Received: 30 January 2020 | Revised: 14 May 2020 | Accepted: 2 June 2020
DOI: 10.1002/cpe.5926

WILEY

SPECIAL ISSUE PAPER

Unequal-interval based loosely coupled control method for auto-scaling heterogeneous cloud resources for web applications

Zhicheng Cai[1,2] | Duan Liu[1] | Yifei Lu[1] | Rajkumar Buyya[2]

[1]School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China

[2]The Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Parkville, Victoria, Australia

Correspondence
Zhicheng Cai, School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China.
Email: caizhicheng@njust.edu.cn

Funding information
Fundamental Research Funds for the Central Universities, Grant/Award Numbers: 30919011235, 30920120180101; National Natural Science Foundation of China, Grant/Award Numbers: 61602243, 61702267, 61972202; Postgraduate Research and Practice Innovation Program of Jiangsu Province, Grant/Award Number: KYCX18_0434

Summary

Most existing quality of service (QoS) control algorithms of Web applications take into account Web Server or database connections which can be released immediately. However, many applications are deployed on virtual machines (VMs) or even Spot VMs elastically rented from public Clouds. To save costs, interval-priced VMs are not released until the ends of rented intervals. Such delays of control effects make existing methods rent or release excess VMs leading to overcontrol. Fluctuated prices make Spot VMs unreliable due to unexpected termination which makes fault-tolerant strategies crucial. In this article, an unequal-interval-based loosely coupled control method is proposed to improve the quality of service (QoS) control ability of fault-tolerant strategies. A queuing model with arrival-rate-adjustment coefficient is used to predict required capacity as a feedforward controller. Another two-threshold and queuing-model-based method is applied to update the coefficient as a loosely coupled feedback controller. Meanwhile, unequal-interval controller collaborating method is proposed to avoid overcontrol and react quickly to workload changes. Our approach is evaluated on both a simulation platform and a real Kubernetes Cluster. Experimental results illustrate that our approach decreases the percentage of waiting times larger than service level agreements with similar or lower rental costs compared with existing algorithms.

KEYWORDS
Cloud computing, feedback control, queuing model, resource provisioning, Spot VM

1 | INTRODUCTION

Cloud computing offers subscription-oriented services and it is widespread to rent Cloud virtual machines (VMs) elastically to support the running of Web applications.[1,2] VMs are generally priced by time intervals and the hour-based pricing model is especially popular in modern commercial public Clouds. Prices of On-demand VMs of Amazon EC2, Microsoft Azure, Aliyun, and so on are fixed. On the contrary, prices of Spot VMs of Amazon EC2[3,4] are dynamic because Spot VMs are auctioned by public Clouds. Spot VMs are cheaper than On-demand VMs but unreliable due to lease terminations when the current market price becomes higher than the bid. Most existing QoS control methods for Web applications only focus on resources inside one server or On-demand VMs. However, it is beneficial to rent Spot VMs to decrease VM rental costs. The principal goal of this article is to provision Spot and On-demand VMs elastically to minimize resource rental cost while guaranteeing the average waiting time of requests and the robustness. The main challenges of provisioning Spot VMs are interval-based pricing models, nonlinear system performances and unreliability of Spot VMs.

Interval-based pricing models make resource provisioning complicated. Most feedback and feedforward control methods of Web applications mainly focus on allocation of processes of servers,[5,6] Web server sessions,[7] or database connections[8] which can be released quickly. However, Cloud VMs are usually priced by hours, Cloud users need to pay for the whole rented hour even if only the first several minutes are used. Therefore, it wastes costs to release rented VMs immediately after releasing decisions are made. VMs are really released at the ends of rented hours which delays the appearance of control effects. Such delays are likely to mislead controllers to release more VMs leading to overcontrol. Meanwhile, in existing control methods, the feedback controller is only used to amend the output of the feedforward controller which means feedback and feedforward controllers should be invoked together every time. However, the interval-pricing models of VMs limit the frequency of invoking the feedback controller which makes the algorithm react slowly to environment changes. Therefore, one of the challenges is to design appropriate feedback and feedforward collaborating methods considering the unique interval-based pricing models.

Nonlinear system performances and unreliability of Spot VMs make the renting of Spot VMs complex too. The relationship between the system performance and the amount of VM resources is nonlinear. Most existing feedback control methods are based on linear,[5-7] inverse proportional models,[9] or M/M/1 model derived linear model[10,11] which perform well in steady states but poorly confronted with large workload changes. Meanwhile, existing control methods for guaranteeing QoS usually focus on stable resources rather than unreliable Spot VMs. QoS control is not the main focus of existing efficient fault-tolerant strategies[12,13] for Spot VMs. Therefore, another challenge is to design algorithms combining appropriate QoS feedback control and fault-tolerant methods.

In this article, an unequal-interval-based loosely coupled control method (UCM) is proposed which considers both fault-tolerant and QoS guarantees simultaneously. In UCM, the M/M/N queuing model with arrival-rate-adjustment coefficient is applied to predict required resource capacity based on current workload as a feedforward controller. To amend the inaccuracy of the M/M/N queuing model, another M/M/1-model-based feedback controller is used to update the coefficient rather than the output of the feedforward control to minimize the output error. Meanwhile, based on the loosely coupled feedforward and feedback structure, an unequal-interval feedforward-and-feedback collaborating method is proposed which avoids overcontrol and reacts quickly to workload changes by the aid of VM releasing status checking and differentiated feedback-and-feedforward invoking frequencies. At last, a two-threshold-based output error computing method is used to decrease fluctuations incurred by coarse-grained VM capacities. Following are key contributions of this article:

1. A novel loosely coupled control method is proposed, which uses the feedback controller to amend the queuing-model-based feedforward controller by adjusting the parameter of the queuing model rather than the output of the queuing model.
2. An unequal-interval collaborating method is developed to select suitable feedback-and-feedforward invoking frequencies considering the delay of control effects and the quick response to workload changes.
3. The function of adjustment ratio to expected change of waiting time is established based on the M/M/1 queuing model to improve the accuracy of feedback control and a two-threshold-based control error computing method is applied to decrease the instability of control.

Following is the structure of rest parts of this article. Section 2 consists of a brief review of related work. An existing group-based fault-tolerant (GFT) Spot VM renting method is described in Section 3 followed by problem description in Section 4. Section 5 introduces the proposed loosely coupled control method. Evaluation results are presented in Sections 6 and 7 is about conclusions and future work.

2 | RELATED WORK

Methods for guaranteeing QoS of Web applications can be categorized into two types: proactive and reactive methods.[14] For example, the queuing theory[15-19] and the reinforcement learning[20-22] based feedforward controls are proactive methods. Linear or nonlinear performance model[5,6,8] and threshold[4] based feedback controls are reactive methods. It has been proved that using the threshold-based feedback controls and the reinforcement-learning-based feedforward controls are both tricky and even fail without much care.[14,23] For example, the threshold-based feedback controls always need users to manually set rules[24] based on information of specific applications. Reinforcement leaning usually requires a very long initialization and learning period which needs careful designing of convergence speed-up techniques.[23] The queuing-model-based feedforward control is an effective and efficient method to predict the required resource capacity for scenarios with dynamic arrival rates.[25,26] Jiang et al[24] proposed a M/M/N-queuing-model-based feedforward VM provisioning method to guarantee QoS and to minimize the rental cost simultaneously. Wang et al[27] developed an unequal-server-based queuing model to allocate resources of a data center to different applications. However, there are unavoidable deviations between queuing models and the real environment. Feedforward-based methods lack the ability to react to real-time performance.

Linear[5-8,28] or inverse proportional[9] performance-model-based feedback control methods have been widely used in traditional computing systems to adjust provisioned resources according to real-time performance. However, there are many challenges when feedback control is applied to Cloud computing applications.[31] Most existing feedback control methods are based on fine-grained resources such as Web Server processes or database connections rather than coarse-grained VMs with fixed capacities, and these fine-grained resources can be allocated and released quickly which are quite different with interval-charged Cloud VMs. Linear-model or multilinear-model-switching-based feedback control was applied by Lu et al[5,6] and Patikirikorala et al[7] to allocate processes or sessions of Web Servers to different classes of requests for providing differentiated services. Pan et al[8] and Karlsson et al[29] developed linear-model-based feedback control methods to distribute limited database connections with fixed or on-line estimated gain parameters. Padala et al[28] applied online parameter estimating techniques to increase the robustness of linear-model-based feedback control for distributing CPU and disk of physical machines to different applications. However, linear or inverse proportional performance models cannot describe the system accurately which make the feedback control response to environment changes inaccurately because computing systems usually have complex nonlinear performance characteristics. Meanwhile, because of the delay of control effect resulted from interval-based pricing models of Cloud VMs, existing feedback control methods are likely to rent or release excess VMs. There are also some existing algorithms designed for provisioning VMs elastically. Lim et al[31] proposed a linear-model-based feedback control method to control the CPU utilization by adjusting the amount of VMs. Al-Shishtawy and Vlassov[32] applied a linear-model-based feedback control method using the ratio of throughput and the number of VMs as control input for online store system. In order to describe the system more accurately, Baresi et al[9] developed an inverse-proportional-performance-model-based proportional-integral (PI) controller to allocate containers to Cloud applications. However, these methods do not take into account Spot VMs or the VM releasing delay.

To guarantee Web application QoS, there is a trend of using feedback-control to compensate the inaccuracies of queuing models[10,11,30] or other proactive methods.[33] The combining of queuing-models and feedback controls is an effective and fast method to make the system follow a referenced average waiting time.[34] However, in existing feedback and feedforward hybrid control algorithms for QoS control of Web applications, the feedback controller is usually only used to amend the output of the feedforward controller (called parallel connection) which is not suitable for interval-charged Cloud VMs. Sha et al[10] applied a first-order linear-model-based PI controller to amend the queuing model. Similarly, the linear-model-based feedback control was used by Lu et al[11] to adjust the output of a queuing-model-based feedforward controller. Xu et al[30] developed a hybrid control method which also uses the proportional-integral derivative controller to correct inaccuracies of queuing models. The interval-based pricing model limits the frequency of invoking feedback controllers to avoid overcontrol. For example, feedback controllers cannot be invoked within at most 1 hour waiting for VMs to be really released at the ends of rented hours. However, the structure of traditional parallel connection limit that feedback and feedforward controllers must be invoked together which delays the response to workload changes.

There are already some fault-tolerant scheduling methods for Cloud Web applications. A GFT Spot VM renting method for Web applications was investigated by Qu et al[12] which rents multiple groups of different types of Spot VMs to increase the robustness of the system. Then, Liu et al[13] extended the group-based method by applying price forecasting and setting minimum VM renting durations to decrease the rental cost further. These two methods mainly focus on the fault-tolerant strategies without taking into account QoS control. It is beneficial to combine fault-tolerant strategies and interval-pricing-model-aware QoS control methods to improve robustness, decrease rental costs and guarantee the QoS.

A comparison between our approach UCM and existing algorithms is shown in Table 1. UCM consists of both interval-pricing-model-aware QoS control and fault-tolerant methods. In UCM, the feedback and feedforward controllers are loosely coupled rather than connected in parallel. The feedback controller is used to update the parameter of the M/M/N-based feedforward controller. The loosely coupled structure makes it possible to invoke feedback and feedforward controllers with unequal time intervals separately. For example, the feedforward controller can be invoked separately and more frequently than the feedback controller to react to workload changes quickly. Meanwhile, the feedback controller applies a M/M/1-based arrival rate coefficient adjustment model to improve the accuracy of feedback control.

3 | BACKGROUND-EXISTING GFT VM RENTING METHOD

An existing GFT Spot VM renting method[12] is extended by adding Spot price prediction and setting minimum renting duration limitations to decrease rental cost further by Liu et al.[13] The main idea is as follows and details of extended GFT can be found in the reference.[13] Multiple groups of Spot VMs are rented simultaneously to increase the robustness. Each group has the same capacity and consists of the same type of VMs. For a required resource capacity R in the unit of million instructions, On-demand VMs with capacity $R_o = C_o \times N_o$ is rented first where C_o is the capacity of each On-demand VM and N_o is the number of rented On-demand VMs. Then, N_s groups of Spot VMs are rented and each group has the subcapacity of $Q = (R - R_o)/N_s$. In order to increase the fault-tolerant ability, f groups of additional Spot VMs are rented and each group has the capacity of Q too. f is called fault-tolerant level.[12] Therefore, the total capacity of rented Spot VM is $[(R - R_o)/N_s] \times (N_s + f)$.

TABLE 1 Comparison of our UCM with existing resource provisioning methods for Web applications

Algorithms	Resource type	Objectives	Feedforward control	Feedback control	Connection type	Control frequency
GFT[12] and extended GFT[13]	Heterogeneous Spot VMs	VM rental cost, response time	×	×	×	Fixed interval or Minimum renting duration
QT[24]	Homogeneous VMs	VM rental cost, connection delay	M/M/N	×	×	Fixed interval
UQueuing[27]	Heterogeneous VMs	response time	Unequal M/M/N	×	×	Fixed interval
PureFeedback[5-8,28,29]	Web server processes, database connections	Relative and absolute connection delay	×	linear models	×	Fixed interval
EcoWare[9]	VMs and containers	CPU-core rental time and response time	×	Inverse proportional model	×	Fixed interval
HC-FFB-A[10]	Server processes	Response time (connection delay and processing time)	G/G/N	M/M/1 derived linear model	Parallel	Fixed amount of events
HC-FFB-R[11]	Server processes	Relative connection delay	M/M/1	M/M/1 derived linear model	Parallel	Fixed interval
eQos[30]	Server processes	Page-view response time	M/G/1	Direct Integral controller	Parallel	Fixed interval
Proposed UCM	Heterogeneous Spot VMs	VM rental cost, connection delay (waiting time)	M/M/N with adjustable arrival rate	M/M/1-based arrival rate coefficient adjustment model	Loosely Coupled	Unequal time intervals

Abbreviations: GFT, group-based fault-tolerant; VM, virtual machine.

Algorithm 1. Group-based fault-tolerant VM renting method (GFT)

```
Input: required capacity $R$, last required VM capacity $R_c$, threshold of plan lasting time $T_u$, lasting time of current plan $T_p$, number of existing Spot VM groups $N_e$
if $R > R_c$ then
    if $T_p > T_u$ then
        for $N_o \in [0, N_{o\_\max}]$ do
            $R_o \leftarrow C_o \times N_o, N_{\min} \leftarrow \max\{N_e, f+1\}, N_{\max} \leftarrow \min\{N_{sp}, N_{al}\}$
            for $N_s \in [N_{\min}, N_{\max}]$ do
                Calculate $Q \leftarrow (R - R_o)/N_s$ of each Spot group
                Calculate the rental cost of existing $N_e$ groups with new capacity $Q$
                Compute rental cost of each unrented Spot type to fulfill the capacity $Q$ using Spot price predicting
                Choose $N_s - N_e$ unrented Spot types with cheapest rental costs
                Compute renting costs of $N_s$ groups
            end for
        end for
        Select the lowest cost plan ($N_o$, $N_s$, Spot types) as current plan
    else
        Increase the capacity of each group by $Q \leftarrow (R - R_o)/N_s$
    end if
else if $R < R_c$ then
    Decrease the capacity of each group by $Q \leftarrow (R - R_o)/N_s$
end if
$R_c \leftarrow R$
```

For given R and f, selecting appropriate N_o, N_s and Spot VM types of different groups has a great impact on the final rental cost. Different combinations of these values lead to different plans. The formal description of extended GFT is shown in Algorithm 1. Let R_c and N_e be the last required VM capacity used to generate the current plan and the number of existing groups. When the required capacity R is larger than R_c, more resources are needed. When the lasting time T_p of the current plan is shorter than a threshold T_u, only capacity of each group is increased based on the new R. Otherwise, a much cheaper plan is selected as follows. Let $N_{o\_\max}$ be the largest number of VMs if only On-demand VMs are rented. For each candidate $N_o \leq N_{o\_\max}$, different numbers of Spot groups N_s are evaluated. N_s is set to be larger than N_e which means that only plans with more groups of Spot VMs are considered to avoid large fluctuations. N_s should also be smaller than N_{sp} (the number of all available Spot VM types in the system) and N_{al} (the maximum number of allowed to rent Spot VM types). For each Spot type, prices of future h hours are first predicted by a time series analysis method.[13] N_{al} is usually set to be smaller than N_{sp} to increase the possibility of selecting stable Spot types by forecasting because the GFT always tends to rent all available groups to save cost if N_{al} limitation is not set. The rental costs of existing N_e groups with new capacity Q are calculated based on predicted Spot prices first. Then, for each unrented Spot type, the number of required VMs is $\lceil Q/[\text{MIPS} \times (1 - d_{\text{margin}}])\rceil$ where MIPS is the number of instructions performed by the VM type per second and d_{margin} is the percentage of margin resources for sudden burst of workloads. And, the rental cost of each unrented Spot type is obtained based on the predicted Spot prices too. Next, $N_s - N_e$ unrented Spot types with cheapest rental costs are selected. The total cost of all N_s groups are obtained by summing the rental costs of N_e existing groups and selected $N_s - N_e$ unrented Spot types. The process iterates and the cheapest plan is chosen at last. On the contrary, when the required capacity R is smaller than R_c, the plan is kept unchanged and only the capacity of each group is decreased using the new capacity R. When a Spot VM is at a pricing point (the end of the rented interval), it is released only when the capacity of left Spot VMs of its group is still not lower than Q.

4 | PROBLEM DESCRIPTION

The considered Web application consists of one or multiple tiers such as graphic interface tier, business logic tier and database tier. Multiple types of Cloud VMs are rented elastically from public Clouds to establish a virtual data center to support the running of each tier. Each tier is deployed in the form of multiple containers in different types and numbers of rented VMs. These containers are organized by Kubernetes[35] to implement automatic request forwarding and life cycle management. To minimize VM renting costs while guaranteeing the average waiting time, a Resource Scheduler is designed to rent or release VMs dynamically according to real-time workloads for each tier. Service level agreements (SLA) of Web applications usually specify a distribution of average waiting times, for example, κ% of waiting time of the requests should be no more than an upper limit W_{SLA}.[15] Although the resource provisioning of different tiers has impact on each other, it is still general to break the overall W_{SLA} of the application into W_{SLA}

of each tier and design the auto-scaling method for each tier separately for simplification.[26] In this article, a hybrid control method is designed for the Resource Scheduler to adjust the VM capacity of each single tier separately considering both VM renting costs and average waiting times. This control method can be used as resource auto-scaler of single-tier Web applications directly or each single tier of multitier applications.

5 | PROPOSED CONTROL METHOD

In this section, an unequal-interval-based loosely coupled feedforward and feedback control method (UCM) is proposed to adjust the required resource capacity R of the GFT increasing GFT's ability of QoS control. At first, architecture of the hybrid control method is described, followed by introduction to the feedforward and feedback controllers, respectively. Then, the collaborating method of two controllers is presented. Finally, the formal description of UCM is given.

5.1 | Architecture of the loosely coupled control method

The structure of proposed loosely coupled controllers is shown in Figure 1. The M/M/N queuing model is applied to predict the resource requirement based on current request arrival rate and maximum expected waiting time. The output of the feedforward controller is the required resource capacity R which will be used to update VMs of a Web application tier using GFT. In order to amend the inaccuracies of M/M/N model, the arrival rate of the feedforward queuing model is adjusted by multiplying a coefficient φ which is learned dynamically by another queuing-model-based feedback controller. In other words, the feedback controller is in charge of learning the parameter of the queuing-model-based feedforward controller. Then, the feedforward controller can be invoked separately and more frequently than the feedback controller based on learned parameters. In order to make the average waiting time smaller than W_{SLA}, the reference waiting time W_r of the feedback controller should be smaller than W_{SLA}. To be consistent with the feedback controller, the maximum expected waiting time of the feedforward queuing model is set to be W_r too.

5.2 | Queuing model with adjustable-arrival-rate-based feedforward controller

Each Web application tier usually consists of multiple VMs with different processing rates. It is very complex to establish a performance model for a cluster with heterogeneous VMs regarding average waiting times, request arrival rates and VM capacities. For simplification, it is assumed that the cluster only consists of identical VMs of the lowest configuration. And time intervals between arrivals of two requests and the request processing times have negative exponential distributions with parameters λ (arrival rate) and μ (processing rate), respectively. Then, the M/M/N model with identical servers are used to describe the heterogeneous system approximately. In M/M/N, each server is a VM with the lowest configuration and there are N identical servers. Based on given arrival rate λ, processing rate μ of each server and reference waiting time W_r, the minimum number of N can be obtained through queuing theories as follow.

For M/M/N, the probability of no waiting request in the queue is

$$P_0 = \left[\sum_{k=0}^{N-1} \frac{1}{k!} \frac{\lambda}{\mu} + \frac{\lambda^N}{N! \left(1 - \frac{\lambda}{N \times \mu}\right) \mu^N} \right]^{-1} . \tag{1}$$

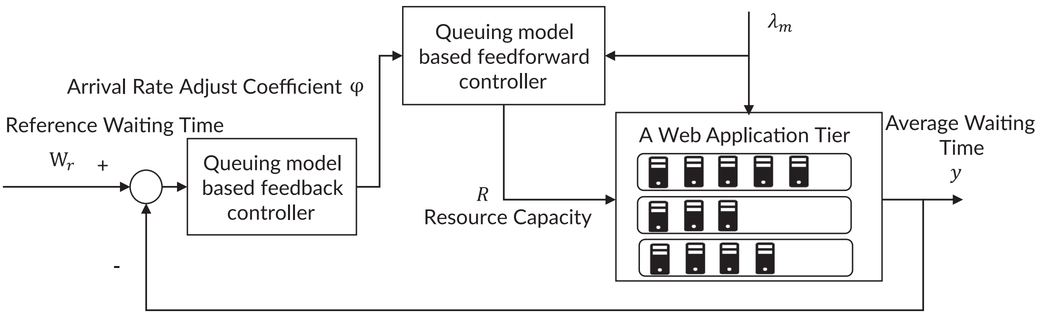

FIGURE 1 Architecture of the loosely coupled control system

The expectation of request number in the queue is

$$L_q = \frac{\left(\frac{\lambda}{\mu}\right)^N \frac{\lambda}{N\times\mu}}{N!\left(1-\frac{\lambda}{N\times\mu}\right)^2} P_0. \tag{2}$$

The expectation of waiting time is

$$W_q = \frac{L_q}{\lambda}. \tag{3}$$

Algorithm 2. Queuing model with adjustable-arrival-rate-based feedforward control (QFC)

```
Input: measured arrival rate $\lambda_m$, processing rate of the lowest configuration VM $\mu$, capacity of the lowest configuration VM $C_l$, current arrival rate adjust coefficient $\varphi(t)$, reference waiting time $W_r$
$\lambda \leftarrow \lambda_m \times \varphi(t)$ and $N \leftarrow \lceil \frac{\lambda}{\mu} \rceil$
while True do
    Calculate $W_q$ based on Equation (3).
    if $W_q \leq W_r$ then
        return $R \leftarrow N \times C_l$
    else
        $N \leftarrow N + 1$
    end if
end while
```

A smallest number of N is found to satisfy $W_q \leq W_r$ to guarantee W_r. Because it is complex to deduce the inverse function of the original function of W_q to N, an exhausted search method[24] is proposed to find the smallest number of servers fulfilling W_r as shown in Algorithm 2. To fix the inaccuracies of queuing model, the measured arrival rate λ_m is multiplied by a coefficient $\varphi(t)$ to generate an adjusted arrival rate λ, that is, $\lambda = \lambda_m \times \varphi(t)$. Then, the basic number of VMs is generated by $N = \lceil \frac{\lambda}{\mu} \rceil$ because the processing rate cannot be smaller than the arrival rate. Next, the number of VMs are increased one by one until the expected waiting time of the queuing model is not larger than W_r. Finally, the required capacity $R = N \times C_l$ is obtained which will be used as the input of the GFT. Figure 2 shows the structure of the feedforward controller.

5.3 | Two-threshold and queuing-model-based feedback controller

Because of the inaccuracies of queuing models and system workload changes, the average waiting time $y(t)$ may deviate from the reference waiting time W_r. The coefficient $\varphi(t)$ should be updated to cope with the current deviation between W_r and $y(t)$. If $y(t) > W_r$, the system is at low-provision state and $\varphi(t)$ should be enlarged to increase R. Otherwise, the system is at overprovision state and $\varphi(t)$ should be shrunk to decrease R. If too much of $\varphi(t)$ is adjusted, the system may skip the normal state and fluctuate between overprovision and low-provision states. Therefore, it is crucial to determine how much to adjust the coefficient $\varphi(t)$ to make the system return back to the normal state. In this article, $\varphi(t)$ is updated by an adjustment ratio ω every time, that is, $\varphi(t) = \varphi(t-1) \times \omega$. Making $y(t)$ follow W_r as much as possible can be modeled as a feedback control problem. ω and $y(t)$ are the input and output of the system, respectively. And the control error is $e(t) = W_q - y(t)$. Because the relationship of $y(t)$ to $\varphi(t)$ is described by a M/M/N-based feedforward controller which is nonlinear, it is very complex to design a feedback controller to transform $e(t)$ to an appropriate control input $\varphi(t)$ directly.

5.3.1 | Linear-model-based abstraction

An abstract linear model is investigated to simplify the analysis in this article. As shown in Figure 3, if the function of adjustment ratio ω to expected change of the waiting time can be established, an ω can be obtained by the function given an expected change of average waiting time $u(t)$. Then, $\varphi(t)$

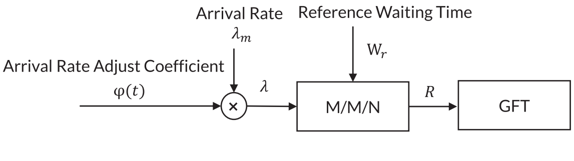

FIGURE 2 Queuing model with adjustable-arrival-rate-based feedforward controller

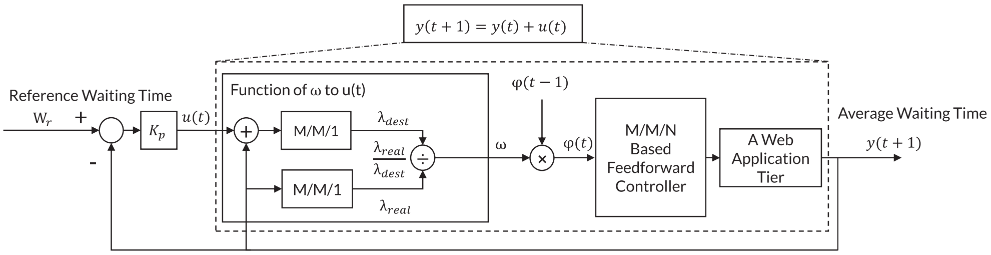

FIGURE 3 Two-threshold and queuing-model-based feedback controller

can be updated by ω based on which resources are updated by the feedforward controller. The real average waiting time of the Web application will change by $u(t)$ if the established function of adjustment ratio ω to changed waiting time is accurate. Therefore, the system can be abstracted into a simple linear model

$$y(t+1) = y(t) + u(t), \tag{4}$$

where $u(t)$ is the expected change of average waiting time as the control input. Then, feedback controllers can be built based on this linear model. In the following sections, the function of ω to changed waiting time and a Proportional controller are proposed.

5.3.2 | Function of adjustment ratio ω to expected change of waiting time $u(t)$

Given a real-time average waiting time $y(t)$ and the current resource capacity, a real arrival rate λ_{real} can be obtained based on queuing models. Meanwhile, given a targeted waiting time $W_d = y(t) + u(t)$ ($u(t)$ is the expected change of waiting time) and the current resource capacity, a targeted arrival rate λ_{dest} can be generated using queuing models too. The ratio $\omega = \lambda_{\text{real}}/\lambda_{\text{dest}}$ describes how much should the real arrival rate be adjusted to change the waiting time from $y(t)$ to W_d under the current resource capacity. In this article, the ratio ω is used to approximately describe how much should the measured arrival rate be adjusted to change the rented amount of resource to catch a targeted waiting time W_d.

However, given $y(t)$ and R, it is complex to calculate the λ_{real} using the M/M/N queuing model because it is much complex to deduce the inverse function of the original function from λ to W_q described in Equations (1), (2), and (3) directly. On the contrary, the similar inverse function of $M/M/1$ can be deduced easily as follows. The function of expected waiting time W_q to λ of $M/M/1$ is

$$W_q = f(\lambda) = \frac{\lambda}{\mu_t(\mu_t - \lambda)}, \tag{5}$$

where μ_t is the total processing rate of all VMs. The inverse function of f is

$$\lambda = f^{-1}(W_q) = \frac{\mu_t^2 W_q}{1 + \mu_t W_q}. \tag{6}$$

Therefore, for simplification, the ratio ω is computed using $M/M/1$ rather than $M/M/N$ as follows establishing the function of adjustment ratio ω to the expected change of waiting time $u(t)$ by substituting $W_d = y(t) + u(t)$.

$$\omega = \frac{\lambda_{\text{real}}}{\lambda_{\text{dest}}} = \frac{f^{-1}(y(t))}{f^{-1}(W_d)} = \frac{\mu_t^2 y(t)}{1 + \mu_t y(t)} \Big/ \frac{\mu_t^2 W_d}{1 + \mu_t W_d} = \frac{y(t)(1 + \mu_t(y(t) + u(t)))}{(y(t) + u(t))(1 + \mu_t y(t))}. \tag{7}$$

5.3.3 | Two-threshold-based proportional controller design

In most existing methods, the output error is the deviation between the real-time waiting time and the reference waiting time. However, because VMs can only be rented or released in discrete numbers, the total capacity of a virtual Cloud data center can only change in coarse-grained scales and output errors may always exist. The traditional computing method of the output error usually leads to system fluctuations. Therefore, a two-threshold-based output error computing method is applied.[31] The output error $e(t)$ is defined to be the difference between y_t and the lower or upper threshold of the interval from $W_r \times$ lower_thr to $W_r \times$ upper_thr. When y_t is inside the interval, $e(t)$ is defined to be zero to avoid a further control which may lead to a bigger output error.

$$e(t) = \begin{cases} W_r \times \text{lower\_thr} - y(t) & y(t) < W_r \times \text{lower\_thr} \\ W_r \times \text{upper\_thr} - y(t) & y(t) > W_r \times \text{upper\_thr} \\ 0 & \text{Otherwise.} \end{cases}$$

The proposed abstract linear model assumes that the control input $u(t)$ will have direct addition impact on the output which simplifies the controller design. Based on the two-threshold-based output error computing method, a proportional controller is applied, that is, $u(t) = K_p \times e(t)$ where K_p is the control gain. The transfer function of the linear-model is

$$\frac{Y}{U} = \frac{1}{z-1}. \tag{9}$$

The transfer function of the proportional controller is

$$\frac{U}{E} = K_p. \tag{10}$$

The transfer function of the whole system with feedback control is

$$\frac{Y}{W} = \frac{K_p \frac{1}{z-1}}{1 + K_p \frac{1}{z-1}} = \frac{K_p}{z - 1 + K_p}. \tag{11}$$

The pole of the whole feedback control system is $1 - K_p$. According to the settle time and overshoot requirements, K_p will be selected based on the pole placement method (PPM)[36] and experiments in the parameter tuning Section.

5.3.4 | Description of the two-threshold and queuing-model-based feedback controller

Algorithm 3 is the formal description of the proposed two-threshold and queuing-model-based feedback control method (TFBC). At first, $e(t)$ is obtained by Equation (8). Then, $u(t)$ is the expected change of waiting time which is the product of $e(t)$ and control gain K_p. The targeted average waiting time W_d after adjustment is the sum of $y(t)$ and $u(t)$. The adjustment ratio ω required to obtain the targeted average waiting time W_d is computed using Equation (7). ω is limited within the interval $[\omega^{\text{upper}}, \omega^{\text{lower}}]$ to avoid too large-scale adjustments incurring fluctuations. At last, $\varphi(t)$ is updated by multiplying the ratio ω.

Algorithm 3. Two-threshold and queuing-model-based feedback control (TFBC)

Input: current arrival rate adjust coefficient $\varphi(t-1)$, reference waiting time W_r, average waiting time $y(t)$, *lower_thr*, *upper_thr*, control gain K_p, last required VM capacity R_c, capacity of the lowest configuration VM C_l

```
$e(t) \leftarrow$ Equation (10), $u(t) \leftarrow K_p \times e(t)$ and $\mu_t \leftarrow \frac{R_c \times \mu}{C_l}$
$\omega \leftarrow$ Calculate adjustment ratio by Equation (5)
$\omega \leftarrow \max(\min(\omega, \omega^{upper}), \omega^{lower})$
$\varphi(t) \leftarrow \varphi(t-1) \times \omega$
return $\varphi(t)$
```

5.4 | Unequal-interval collaborating strategy

For control problems, frequent adjustments on the input are required to cope with output errors incurred by dynamic workloads. However, input adjustments usually have delays to take effect, for example, it takes about 1 to 2 minutes for newly requested VMs to become available and at most 1 hour for VMs to be really released. Too frequent adjustments on the input are likely to lead to overcontrol, for example, renting or releasing excess VMs before adjustments take effect. Therefore, it is crucial to determine the time interval between two control actions (called control interval). An unequal-interval feedforward-and-feedback collaborating method is proposed based on the loosely coupled structure of two controllers. This collaborating method assigns different control intervals to feedback and feedforward controllers considering the renting and releasing delays of Cloud VMs, respectively. The time intervals of invoking feedback controllers should be longer than the VM preparation time to make the newly rented VM take effect. Meanwhile, before the feedback controller can be invoked, it should be checked whether the last VM releasing action has taken effect, that is, VMs determined to be released are really released. The VM releasing status checking method (RC) is as follows. For a VM group, if the releasing of any VM at its end of rented hour makes the group capacity lower than Q, it means no VMs can be released anymore for fulfilling the capacity Q. If VMs of all groups cannot be released, it means that the VMs required to be released determined by previous control actions are all really released. On the contrary, the feedforward controller can be invoked more frequently than the feedback controller to response to workload changes quickly since two controllers are loosely coupled and the feedforward controller can work temporarily well separately.

5.5 | Description of unequal-interval loosely coupled control method

Based on the proposed loosely coupled feedforward and feedback controllers and unequal-interval collaborating strategy, the proposed unequal-interval-based loosely coupled control method (UCM) is formally described in Algorithm 4. The main goal of feedforward controller is to adjust the required VM capacity to cope with the changes of workloads. Therefore, when B_s = TRUE (the fast feedforward is enabled), the feedforward controller is invoked at every workload sampling interval α (eg, 1 minute) to response to workload changes quickly. The minute-based fast feedforward ($\alpha = 60$ seconds) is called MF. Every β (eg, 300) seconds, it is checked whether the feedback controller can be invoked. β should be larger than the VM preparation time and is usually much longer than α. When the system is under-provisioning ($y(t) > W_r$), the feedback controller TFBC is called directly. Otherwise, the VM releasing status B_r is checked by RC when B_c = TRUE. Then, if B_c = FALSE (not to check VM releasing status) or B_r = TRUE (VMs are already released), the feedback controller is invoked. Otherwise, the feedback controller is not called. Finally, because the proposed feedback and feedforward controller are loosely coupled, the feedforward controller QFC and group-based resource renting method GFT are called no matter whether the feedback controller has been invoked.

Algorithm 4. Unequal-interval-based loosely coupled control method (UCM)

```
Input: Measured arrival rate $\lambda_m$, reference waiting time $W_r$, current average waiting time $y(t)$, is release status checking enabled $B_c$, is fast feedforward enabled $B_s$, lasting time of current plan $T_p$, $N_e$, $R_c$, $C_l$, $T_u$, $\varphi(t-1)$, $\mu$
while True do
    if $T_c - Tf \geq \alpha$ and $B_s = TRUE$ then
        $Tf \leftarrow T_c$
        $R \leftarrow$Call QFC($\lambda_m, \mu, C_l, \varphi(t), W_r$)
        Call GFT($R, R_c, T_u, T_p, N_e$)
    else if $T_c - Tb \geq \beta$ then
        $Tb \leftarrow T_c$, $lower\_thr \leftarrow 0.75$ and , $upper\_thr \leftarrow 1.25$
        if $y(t) > W_r$ then
            $\varphi(t) \leftarrow$ TFBC($\varphi(t-1), W_r, y(t), lower\_thr, upper\_thr, K_p, R_c, C_l$)
        else
            if $B_c = TRUE$ then
                $B_r \leftarrow$ Check VM releasing Status using RC
            end if
            if $B_c = FALSE$ or $B_r = TRUE$ then
                $\varphi(t) \leftarrow$ TFBC($\varphi(t-1), W_r, y(t), lower\_thr, upper\_thr, K_p, R_c, C_l$)
            end if
        end if
        $R \leftarrow$Call QFC($\lambda_m, \mu, C_l, \varphi(t), W_r$)
        Call GFT($R, R_c, T_u, T_p, N_e$)
    end if
end while
```

6 | PERFORMANCE EVALUATION

UCM and existing algorithms are first evaluated in a simulation environment, created using CloudSim[37] which supports the modeling of Spot VMs. Meanwhile, algorithms are also compared in a real cluster consisting of six VMs which are organized by Kubernetes[35] to implement elastic resource provisioning by the aid of container and request automatic forwarding techniques. One VM with four 2.6 GHz Intel i7-9750H virtual processors and 2 GB of Memory is the master node. Four VMs with three Intel 2.4 GHz i5-6300 or 2.6 GHz i7-9750H virtual processors and 1 GB Memory are worker nodes. A Web application for calculating Fibonacci numbers is deployed in worker nodes in the form of containers. Since creating more containers in one VM will not increase the total capacity, each VM is limited to accommodate only one application container.

Worker VMs are dynamically allocated to (deallocated from) the Web application through creating (or deleting) containers of the application on VMs.[35] The ingress-nginx-controller[38] in the master node is used to forward requests to containers belonging to the application currently. The Kubernetes-based resource management method can be used to manage VMs rented from public Clouds too.[39] JMeter[40] serves as a concurrent request generator.

Because the Wikipedia user access traces[41,42] have common fluctuation characteristics, user access traces during September and October in 2007 are used as workloads. Since the original trace data contains 1500 to 3500 requests every second, about 5% of the original requests are sampled randomly to speed up the simulation. Because the Wikipedia trace has a significant weekly seasonal pattern, 2 weeks of nonoverlapping traces are used, called Workload 1 and 2 for CloudSim evaluation. Eight types of Amazon EC2 Spot and On-demand VMs as shown in Table 2 with different configurations and prices are simulated in CloudSim. Real prices of Spot VMs are obtained by the interface of Amazon EC2 and used to evaluate rental costs. For experiments on the real Kubernetes Cluster, access traces of each hour are compressed into 10 minutes to accelerate the evaluation by sampling. And the experiment on the Kubernetes cluster lasted 12 hours for each algorithm. JMeter generates requests according to the number of requests of each minute in the access traces. For simplification, prices of two types of VMs in the Kubernetes Cluster are set to be 1 and 2 per 10 minutes, respectively.

The UCM is first compared with the GFT method[12,13] which is one of the few works considering the heterogeneous Spot VMs. The total required resource capacity R of GFT is the multiplication of the last λ_m and the average request length (million instructions). GFT mainly focuses on fault-tolerant strategies without considering the prediction of required VM capacity to control the average waiting time in a given interval. Therefore, UCM is then compared with QT[24] which use a M/M/N-model-based feedforward controller to predict required VM capacities. Because QT is not tailored for Spot VMs, it is extended by adding GFT as a fault-tolerant strategy and the extended QT is called GFT-FF. Next, UCM is compared with another unequal-M/M/N-queuing-model-based feedforward method UQueuing proposed by Wang et al.[27] UCM is also compared with EcoWare[9] which is one of the classical simple nonlinear-model-based feedback control methods. At last, UCM is compared with HC-FFB proposed by Sha et al[10] which obtains well performance in the control of traditional server processes. HC-FFB applies a G/G/N model as feedforward control and a queuing-model-derived-linear-model-based PI controller as feedback control. Since the comparison of feedforward-control-based on different queuing models is not the main concern of this article, G/G/N model of HC-FFB is replaced by M/M/N used in QT[24] for fair comparison. Because HC-FFB is designed to control the response time rather than waiting time, the first-order linear model[10] established based on M/M/1 that describe the effect of the amount of increased (decreased) processing rate to the reduction (increase) in output response time cannot be used directly. In this article, a similar first-order linear model is built to describe the effect of changes in processing rate to changes in average waiting time as follows. The average waiting time of M/M/1 is

$$W_q = \frac{\lambda}{\mu_t^2 - \mu_t \lambda}, \tag{12}$$

where μ_t and λ are processing and arrival rate, respectively. Because μ_t is similar with λ in stable state, the first derivative of the waiting time vs μ_t is

$$\frac{dW_q}{d\mu_t} = -\lambda\left(\frac{1}{\mu_t^2 - \mu_t\lambda}\right)^2 (2\mu_t - \lambda) \approx -\lambda\left(\frac{1}{\mu_t^2 - \mu_t\lambda}\right)^2 (2\lambda - \lambda) = -\left(\frac{\lambda}{\mu_t^2 - \mu_t\lambda}\right)^2 = -(W_q)^2. \tag{13}$$

Then, the linear model $dW_q = -(W_q)^2 \times d\mu_t$ is used to build a PI feedback controller. The HC-FFB extended with GFT and the new linear-model-based PI feedback controller is called HC-GFT-FFB. To evaluate the performance of the proposed VM RC and minute-based fast feedforward (MF), the variants of UCM, HC-GFT-FFB, GFT, UQueuing, and EcoWare shown in Table 3 are compared. The parameters of existing GFT are set with $f = 1$, $d_{\text{margin}} = 0.2$, $N_{al} = 3$, and $h = 4$ consistent with existing work.[13]

TABLE 2 Simulated virtual machine types of Amazon EC2

VM type	MIPS	On-demand Price
c4.L	4000	0.1
c4.l-linux-unix	4000	0.1
c4.xl-linux-unix	8000	0.21
c4.2xl-linux-unix	15 500	0.419
c4.4xl-linux-unix	32 000	0.838
m4.xl-linux-unix	6500	0.2
m4.2xl-linux-unix	13 000	0.4
m4.4xl-linux-unix	26 750	0.8

Abbreviation: VM, virtual machine.

TABLE 3 Details of compared algorithms

Name	Feedforward control	Feedback control	Release status check	Fast feedforward
GFT[12,13]	×	×	×	×
GFT-FF (GFT and QT[24])	M/M/N	×	×	×
UQueuing-RC[27]	unequal M/M/N	×	*RC*	×
EcoWare-RC[9]	×	Inverse proportional model	*RC*	×
HC-GFT-FFB-NRC[10]	M/M/N	M/M/1 derived linear model (Equation (13))	×	×
HC-GFT-FFB-RC[10]	M/M/N	M/M/1 derived linear model (Equation (13))	*RC*	×
UCM-NRC	QFC	TFBC	×	×
UCM-RC	QFC	TFBC	*RC*	×
UCM-RC-M	QFC	TFBC	*RC*	*MF*

Abbreviation: GFT, group-based fault-tolerant.

Because of the complexity of Web applications, the real average waiting time always fluctuates around the given reference waiting time W_r. Therefore, appropriate reference waiting time W_r should be selected to avoid violating SLA which is usually smaller than W_{SLA}.[9] In CloudSim, it is assumed that $W_{SLA} = 0.1$ second and $\kappa = 95$ are defined in the SLA, that is, 95% of waiting times should be smaller than 0.1 second. A larger W_r increases the ability of SLA violation while a smaller W_r increases the rental cost of VMs. W_r with different values from {0.005,0.01,0.02,0.04,0.08} are tested by experiments and $W_r = 0.02$ second with appropriate cost and waiting times is selected. Lengths of requests are randomly generated following an exponential distribution with a mean of 2000 MI, which means it takes 0.5 second to process on the smallest VM c4.L with the processing rate of 4000 MIPS. The VM preparation time including requesting time to public Clouds is set to be 150 seconds.[9] The arrival rate sampling interval and the fast feedforward invoking interval is $\alpha = 60$ seconds and the feedback control interval is $\beta = 300$ seconds. For experiments on the real Kubernetes Cluster, $W_{SLA} = 0.045$ s and $W_r = 0.03$ s are selected based on the characteristics of the Fibonacci Web application and experiments. The processing rates of requests on the two types of Kubernetes Cluster's VMs are about 20 /s and 40 /s, respectively obtained by experiments. And the average processing time of one request on the fastest VM is about 0.02 s. For fair comparison, the average waiting time is obtained by subtracting 0.02 s from the average response time of ingress-nginx-controller[38] for all algorithms. Because containers can be allocated to the application within half minute, the control interval is shortened to $\beta = 180$ s and fast feedforward is disabled by setting $B_s = \text{FALSE}$.

6.1 | Parameter tuning

The control gains of UCM's P controller and HC-GFT-FFB's PI controller are determined by the PPM[36] and experimental evaluations together. According to PPM, poles of control systems have a great impact on stability, settle times and maximum overshoots. Poles should be within the unit circle to make the system stable and positive for first-order systems to avoid overshoots. Because of the coarse-grained capacity adjustment scale and complexity of Web systems, the real performance of candidate poles {0.9,0.7,0.5,0.3,0.1,0} fulfilling the above PPM's criteria is evaluated by experiments. For UCM, the control gain is generated by $K_p = 1 - pole$, that is, $K_p \in \{0.1,0.3,0.5,0.7,0.9,1\}$. Similarly, the control gains of HC-GFT-FFB's PI controllers can be obtained.

When the percentages of average waiting times (PAWT) larger than 0.1 second (denoted by PAWT[0.1,∞] for brief) is larger than 95%, it means that the SLA is violated for the CloudSim simulation. A larger PAWT[0.05,0.1] means a higher possibility of SLA violation when there are sudden burst of requests. A larger PAWT[0,0.005] means that more resources are rented incurring higher rental costs. Therefore, this article aims to control the waiting time in a desired interval such as [0.005,0.05]s to save the renal cost and guarantee the SLA simultaneously. Figure 4A shows the PAWT of our approach UCM-RC-M with different K_p on CloudSim. In total, UCM-RC-M with $K_p = 1$ obtains the minimum PAWT[0.05,∞] = 10.4% (the sum of 6.3% and 4.1%) and the costs of all UCM-RC-M are similar. The PAWTs of the existing algorithm HC-GFT-FFB with different *pole*s are shown in Figure 4B which shows that HC-GFT-FFB with *pole* = 0.9 gets the minimum PAWT[0.05,∞] which is 17% (the sum of 8.3% and 8.7%). Therefore, $K_p = 1$ and *pole* = 0.9 are selected for UCM and HC-GFT-FFB, respectively. Although, the pole α of EcoWare-RC[9] is equal to 0.95 originally, in this article, α is set to be 0.9 consistent with HC-GFT-FFB. Based on experimental comparison, for the CloudSim, lower_thr = 0.75, upper_thr = 1.25, $\omega^{\text{lower}} = 0.95$ and $\omega^{\text{upper}} = 1.05$ are chose. For the Kubernetes Cluster, lower_thr = 0.5, upper_thr = 1, $\omega^{\text{lower}} = 0.95$ and $\omega^{\text{upper}} = 2$ are selected.

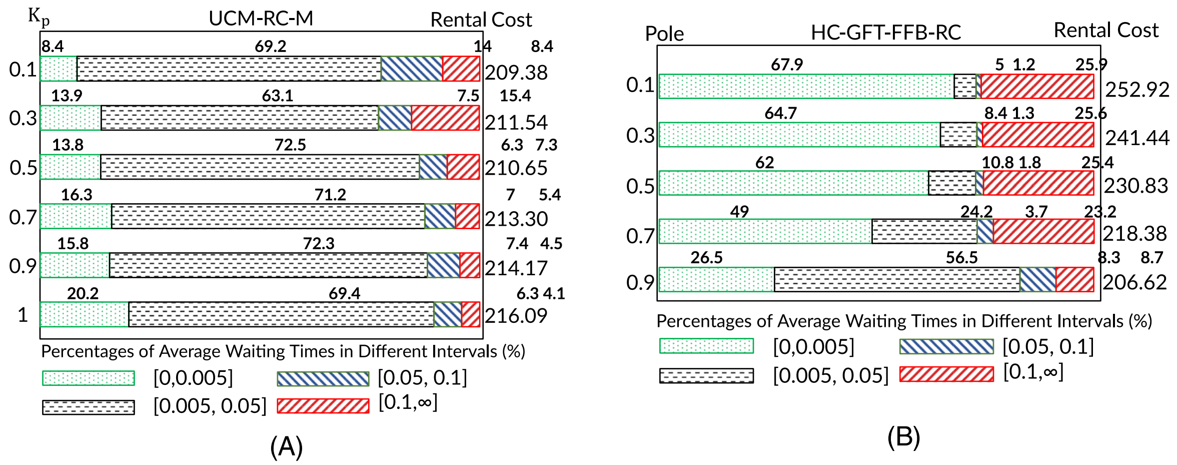

FIGURE 4 The distribution of average waiting times and rental costs of UCM-RC-M and HC-GFT-FFB

6.2 | Results on CloudSim

Figure 5A-C show PAWTs and rental costs of compared algorithms on CloudSim using Workload 1 and 2. Experimental results show that existing GFT and GFT-FF usually rent too much resource with highest VM rental costs and more than 99.9% average waiting times are smaller than 0.005 s. The reason is that the total capacities of GFT and GFT-FF are determined only according to historical workloads or queuing model-based

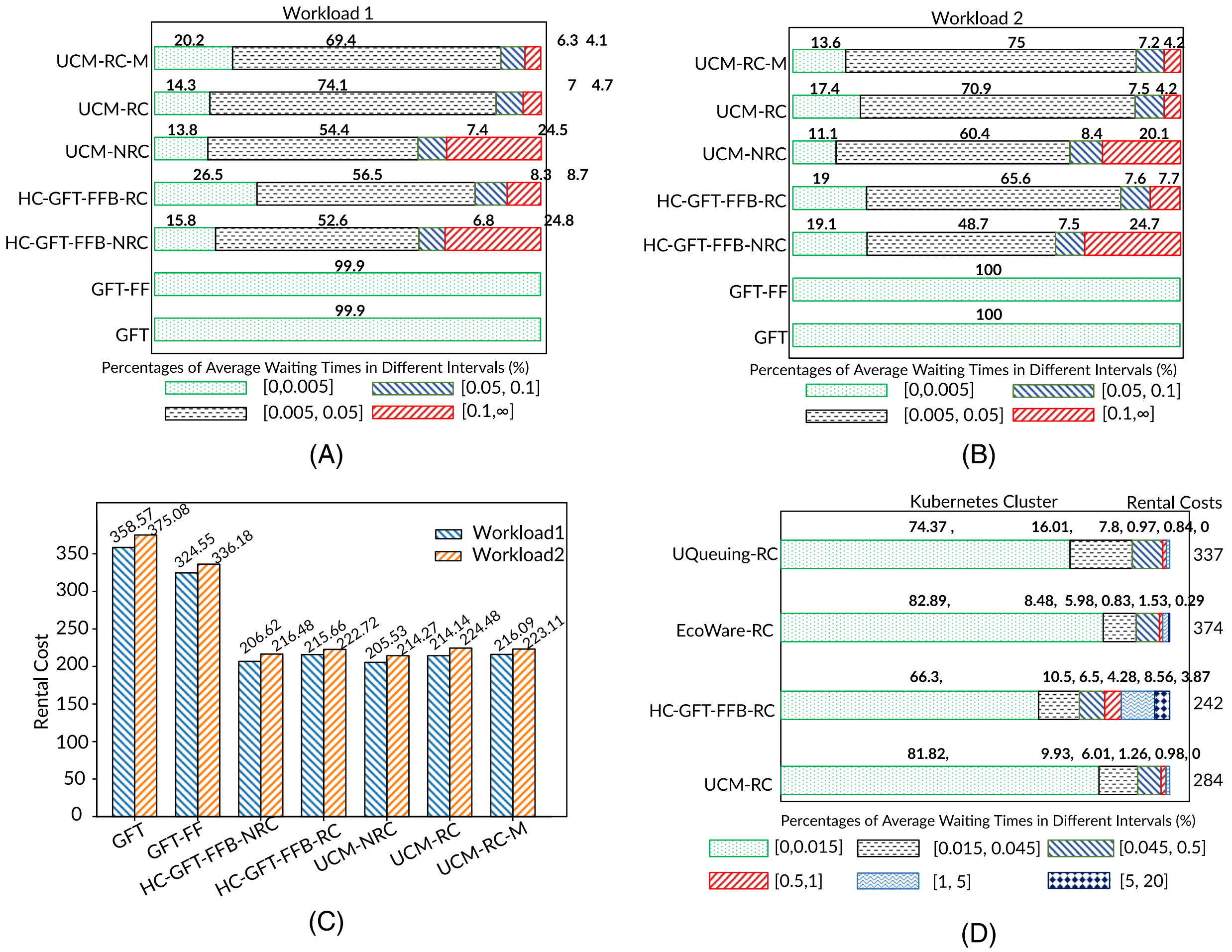

FIGURE 5 The distribution of average waiting times and rental costs of compared algorithms on CloudSim and the Kubernetes Cluster

feedforward controllers without reacting to output errors. On the contrary, all the other methods with feedback controllers can decrease rental costs greatly by trying to rent appropriate amounts of resources to keep more average waiting times in the desired interval [0.005,0.05]s. The PAWT[0.05,∞] of UCM-RC and HC-GFT-FFB-RC are much smaller than those of UCM-NRC and HC-GFT-FFB-NRC. For example, the PAWT[0.05,∞] of HC-GFT-FFB-RC is 17% (the sum of 8.3% and 8.7%) which is much smaller than 31.6% (the sum of 6.8% and 24.8%) of the traditional HC-GFT-FFB-NRC on Workload 1. This illustrates that the traditional HC-GFT-FFB-NRC has violated the SLA (95% no larger than 0.1 second) greatly and is not suitable for controlling scenarios with hourly priced VMs although HC-GFT-FFB-NRC has both feedback and feedforward controllers. And the VM RC improves the performance of existing HC-GFT-FFB-NRC greatly. Similarly, PAWT[0.05,∞] of our approach UCM-RC is decreased greatly compared with UCM-NRC by applying RC. For instance, PAWT[0.05,∞] is decreased from 28.5% of UCM-NRC (the sum of 8.4% and 20.1%) to 11.7% of UCM-RC (the sum of 7.5% and 4.2%) on Workload 2.

Experimental results also show that UCM-RC has lower PAWT[0.05,∞] than that of HC-GFT-FFB-RC. For example, on Workload 1, the PAWT[0.05,∞] of UCM-RC is 11.7% (the sum of 7% and 4.7%) which is smaller than 17% (the sum of 8.3% and 8.7%) of HC-GFT-FFB-RC, that is, UCM-RC is more powerful at avoiding SLA violation. Furthermore, UCM-RC-M has a much smaller PAWT[0.05,∞] than UCM-RC. For example, PAWT[0.05,∞] of UCM-RC-M is 10.4% (the sum of 6.3% and 4.1%) which is smaller than 11.7% (the sum of 7% and 4.7%) of UCM-RC on Workload 1. It proves that the fast feedforward (MF) is helpful to decrease PAWT[0.05,∞]. Meanwhile, HC-GFT-FFB-RC has a much more larger PAWT[0,0.005] compared with UCM-RC and UCM-RC-M, that is, HC-GFT-FFB-RC wastes more resources sometimes. On the contrary, UCM-RC and UCM-RC-M make more average waiting times cluster in the desired interval [0.005,0.05]s with a percentage of 74.1% and 69.4%, respectively, which are larger than the 56.5% of HC-GFT-FFB-RC on Workload 1. Meanwhile, UCM-RC, UCM-RC-M, and HC-GFT-FFB-RC have similar rental costs finally. These illustrate that our approach UCM-RC and UCM-RC-M control the system more stably than HC-GFT-FFB-RC with similar rental costs. For example, Figure 6A shows the workloads and VM capacities of 700 control steps which denote the total VM capacity of UCM-RC-M changes more stably compared with HC-GFT-FFB-RC as the workload changes. Therefore, as shown in Figure 6B,C, there are many successive average waiting times of HC-GFT-FFB-RC smaller than 0.005 second while the average waiting times of UCM-RC-M are nearly uniformly distributed around the reference point 0.02 second. The reason is that the proposed M/M/1-based TFBC can update the arrival rate adjustment coefficient appropriately to cope with environment and workload changes more stably than the existing M/M/1-derived-linear-model-based feedback controller. Figure 6 also denotes that the total VM capacity of UCM-RC-M updates not only stably but frequently. This is because the loosely coupled feedforward and feedback architecture allows the feedforward controller be called separately and more frequently to react to workload changes more quickly.

6.3 Results on real Kubernetes Cluster

Figure 5D shows PAWTs and rental costs of compared algorithms on a real Kubernetes Cluster. GFT and GFT-FF are not evaluated on the real Cluster because of poor performance on the CloudSim. UCM-RC's PAWT[0,0.045] is 91.75% (sum of 81.82% and 9.93%) larger than those of all

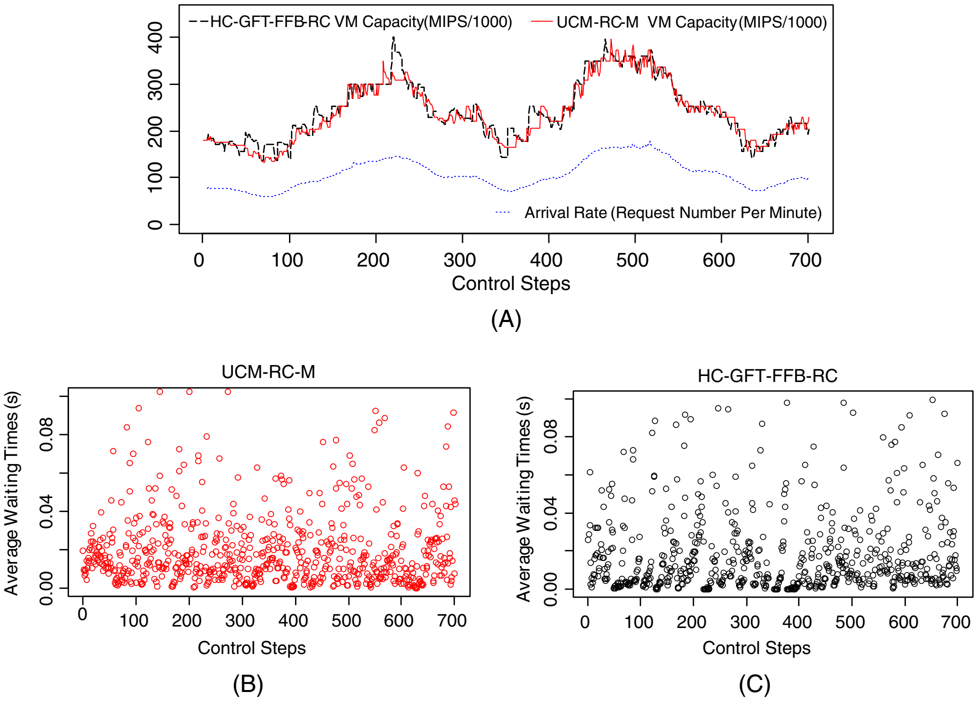

FIGURE 6 A sample of the workloads and VM capacities within 700 control steps. VM, virtual machine

other algorithms, that is, UCM-RC gets the minimum percentage of average waiting times larger than $W_{SLA} = 0.045$ s. Meanwhile, the rental cost of UCM-RC is the lowest one except that of HC-GFT-FFB-RC. These prove that UCM-RC can adjust resources appropriately by the aid of TFBC- based coefficient adjustment on the real Cluster. The comparison result between UCM-RC and HC-GFT-FFB-RC on the real Cluster is consistent with that on CloudSim. Although EcoWare-RC's PAWT[0,0.045] is 91.37% close to that of UCM-RC, EcoWare-RC's rental cost is 374 which is 31% higher than UCM-RC's cost. The reason is that EcoWare-RC reacts to waiting times smaller than W_r too quickly leading to more larger SLA violations and higher rental costs because of the characteristic of the inverse-proportional-performance model. Uqueuing-RC's PAWT[0,0.045] is 90.38% which denotes that more average waiting times are larger than W_{SLA} compared with UCM-RC and EcoWare-RC, and the rental cost of Uqueuing-RC is 337 which is much higher than 284 of UCM-RC. The reason is that there is unavoidable deviation between the unequal M/M/N model of Uqueuing-RC and the real system because of the complexity of the system and inaccurate estimations of VM's processing rates. Although Uqueuing-RC reacts quickly by estimating waiting times considering the real-time queuing length when the system becomes unstable, Uqueuing-RC cannot amend the deviation between the estimated and real waiting times if the deviation is the reason of model inaccuracy when the system is stable.

7 | CONCLUSIONS AND FUTURE WORK

In order to decrease the VM rental cost while guaranteeing the SLA and robustness, a hybrid control method UCM is proposed which takes advantages of queuing-model-based loosely coupled controllers, unequal-interval-based collaborating method and an existing GFT strategy. Experimental results show that feedback controls make more average waiting times stay in a reasonable interval to save rental cost. And our approach decreases the percentage of waiting times larger than the SLA from about 24.7%-24.8% and 23.2% to 4.1%-4.2% and 9.6% compared with HC-GFT-FFB-NRC and HC-GFT-FFB-RC, respectively, on CloudSim and the Kubernetes Cluster with similar rental costs. Our approach obtains lower SLA violation and 18.6% to 31% lower rental costs compared with EcoWare-RC and Uqueuing-RC. These experimental results prove that the proposed VM releasing status checking is helpful to avoid overcontrol for Web applications with interval-priced Cloud VMs. The proposed function of the adjustment ratio to expected change of the waiting time describes the Web system more accurately than the existing queuing model derived linear model. The loosely coupled structure of feedforward and feedback controllers allowing the feedforward controller to be called separately and frequently is helpful to react to workload changes quickly. The promising future work is to apply the interval-pricing-model aware feedback methods to control QoS of other complex Cloud applications with MapReduce or Directed-Acyclic-Graph-based tasks.

ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China (Grant No.61972202, 61602243, and 61702267), the Fundamental Research Funds for the Central Universities (No.30919011235 and NO.30920120180101) and the Postgraduate Research and Practice Innovation Program of Jiangsu Province (No.KYCX18_0434). This work is mainly accomplished during the visiting of the first author in the CLOUDS Laboratory of the University of Melbourne.

ORCID

Zhicheng Cai https://orcid.org/0000-0002-8702-6216
Yifei Lu https://orcid.org/0000-0002-1352-5418
Rajkumar Buyya https://orcid.org/0000-0001-9754-6496

REFERENCES

1. Gill SS, Chana I, Singh M, Buyya R. RADAR: self-configuring and self-healing in resource management for enhancing quality of cloud services. *Concurrency Comput Pract Exper*. 2019;31(1):1-29.
2. Li Z, Zhang Y, Liu Y. Towards a full-stack devops environment (platform-as-a-service) for cloud-hosted applications. *Tsinghua Sci Technol*. 2017;22(01):1-9.
3. Singh VK, Dutta K. Dynamic price prediction for Amazon Spot instances. Paper presented at: Hawaii International Conference on System Sciences; 2015;1513-1520; Kauai, HI, USA: IEEE.
4. AWS Auto Scaling. Amazon EC2 Web site; 2020. https://aws.amazon.com/autoscaling/.
5. Lu C, Abdelzaber T, Stankovic JA, Son SH. A feedback control approach for guaranteeing relative delays in Web servers. Paper presented at: Proceedings Seventh IEEE Real-Time Technology and Applications Symposium; 2001;51-62; Taipei: IEEE.
6. Lu C, Lu Y, Abdelzaher TF, Stankovic JA, Son SH. Feedback control architecture and design methodology for service delay guarantees in web servers. *IEEE Trans Parallel Distribut Syst*. 2006;17(9):1014-1027.
7. Patikirikorala T, Colman A, Han J, Wang L. A multi-model framework to implement self-managing control systems for QoS management. Paper presented at: Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems; 2011;218-227; Honolulu, HI: ACM.
8. Pan W, Mu D, Wu H, Yao L. Feedback control-based QoS guarantees in web application servers. Paper presented at: 10th IEEE International Conference on High Performance Computing and Communications; 2008;328-334; Dalian, China: IEEE.
9. Baresi L, Guinea S, Leva A, Quattrocchi G. A discrete-time feedback controller for containerized cloud applications. Paper presented at: 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering; 2016;217-228; New York, USA: ACM.

10. Sha L, Liu X, Lu Y, Abdelzaher T. Queueing model based network server performance control. Paper presented at: 23rd IEEE Real-Time Systems Symposium; 2002;81-90; Austin, TX: IEEE.
11. Lu Y, Abdelzaher T, Lu C, Sha L, Liu X. Feedback control with queueing-theoretic prediction for relative delay guarantees in web servers. Paper presented at: The 9th IEEE Real-Time and Embedded Technology and Applications Symposium; 2003;208-217; Toronto, ON: IEEE.
12. Qu C, Calheiros RN, Buyya R. A reliable and cost-efficient auto-scaling system for web applications using heterogeneous spot instances. *J Netw Comput Appl*. 2016;65:167-180.
13. Liu D, Cai Z, Lu Y. Spot price prediction based dynamic resource scheduling for web applications. Paper presented at: Seventh International Conference on Advanced Cloud and Big Data (CBD); 2019;78-83; Suzhou, China: IEEE.
14. Al-Dhuraibi Y, Paraiso F, Djarallah N, Merle P. Elasticity in cloud computing: state of the art and research challenges. *IEEE Trans Serv Comput*. 2018;11(2):430-447.
15. Urgaonkar B, Shenoy P, Chandra A, Goyal P, Wood T. Agile dynamic provisioning of multi-tier internet applications. *ACM Trans Auton Adapt Syst*. 2008;3(1):1-39.
16. Lakew EB, Klein C, Hernandez-Rodriguez F, Elmroth E. Towards faster response time models for vertical elasticity. Paper presented at: IEEE/ACM 7th International Conference on Utility and Cloud Computing; 2014;560-565; London, UK: IEEE.
17. Ali-Eldin A, Tordsson J, Elmroth E. An adaptive hybrid elasticity controller for cloud infrastructures. Paper presented at: IEEE Network Operations and Management Symposium; 2012;204-212; Maui, HI: IEEE.
18. Xue C, Lin C, Hu J. Scalability analysis of request scheduling in cloud computing. *Tsinghua Sci Technol*. 2019;24(3):249-261.
19. Huang G, Wang S, Zhang M, et al. Auto scaling virtual machines for web applications with queueing theory. Paper presented at: 3rd International Conference on Systems and Informatics (ICSAI); 2016;433-438; Shanghai, China: IEEE.
20. Tesauro G, Jong NK, Das R, Bennani MN. A hybrid reinforcement learning approach to autonomic resource allocation. Paper presented at: IEEE International Conference on Autonomic Computing; 2006;65-73; Dublin, Ireland: IEEE.
21. Li H, Venugopal S. Using reinforcement learning for controlling an elastic web application hosting platform. Paper presented at: The 8th ACM International Conference on Autonomic Computing; 2011;205-208; New York, USA: ACM.
22. Barrett E, Howley E, Duggan J. Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. *Concurr Comput Pract Exper*. 2013;25(12):1656-1674.
23. Dutreilh X, Moreau A, Malenfant J, Rivierre N, Truck I. From data center resource allocation to control theory and back. Paper presented at: IEEE 3rd International Conference on Cloud Computing; 2010;410-417; Miami, FL: IEEE.
24. Jiang J, Lu J, Zhang G, Long G. Optimal cloud resource auto-scaling for web applications. Paper presented at: 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing; 2013;58-65; Delft, Netherlands: IEEE.
25. Jafarnejad Ghomi E, Rahmani AM, Qader NN. Applying queue theory for modeling of cloud computing: A systematic review. *Concurr Comput Pract Exper*. 2019;31(17):e5186.
26. Qu C, Calheiros RN, Buyya R. Auto-scaling web applications in clouds: a taxonomy and survey. *ACM Comput Surv (CSUR)*. 2018;51(4):1-33.
27. Wang X, Du Z, Chen Y, et al. An autonomic provisioning framework for outsourcing data center based on virtual appliances. *Clust Comput*. 2008;11(3):229-245.
28. Padala P, Hou KY, Shin KG, et al. Automated control of multiple virtualized resources. Paper presented at: Proceedings of the 4th ACM European Conference on Computer systems. 2009;13-26; Nuremberg, Germany: ACM.
29. Karlsson M, Karamanolis C, Zhu X. Triage: performance differentiation for storage systems using adaptive control. *ACM Trans Storage (TOS)*. 2005;1(4):457-480.
30. Xu CZ, Liu B, Wei J. Model predictive feedback control for QoS assurance in webservers. *Computer*. 2008;41(3):66-72.
31. Lim HC, Babu S, Chase JS, Parekh SS. Automated control in cloud computing: challenges and opportunities. Paper presented at: Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds. 2009;13-18; Barcelona, Spain: ACM.
32. Al-Shishtawy A, Vlassov V. Elastman: elasticity manager for elastic key-value stores in the cloud. Paper presented at: Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference. 2013;1-10; Miami, FL: ACM.
33. Oh J, Kang KD. A predictive-reactive method for improving the robustness of real-time data services. *IEEE Trans Knowl Data Eng*. 2013;25(5):974-986.
34. Shevtsov S, Berekmeri M, Weyns D, Maggio M. Control-theoretical software adaptation: a systematic literature review. *IEEE Trans Softw Eng*. 2018;44(8):784-810.
35. Kubernetes: Production-Grade Container Orchestration. Kubernetes Web site; 2020. https://kubernetes.io/.
36. Hellerstein JL, Diao Y, Parekh S, Tilbury DM. *Feedback Control of Computing Systems*. Hoboken, NJ: John Wiley & Sons, Inc.; 2004.
37. Buyya R, Ranjan R, Calheiros RN. Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities. Paper presented at: International Conference on High Performance Computing & Simulation; 2009; 1-11; Leipzig, Germany: IEEE.
38. NGINX Ingress Controller. NGINX Ingress Controller Web site; 2020. https://kubernetes.github.io/ingress-nginx/.
39. Containers on AWS. Amazon Containers Web site; 2020. https://aws.amazon.com/containers/.
40. Apache JMeter: Workload generator. Apache JMeter Web site; 2020. https://jmeter.apache.org/.
41. Urdaneta G, Pierre G, Steen VM. Wikipedia workload analysis for decentralized hosting. *Elsevier Comput Netw*. 2009;53(11):1830-1845. http://www.globule.org/publi/WWADH_comnet2009.html.
42. Wikipedia access traces. WikiBench Web site; 2020. http://www.wikibench.eu/?page_id=60.

How to cite this article: Cai Z, Liu D, Lu Y, Buyya R. Unequal-interval based loosely coupled control method for auto-scaling heterogeneous cloud resources for web applications. *Concurrency Computat Pract Exper*. 2020;e5926. https://doi.org/10.1002/cpe.5926