloudSim as follows:

1) We set three datacenters DT1, DT2, and DT3 with a group of $n = 4 - 5$ multicore execution servers and a main scheduler server.
2) The parameters settings of DT1, DT2, and DT3 are shown in Table 1.
3) The three datacenters settings of the Host#i $(1 \le i \le 5)$ are shown in Table 2.
4) It uses space-shared as the utilization model in the VM-scheduler and makes one task in one VM.
5) The VM allocation policy is FCFS, and we use the multi-server control model to schedule the tasks.
6) The tasks arrival rate λ (the number of tasks requesting per second) follows an exponential distribution generated by Matlab14.0. The cloudlet is created by a task generator according to the settings of the task arrive interval time and the mean length of the tasks as $\overline{I} = 2$ (Giga instructions).
7) Let t_{k_a} denotes the arrival time of a task assigned to a computing node; t_{k_o} denotes the time when a task is completed and left the system; t_{k_e} denotes the time executed in a computing node. Accordingly, the system response time of a task is $t_{k\_o} - t_{k\_a}$, and the waiting time of a task is $t_{k\_o} - t_{k\_a} - t_{k\_e}$.

TABLE 2
Datacenters Setting

HostID	Setting : DT1		Setting : DT2		Setting : DT3	
	VMs	GIPS, f_i	VMs	GIPS, f_i	VMs	GIPS, f_i
Host#1	1	1Core/2G	2	1Core/2G	3	1Core/2G
Host#2	2	1Core/2G	5	1Core/1G	5	1Core/2G
Host#3	3	1Core/2G	3	1Core/2G	7	1Core/2G
Host#4	4	1Core/2G	7	1Core/1G	8	1Core/2G
Host#5			9	1Core/1G		

TABLE 3
The Simulations and Analytical Results of TW_q

λ	Strategies	Simlation 95% C.I. for TW_q			Analytical
		Mean	Lower	Upper	
3	$L(P)$	0.037760305	0.037612359	0.03790825	0.037815126
	$L(Max)$	0.234571288	0.233762156	0.23538042	0.234009306
	$L(LDS_{(0.7)})$	0.442895401	0.441451485	0.444339317	0.44282978
6	$L(P)$	0.096340595	0.09604497	0.09663622	0.096256684
	$L(Max)$	0.453713756	0.451853996	0.455573516	0.453835385
	$L(LDS_{(0.7)})$	0.871720255	0.868365043	0.875075468	0.871430159
10	$L(P)$	0.252291416	0.251345277	0.253237555	0.25210084
	$L(Max)$	0.439641987	0.438109356	0.441174618	0.439701512
	$L(LDS_{(0.7)})$	0.821201543	0.817731314	0.824671773	0.821418139

TABLE 4
The Simulations and Analytical Results of TW_s

λ	Strategies	Simlation 95% C.I. for TW_s			Analytical
		Mean	Lower	Upper	
3	$L(P)$	0.21450975	0.213643288	0.215376213	0.214285714
	$L(Max)$	0.470989755	0.468896695	0.473082816	0.470663564
	$L(LDS_{(0.7)})$	0.934879436	0.931511882	0.93824699	0.932063778
6	$L(P)$	0.272723498	0.271524889	0.273922108	0.272727273
	$L(Max)$	0.677850658	0.674511593	0.681189722	0.676783649
	$L(LDS_{(0.7)})$	1.278779518	1.272516715	1.285042322	1.276946685
10	$L(P)$	0.427856213	0.426306032	0.429406394	0.428571429
	$L(Max)$	0.62903687	0.626605339	0.631468401	0.629095459
	$L(LDS_{(0.7)})$	1.098952226	1.093767233	1.104137219	1.099430242

The hardware environment is: 2×Dell PowerEdge T720 (2×CPUs: Xeon 6-core E5-2630 2.3G, 12 cores in total). Simulation experiments settings:

1) Tasks number: cloudletsNo=100000, tasks length (GIPS): -log(rand(1,cloudletNo))/$(f_i/\overline{I})$), and λ : {0.5,1,2,3,4,5,6,7,8,9,10,11,12,13}.
2) The load distribution strategies in the multi-server control model are $L(P)$, $L(Max)$ and $L(LDS_{(0.7)})$ respectively.
3) Process: simulation experiment times: 20. We recorded the response time and the waiting time of each host in each datacenter. After 20 runs, we calculated the average values of the mean queue waiting time (TW_q) and mean response time (TW_s). Then, we calculated the 95 percent confidence interval (95 percent C.I.) for TW_q and TW_s to validate these two metrics in the control model.

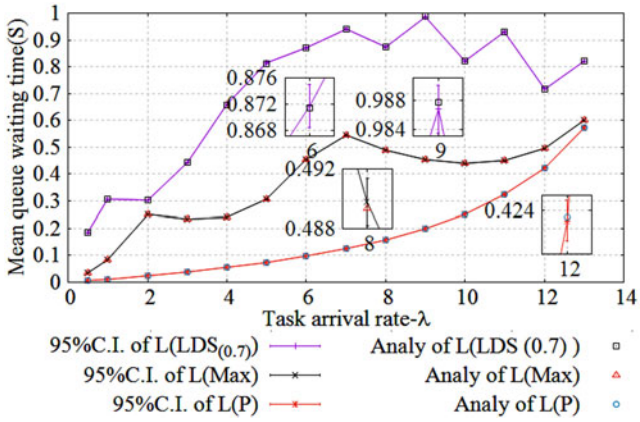

Fig. 3. Simulation 95 percent C.I. for TW_q versus analytical.

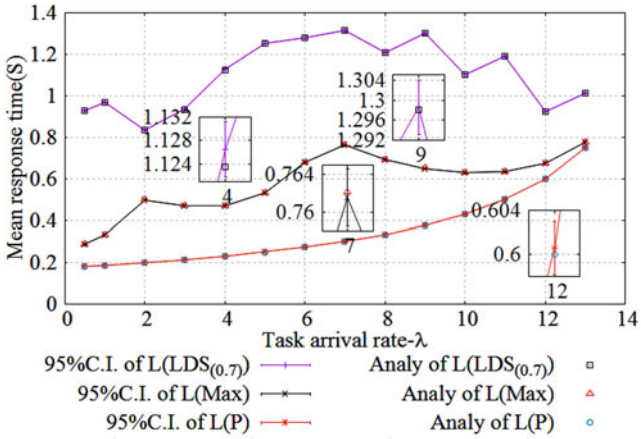

Fig. 4. Simulation 95 percent C.I. for TW_s versus analytical.

4) The results were calculated using strategies $L(P)$, $L(Max)$ and $L(LDS_{(0.7)})$ respectively.

The simulations and the results (λ : 3, 6,10) are shown in Tables 3 and 4.

The comparisons between the analytical and the simulations results are shown in Figs. 3 and 4 (λ: 0.5,1,2,3,4,5,6,7,8, 9,10,11,12,13).

From the simulation results and the analytical results shown in Tables 3 and 4 and Figs. 3 and 4, we can observe that the analytical results of TW_q and TW_s are all in the range of the 95 percent C.I. It demonstrates that the analytical results are consistent with the CloudSim simulation results. It also confirms that the metrics of TW_q and TW_s in the control model can be trusted within the 95 percent C.I.

5.2 Comparison Between $L(P)$ and $L(Max)$ Under C_i=0

The cost coefficient C_i can be treated as the tradeoff factor between the queue waiting cost and the cost of operation and maintenance. If C_i=0, the cost of operation and maintenance is not considered, and (5) is rewritten as

$$\min_{\{\lambda,\overline{\mu_c^i}\}} F(\lambda) = \sum_{i=1}^{n} \left(\frac{\lambda_i^2}{\overline{\mu_c^i}\left(\overline{\mu_c^i} - \lambda_i\right)\lambda} \right), \lambda > 0. \tag{13}$$

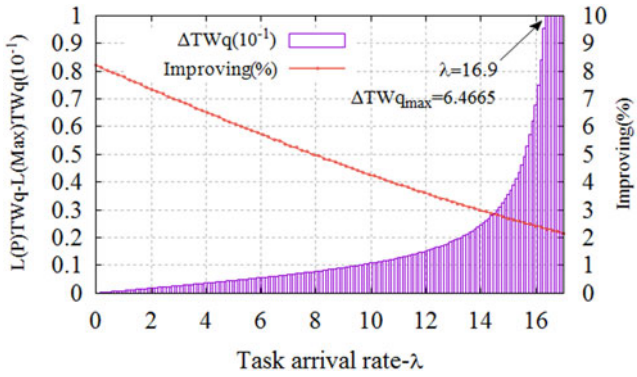

Fig. 5. TWq and its improvement.

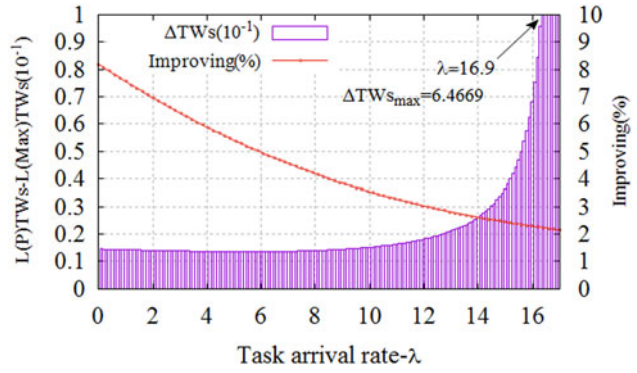

Fig. 6. TWs and its improvement.

Thus, the optimal dispatching strategy $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n)$ is to minimize the queue waiting time. Figs. 5 and 6 illustrate the subtraction results of the $L(P)$ load distribution and the $L(Max)$ load distribution with different λ values in the case of mean queue waiting time (TWq) and mean response time (TWs). Let $\Delta TWq = TWq_{L(P)} - TWq_{L(Max)}$, $\Delta TWs = TWs = TWs_{L(P)} - TWs_{L(Max)}$.

As shown in Figs. 5 and 6, the values of TWq and TWs increase with the task arrival rate λ. The improvement in both mean queue waiting time and mean response time ranges from 2.15 percent to 8.21 percent. The load proportion of S_1, S_2, and S_3 using these two strategies are shown in Fig. 7.

Numerical simulation results show that the load distribution strategy of $L(Max)$, as used by Algorithm $Calculate\_lds\ (C_i[\,],\ \mu_c^i[\,],\ \lambda,\ n)$ to assign load to each server in our model, is better than the proportion load distribution $L(P)$ both in terms of the mean queue waiting time and the mean response time under the condition of C_i=0.

5.3 Comparison of Performance Metrics Among the Strategies

If we consider the cost of operation and maintenance and the mean queue waiting time (the mean system waiting time), the cost coefficients will be taken into account in (5). Since the servers are all configured with variable service rates in the $L(LD)$, $L(LDS_{(1)})$, and $L(LDS_{(0.7)})$ strategies, by applying Little's result the mean queue waiting time and the mean response time are formulated as follows:

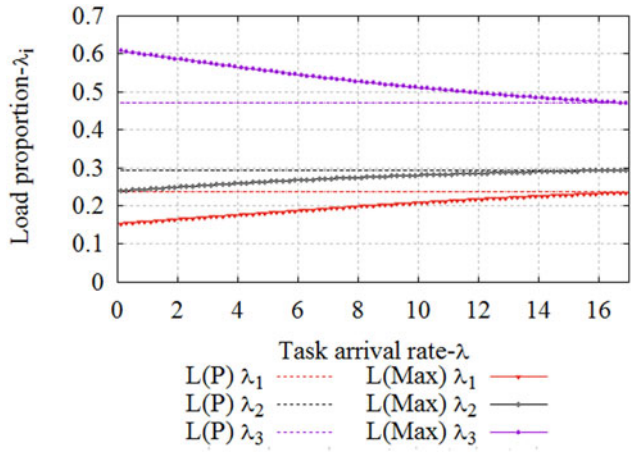

Fig. 7. Load proportions of $L(P)$ and $L(Max)$.

1) The mean queue waiting time:

$$TW_q = \sum_{i=1}^{n} \frac{\lambda_i}{\lambda} \frac{L_q(i)}{\lambda_i} = \begin{cases} \frac{1}{\lambda}\sum_{i=1}^{n}\left(F_l^i(k_i) - 1 + p(i,0,0)\right), \lambda = \sum_{i=1}^{n}\lambda_i, \\ \qquad \lambda > 0, k_i \neq 0 \\ \frac{1}{\lambda}\sum_{i=1}^{n} \frac{\lambda_i^2}{\mu_{M_i}^i\left(\mu_{M_i}^i - \lambda_i\right)}, \lambda = \sum_{i=1}^{n}\lambda_i, \lambda > 0, k_i = 0. \end{cases} \tag{14}$$

2) The mean response time:

$$TW_s = \sum_{i=1}^{n} \frac{\lambda_i}{\lambda} \frac{L_s(i)}{\lambda_i} = \begin{cases} \frac{1}{\lambda}\sum_{i=1}^{n} F_l^i(k_i), \lambda = \sum_{i=1}^{n}\lambda_i, \lambda > 0, k_i \neq 0 \\ \frac{1}{\lambda}\sum_{i=1}^{n} \frac{1}{\mu_{M_i}^i - \lambda_i}, \lambda = \sum_{i=1}^{n}\lambda_i, \lambda > 0, k_i = 0. \end{cases} \tag{15}$$

The $L_q(i)$ denote the length of tasks waiting in the queue and $L_s(i)$ [which equates to $F_l^i(k_i)$] denote the length of the task queue (including the task of being executed in the server).

The mean queue waiting time and the mean response time of the compared load distribution strategies are shown in Figs. 8 and 9, respectively.

Since $C_i \neq 0$ ($C_1 = 0.4$, $C_2 = 0.5$, and $C_3 = 0.6$), the cost of operation and maintenance and the mean queue waiting time are all taken into account in the model, which is shown in the target function (5), the mean queue waiting time and the mean response time are all slightly greater than the proportion load distribution ($L(P)$). In the load distribution strategies $L(LD)$, $L(LDS_{(1)})$ and $L(LDS_{(0.7)})$, the server is treated as a $M/M/1$ queuing system with variable service rates, and the service rate is adjusted by the state of the task queue. As shown in Figs. 8 and 9, the mean queue waiting time and the mean response time of $L(LD)$, $L(LDS_{(1)})$, and $L(LDS_{(0.7)})$ are slightly greater than $L(Max)$, where the service rate of each server is configured with the max service rate $\mu_{M_i}^i$.

We also observe that the mean queue waiting time and the mean response time of $L(LD)$ and $L(LDS_{(1)})$ will oscillate, especially increasing with task arrival rate λ. Since the server is configured with a variable service rate and the load balancing will be more important under high-load conditions, the

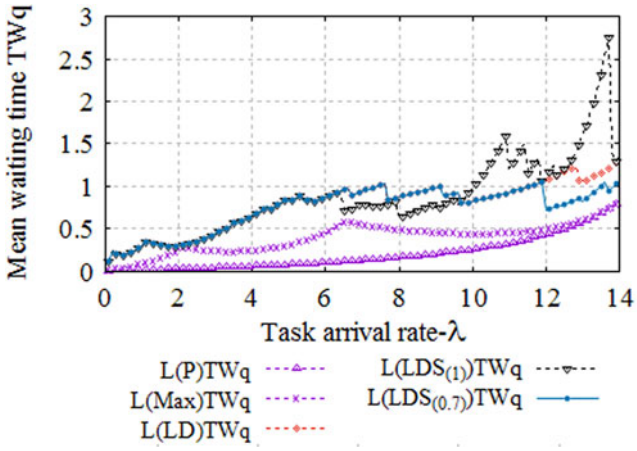

Fig. 8. Mean queue waiting time.

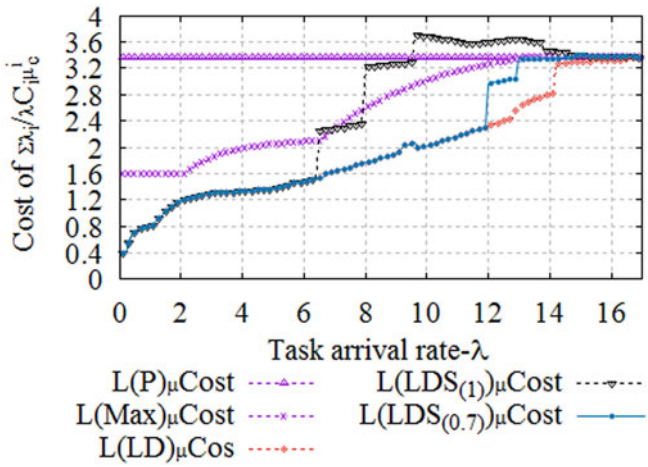

Fig. 10. Cost of operation and maintenance.

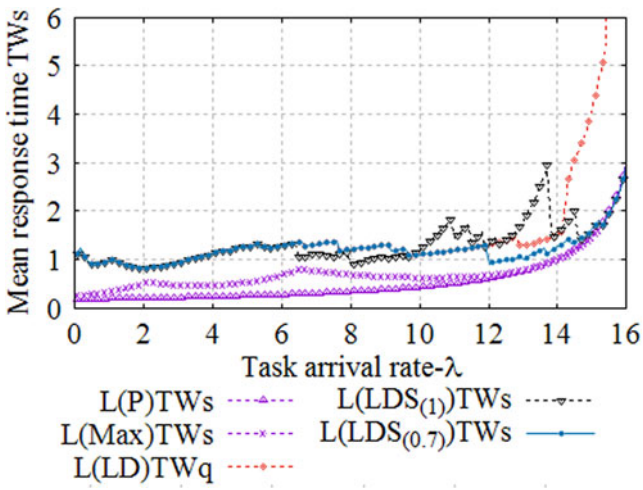

Fig. 9. Mean response time.

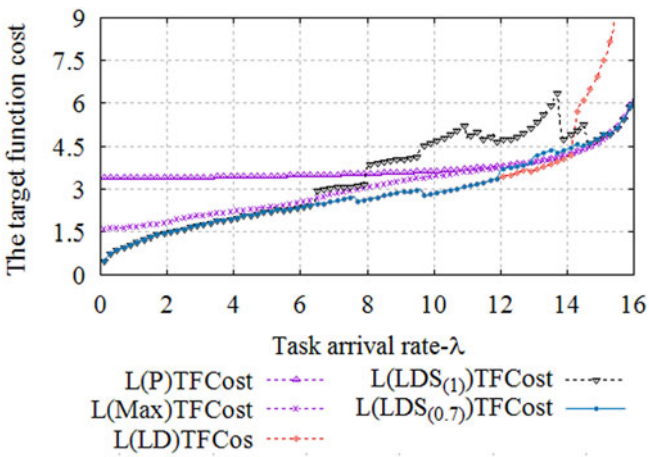

Fig. 11. Target cost.

mean service rate of each server will affect the DFC controlling the dispatcher of each server. Thus, it will bring about oscillation in $L(LD)$ and $L(LDS_{(1)})$. However, if the switch threshold in the SRC is set as $\xi = 0.7$, most of the oscillation will be avoided. This can be observed in Figs. 8 and 9.

5.4 Cost Comparison

We consider the cost of operation and maintenance, the part of $\sum_{i=1}^{n}(\frac{\lambda_i}{\lambda}C_i\overline{\mu_c^i})$, and the value of the target function (5).

As shown in Fig. 10, since the $L(LDS_{(1)})$ controls SRC-directed alterations of the service rate by using the utilization threshold $[\Theta_l^i, \Theta_h^i]$, it will make the service rate increases faster than those of the other strategies; the cost of operation and maintenance will be greater than the cost of $L(P)$ with the task arrival rate. In Fig. 11, it can be observed that the costs of $L(LD)$ and $L(LDS_{(1)})$ are greater than the others since $\lambda > 14.2$ and $\lambda > 8$, respectively. These oscillating behaviors are caused by the service rate increase and the oscillation of the mean queue waiting time. Although $L(LD)$ gives the best result shown in the Fig. 11, the mean queue waiting time is the worst when $\lambda > 14.2$, making the target cost the worst one.

The increase shown in Fig. 10 is impacted by the gap between the two neighboring service rates. If the gap between the two neighboring service rates is small, the increase will be small. From Figs. 8, 9, 10, and 11, we choose the $L(LDS_{(0.7)})$ as the strategy in our multi-server control model, in which the switch threshold ξ in the SRC can be set according to the needs of the cloud computing environment.

5.5 Load Proportion and Mean Service Rate of $L(LDS_{(0.7)})$

The comparisons of the load proportion in each server and the mean service rate of each server between $L(P)$ and $L(LDS_{(0.7)})$ are shown in Figs. 12 and 13, respectively.

As shown in Fig. 12, according to the cost of the server, when the task arrival rate is light, most of the tasks are dispatched to S_1. With increasing task arrival rate, the tasks will be dispatched to S_2 and S_3. In the full load state, all the servers are at full load and the load proportion will be close to $L(P)$. Since the servers are configured with variable service rates and controlled by the SRC according to the task arrival, the mean service rate of each server will increase with the increase of λ.

In Fig. 13, we can observe that the increase is much greater than the increase of the other servers near $\lambda = 12$ and 13. This is because the gap between the two neighboring service rates in the S_3' service rate sequence is greater than that of the other servers. However, if the gap between

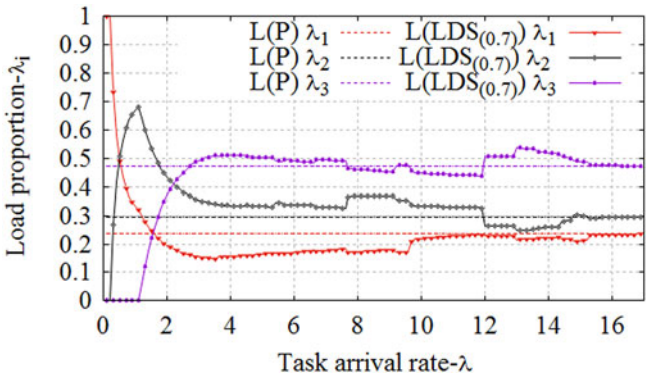

Fig. 12. Load proportions of $L(P)$ and $L(LDS_{(0.7)})$.

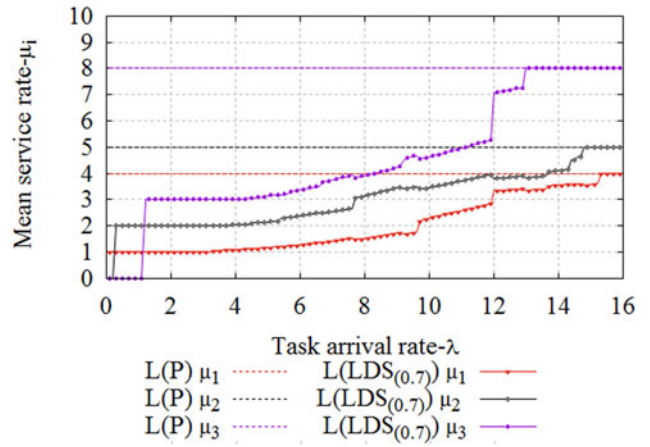

Fig. 13. Mean service rates of $L(P)$ and $L(LDS_{(0.7)})$.

TABLE 5
Parameters Settings 2

i	$\mu^i[\,]$	$M_i^{'}$
1	$\{0.1+\frac{\mu^1_{max}}{M_1^{'}-1} * j\}, j=0,1,\ldots,M_1^{'}, \mu^1_{max}=4$	9
2	$\{0.1+\frac{\mu^2_{max}}{M_2^{'}-1} * j\}, j=0,1,\ldots,M_2^{'}, \mu^2_{max}=5$	11
3	$\{0.1+\frac{\mu^3_{max}}{M_3^{'}-1} * j\}, j=0,1,\ldots,M_3^{'}, \mu^3_{max}=8$	17

the two neighboring service rates is small, the curve of mean service rate will be smooth. This illustrates that we should increase the service rate in the sequence for each server in order to decrease the gap between the two neighboring service rates, as suggested, to smooth the curve in Figs. 8, 9, 10, 11, 12, and 13.

5.6 Impact of M_i

In the following numerical examples, we changed the parameters settings of M_i and $\mu^i[\,]$ based on the parameters shown in Table 1 to compare the impact of selecting different values of M_i. The parameter settings of $M_i^{'}$ and $\mu^i[\,]^{'}$ are shown in Table 5.

We used these settings and ran them on random task arrival rate λ instances for simulation in $200\Delta\tau$ (time units), and the results are shown in Figs. 14, 15, and 16.

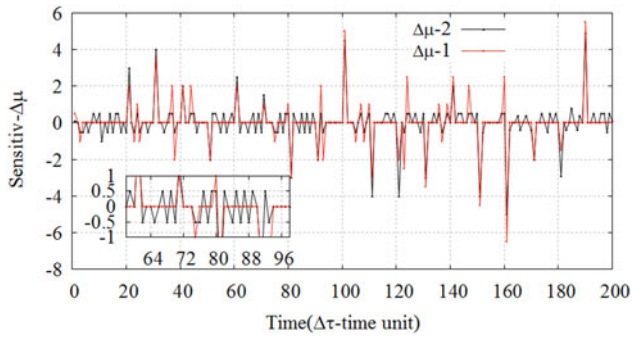

Fig. 14. Sensitivity of $M_i^{'}$ and M_i.

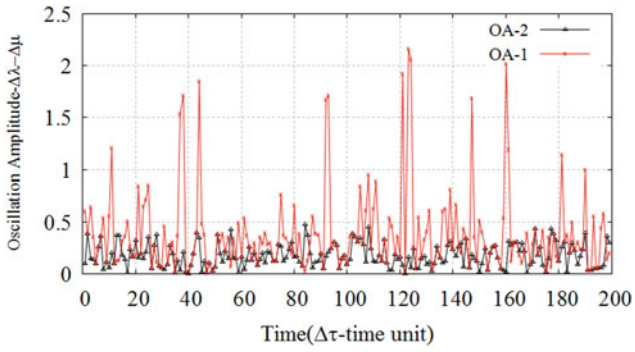

Fig. 15. Oscillation amplitude of $M_i^{'}$ and M_i.

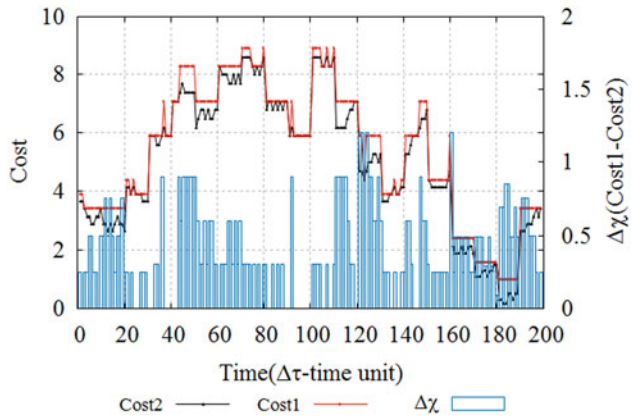

Fig. 16. Instant costs of $M_i^{'}$ and M_i.

As shown in Fig. 14, $\Delta\mu\_1$ and $\Delta\mu\_2$ represent the selections of M_i and $M_i^{'}$, respectively. Since, with the setting of $M_i^{'}$, $\Delta\mu$ is higher than with the setting of M_i, $\Delta\mu\_2$ is more highly sensitive to the slightest change of task arrival rate than $\Delta\mu\_1$. However, the higher sensitivity will create a slight oscillation, as shown in the inset of Fig. 14. The M_i selection impacts the responsiveness of the system to λ, and it should be set within a reasonable scope by the cloud provider for acceptable system jitter according to the status of the server and the features of λ.

In Fig. 15, OA_2 and the OA_1 represent the oscillation amplitudes of $M_i^{'}$ and M_i, respectively. To meet the change of $\Delta\lambda$, the oscillation amplitude of OA_1 is greater than that of OA_2, and since the adjustment is less in OA_2 than in OA_1, its adjustment can better accommodate the change of λ and the excess energy consumption is less than that in OA_1.

TABLE 6
The Configuration of Server Cluster

Num of Servers	CPU	$Memory$	$u_{max}^{s_i}$	C_{si}	M_{si}	(c_p^i, c_a^i, c_h^i)
6732	0.50	0.50	0.50	0.165	10	(0.35,0.35,0.3)
3863	0.50	0.25	0.29	0.096	6	(0.45,0.25,0.3)
1001	0.50	0.75	0.71	0.234	14	(0.40,0.36,0.24)
795	1.00	1.00	1	0.33	20	(0.35,0.33,0.32)
126	0.25	0.25	0.25	0.083	5	(0.42,0.35,0.23)
52	0.50	0.12	0.18	0.059	4	(0.37,0.33,0.3)
5	0.50	0.03	0.11	0.036	2	(0.5,0.35,0.15)
5	0.50	0.97	0.89	0.294	18	(0.42,0.35,0.23)
3	1.00	0.50	0.58	0.191	12	(0.39,0.36,0.25)
1	0.50	0.06	0.13	0.043	3	(0.41,0.37,0.22)

The instant costs of $M_i{}'$ and M_i and the cost difference between them under different values of λ are shown in Fig. 16. In this paper, we only consider the service cost and the energy consumption. Although the instant costs of $M_i{}'$ are less than those of M_i, the additional consumption would be greater and the adjustment strategy of DVFS would be more difficult to realize.

Based on the analysis of the impact of M_i selection, the M_i selection can be adjusted by cloud providers according to server status to achieve reasonable cost, system stability, energy consumption, and the ability to implement the DVFS adjustment strategy.

5.7 Evaluation Using Real Traffic Trace

To verify the multi-server control model working in the actual tasks workload, we use the Google cluster dataset [40], [41] which is a real traffic trace with large scale heterogeneous servers as a task flow to run in the cloud computing environment simulated by MultiRECloudSim and Matlab with the hardware environment shown in Section 5.1. The task workload is scheduled by LDS and the two widely-used representative algorithms: SRPT [42] and PF (Proportional Fair scheduling mechanism) [43], [44] for testing and comparing the target cost index in the heterogeneous datacenter.

The corresponding pretreatment data sets:

1) The configuration of server cluster: In one of the Googles clusters, the configurations of servers and its composition are shown in the first three columns of Table 6. According to the maximum machine which is scaled to 1, it normalized CPU and memory units in the computing node. Since the resources utilization of CPU and memory are the two main factors affecting the service rate of a computing node, and it can obtain the requested CPU-core and memory of each task from the Google clusters dataset, we can used CPU and memory to evaluate the service rate of a computing node $u_{max}^{s_i}$ and the size of a task I_{task_i}. There are more than 80 percent jobs with Memory-Core Ratio per job is less than 5 in Google clusters dataset [45]. Based on these information, we defined the max service rate of a computing node: $u_{max}^{s_i} = (CPU_{si} + Memory_{si} * 5)/6$, which CPU_{si}, $Memory_{si}$ are the CPU and memory units respectively, and the size of a task: $I_{task_i} = (CPU_r + Memory_r * 5)/6$,which CPU_r, $Memory_r$ are the requested CPU-core and memory of a task respectively. By these definitions and the price of a server in Aliyun, we use an analog sampling method to add four columns: the service rate usimax, the cost coefficient C_{si}, the number of variable service rates M_{si} and the adjustment cost coefficient (c_p^i, c_a^i, c_h^i). The configuration of server cluster in our experiment is shown in Table 6.
2) Task sequence: We filtered to obtain about 1 million which with non-production priority and complete normally without stopping or resuming in a life-cycle after it had been submitted in the Google cluster dataset as the arrival task sequence shown with the button lines in Figs. 17 and 18 for the experiment. All the tasks will be sent to the main node and then be dispatched to the computing node based on its current status for execution. The tasks arrival rate λ will be evaluated by the arrival tasks number with the internal 5 minutes and the size of a task I_{task_i} which has been defined above is set by its resource request.

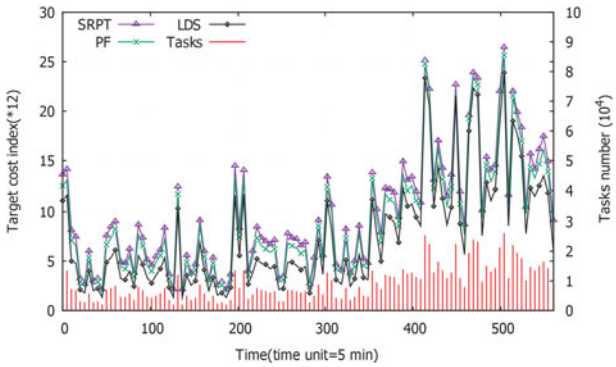

Fig. 17. The target cost index of LDS, SRPT, and PF.

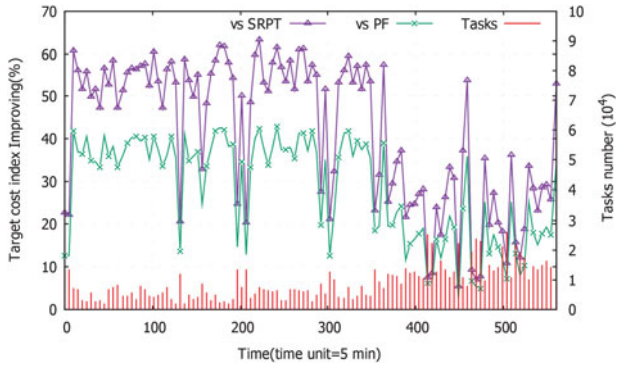

Fig. 18. The target cost index improving than SRPT and PF.

In the experiment, it sets the heterogeneous server cluster with 12583 nodes and its configurations are shown in Table 6, and the target function cost of (5)- target cost index which represents the cost of nodes in a unit time is calculated with the internal 5 minutes. The results of the experiments are shown in Figs. 17 and 18.

From Figs. 17 and 18, it can observe that the target cost index of LDS improved about 45-65 percent than SRPT and 42 percent than PF when the cluster work in low task intensity, especially in X axis(times) from 40 to 200. It is because LDS assigns tasks according to the cost of each node in the cluster with tradeoff between the cost saving and the waiting time of a task, while SRPT and PT are mainly to improve

the response time without consider the cost of node. The nodes with high cost coefficients will be dormant controlled by LDS in low task intensity. The target cost indices of LDS, SRPT and PF are similar in high task intensity. For example in X-time=420, 475 and 525, the improving of LDS than SRPT and PF are down to 5-7 percent, since it is more nodes to participate in work.

As shown in Figs. 10, 11, 17 and 18, the effect of LDS is significant which improving more than 40 percent with dealing middle-low task intensity. It can effectively guide the system to deploy suitable server clusters dynamically in heterogeneous cloud data centers to respond to the appropriate task intensity to obtain a better performance ratio. Since the tasks had been filtered and selected for the experiment only accounts for about 10-15 percent of the service capacity of the server cluster [46], it can obtain such a high improvement, partly as high as 65 percent, which verifies this characteristic of the algorithm presented in this paper.

6 Conclusion

Optimal load distribution and speed scaling in a heterogeneous data center that considers the tradeoff between performance and the cost of operation and maintenance are crucial aspects of cloud computing for both cloud providers and cloud service customers. A dynamic feedback and queue waiting cost-aware optimal load distribution and multi-server control model is presented for cloud computing in heterogeneous data centers. Each server in a data center is configured with variable service rates as an $M/M/1$ queue system with variable service rate, and the service rate adjustment is controlled by the length of the task queue. After receiving dynamic feedback from the SRC of each server, the DFC controls the dispatcher to assign tasks to the servers. By using the theory of convex optimization and queuing theory, we have demonstrated an analytical approach that considers the tradeoff between the queue waiting time and the consumption of operation and maintenance, to study optimal load and cost distribution. The two optimization problems—the optimal load distribution problem and the optimal controlling service rate problem—have been formulated in this paper. The main algorithms have been provided in the model to solve the optimization problems and to adjust the server's service rate. We also have demonstrated some numerical and simulation examples to validate our model. Numerical and simulation results show that in our model, if one only wants to optimize the queue waiting time and response time ($C_i = 0$), one should choose the $L(Max)$ load distribution strategy. In addition, if one considers the tradeoff between performance and the cost of operation and maintenance, one should choose the $L(LDS_{(0.7)})$ load distribution strategy with reasonable M_i and $\mu^i[\,]$ selection for cloud computing in heterogeneous data centers.

Acknowledgments

This work was supported by National Natural Science Foundation (61902232, 61662013, U1811264, 61662015, 61562014); Guangxi Innovation-Driven Development Project (Science and Technology Major Project)(AA17202024). The Guangxi Natural Science Foundation (2017GXNSFAA198372).

References

[1] W. Voorsluys, J. Broberg, and R. Buyya, "Introduction to cloud computing," *Cloud Comput., Principles Paradigms*, 2011, pp. 1–44.
[2] C. Delimitrou and C. Kozyrakis, "Paragon: QoS-aware scheduling for heterogeneous datacenters," in *Proc. 18th Int. Conf. Architect. Support Programm. Lang. Operating Syst.*, 2013, pp. 77–88.
[3] J. Mars, L. Tang, and R. Hundt, "Heterogeneity in "homogeneous" warehouse-scale computers: A performance opportunity," *IEEE Comput. Archit. Lett.*, vol. 10, no. 2, pp. 29–32, Jul. 2011.
[4] H. Khazaei, J. Misic, V. Misic, and S. Rashwand, "Analysis of a pool management scheme for cloud computing centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 5, pp. 849–861, May 2013.
[5] Y. Chunxia and J. Shunfu, "An energy-saving strategy based on multi-server vacation queuing theory in cloud data center," *J. Supercomputing*, vol. 74, no. 12, pp. 6766–6784, 2018.
[6] N. Megow and J. Verschae, "Dual techniques for scheduling on a machine with varying speed," *SIAM J. Discrete Math.*, vol. 32, no. 3, pp. 1541–1571, 2018.
[7] K. Li, "Optimal load distribution for multiple classes of applications on heterogeneous servers with variable speeds," *Softw., Practice Experience*, vol. 48, no. 10, pp. 1805–1819, 2018.
[8] 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. Comput.*, vol. 63, no. 1, pp. 45–58, Jan. 2014.
[9] X. Zhu, L. T. Yang, H. Chen, J. Wang, S. Yin, and X. Liu, "Real-time tasks oriented energy-aware scheduling in virtualized clouds," *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 168–180, Third Quarter 2014.
[10] L. Ruan, X. Xu, L. Xiao, F. Yuan, Y. Li, and D. Dai, "A comparative study of large-scale cluster workload traces via multiview analysis," in *Proc. IEEE 21st Int. Conf. High Perform. Comput. Commun.; IEEE 17th Int. Conf. Smart City; IEEE 5th Int. Conf. Data Sci. Syst.*, 2019, pp. 397–404.
[11] H. Kameda, J. Li, C. Kim, and Y. Zhang, *Optimal Load Balancing in Distributed Computer Systems*, Berlin, Germany: Springer, 1997.
[12] C. Kim and H. Kameda, "An algorithm for optimal static load balancing in distributed computer systems," *IEEE Trans. Comput.*, vol. 41, no. 3, pp. 381–384, Mar. 1992.
[13] X. Tang and S. Chanson, "Optimizing static job scheduling in a network of heterogeneous computers," in *Proc. Int. Conf. Parallel Process.*, 2000, pp. 373–382.
[14] K. Nuaimi, N. Mohamed, M. Nuaimi, and J. Al-jaroodi, "A survey of load balancing in cloud computing: Challenges and algorithms," in *Proc. 2nd Symp. Netw. Cloud Comput. Appl.*, 2012, pp. 137–142.
[15] V. Kumar, A. Grama, and N. Vempaty, "Scalable load balancing techniques for parallel computers," *J. Parallel Distrib. Comput.*, vol. 22, no. 1, pp. 60–79, 1994.
[16] G. Xu, J. Pang, and X. Fu, "A load balancing model based on cloud partitioning for the public cloud," *Tsinghua Sci. Technol.*, vol. 18, no. 1, pp. 34–39, 2013.
[17] D. Grosu and A. Chronopoulos, "Noncooperative load balancing in distributed systems," *J. Parallel Distrib. Comput.*, vol. 65, no. 9, pp. 1022–1034, 2008.
[18] V. Cardellini, M. Colajanni, and P. Yu, "Dynamic load balancing on web-server systems," *IEEE Internet Comput.*, vol. 3, no. 3, pp. 28–39, May 1999.
[19] L. Chen and N. Li, "On the interaction between load balancing and speed scaling," *IEEE J. Sel. Areas Commun.*, vol. 33, no. 12, pp. 2567–2578, Dec. 2015.
[20] B. Guenter, N. Jain, and C. Williams, "Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning," in *Proc. IEEE INFOCOM*, 2011, pp. 1332–1340.
[21] S. Albers, F. Müller, and S. Schmelzer, "Speed scaling on parallel processors," *Algorithmica*, vol. 68, no. 2, pp. 404–425, 2014.
[22] K. Li, "Optimal power allocation among multiple heterogeneous servers in a data center," *Sustain. Comput. Inform. Syst.*, vol. 2, no. 1, pp. 13–22, 2012.
[23] A. Wierman, L. Andrew, and A. Tang, "Power-aware speed scaling in processor sharing systems: Optimality and robustness," *Perform. Eval.*, vol. 69, no. 12, pp. 601–622, 2012.
[24] K. Son and B. Krishnamachari, "Speedbalance: Speed-scaling-aware optimal load balancing for green cellular networks," in *Proc. IEEE INFOCOM*, 2012, pp. 2816–2820.
[25] J. George and J. Harrison, "Dynamic control of a queue with adjustable service rate," *Operations Res.*, vol. 49, no. 5, pp. 720–731, 2001.

[26] B. Mitchell, "Optimal service-rate selection in an M|G|1̂ Queue" *SIAM J. Appl. Math.*, vol. 24, no. 1, pp. 19–35, 1973.
[27] L. Li, J. Wang, and F. Zhang, *Customers' Equilibrium Balking Strategies in an M/M/1 Queue with Variable Service Rate*. Berlin, Germany: Springer, 2013.
[28] X. Zhang, J. Wang, and Q. Ma, *Convexity Results for Queueing System with Variable Service Rate*. Berlin, Germany: Springer, 2014.
[29] A. Greenberg, J. Hamilton, D. Maltz, and P. Patel, "The cost of a cloud," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68–73, 2008.
[30] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, Nov. 2011, pp. 1–12.
[31] G. Jung, M. Hiltunen, K. Joshi, R. Schlichting, and C. Pu, "Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2010, pp. 62–73.
[32] H. Goudarzi, M. Ghasemazar, and M. Pedram, "SLA-based optimization of power and migration cost in cloud computing," in *Proc. IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2012, pp. 172–179.
[33] A. Aral and T. Ovatman, "Network-aware embedding of virtual machine clusters onto federated cloud infrastructure," *J. Syst. Softw.*, vol. 120, pp. 89–104, 2016.
[34] L. Nonde, T. E. El-Gorashi, and J. M. Elmirghani, "Energy efficient virtual network embedding for cloud networks," *J. Lightwave Technol.*, vol. 33, no. 9, pp. 1828–1849, 2014.
[35] Y. Hong, J. Xue, and M. Thottethodi, "Dynamic server provisioning to minimize cost in an IaaS cloud," in *Proc. ACM SIGMETRICS Joint Int. Conf. Meas. Modeling Comput. Syst.*, 2011, pp. 147–148.
[36] A. Wolke and G. Meixner, *TwoSpot: A Cloud Platform for Scaling Out Web Applications Dynamically*. Berlin, Germany: Springer, 2010.
[37] P. Arroba, J. Moya, J. Ayala, and R. Buyya, "Dynamic voltage and frequency scaling-aware dynamic consolidation of virtual machines for energy efficient cloud data centers," *Concurrency Comput.*, vol. 29, no. 10, 2016, Art. no. e4067.
[38] A. Florence, V. Shanthi, and C. Simon, "Energy conservation using dynamic voltage frequency scaling for computational cloud," *Sci. World J.*, vol. 2016, no. 1, 2016, Art. no. 9328070.
[39] W. Lin, S. Xu, L. He, and J. Li, "Multi-resource scheduling and power simulation for cloud computing," *Inf. Sci.*, vol. 397, no. 1, pp. 168–186, 2017.
[40] C. Reiss, J. Wilkes, and J. Hellerstein, "Google cluster-usage traces: Format+ schema," *Google Inc., White Paper*, vol. 2011, no. 1, pp. 1–14, 2011.
[41] K. Psychas and J. Ghaderi, "Randomized algorithms for scheduling multi-resource jobs in the cloud," *IEEE/ACM Trans. Netw.*, vol. 26, no. 5, pp. 2202–2215, Oct. 2018.
[42] X. Ren, G. Ananthanarayanan, A. Wierman, and M. Yu, "Hopper: Decentralized speculation-aware cluster scheduling at scale," *Comput. Commun. Rev.*, vol. 45, no. 5, pp. 379–392, 2015.
[43] R. Margolies *et al.*, "Exploiting mobility in proportional fair cellular scheduling: Measurements and algorithms," *IEEEACM Transactions on Networking*, vol. 24, no. 1, pp. 355–367, Feb. 2016.
[44] S. Singh, M. Geraseminko, S.-P. Yeh, N. Himayat, and S. Talwar, "Proportional fair traffic splitting and aggregation in heterogeneous wireless networks," *IEEE Commun. Lett.*, vol. 20, no. 5, pp. 1010–1013, May 2016.
[45] Y. Chen and A. S. Ganapathi, R. Griffith, and R. H. Katz, "Analysis and lessons from a publicly available google cluster trace," EECS Department, Univ. California, Berkeley, CA, Tech. Rep. UCB/EECS-2010–95, 2010.
[46] Z. Liu and S. Cho, "Characterizing machines and workloads on a google cluster," in *Proc. 41st Int. Conf. Parallel Process. Workshops*, 2012, pp. 397–403.

Weihua Bai received the ME degree from the School of Computer Science, South China Normal University, in 2006, and the PhD degree from the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China, in 2017. He is currently an associate professor with the School of Computer Science, Zhaoqing University. His research interests include cloud computing, parallel scheduling, and software architecture. He is a member of the China Computer Federation.

Jiaxian Zhu received the BE degree from the Department of Computer Science, South China Normal University, in 2000, and the ME degree from the School of Computer Science, Guangdong University of Technology, Guangzhou, China, in 2006. She is currently an associated professor with the School of Computer Science, Zhaoqing University. Her research interests include cloud computing, parallel scheduling, and semantic web.

Shaowei Huang received the BE degree from the Department of Electrical and Electronics Engineering, Nankai University, Tianjin, China, in 2002, and the ME and PhD degrees from the Department of Electrical, Electronics and Information Engineering, Osaka University, Osaka, Japan, in 2006 and 2008, respectively. He is currently an associate professor with the School of Computer Science of Zhaoqing University. His current research interests include large-scale and complex network architecture.

Huibing Zhang received the MS degree in computer science and technology from the Guilin University of Electronic Technology, Guilin, China, in 2005, and the PhD degree in computer science and technology from the Beijing University of Technology, Beijing, China, in 2012. He is currently an associate professor with the Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology. His research interests include Service-Oriented Computing and Social Computing.