Inverse Queuing Model-Based Feedback Control for Elastic Container Provisioning of Web Systems in Kubernetes

Zhicheng Cai, *Member, IEEE* and Rajkumar Buyya, *Fellow, IEEE*

Abstract—Container orchestration platforms such as Kubernetes and Kubernetes-derived KubeEdge (called Kubernetes-based systems collectively) have been gradually used to conduct unified management of Cloud, Fog, and Edge resources. Container provisioning algorithms are crucial to guaranteeing quality of services (QoS) of such Kubernetes-based systems. However, most existing algorithms focus on placement and migration of fixed number of containers without considering elastic provisioning of containers. Meanwhile, widely used linear-performance-model-based feedback control or fixed-processing-rate-based queuing model on diverse platforms cannot describe the performance of containerized Web systems accurately. Furthermore, a fixed reference point used by existing methods is likely to generate inaccurate output errors incurring great fluctuations encountered with large arrival-rate changes. In this article, a feedback control method is designed based on a combination of varying-processing-rate queuing model and linear-model to provision containers elastically which improves the accuracy of output errors by learning reference models for different arrival rates automatically and mapping output errors from reference models to the queuing model. Our approach is compared with several state-of-art algorithms on a real Kubernetes cluster. Experimental results illustrate that our approach obtains the lowest percentage of service level agreement (SLA) violation (decreasing no less than 8.44 percent) and the second lowest cost.

Index Terms—Cloud, fog and edge computing, kubernetes, container auto-scaling, Qos control, queuing theory, feedback control

1 Introduction

One of the most effective approaches to share resources of diverse systems such as private data centers, Cloud Computing, Fog and Edge Computing [1] among multiple applications is using containers which are more lightweight and portable than virtual machines (VMs) [2]. Kubernetes [3] is one of the popular container orchestrating systems and is gradually used to manage Cloud, Fog and Edge resources by containers seamlessly leading to many derivative platforms such as KubeEdge [4]. Kubernetes's Pods (consisting of one or more containers) of different applications are deployed on VMs rented from public Clouds or physical machines (PMs) of Fog and Edge nodes. Meanwhile, Web applications and services (called Web systems collectively) are very common in Cloud, Fog and Edge Computing providing various functions to end users via different micro-services deployed in the form of containers. One of the most crucial problems is to design container auto-scaling algorithms to control response time of each micro-service in Kubernetes-based platforms.

Most existing container provisioning algorithms focus on placement and migration of fixed number of containers on PMs or VMs rather than auto-scaling of containers [5], [6], [7]. However, the number of containers allocated to each micro-service has a great impact on request response times. Kubernetes's build-in auto-scaling scheduler [8] and most threshold-based methods [9] only add or remove container replicas based on resource usage rates which are indirect metrics for controlling response times. Therefore, the main goal of this paper is to design container auto-scaling methods for Kubernetes which adjust the number of containers allocated to each micro-service automatically to decrease resource consumption while guaranteeing quality of services (QoS). The main challenges of designing such auto-scaling algorithms include non-linear performance model of multi-container systems and finding appropriate output error computing methods.

Non-linear performance characteristics of Web systems make resource auto-scaling complex. QoS control has been studied extensively for traditional Web systems involving elastic provisioning of application resources, PMs or VMs. However, most existing methods belong to pure queuing-theory-based feed-forward control [10], [11], [12], [13] or linear-model-based feedback control [14], [15], [16] which lack feedback ability or cannot describe complex non-linear multi-container systems accurately. Although linear-model-based feedback control has been used to amend the inaccuracy of queuing models taking advantage of queuing and control theory together, the reference-point-derived linear

- *Zhicheng Cai is with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China, and also with the University of Melbourne, Parkville, VIC 3010, Australia. E-mail: caizhicheng@njust.edu.cn.*
- *Rajkumar Buyya is with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, University of Melbourne, Parkville, VIC 3010, Australia. E-mail: rbuyya@unimelb.edu.au.*

Manuscript received 9 Sept. 2020; revised 12 Dec. 2020; accepted 3 Jan. 2021. Date of publication 6 Jan. 2021; date of current version 14 Jan. 2022. (Corresponding author: Zhicheng Cai.) Recommended for acceptance by J. Chen. Digital Object Identifier no. 10.1109/TC.2021.3049598

performance model only works well when the system is near the reference point [17], [18].

The deviation between the reference point and the real-time response time is called output error which has a great influence on control performance. Selecting appropriate reference points to compute output errors is helpful to obtaining a stable control performance. In existing methods [19], reference points are usually selected manually and kept unchanged for different arrival rates. However, a fixed reference point is likely to incur great control fluctuations because there are different stable working points for different arrival rates. Meanwhile, the inconsistency between the profiled performance model and the real system is unavoidable which makes output errors mismatch with the profiled performance model.

In this paper, an inverse-queuing-model-based feedback control method (FeedBack_InverseQM) is proposed to guarantee the QoS of container-based Web systems in Kubernetes which decreases the percentage of SLA violation by 8.44 percent. The main contributions of our work are as follows.

1) A hybrid of varying-processing-rate-based queuing model and linear model is applied to describe the performance of the multi-container system more accurately. Inverse-queuing model is used to linearize the control system to simplify the controller design.
2) An online reference model learning method is designed to find appropriate reference points for different arrival rates increasing the accuracy of output errors.
3) An adaptive output-error-mapping method is proposed to amend the inconsistency between sampled reference models and the profiled performance model avoiding fierce control fluctuations.

The rest of this paper is organized as follows. Section 2 is the related work and Section 3 describes Web systems in Kubernetes. The proposed method is introduced in Section 4. Sections 5 and 6 include performance evaluation on a real Kubernetes cluster, conclusions and future work.

2 Related Work

Most of existing works for container scheduling assume that containers consumed by each application is fixed. Deploying and migrating a fixed number of containers on private or elastically rented underlying resources can be modeled as a bin-packing problem [7] and solved by CPLEX [6]. For Kubernetes, heuristic methods such as best fit decreasing bin packing (BFD) and time-bin BFD [5], [20] have been proposed for the container deploying and migrating. Reinforcement learning [7] and game theory [21] are also used to deploy and migrate a fixed number of containers considering geography locations of Fog nodes, user mobility, energy consumption or SLA violations.

Compared with assigning fixed number of containers, providing scalable computing and storage capacities to applications is beneficial to saving resource cost and improving efficiency. For example, the authors of [22] propose a scheme to support data auditing processing in Cloud with fine-grained data updates (i.e., flexible size data updates). It saves huge storage and computation overheads of data auditing processing against fixed-size update where every small update will cause re-computation and updating for an entire file block. This scheme offers highly scalable and efficient data auditing processing in Cloud. To provide scalable computing capacity to applications in Kubernetes, genetic algorithms [23] and linear programming [24] have been used to find the optimal container distributing policy among multiple applications assuming that each application has a performance-degrading resource threshold or the utility of provisioning different numbers of containers to each application has been profiled in advance. In other words, these works mainly focus on the distribution of resources among competitive applications while container auto-scaling of one application is considered in this paper. Since auto-scaling methods of traditional application resources, PMs and VMs can be migrated to deal with container auto-scaling, auto-scaling techniques are surveyed in total.

2.1 Auto-Scaling Techniques for QoS Control

Threshold is one of the most simplest auto-scaling technique. Kubernetes's build-in auto-scaling scheduler adjusts container numbers based on thresholds of CPU or Memory usage rates [8]. One container is added when existing resources have been used up [9] or a fixed number of containers are added when the real-time response time is larger than a threshold [25]. The difficulty of threshold-based methods is how to select appropriate threshold values suitable for different arrival rates [26].

Queuing models describe the relations among the average response time, the request arrival rate and allocated resources of Web systems. M/M/1 [12], M/M/N [10], [11], heterogeneous M/M/N [27] queuing models and queuing networks [13] have been used to determine the minimum number of VMs or PMs for guaranteeing QoS of Web systems. Nonetheless, pure queuing-model-based methods lack the ability of reacting to real-time output errors.

Feedback control is an essential method for application resource (server processes, database connections, etc) auto-scaling. Linear-performance-model-based fixed gain [14], [15], adaptive [16] or multi-model switching [28] feedback control methods have been widely used in application resource auto-scaling [29]. An inverse-proportional performance model, which is more accurate than linear models, is used to design a feedback control method to auto-scale containers of Web applications [19]. Queuing models are more accurate than linear or inverse-proportional models, and have been used to improve the performance of feedback control. For example, M/M/1 queuing model-derived linear model, which describes the linear relation of output-error changes and the adjustment of allocated resources near the reference point, is used to design feedback controllers, and only works well near reference points [17], [18]. For Kubernetes, M/M/1-model-based feedback controller is designed to adjust the arrival-rate adjustment coefficient of a loosely coupled M/M/N model [30] which is tailored for avoiding over-control incurred by interval-based charging models of Cloud VMs.

Q-table and deep-neural-network based reinforcement learning methods have been used to allocate VMs or PMs to Web systems elastically [12], [31], [32]. However, these

TABLE 1
Comparison of Our Approach With Existing Resource Provisioning Algorithms for Web Systems

Resources	Problems	Objectives	Algorithms and Complexity	Platforms	Works
Application resources	Auto-scaling	Absolute or relative average response time	Linear-model-based feedback control, O(1)	A single Web Server	[14], [15], [16], [28]
			Queuing-model-derived linear model based hybrid control, O(n^2)	A single Web Server	[17], [18]
VMs or PMs	Auto-scaling	Average response time	Threshold, O(1)	Private VM clusters	[33]
			Reinforcement learning	Private VM or PM clusters, Public Clouds, MATLAB	[12], [31], [32], [33]
			Queuing models, O(n^2)	Simulation, OpenStack, Private VM clusters	[10], [11], [12], [13], [27]
			Queuing-length-based feedback control, O(1)	CometCloud	[35]
			Queuing-model-arrival-rate-adjusting-coefficient based hybrid control, O(n^2)	CloudSim, Private VM clusters	[30]
Containers	Fixed containers per application	Power consumption, etc	Deep Q-learning	Private VM clusters	[7]
		Deployment Cost of VMs	CPLEX, BFD, Time-bin BFD, Greedy algorithms	Kubernetes, Simulation	[5], [6], [20]
	Auto-scaling	Resource usage, Average response time	Threshold, O(1)	IaaS, Docker Swarm	[9] [25]
		Total network latency, etc	Threshold, Genetic algorithm	Simulation	[23]
		Average response time	Simple non-linear model based feedback control, O(1)	ECOWARE	[19]
			Inverse-M/M/N-model based feedback control, O(n^2)	Kubernetes	Our approach

methods need a long sampling and training period to obtain good performance [33]. Therefore, exact model based methods are still necessary and can be used to guide the deep learning based methods.

2.2 Comparison With Existing Algorithms

Table 1 shows a comparison of our approach with existing resource provisioning algorithms. First, existing linear or queuing-model-derived linear models cannot describe the performance of container based systems accurately [34]. On the contrary, our approach applies a feedback controller based on a more accurate performance model which is the hybrid of varying-processing-rate M/M/N model and linear model. Second, most existing works use a fixed reference point obtained by experience to compute output errors directly, while our approach applies an automatic reference point identification method and an adaptive output error mapping method to improve the control stability. Finally, our approach is implemented as a user-level scheduler of Kubernetes rather than simulation platforms.

3 Web Systems in Kubernetes

It is flexible and inter-operable to organize diverse Web systems using micro-service-based architecture. Each tier of such Web systems can be implemented as a micro-service which runs in parallel containers to support large-scale requests. Each micro-service provides a single function and is usually encapsulated as a RESTful or SOAP-based Web service. Kubernetes is widely used to manage containers which includes one Master and multiple Worker nodes as shown in Fig. 1. Pod is the basic unit of resource management which consists of one or multiple containers [3]. Containers of micro-services are embedded in Pods which are deployed on Worker nodes (PMs or VMs). Containers of the same color belong to the same micro-services. User requests of the same micro-service are distributed to parallel containers by the nginx-ingress-controller [36]. Real-time performance of each micro-service is monitored through Kubernetes java-client-interface [37], based on which the

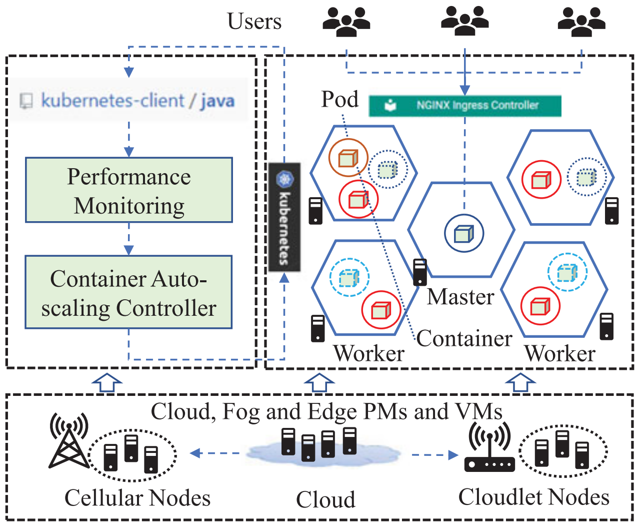

Fig. 1. Architecture of container-based web systems in Kubernetes.

TABLE 2
Common Notations

Label	Description
λ	Real-time request arrival rate
μ	Varying processing rate
W_{sla}	Maximum response time described in SLA
k	Index of control steps
y_k	Real-time average response time of step k
N_k	Output container number of control step k
N_m	Available containers for one microservice
L_r	Real-time queuing length
u_k	Control input
m_k	Reference model of step k with maximum arrival rate deviation of $5/s$
m'_k	Mature reference model of step k with maximum arrival rate deviation of $20/s$
$W^r_{m'_k}$	Reference response time of m'_k
$n^r_{m'_k}$	Maximum number of containers not fulfilling SLA in m'_k (called lower bound)

container Auto-scaling Controller (ASC) is implemented as a user-level plugin to allocate appropriate number of containers to each micro-service separately. Each micro-service has its own ASC, i.e., a decentralized controlling method is applied [19].

The average response time and resource cost are two crucial metrics of containerized Web systems. In the SLA of a micro-service, it is usually defined that $x\%$ of response times should be smaller than a given threshold W_{sla}. Because identical containers are usually required by the same micro-service, one Container Unit (CU) is defined to be the cost of allocating one container to a micro-service in one control interval. The objective of this article is to design container auto-scaling algorithms for ASCs to decrease the number of consumed CUs while guaranteeing SLAs. Common notations of this article are shown in Table 2.

4 Proposed Feedback Control Method

In this paper, an Inverse-Queuing-Model-based feedback control method (Feedback_InverseQM) is proposed. A hybrid of varying-processing-rate-based M/M/N model and linear model is first adopted to describe the system accurately. Then an automatic reference model learning method is developed to generate accurate output errors. Based on the performance model and output errors, an inverse-queuing-model-based Integral controller is designed. Next an adaptive output error mapping method is investigated to amend the inconsistency between sampled reference models and the profiled queuing model. Finally, a queuing-length-based scheduling method is used to provision containers when the system is unstable.

4.1 Varying-Processing-Rate-Based Performance Model

Performance model is the basis of feedback control. The average processing rate of one container decreases as the request arrival rate increases in Kubernetes because of scheduling overhead. Traditional queuing models with fixed processing rates-based performance models cannot describe the system accurately. Let λ be the request arrival rate and N be the number of containers. In this paper, the processing rate of each container is defined to be inversely-proportional to the arrival rate as

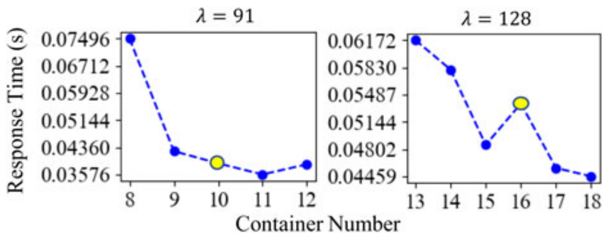

Fig. 2. Average response times of different arrival rates given different numbers of containers.

$$\mu = \mu_b + c/\lambda, \tag{1}$$

where μ_b is the basic processing rate and c is the inverse-proportional coefficient of λ. According to existing M/M/N model [10], [11], the probability of no requests in the whole system is

$$P_0 = \left[\sum_{k=0}^{N-1} \frac{1}{k!}\left(\frac{\lambda}{\mu}\right)^k + \frac{\lambda^N}{N!(1-\frac{\lambda}{N\times\mu})\mu^N}\right]^{-1}. \tag{2}$$

The expectation of the number of requests in the waiting queue and under processing is

$$L_s(N,\lambda,\mu) = \frac{(\frac{\lambda}{\mu})^N \frac{\lambda}{N\times\mu}}{N!(1-\frac{\lambda}{N\times\mu})^2} P_0 + \frac{\lambda}{\mu}. \tag{3}$$

The expectation of the average response time of requests is

$$W_s(N,\lambda,\mu) = \frac{L_s(N,\lambda,\mu)}{\lambda}. \tag{4}$$

Meanwhile, the current average response time is also affected by past values. Therefore, in this paper, the average response time y_k of control step k is defined to be a weighted combination of the past value y_{k-1} and $W_s(N,\lambda,\mu)$ as

$$y_k = a \times y_{k-1} + (1-a) \times W_s(N,\lambda,\mu). \tag{5}$$

Equation (5) is a hybrid of inverse-proportional-processing-rate (varying processing rate) based M/M/N queuing model and linear model which is applied to be the performance model of this paper.

4.2 Automatic Reference-Model Learning

Reference points are crucial to compute output errors which are the basis of feedback control. Because containers are provisioned in a coarse-grained granularity, average response times (called stable points) cannot change smoothly when the number of provisioned containers is adjusted as shown in Fig. 2. Therefore, the given W_{sla} might be far from any stable point, and using W_{sla} as the reference point directly is likely to generate great fluctuations. Appropriate reference response times should be selected from stable points smaller than W_{sla}. Fig. 2 illustrates that stable points of different arrival rates are different, the reference point W^r should be found for each arrival rate individually. For each λ, the maximum number of containers n^r violating SLA is called the lower bound which is able to avoid releasing too many containers. For each λ, stable points of different numbers of containers can be studied from historical data.

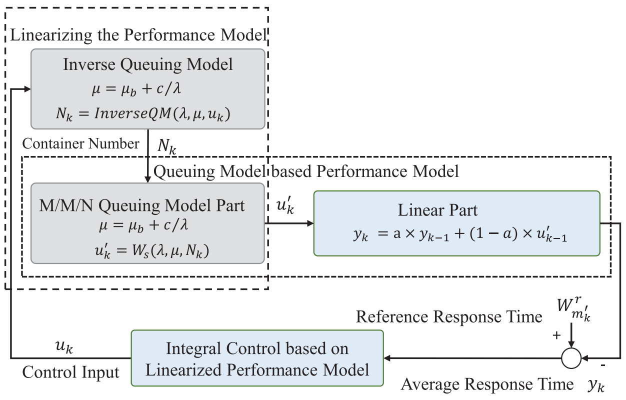
Linearizing the Performance Model
Inverse Queuing Model
$\mu = \mu_b + c/\lambda$
$N_k = InverseQM(\lambda, \mu, u_k)$
Container Number N_k
Queuing Model based Performance Model
M/M/N Queuing Model Part
$\mu = \mu_b + c/\lambda$
$u'_k = W_s(\lambda, \mu, N_k)$
u'_k
Linear Part
$y_k = a \times y_{k-1} + (1-a) \times u'_{k-1}$
Reference Response Time $W^r_{m'_k}$
u_k
Control Input
Integral Control based on Linearized Performance Model
Average Response Time y_k

Fig. 3. Architecture of hybrid control for container-based web systems.

W^r and n^r can be obtained base on stable points of each λ. The set of stable points, W^r and n^r is called a reference model of λ which is used to guide the feedback control.

An automatic reference-model learning method (ARML) is proposed to profile reference models based on samples in the form of $s = (n_s, \lambda_s, y_s)$ in which n_s is the container number, λ_s is the arrival rate and y_s is the average response time. M is the set of reference models. In order to decrease the number of studied reference models, the difference between any two models' reference arrival rates should be larger than a gap g (e.g., 5/s). Samples with similar arrival rates are used to profile the same model.

Formal description of ARML is shown in Algorithm 1. In auto-scaling step k, a new sample s will be collected and a corresponding reference model $m_k = \arg\min_{m\in M}\{\lambda_d = |\lambda_m - \lambda|, \lambda_d < g\}$ is tried to be found. If $m_k = null$, a new model m_k with reference arrival rate λ is created and added to M. If $m_k = m_{k-1}$ which means that the system's arrival rate is stable in two consecutive steps, s will be processed as follows. If $y_s > W_{sla}$ and $n_s > n^r_{m_k}$, n_s is a larger number which violates SLA and $n^r_{m_k}$ is replaced by n_s. For example, $n^r_{m_k} = 10$ means that providing 10 containers will lead to SLA violations, but allocating 11 containers is able to guarantee SLA. When a new sample is obtained with $n_s = 13$ and $y_s > W_{sla}$, it means that allocating 13 containers is already not able to fulfill SLA. Therefore, 13 is a new maximum number not fulfilling SLA. When $y_s \leq W_{sla}$, s is stored in a hashmap $h_{m_k} =< n, bt_n >$ in which container number n is the key and bucket bt_n contains average response times of samples with container number n. For each n, only latest ten average response times are stored in bt_n and values with distances larger than $0.4 \times average(bt_n)$ from the average value $average(bt_n)$ are filtered. Then $average(bt_n)$ is updated based on filtered values for each n and added to a set Y. Finally, the third largest value in Y is taken as the reference response time $W^r_{m_k}$. For example, in Fig. 2, the hollow circles are the third largest average response times when $W_{sla} = 0.1$ s. The reason of not selecting the first two largest stable points is that small workload fluctuations might lead to SLA violations because the first two largest stable points are near W_{sla}. If $n_s \leq n^r_{m_k}$ which means a smaller or equal number of containers already can fulfill SLA compared with $n^r_{m_k}$, $n^r_{m_k}$ is updated to be $n_s - 1$. Whenever a new sample s is collected from the system, $W^r_{m_k}$ and $n^r_{m_k}$ are updated as mentioned above. Only when $n^r_{m_k}$ has been assigned a value and the number of hashmap's keys ($h^{keys}_{m_k}$) is larger than four, the model m_k is called mature and added to a mature model set M'. Because immature models do not have sufficient samples to provide appropriate reference points and lower bounds, they cannot be used.

After s is added to m_k, a mature reference model $m'_k = \arg\min_{m\in M'}\{\lambda_d = |\lambda_m - \lambda|, \lambda_d < g'\}$ is tried to be found. g' is usually set to be larger than g (e.g., 20/s) to allow mature models to guide more scenarios with larger arrival-rate differences. $m'_k = m_k$ only when m_k is mature. If m'_k can be found, $W^r_{m'_k}$ is selected as the reference response time which will be used as the reference point of feedback control. Otherwise, a sampling method is activated to collect more samples for m_k. Let $n^s_{m_k}$ and $n^l_{m_k}$ be the smallest and largest keys of h_{m_k}. If $n^s_{m_k} - 1 > n^r_{m_k}$ or $n^r_{m_k} = null$, container number N_k of current step k is set to be $n^s_{m_k} - 1$. Otherwise, $N_k = n^l_{m_k} + 1$.

Algorithm 1. Automatic Reference Model Learning (ARML)

```
   input: $s$, $\lambda$, $m_{k-1}$, $W_{sla}$
1  Initialize $g \leftarrow 5/s$, $g' \leftarrow 20/s$, $Y \leftarrow \emptyset$ ;
2  $m_k \leftarrow \arg\min_{m\in M}\{\lambda_d = |\lambda_m - \lambda|, \lambda_d < g\}$;
3  if $m_k = null$ then
4      Create $m_k$, $\lambda_{m_k} \leftarrow \lambda$ and $M \leftarrow M \cup \{m_k\}$;
5  if $m_k = m_{k-1}$ then
6      if $y_s > W_{sla}$ and ($n_s > n^r_{m_k}$ or $n^r_{m_k} = null$) then
7          $n^r_{m_k} \leftarrow n_s$;
8      if $y_s \leq W_{sla}$ then
9          Store $s$ in hashmap $h_{m_k}$;
10         for each $n \in h^{keys}_{m_k}$ do
11             Calculate $average(bt_n)$;
12             $Y \leftarrow Y \cup \{average(bt_n)\}$;
13         $W^r_m \leftarrow$ the third largest value in $Y$;
14         if $n_s \leq n^r_{m_k}$ then
15             $n^r_{m_k} \leftarrow n_s - 1$;
16         if $n^r_{m_k} \neq null$ and $|h^{keys}_{m_k}| > 4$ then
17             $M' \leftarrow M' \cup \{m_k\}$;
18 $m'_k = \arg\min_{m\in M'}\{\lambda_d = |\lambda_m - \lambda|, \lambda_d < g'\}$;
19 if $m'_k = null$ then
20     if $n^s_{m_k} - 1 > n^r_{m_k}$ or $n^r_{m_k} = null$ then
21         $N_k \leftarrow n^s_{m_k} - 1$;
22     else
23         $N_k \leftarrow n^l_{m_k} + 1$ ;
24     return $null$, $N_k$;
25 else
26     return $m'_k$, $null$;
```

4.3 Inverse Queuing Model-Based Controller Design

If a mature reference performance m'_k can be found, feedback control is applied to follow the reference time $W^r_{m'_k}$. The control error is

$$e_k = W^r_{m'_k} - y_k. \tag{6}$$

A controller should be designed based on performance model in Equation (5) and e_k. However, designing a feedback controller based on the hybrid non-linear performance model directly is very complex. Therefore, an inverse function of queuing model is used to linearize the performance model which simplifies the design of feedback controllers [19], [30] as shown in Fig. 3. Because it is complex to deduce

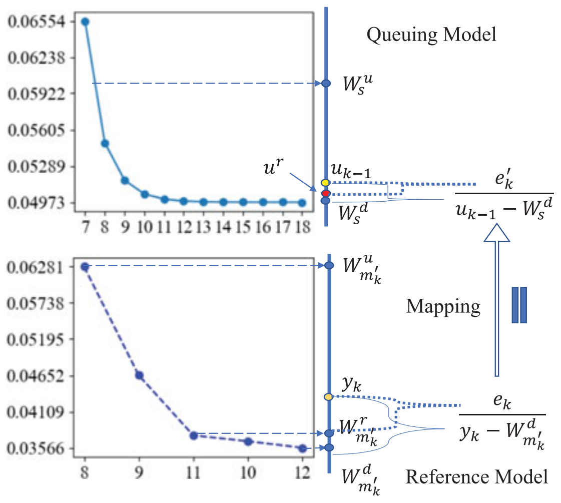

Fig. 4. Mapping from the reference model to profiled queuing model.

an inverse-queuing function from Equation (4) directly, an exhausted search method is used to implement the inverse-queuing function to find the corresponding container number N_k given average response time u_k (control input) as shown in Algorithm 2. According to queuing theory, the total processing rate $N_k \times \mu$ should be larger than arrival rate λ to make system stable. Therefore, N_k is first initialized to be $\lceil \frac{\lambda}{\mu} \rceil$. Then N_k is increased one by one based on $\lceil \frac{\lambda}{\mu} \rceil$ until $W_s(N_k, \lambda, \mu) > u_k$. The queuing function from N_k to u_k' (queuing model part) in the performance model is counteracted by the inverse-queuing function from u_k to N_k. In other words, $u_k' = u_k$, if queuing model part is accurate in describing multi-container systems. Then the linearized performance model is

$$y_k = a \times y_{k-1} + (1-a)u_{k-1}' = a \times y_{k-1} + (1-a)u_{k-1}, \tag{7}$$

and the Z-transfer function [29] of the performance model is

$$\frac{Y}{U} = \frac{1-a}{z-a}. \tag{8}$$

Algorithm 2. Inverse Queuing Model (InverseQM)

```
input: $\lambda$, $\mu$, $u_k$
1 $N_k \leftarrow \lceil \frac{\lambda}{\mu} \rceil$;
2 while $W_s(N_k, \lambda, \mu) > u_k$ do
3     $N_k \leftarrow N_k + 1$;
4 return $N_k$
```

An integral feedback controller is designed based on the linear part merely ignoring the queuing model part as follows:

$$u_k = u_{k-1} + K_I \times e_k. \tag{9}$$

Z-transfer function of the integeral controller is

$$\frac{U}{E} = \frac{K_I z}{z-1}. \tag{10}$$

Z-transfer function of the whole feedback system is

$$F_R(z) = \frac{Y}{R} = \frac{\frac{1-a}{z-a}\frac{K_I z}{z-1}}{1 + \frac{1-a}{z-a}\frac{K_I z}{z-1}} = \frac{(1-a)K_I z}{z^2 + [(1-a)K_I - 1 - a]z + a}. \tag{11}$$

Through linearization, queuing theory and feedback control are combined to provide accurate performance modeling and feedback abilities simultaneously.

4.4 Adaptive Output Error Mapping Method

In order to keep the system stable, control input u_k should be smaller than $W_s^u = W_s(n^r_{m_k'}, \lambda, \mu)$ to make N_k larger than $n^r_{m_k'}$. Meanwhile, u_k cannot be smaller than $W_s^d = W_s(N_m, \lambda, \mu)$ to allocate no more than the maximum number of available containers N_m. Because $n^r_{m_k'}$ is determined by one sample and may be not accurate. To allow sampling on $n^r_{m_k'}$ again, W_s^u is set to be

$$W_s^u = \frac{W_s(n^r_{m_k'}, \lambda, \mu) + W_s(n^r_{m_k'} - 1, \lambda, \mu)}{2}. \tag{12}$$

The output error e_k is used to adjust u_k by Equation (9). However, there are still unavoidable deviation between reference model m_k' and profiled performance model as shown in Fig. 4. When the deviation is too large, u_k is likely to change greatly and exceed the lower or upper bounds $[W_s^d, W_s^u]$ of the queuing model part. Let $[W^d_{m_k'}, W^u_{m_k'}]$ be the lower and upper bounds of reference model m_k' which are the smallest and largest response times smaller than W_{sla}, respectively. To guarantee the bounds $[W_s^d, W_s^u]$, an adaptive output-error mapping method (AOM) is proposed to map e_k from the reference model space to output error e_k' in the queuing model space.

Theorem 1. *If output errors of the reference model are mapped to the performance model in proportional, there are*

$$e_k' = e_k \times K_a \tag{13}$$

$$K_a = \begin{cases} \frac{u_{k-1} - W_s^d}{y_k - W^d_{m_k'}} & y_k > W^r_{m_k'} \\ \frac{W_s^u - u_{k-1}}{W^u_{m_k'} - y_k} & \text{Otherwise} \end{cases}. \tag{14}$$

Proof 1. It is assumed that u^r (unknown) is the target reference point in queuing model space which is able to make the system get response time $W^r_{m_k'}$. Then $e_k' = u^r - u_{k-1}$. The main objective of the mapping is to find an appropriate e_k' to make u_k more and more close to u^r. According to Equation (5), $a \times y_{k-1} + (1-a)u^r$, $a \times y_{k-1} + (1-a)W_s^d$ and $a \times y_{k-1} + (1-a)u_{k-1}$ are corresponding values in the complete performance model (the total of queuing model and linear part) of u^r, W_s^d and u_{k-1} in the queuing model space, respectively. The output error in the performance model is $e_k^p = (a \times y_{k-1} + (1-a)u^r) - (a \times y_{k-1} + (1-a)u_{k-1})$. If values of the reference model is mapped to the performance model in proportional, the ratio of e_k to $y_k - W^d_{m'_k}$ is equal to the ratio of e^p_k to $(a \times y_{k-1} + (1-a)u_{k-1}) - (a \times y_{k-1} + (1-a)W^d_s)$ as follows:

$$\begin{aligned}&\frac{e_k}{y_k - W^d_{m'_k}}\\ &= \frac{e^p_k}{(a \times y_{k-1} + (1-a)u_{k-1}) - (a \times y_{k-1} + (1-a)W^d_s)}\\ &= \frac{(1-a)(u^r - u_{k-1})}{(1-a)(u_{k-1} - W^d_s)}\\ &= \frac{e'_k}{u_{k-1} - W^d_s}.\end{aligned} \tag{15}$$

When $y_k \leq W^r_{m'_k}$, the prove is similar. □

In order to avoid large fluctuations, K_a is trimmed to 1 when $K_a > 1$. By replacing e_k of Equation (9) using e'_k, the integral controller becomes

$$u_k = u_{k-1} + K'_I \times e_k, \tag{16}$$

where $K'_I = K_I \times K_a$ is called adaptive control gain.

4.5 Queuing-Length-Based Unstable State Scheduling

When the arrival rate λ is larger than the total processing ability μ, the system is unstable and Equations (2), (3) and (4) are not valid. Therefore, it is not suitable to find N_k based on e_k directly. In unstable states, the real queuing length L_r is larger than $L_s(N_k, \lambda, \mu)$ and increases continually until the allowed maximum number of waiting connections is reached which represents the blocking degree. Therefore, a queuing-length-based unstable state provisioning method is applied as shown in Algorithm 3. First, if $n_s > n^r_{m_k}$ which means n_s is a larger container number not fulfilling SLA, $n^r_{m_k}$ is replaced by n_s. Given N_k containers, the theoretical response time y_{k+1} of next step is the weighted combination of current response time y_k and expected response time $W_s(N_k, \lambda, \mu)$ of queuing models according to Equation (5). An iterative search is used to find the minimum number of containers N_k making $y_{k+1} > W_{sla}$ based on $\lceil \frac{\lambda}{\mu} \rceil$. When $N_k \leq N_{k-1}$, at least one new container should be added every step. Each second, newly added $N_k - N_{k-1}$ containers in this step can process additional $\mu \times (N_k - N_{k-1})$ requests. N_k is increased one by one to make sure that real-time queuing length L_r can decrease to theoretical queuing length L_s in T_r (e.g., 10) seconds by processing $\mu \times (N_k - N_{k-1}) \times T_r$ additional requests on $N_k - N_{k-1}$ containers. Larger T_r means decreasing queuing length more quickly. Finally, u_k is updated to guarantee that N_k containers are still rented in the next control step if there is no output errors.

4.6 Formal Description of Feedback_InverseQM

Formal description of Feedback_InverseQM is shown in Algorithm 4. At first, performance model is profiled using historical data. For queuing systems, when the real queuing length L_r is larger than theoretical queuing length $L_s(N_k, \lambda, \mu)$, the system is unstable. Given arrival rate λ, at least $N = \lceil \frac{\lambda}{\mu} \rceil$ containers are required to make system stable. The theoretical response time given N containers is $W_s(N, \lambda, \mu)$, which is called the largest stable response time. When y_k is larger than the largest stable response time, the system is unstable too. Because there are still deviations between the queuing model and real system, the queuing length is likely to be a little bit larger than the theoretical queuing length occasionally even if the system is stable. The above criterion of unstable is too strict, which is likely to lead to frequent switching between stable and unstable control making the system fluctuate greatly. Therefore, the system is judged to be unstable only when L_r is larger than α (e.g., 10) times of $L_s(N_k, \lambda, \mu)$ or y_k is larger than β (e.g., 1.2) times of $W_s(N, \lambda, \mu)$. Giving larger α and β means a more looser criterion of judging the system to be unstable. If the system is unstable, QLP is invoked. Otherwise, ARML is called to get a reference model m'_k or obtain a sampling N_k. If $m'_k \neq null$, Equation (16) is used to obtain u_k, and InverseQM is applied to get the real control action N_k. Finally, the number of containers allocated to the Web system is adjusted to be N_k.

Algorithm 3. Queuing-Length-Based Provisioning (QLP)

```
   input: $\lambda$, $\mu$, $W_{sla}$, $N_{k-1}$, $L_r$, $y_k$
1  if $n_s > n^r_{m_k}$ or $n^r_{m_k} = null$ then
2      $n^r_{m_k} \leftarrow n_s$;
3  $N_k \leftarrow \lceil \frac{\lambda}{\mu} \rceil$, $T_r \leftarrow 10$ ;
4  $y_{k+1} \leftarrow a \times y_k + W_s(N_k, \lambda, \mu) \times (1-a)$;
5  while $y_{k+1} > W_{sla}$ do
6      $N_k \leftarrow N_k + 1$;
7      $y_{k+1} \leftarrow a \times y_{k-1} + W_s(N_k, \lambda, \mu) \times (1-a)$;
8  if $N_k \leq N_{k-1}$ then
9      $N_k \leftarrow N_{k-1} + 1$
10 while $\mu \times (N_k - N_{k-1}) \times T_r < L_r - L_s(N_k, \lambda, \mu)$ do
11     $N_k \leftarrow N_k + 1$;
12 if $(N_k - 1) \times \mu > \lambda$ then
13     $u_k \leftarrow \frac{W_s(N_k,\lambda,\mu) + W_s(N_k - 1,\lambda,\mu)}{2}$;
14 else
15     $u_k \leftarrow W_s(N_k, \lambda, \mu) + \epsilon$, $\epsilon$ is infinitely small;
16 return $N_k$
```

Algorithm 4. Inverse Queuing Model Based Feedback Control (FeedBack_InverseQM)

```
   input: $\lambda$, $L_r$, $s$, $y_{k-1}$
1  Initialize $\alpha \leftarrow 10$, $\beta \leftarrow 1.2$, $y_k \leftarrow y_s$;
2  $\mu_b, c, a \leftarrow$Profiling the performance model;
3  $\mu = \mu_b + c/\lambda$, $N \leftarrow \lceil \frac{\lambda}{\mu} \rceil$;
4  if $L_r > L_s(N_k, \lambda, \mu) \times \alpha$ or $y_k > W_s(N, \lambda, \mu) \times \beta$ then
5      $N_k \leftarrow$ QLP($\lambda$, $\mu$, $W_{sla}$, $N_{k-1}$, $L_r$, $y_{k-1}$);
6  else
7      $m'_k, N_k \leftarrow$ ARML($s$, $\lambda$, $m_{k-1}$, $W_{sla}$, $y_k$);
8      if $m'_k \neq null$ then
9          $u_k \leftarrow$ Equation (16);
10         $N_k \leftarrow$ InverseQM($\lambda$, $\mu$, $u_k$);
11 Allocate $N_k$ containers to the system;
```

5 Performance Evaluation

Our proposed FeedBack_InverseQM is implemented as a user-level scheduler for Kubernetes. It is compared with

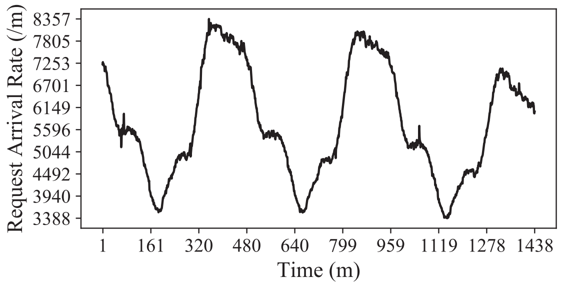

Fig. 5. Request arrival rates of applied Wikipedia access traces.

existing algorithms on a real Kubernetes cluster which locates on four physical machines with the configuration of 6∼12 virtual CPU cores and 8∼16 GB Memory. The Kubernetes cluster consists of one Master and four Worker nodes. A service for calculating Fibonacci numbers is adopted as the test-bed and the input of the service is the length of the generated Fibonacci series which is selected from the interval [26,33] for each request randomly. W_{sla} is 0.1 second. Requests of the service are redirected to different containers by the nginx-ingress-controller using exponentially weighted moving average (EWMA) as the load-balancing algorithm [36]. The response time of every request is stored in the log of nginx, and ASC obtains the average response time of every minute by reading logs through the java client-interface of Kubernetes (JCI). The queuing length and request arrival rate are obtained by reading status information of nginx using the http protocal. Pod auto-scaling commends generated by ASC are sent to Kubernetes by changing the value of deployment's replicas through JCI. The connection timeout time of nginx is 10 seconds and the allowed maximum number of connections is 500 per container. The user access traces of Wikipedia [38] as shown in Fig. 5 with common peaks and valleys of Web systems are used to generate requests through JMeter [39].

FeedBack_InverseQM is first compared with Feedback_QMDL which is a classical feedback control method for QoS control based on queuing-model-derived linear models [17], [18]. Although Feedback_QMCA [30] is tailored for hourly-priced VM provisioning, FeedBack_InverseQM is still compared with Feedback_QMCA by removing the VM-releasing status checking. Finally, Feedback_InverseQM is also compared with Feedback_InverseP [19] which is one of the elastic container provisioning algorithms considering QoS control. Average response times and the total consumed CUs are metrics of algorithm comparison. The length of control interval of all algorithms is 250 seconds. Since FeedBack_InverseQM has an initial sampling period, the cost of each compared algorithm is only the accumulation of consumed CUs after the initial 500 minutes for fair comparison.

5.1 Parameter Tunning

The control gain K_I determines the poles of the system which has a great impact on the settle time and overshoot. Poles can be derived by setting $z^2 + [(1-a)K_I - 1 - a]z + a = 0$ given K_I. For example, poles $p_1 = 0.44 + 0.32619013j$ and $p_2 = 0.44 - 0.32619013j$ when $K_I = 0.6$. According to root locus which draws the figure of poles as K_I changes [29], larger K_I usually means larger overshoots, and too small or too large K_I are likely to incur long settle times. A set of candidate values $S_{ki} = \{0.05, 0.1, 0.15, 0.3, 0.6\}$ are selected based on root locus. Then the real perfromance of $K_I \in S_{ki}$ is evaluated by experiments. Figs. 6 and 7 show consumed container numbers and average response times of FeedBack_InverseQM with different K_I which illustrate that container numbers of FeedBack_InverseQM changes more quickly as K_I increases in total. For smaller K_I, FeedBack_InverseQM reacts to excess containers and SLA violations very slowly. For example, excess containers cannot be released in time consuming the most cost (2454 CUs) when $K_I = 0.05$ and SLA is violated for many periods when $K_I = 0.1$. On the contrary, for larger K_I, the container number fluctuate fiercely leading to more SLA violations when $K_I \geq 0.3$ and extremely large $K_I = 0.6$ even incurs the second most cost (2290 CUs) because of frequent container allocating and deallocating. Therefore, $K_I = 0.15$ is finally selected which obtains the most appropriate control strength leading to fewer SLA violations and a lower cost simultaneously.

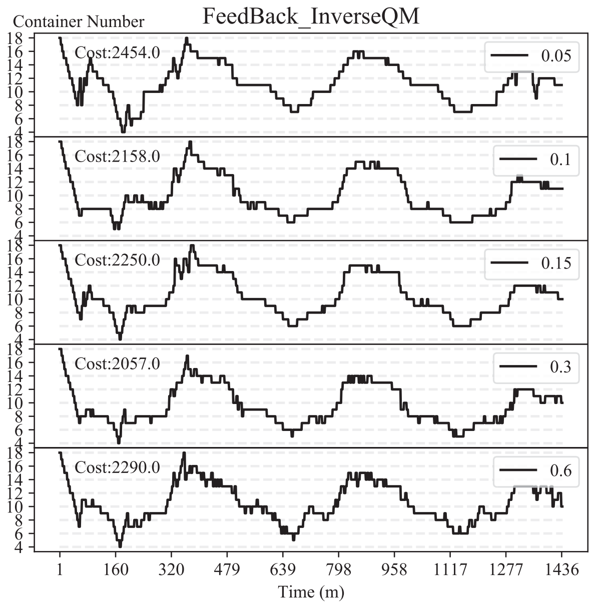

Fig. 6. Container numbers of FeedBack_InverseQM with different K_I.

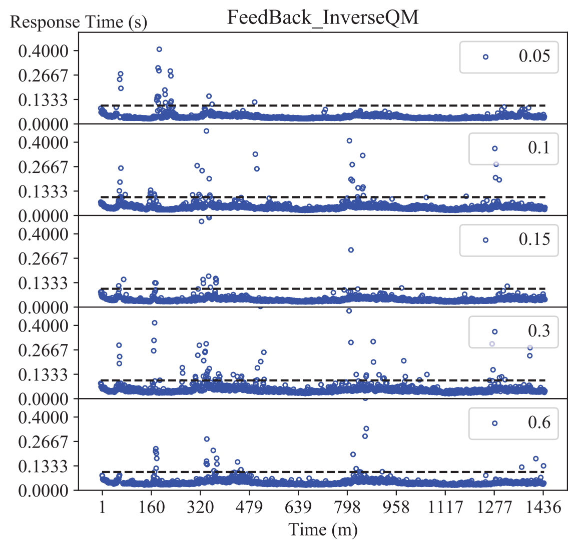

Fig. 7. Response times of FeedBack_InverseQM with different K_I.

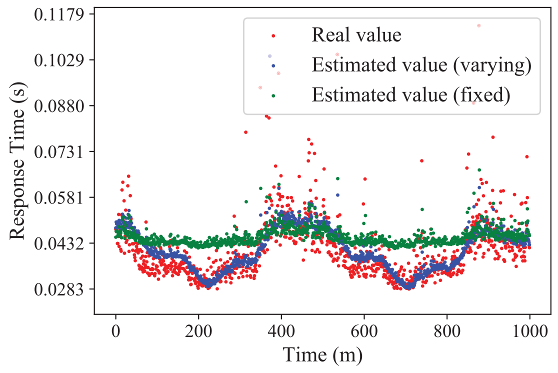

Fig. 8. Real response times and estimated values of profiled performance models using fixed and varying processing rates, respectively.

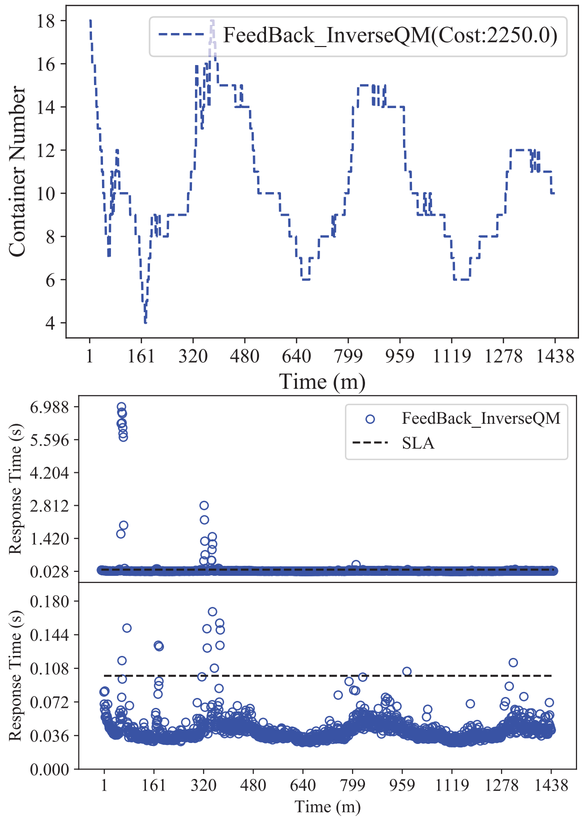

Fig. 9. Container numbers and response times of FeedBack_InverseQM.

According to the performance tuning results of [30], poles of Feedback_QMDL and Feedback_QMCA are set to be 0.9 and 0, respectively. The pole of Feedback_InverseP is set to be 0.95 consistent with [19].

To obtain vaules of performance model's parameters, the least square method is applied based on historical data (including average response times, container numbers and arrival rates) collected from Kubernetes platforms under $0 < a < 0.3$, $\mu_b > 0$ and $c > 0$ constraints. Without constraint $0 < a < 0.3$, $a > 0.9$ is likely to be obtained because of the similarity between two consecutive response times which will suppress the impact of the queuing model part. Constraint $c > 0$ is used to guarantee that the processing ability decreases as the arrival rate increases. If μ_b of the trained model is smaller than zero, processing ability μ will be smaller than zero when $\lambda > \frac{c}{-\mu_b}$ which makes the model meaningless. Therefore, $\mu_b > 0$ is necessary. For our kubernetes-based platform, $\mu_b = 7.771$, $c = 1574.510$ and $a = 0.215$ are obtained based on the given historical data which may change over time. Parameters of the fixed-processing-rate-based performance model can be acquired similarly by setting $c = 0$. Fig. 8 shows real response times and estimated values of profiled performance models which illustrates that the proposed varying-processing-rate-based method (blue dots) describes the system more accurately than the fixed-request-processing-rate based method (green dots).

TABLE 3
Percentages of SLA Violations and Costs

Algorithms	SLA Violations	Costs
Feedback_InverseQM	**2.36%**	2250 CUs
Feedback_QMDL	10.80%	2289 CUs
Feedback_QMCA	21.26%	**1989 CUs**
Feedback_InverseP	52.99%	2487 CUs

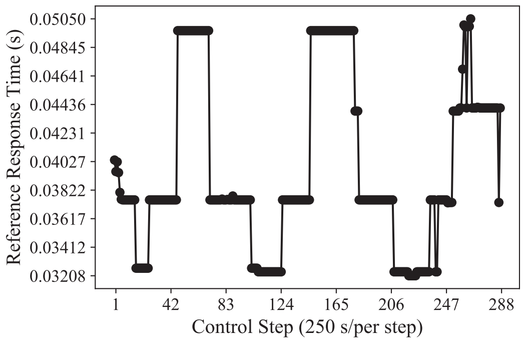

Fig. 10. Reference response times of control steps with mature models.

5.2 Experimental Results

Table 3 shows percentages of SLA violations and costs of compared algorithms which illustrate that proposed FeedBack_InverseQM obtains the lowest percentage of SLA violation (8.44 percent smaller than that of the best existing algorithm) with the second lowest cost (2250 CUs) in total.

Fig. 9 shows container numbers and response times of FeedBack_InverseQM which denotes that most response times are smaller than W_{sla}. Reasons of FeedBack_InverseQM's best performance are as follows. First, ARML of FeedBack_InverseQM is helpful to improving the accuracy of output errors. Because different arrival rates have different stable points, samples are collected by ARML to study reference response times for diverse arrival rates leading to some fluctuations in the initial stage of Fig. 9. Fig. 2 shows stable points of two mature reference models with different arrival rates. In Fig. 10, the studied reference time changes over time which increases the accuracy of output errors. Second, AOM of FeedBack_InverseQM is able to improve control stability. FeedBack_InverseQM is sensitive to output errors when u_{k-1} is near relatively flatten parts of the queuing model as shown in Fig. 4. Flatten parts of performance models reflect the nature of queuing systems which enable controllers react to SLA violations quickly. However, there are unavoidable deviations between the reference model and the profiled queuing model. Sometimes, the original output errors based on the reference model are

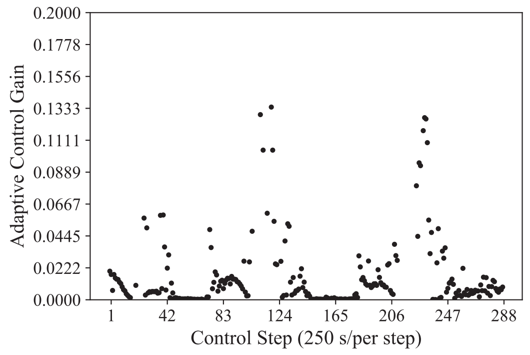

Fig. 11. Adaptive control gains ($K_I \times K_a$) of different control steps.

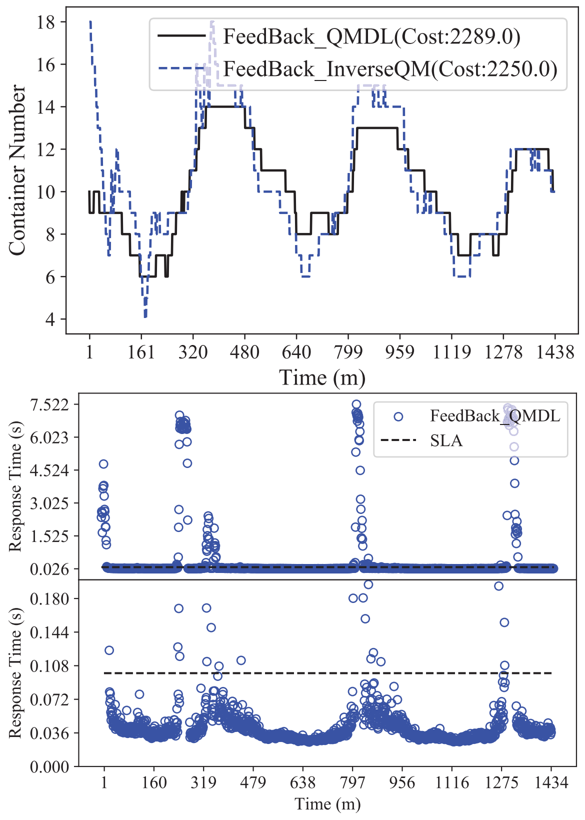

Fig. 12. Container numbers and response times of FeedBack_QMDL.

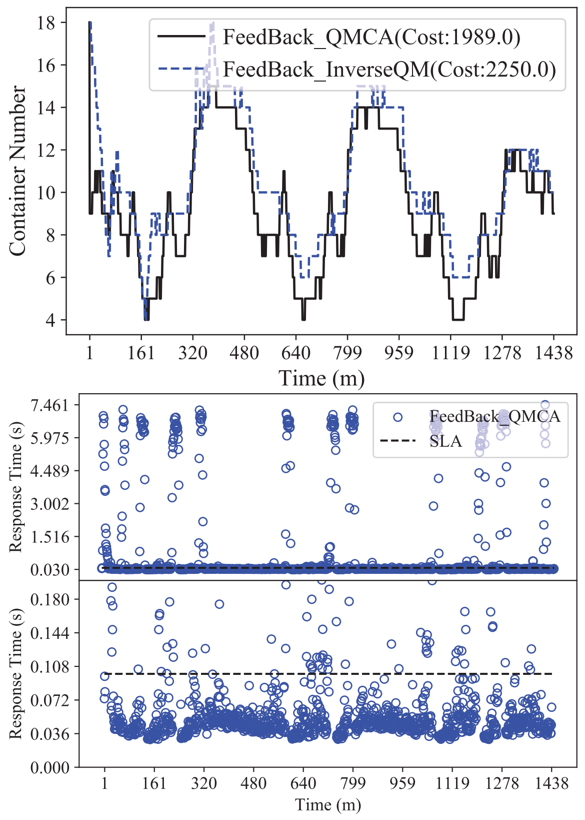

Fig. 13. Container numbers and response times of FeedBack_QMCA.

so large which makes u_k fluctuate drastically. Fig. 11 shows adaptive control gains of different steps generated by AOM which are used to map output errors from the reference model to the queuing model in proportion to avoid such fierce fluctuations. Both ARML and AOM make FeedBack_InverseQM obtain the most stable performance as shown in Fig. 9 except the initial sampling stage.

Fig. 12 illustrates that FeedBack-QMDL reacts slowly to fast arrival-rate changes leading to SLA violations or delaying the release of excess containers. The reason is that the output of FeedBack-QMDL is the plus of M/M/N model's output and an additional value determined by feedback control. Meanwhile, different arrival rates need quite different additional values, and the changing speed of the additional value cannot meet the requirement when the arrival rate changes quickly. However, the changing speed of the additional value cannot be increased by setting smaller poles (e.g., 0.6) anymore, because it is likely to allocate or release excess containers incurring more SLA violations or higher costs for scenarios with slow arrival rate changing speeds. The main reason is that the changing speed of the additional value is fixed and linear to the output errors without considering various arrival rates with different changing speeds.

Fig. 13 demonstrates that FeedBack_QMCA is very likely to allocate or deallocate excess containers leading to frequent large fluctuations at periods with slow arrival-rate changing speeds of which the reasons are as follows. In FeedBack_QMCA, the inaccuracy of queuing model is fixed by an arrival-rate adjustment coefficient. Experimental results illustrate that different arrival rates need quite different adjustment coefficients. When the arrival rate changes quickly, it takes a long time to adjust the coefficient leading to SLA violations or higher costs given small control gains. Therefore, the current proportional control gain has been increased as large as possible to speed up the changing speed of the adjustment coefficient to meet the requirement of fast arrival rate changes. However, the controller gain, which fulfills the fast-changed arrival rates, makes the system fluctuate when arrival rates change slowly. It is hard to find an appropriate gain suitable for different arrival-rate changing speeds. Meanwhile, a fixed reference point is not able to generate suitable output errors for all arrival rates which misleads the controller to release or rent excess containers incurring SLA violations or higher costs.

Fig. 14 shows that the container number of FeedBack_InverseP fluctuates fiercely and SLA is violated frequently although a large pole of 0.95 (long settle times) has been given. Both FeedBack-QMDL and FeedBack_QMCA use a gradually

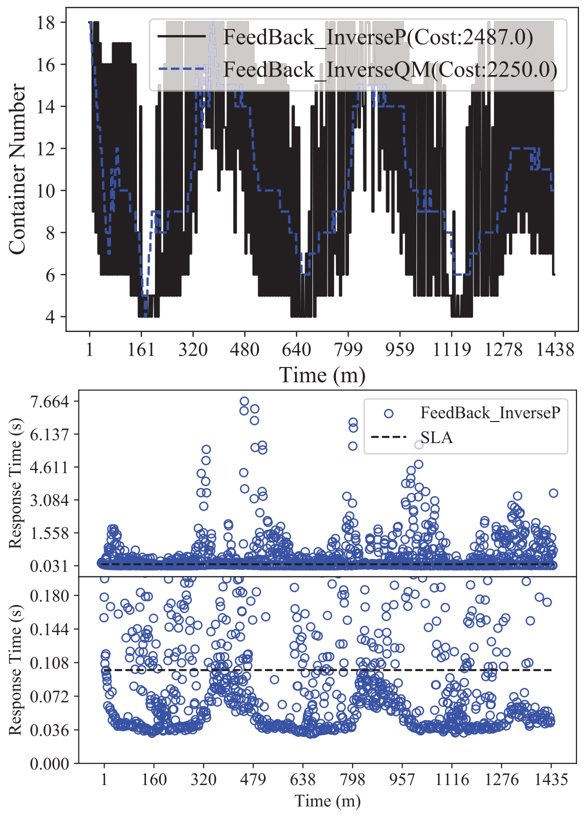

Fig. 14. Container numbers and response times of FeedBack_InverseP.

changed additional value or arrival-rate coefficient to amend the inaccuracy of queuing models which avoids fierce fluctuation of container numbers. On the contrary, similar with FeedBack_InverseQM, FeedBack_InverseP's inverse-proportional performance model is also very sensitive to output errors when u_{k-1} is near the relatively flatten parts of the performance model. However, a fixed reference point and inconsistency between the inverse-proportional performance model and the real system cannot always generate appropriate output errors for different arrival rates incurring fierce fluctuations of container numbers.

6 Conclusion and Future Work

In this paper, an inverse-queuing-model-based feedback control method has been proposed to provision containers to Web systems in Kubernetes-based platforms elastically for guaranteeing QoS. Experimental results show that the hybrid of varying-processing-rate-based queuing model and linear model is able to increase the accuracy of performance model. Meanwhile, automatic reference-model learning and adaptive output-error mapping increase the accuracy of output errors which decreases the percentage of SLA-violation by 8.44 percent and obtains the second lowest cost. Designing container auto-scaling algorithms considering geography distribution of Cloud, Fog and Edge resources is promising future work.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 61972202, in part by the Fundamental Research Funds for the Central Universities under Grant 30919011235, and in part by the China Scholarship Council. This work is mainly accomplished during the visit of the first author to the CLOUDS Laboratory of the University of Melbourne.

References

[1] R. Buyya and S. N. Srirama, *Fog and Edge Computing: Principles and Paradigms*. Hoboken, NJ, USA: Wiley, 2019.
[2] A. Hegde, R. Ghosh, T. Mukherjee, and V. Sharma, "SCoPe: A decision system for large scale container provisioning management," in *Proc. IEEE 9th Int. Conf. Cloud Comput.*, 2016, pp. 220–227.
[3] Kubernetes: Production-grade container orchestration, Kubernetes Web site, Kubernetes, 2020. [Online]. Available: https://kubernetes.io/
[4] KubeEdge, a kubernetes native edge computing framework, 2020. [Online]. Available: https://kubeedge.io/en/
[5] Z. Zhong and R. Buyya,"A cost-efficient container orchestration strategy in kubernetes-based cloud computing infrastructures with heterogeneous resources," *ACM Trans. Internet Technol.*, vol. 20, no. 2, pp. 1–24, 2020.
[6] M. Nardelli, C. Hochreiner, and S. Schulte, "Elastic provisioning of virtual machines for container deployment," in *Proc. 8th ACM/SPEC Int. Conf. Perform. Eng. Companion*, 2017, pp. 5–10.
[7] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, "Migration modeling and learning algorithms for containers in fog computing," *IEEE Trans. Services Comput.*, vol. 12, no. 5, pp. 712–725, Sep./Oct. 2019.
[8] U. Altaf *et al.*, "Auto-scaling a defence application across the cloud using docker and kubernetes," in *Proc. IEEE/ACM Int. Conf. Utility Cloud Comput. Companion*, 2018, pp. 327–334.
[9] S. He, L. Guo, Y. Guo, C. Wu, M. Ghanem, and R. Han, "Elastic application container: A lightweight approach for cloud resource provisioning," in *Proc. IEEE 26th Int. Conf. Adv. Inf. Netw. Appl.*, 2012, pp. 15–22.
[10] J. Jiang, J. Lu, G. Zhang, and G. Long, "Optimal cloud resource auto-scaling for web applications," in *Proc. 13th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2013, pp. 58–65.
[11] G. Huang, *et al.*, "Auto scaling virtual machines for web applications with queueing theory," in *Proc. 3rd Int. Conf. Syst. Informat.*, 2016, pp. 433–438.
[12] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani, "A hybrid reinforcement learning approach to autonomic resource allocation," in *Proc. IEEE Int. Conf. Autonomic Comput.*, 2006, pp. 65–73.
[13] J. Vilaplana, F. Solsona, I. Teixidó, J. Mateo, F. Abella, and J. Rius, "A queuing theory model for cloud computing," *J. Supercomput.*, vol. 69, no. 1, pp. 492–507, Jul. 2014.
[14] C. Lu, Y. Lu, T. F. Abdelzaher, J. A. Stankovic, and S. H. Son, "Feedback control architecture and design methodology for service delay guarantees in web servers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 9, pp. 1014–1027, Sep. 2006.
[15] W. Pan, D. Mu, H. Wu, and L. Yao, "Feedback control-based QoS guarantees in web application servers," in *Proc. 10th IEEE Int. Conf. High Perform. Comput. Commun.*, 2008, pp. 328–334.
[16] Y. Hu, G. Dai, A. Gao, and W. Pan, "A self-tuning control for web QoS," in *Proc. Int. Conf. Inf. Eng. Comput. Sci.*, 2009, pp. 1–4.
[17] L. Sha, X. Liu, Y. Lu, and T. Abdelzaher, "Queueing model based network server performance control," in *Proc. 23rd IEEE Real-Time Syst. Symp.*, 2002, pp. 81–90.
[18] C. Xu, B. Liu, and J. Wei, "Model predictive feedback control for QoS assurance in webservers," *Computer*, vol. 41, no. 3, pp. 66–72, Mar. 2008.
[19] L. Baresi, S. Guinea, A. Leva, and G. Quattrocchi, "A discrete-time feedback controller for containerized cloud applications," in *Proc. 24th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2016, pp. 217–228.
[20] A. Chung, J. W. Park, and G. R. Ganger, "Stratus: Cost-aware container scheduling in the public cloud," in *Proc. ACM Symp. Cloud Comput.*, 2018, pp. 121–134.
[21] K. Kaur, T. Dhand, N. Kumar, and S. Zeadally, "Container-as-a-service at the edge: Trade-off between energy efficiency and service availability at fog nano data centers," *IEEE Wireless Commun.*, vol. 24, no. 3, pp. 48–56, Jun. 2017.
[22] C. Liu *et al.*, "Authorized public auditing of dynamic big data storage on cloud with efficient verifiable fine-grained updates," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 9, pp. 2234–2244, Sep. 2014.

[23] C. Guerrero, I. Lera, and C. Juiz,"Genetic algorithm for multi-objective optimization of container allocation in cloud architecture," *J. Grid Comput.*, vol. 16, no. 1, pp. 113–135, 2018.

[24] G. Santos, H. Paulino, and T. Vardasca, "QoE-aware auto-scaling of heterogeneous containerized services (and its application to health services)," in *Proc. 35th Annu. ACM Symp. Appl. Comput.*, 2020, pp. 242–249.

[25] M. Abdullah, W. Iqbal, and F. Bukhari, "Containers vs virtual machines for auto-scaling multi-tier applications under dynamically increasing workloads," in *Proc. Int. Conf. Intell. Technol. Appl.*, 2018, pp. 153–167.

[26] M. Imdoukh, I. Ahmad, and M. G. Alfailakawi, "Machine learning-based auto-scaling for containerized applications," *Neural Comput. Appl.*, vol. 32, pp. 9745–9760, 2020.

[27] X. Wang, *et al.*, "An autonomic provisioning framework for outsourcing data center based on virtual appliances," *Cluster Comput.*, vol. 11, no. 3, pp. 229–245, 2008.

[28] T. Patikirikorala, A. Colman, J. Han, and L. Wang, "A multi-model framework to implement self-managing control systems for QoS management," in *Proc. 6th Int. Symp. Softw. Eng. Adaptive Self-Manag. Syst.*, 2011, pp. 218–227.

[29] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. Hoboken, NJ, USA: Wiley, 2004.

[30] Z. Cai, D. Liu, Y. Lu, and R. Buyya, "Unequal-interval based loosely coupled control method for auto-scaling heterogeneous cloud resources for web applications," *Concurrency Comput.: Practice Experience*, vol. 32, no. 23, 2020, Art. no. e5926.

[31] H. Li and S. Venugopal, "Using reinforcement learning for controlling an elastic web application hosting platform," in *Proc. 8th ACM Int. Conf. Autonomic Comput.*, 2011, pp. 205–208.

[32] E. Barrett, E. Howley, and J. Duggan,"Applying reinforcement learning towards automating resource allocation and application scalability in the cloud," *Concurrency Comput.: Practice Experience*, vol. 25, no. 12, pp. 1656–1674, 2013.

[33] X. Dutreilh, A. Moreau, J. Malenfant, N. Rivierre, and I. Truck, "From data center resource allocation to control theory and back," in *Proc. IEEE 3rd Int. Conf. Cloud Comput.*, 2010, pp. 410–417.

[34] M. Litoiu, M. Mihaescu, D. Ionescu, and B. Solomon, "Scalable adaptive web services," in *Proc. 2nd Int. Workshop Syst. Develop. SOA Environ.*, 2008, pp. 47–52.

[35] R. Tolosana-Calasanz, J. Diaz-Montes, O. F. Rana, and M. Parashar, "Feedback-control queueing theory-based resource management for streaming applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 1061–1075, Apr. 2017.

[36] NGINX ingress controller, NGINX Ingress Controller Web site, 2020. [Online]. Available: https://kubernetes.github.io/ingress-nginx/

[37] Kubernetes java client, Kubernetes Java Client Interface Web site, 2020. [Online]. Available: https://github.com/kubernetes-client/java

[38] Wikipedia access traces, WikiBench Web site, WikiBench, 2020. [Online]. Available: http://www.wikibench.eu/?page_id=60

[39] Apache JMeter: Workload generator, Apache JMeter Web site, Apache, 2020. [Online]. Available: https://jmeter.apache.org/

Zhicheng Cai (Member, IEEE) received the PhD degree in computer science and engineering from Southeast University, Nanjing, China, in 2015. He is an associate professor with the Nanjing University of Science and Technology, China. He is also a visiting scholar with the University of Melbourne from September 2019 to September 2020. His research interests focus on resource scheduling of web systems and batch tasks in cloud, fog, and edge computing. He is the author of more than 15 publications in journals such as the *IEEE Transactions on Services Computing*, the *IEEE Transactions on Cloud Computing*, the *IEEE Transactions on Automation Science and Engineering*, the *Future Generation Computer Systems*, the *Journal of Grid Computing*, the *Concurrency and Computation: Practice and Experience* and at conferences such as ICSOC, ICPADS, ISPA, ICA3PP, SMC, CBD, and CASE.

Rajkumar Buyya (Fellow, IEEE) is a Redmond Barry distinguished professor and director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, University of Melbourne, Australia. He has authored more than 625 publications and seven text books including "*Mastering Cloud Computing*". He also edited several books including "*Cloud Computing: Principles and Paradigms*". He is one of the highly cited authors in computer science and software engineering worldwide (h-index=137, g-index=304, more than 100,700 citations). Microsoft Academic Search Index ranked him as ♯1 author in the world (2005-2016) for both field rating and citations evaluations in the area of distributed and parallel computing. A Scientometric Analysis of Cloud Computing Literature by German scientists ranked him as the World's Top-Cited (♯1) author and the World's Most-Productive (♯1) author in cloud computing. He is recognized as a "Web of Science Highly Cited researcher" in 2016, 2017, and 2018 by Thomson Reuters, and Scopus researcher of the Year 2017 with Excellence in Innovative Research Award by Elsevier for his outstanding contributions to cloud computing. He is currently serving as editor-in-chief of the *Journal of Software: Practice and Experience*.