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

Zhicheng Cai, *Member, IEEE,* 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 paper, 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 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 performance model only works well when the system is near the reference point [17], [18].

The deviation between the reference point (e.g., reference response time) 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

- *Zhicheng Cai is an Associate Professor of School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. He is also a visiting scholar of the University of Melbourne, Australia from Sep 2019 to Sep 2020. (caizhicheng@njust.edu.cn)*
- *Rajkumar Buyya is a Professor and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, the University of Melbourne, Australia*

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 sampled reference models is unavoidable which makes output errors mismatch with the profiled performance model leading to fierce fluctuations in some cases.

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. 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. Section 5 and 6 include performance evaluation on a real Kubernetes cluster, conclusions and future work.

2 Related Work

Resource provisioning algorithms have been designed for Web systems in private data centers, Cloud, Fog and Edge Computing which mainly involve with allocation of application resources, PMs, VMs or containers.

2.1 Application Resource Provisioning

QoS control is one of the most important objectives of building self-adaptive computing systems [20], [21]. QoS control of traditional Web applications deployed on a single or multiple PMs usually involves application resources like Web server processes, sessions or database connections [22]. Feedback control is an essential method for application resource allocation problems. Linear-performance-model-based fixed gain [14], [15], adaptive [16] or multi-model switching [23] feedback control methods have been proposed for application resource auto-scaling.

Queuing models describe relations among the average response time, the request arrival rate and allocated resources of Web systems more accurately than linear models. Therefore, queuing models have been used to improve the performance of feedback control. For example, feedback control was used by Sha et al [17] and Xu et al. [18] to minimize residential errors produced by queuing models. Linear models, derived from queuing models which describe the linear relation of output-error changes and the adjustment of allocated resources near the reference point, are usually used to design feedback controllers. However, queuing-model-derived linear models only work well near reference points.

2.2 VM and PM Provisioning

VM placement, migration and auto-scaling are three important VM provisioning problems in Cloud, Fog and Edge Computing. The main objective of VM placement and migration is to decrease power consumption, total network latency [24] or improve resource utilization by placing or migrating a fixed number of VMs appropriately. For example, a fixed number of VMs are migrated among Fog Cloudlets to trace their mobile users [24] which only focuses on network latency affected by geography locations of VMs. Game theory is applied to deploy fixed number of VMs on Cloud and Edge hosts in terms of energy, performance and cost [25]. VM auto-scaling algorithms usually aim to minimize resource consumption by adjusting the number of allocated VMs to each application while guaranteeing QoS. Auto-scaling algorithms are the basis of VM placement and migration algorithms for improving the QoS collaboratively.

Threshold, reinforcement learning, queuing-and-control-theory-based methods have been widely used for VM auto-scaling problems. For threshold-based method, it is very tricky to set thresholds and control strength [20], [26]. Q-table and deep-neural-network based reinforcement learning methods proposed by Li et al. [27], Barrett et al. [28] and Tesauro et al. [12] have been used to allocate VMs or PMs to Web systems elastically. However, both Q-table and neural-network based methods need long training periods [26]. M/M/1 [12], M/M/N [10], [11], heterogeneous M/M/N [29] 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. Therefore, 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] based on output errors. However, this method is tailored for avoiding over-control incurred by interval-based charging models of Cloud VMs. For guaranteeing QoS of stream applications, VMs are allocated by comparing the real-time queuing length with the Little's Law determined queuing length [31]. However, the number of VMs adjusted each time is determined by experience.

2.3 Container Provisioning

Container deployment, migration and auto-scaling problems considering load balancing, power consumption or QoS are becoming increasingly common on diverse underlying resources such as Cloud VMs [6], Fog Cloudlets [7], Edge nodes [32] and private data centers [9]. The number of containers allocated to each application are fixed or variable, and the total capacity of underlying resources is fixed or elastic too.

Most existing works focus on deploying and migrating a fixed number of containers on private or elastically rented underlying resources which can be modeled as bin-packing problems [7] and solved by CPLEX [6] or heuristic methods such as best fit decreasing bin packing (BFD) and time-bin BFD [5], [33]. These bin-packing methods are usually combined with VM auto-scaling algorithms such as cost-efficiency-based greedy method [5], [33] to implement container deployment on elastically rented VMs. Reinforcement

learning is another method to deploy and migrate containers considering geography locations of Fog nodes and user mobility [7]. Game theory is also used to schedule batch tasks to a fixed number of containers and deploy these containers on Edge hosts to minimize energy consumption and SLA violations [32]. For Web systems, response times are usually not only affected by geography distances, but also the number of deployed container replicas. However, auto-scaling of containers is not considered in these works.

Container auto-scaling algorithms mainly include workload prediction based proactive methods [34], threshold [9], [35] and control theory [19] based reactive methods. Time series analysis and machine learning methods have been used to predict workloads based on which containers are auto-scaled [34]. However, proactive methods lack the ability of reacting to real-time errors and workloads are not predictable for some cases. Some other works set a threshold to each application regarding CPU, Memory or performance metrics. 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 [36]. Genetic algorithms [35] and linear programming [37] have been used to find the optimal container distribution 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. The difficulty of threshold-based methods is how to select appropriate threshold values suitable for different arrival rates [34]. An inverse proportional performance model, which is more accurate than linear models, was used to design a feedback control method by Baresi et al. [19] to allocate containers to Web applications with fixed reference point of $0.9\times SLA$.

2.4 Comparison with Existing Algorithms

Table 1 shows a comparison of our approach with existing resource provisioning algorithms. Firstly, linear or queuing-model-derived linear models used in most existing feedback control methods cannot describe the container based systems accurately [38]. 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. Secondly, most existing works use a fixed reference point obtained by experience to compute output errors directly. However, appropriate reference points are different for various request arrival rates. Therefore, our approach applies an automatic reference point identification method and an adaptive output error mapping method to improve the control stability. Finally, performance of many container auto-scaling algorithms is only evaluated on simulation platforms while our approach is implemented as a user-level scheduler of real Kubernetes-based platforms.

3 Web Systems in Kubernetes

It is flexible and inter-operable to organize diverse Web systems using micro-service-based architecture as shown in Figure 1. Each tier of such Web systems can be implemented

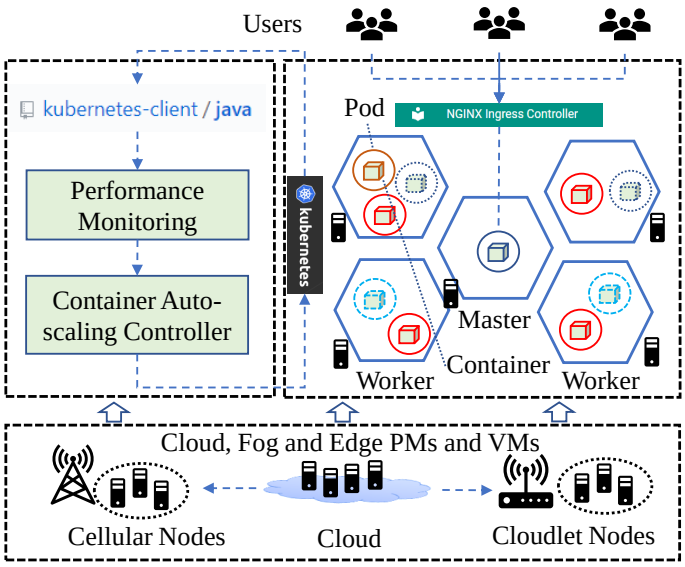

Fig. 1. Architecture of container-based Web Systems in Kubernetes

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. Since Kubernetes manages resources in the form of Pods [3], containers of micro-services are embedded in Pods and deployed on PMs or VMs of Cloud, Fog and Edge Computing which are managed by Kubernetes-based systems uniformly. It is very common that one Pod only contains one container. Pods with different colors in Figure 1 consist of containers of different micro-services. User requests of the same micro-service are distributed to containers using different balancing algorithms by nginx-ingress-controllers [41]. Real-time performance of each micro-service is monitored through Kubernetes java-client-interface [42]. Based on collected performance metrics, the container Auto-scaling Controller (ASC) is implemented as a user-level plugin to allocate appropriate number of containers (Pods) to each micro-service separately. Although geography locations of containers influence the response time either, it is assumed that containers with the same configuration on Cloud and Edge hosts have the same performance. The auto-scaling considering geography locations will be considered in future works. 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 $k\%$ 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.

4 Proposed Feedback Control Method

In this paper, an Inverse-Queuing-Model-based feedback control method (Feedback_InverseQM) is proposed. A hy-

TABLE 1
Comparison of our approach with existing resource provisioning algorithms for Web Systems

Resource Types	Problems	Objectives	Techniques	Platforms	Works
Application resources	Auto-scaling	Absolute or relative average response time	Linear-model-based feedback control	A single Web Server	[14], [15], [16], [23]
			Queuing-model-derived linear model based hybrid control	A single Web Server	[17], [18]
VMs or PMs	Deployment and migration	Power consumption, Load balancing, Migration costs, Resource utilization	Heuristics, meta-heuristics	CloudSim, iFogSim, Other simulation platforms	[24], [25], [39], [40]
	Auto-scaling	Average response time	Threshold	Private VM clusters	[26]
			Reinforcement learning	Private VM or PM clusters, Public Clouds, MATLAB	[12], [26], [27], [28]
			Queuing models	Simulation, OpenStack, Private VM clusters	[10], [11], [12], [13], [29]
			Queuing-length-based feedback control	CometCloud	[31]
			Queuing-model-arrival-rate-adjusting-coefficient based hybrid control	CloudSim, Private VM clusters	[30]
Containers	Deployment and migration, Fixed capacity	Power consumption, Total network latency	Deep Q-learning	Private VM clusters	[7]
	Deployment and migration, Elastic capacity	Deployment Cost	CPLEX, BFD, Time-bin BFD, Cost-efficiency-based greedy algorithms	Kubernetes, Simulation	[5], [6], [33]
	Auto-scaling	Resource usage, Average response time	Threshold	IaaS, Docker Swarm	[9] [36]
		Threshold distance, Load balance, Reliability, Total network latency	Threshold, Genetic algorithm	Simulation	[35]
		Average response time	Simple non-linear model based feedback control	ECOWARE	[19]
			Inverse-M/M/N-model based feedback control	Kubernetes	Our approach

brid 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

Experimental results illustrate that average processing rates of each container decreases as the request arrival rate increases in Kubernetes because of scheduling overhead. Meanwhile, the current average response times is also affected by past values. Traditional queuing models with fixed processing rates cannot describe this system accurately. Therefore, a hybrid of inverse-proportional-processing-rate-based M/M/N queuing model and linear model is proposed.

Let λ be the request arrival rate and N be the number of containers. The processing rate of each container is

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

where μ_b is the basic processing rate and c is the inverse-proportional coefficient of λ. Larger arrival rates mean smaller processing rates. According to M/M/N model, the probability of no requests in the whole system is

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

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 is

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

TABLE 2
Common Notations

Label	Description
λ	Real-time request arrival rate
μ	Studied varying processing rate which is the function of arrival rates
W_{sla}	Maximum response time of the micro-service 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	Maximum number of available containers for one microservice
L_r	Real-time queuing length
u_k	Integeral controller output
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 $m_k^{'}$
$n^r_{m_k^{'}}$	Minimum number of containers fulfilling SLA in $m_k^{'}$

The average response time y_k of control step k is set to be a weighted combination of the past value y_{k-1} and M/M/N queuing model with varing processing rates as

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

To obtain values of parameters a, μ_b and c, the least square method is applied based on historical data collected from Kubernetes platforms under $0 < a < 0.3$, $\mu_b > 0$ and $c > 0$ constraints.

4.2 Automatic Reference-Model Learning

For container-based queuing systems, the given W_{sla} might be far from response times of stable working points. Appropriate reference response times smaller than W_{sla} should be found to compute output errors which is the basis of feedback control rather than using W_{sla} directly. Because request arrival rates have a great impact on stable points, different reference points should be found for different arrival rates. An automatic reference-model learning method (ARML) is proposed to profile reference models for different arrival rates 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. For model m with reference arrival rate λ_m, a reference response time W^r_m and a lower bound of container-number n^r_m will be studied from samples. A reference-model set M is used to store studied models and new models are gradually added. 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. Otherwise, s will be added to m_k only if $m_k = m_{k-1}$. In each model, samples are stored in a hashmap $h_{m_k} =< n, bt_n >$ where 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 10 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 $i-$th (e.g., third) largest value in Y is taken as the reference response time $W^r_{m_k}$. When $y_s > W_{sla}$, n_s is used to update the lower bound of container number $n^r_{m_k}$ if $n_s < n^r_{m_k}$ or $n^r_{m_k} = null$. 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 4, the model m_k is called mature and added to a mature model set $M^{'}$.

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, $N_k = n^l_{m_k} + 1$.

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

$$e_k = W^r_{m_k^{'}} - y_k \quad (6)$$

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 Figure 2. Because it is complex to deduce an inverse-queuing function from Equation (4) directly, an exhausted search method is used to implement the inverse-queuing function to find the corresponding N_k given u_k as shown in Algorithm 2. 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

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

and the Z-transfer function of the performance model is

$$\frac{Y}{U} = \frac{1-a}{z-a} \quad (8)$$

Algorithm 1: Automatic Reference Model Learning (ARML)

input : s, λ, m_{k-1}, W_{sla}
Initialize $i \leftarrow 3$, $g \leftarrow 5/s$, $g^{'} \leftarrow 20/s$, $Y \leftarrow \emptyset$;
$m_k \leftarrow \arg\min_{m \in M}\{\lambda_d = |\lambda_m - \lambda|, \lambda_d < g\}$;
if $m_k = null$ **then**
  Create a new model m_k;
  $\lambda_{m_k} \leftarrow \lambda$, $M \leftarrow M \cup \{m_k\}$;
if $m_k = m_{k-1}$ **then**
  Store s in hashmap h_{m_k};
  foreach $n \in h^{keys}_{m_k}$ **do**
    Calculate $average(bt_n)$;
    $Y \leftarrow Y \cup \{average(bt_n)\}$;
  $W^r_m \leftarrow$ the i-th largest value in Y ;
  if $y_s > W_{sla}$ *and* ($n_s < n^r_{m_k}$ *or* $n^r_{m_k} = null$) **then**
    $n^r_{m_k} \leftarrow n_s$;
  if $n^r_{m_k} \neq null$ *and* $|h^{keys}_{m_k}| > 4$ **then**
    $M^{'} \leftarrow M^{'} \cup \{m_k\}$;
$m^{'}_k = \arg\min_{m \in M'}\{\lambda_d = |\lambda_m - \lambda|, \lambda_d < g^{'}\}$;
if $m^{'}_k = null$ **then**
  if $n^s_{m_k} - 1 > n^r_{m_k}$ *or* $n^r_{m_k} = null$ **then**
    $N_k \leftarrow n^s_{m_k} - 1$;
  else
    $N_k \leftarrow n^l_{m_k} + 1$;
  return $null$, N_k;
else
  return $m^{'}_k$, $null$;

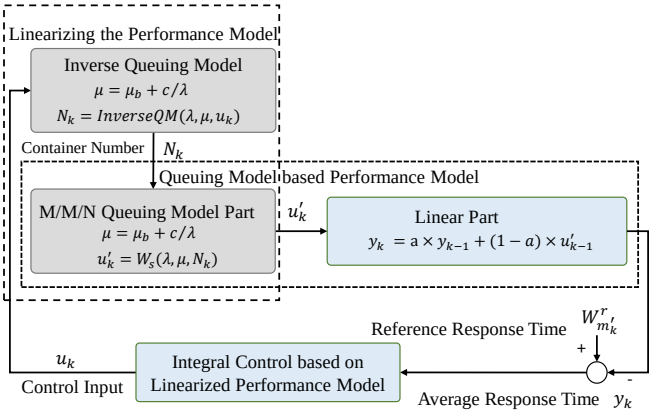

Fig. 2. Architecture of hybrid control for container-based Web Systems

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

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

Algorithm 2: Inverse Queuing Model (InverseQM)

input : λ, μ, u_k
$N_k \leftarrow \lceil \frac{\lambda}{\mu} \rceil$;
while $W_s(N_k, \lambda, \mu) > u_k$ **do**
  $N_k \leftarrow N_k + 1$;
return N_k

Z-transfer function of the whole feedback system is

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

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, u_k should be smaller than $W^u_s = W_s(n^r_{m^{'}_k}, \lambda, \mu)$ to make N_k larger than $n^r_{m^{'}_k}$. Meanwhile, u_k cannot be smaller than $W^d_s = W_s(N_m, \lambda, \mu)$ 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^u_s is set to be

$$W^u_s = \frac{W_s(n^r_{m^{'}_k}, \lambda, \mu) + W_s(n^r_{m^{'}_k} - 1, \lambda, \mu)}{2} \tag{12}$$

The output error e_k, which is calculated based on reference model $m^{'}_k$, will be used to adjust u_k of the queuing model by Equation (9). However, there are still unavoidable deviation between reference model $m^{'}_k$ and profiled performance model as shown in Figure 3. When the deviation is too large, u_k is likely to change greatly and exceed the lower or upper bounds $[W^d_s, W^u_s]$ 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 $[W^d_s, W^u_s]$, an adaptive output-error mapping method (AOM) is proposed to map e_k from the reference model space to $e^{'}_k$ in the queuing model space.

Theorem 1. If e_k is mapped to $e^{'}_k$ in proportional, there is

$$e^{'}_k = e_k \times K_a \tag{13}$$

$$K_a = \begin{cases} \frac{u_{k-1} - W^d_s}{y_k - W^d_{m^{'}_k}} & y_k > W^r_{m^{'}_k} \\ \frac{W^u_s - u_{k-1}}{W^u_{m^{'}_k} - y_k} & \text{Otherwise} \end{cases} \tag{14}$$

Proof 1. Let u^r be the reference point in queuing model space which is the corresponding map of $W^r_{m^{'}_k}$ of reference model. When $y_k > W^r_{m^{'}_k}$, $e^{'}_k = u_{k-1} - u^r$. If mapping is done in proportion, the ratio of e_k to $y_k - W^d_{m^{'}_k}$ is equal to the ratio of the output error to the maximum decreasing range of response times in the

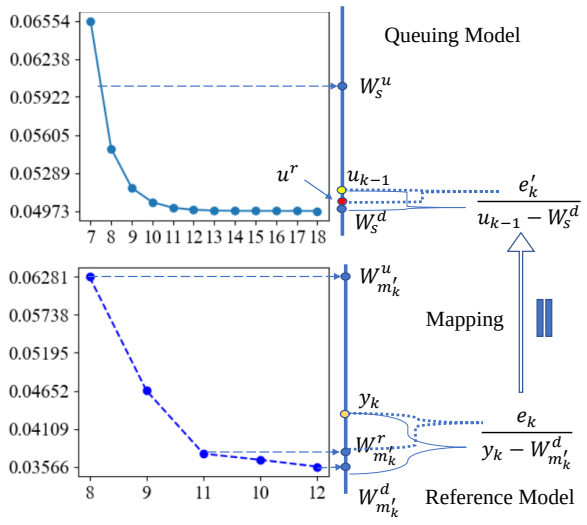

Fig. 3. Mapping from the reference model to profiled queuing model

profiled performance model, then equal to the ratio of $e_k^{'}$ to $u_{k-1} - W_s^d$ of the queuing model part as follows

$$\begin{aligned}
&\frac{e_k}{y_k - W^d_{m_k^{'}}} \\
&= \frac{(a \times y_{k-1} + (1-a)u_{k-1}) - (a \times y_{k-1} + (1-a)u^r)}{(a \times y_{k-1} + (1-a)u_{k-1}) - (a \times y_{k-1} + (1-a)W_s^d)} \\
&= \frac{(1-a)(u_{k-1} - u^r)}{(1-a)(u_{k-1} - W_s^d)} \\
&= \frac{e_k^{'}}{u_{k-1} - W_s^d}
\end{aligned} \tag{15}$$

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 substituting Equation (13) into Equation (9), 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 determine 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, i.e., queuing lengths represent the blocking degrees. Therefore, a queuing-length-based unstable state provisioning method is applied as shown in Algorithm 3. In lines 1 to 5, the minimum number of containers N fulfilling W_{sla} according to M/M/N queuing model is first obtained. Then N_k should be larger than N_{k-1} and not smaller than N in lines 6 to 8. Next N_k is increased one by one to make sure that L_r can decrease to L_s in T_r (e.g., 10) seconds by increasing $N_k - N_{k-1}$ containers. 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.

Algorithm 3: Queuing-length-based provisioning (QLP)

input : λ, μ, W_{sla}, N_{k-1}, L_r
1 $N \leftarrow \lceil \frac{\lambda}{\mu} \rceil$, $T_r \leftarrow 10$;
2 $W_s^{'} \leftarrow a \times y_{k-1} + W_s(N, \lambda, \mu) \times (1-a)$;
3 **while** $W_s^{'} > W_{sla}$ **do**
4 $\quad N \leftarrow N + 1$;
5 $\quad W_s^{'} \leftarrow a \times y_{k-1} + W_s(N, \lambda, \mu) \times (1-a)$;
6 $N_k \leftarrow N_{k-1} + 1$;
7 **if** $N_k < N$ **then**
8 $\quad N_k \leftarrow N$
9 **while** $\mu \times (N_k - N_{k-1}) \times T_r < L_r - L_s(N_k, \lambda, \mu)$ **do**
10 $\quad N_k \leftarrow N_k + 1$;
11 **if** $(N_k - 1) \times \mu > \lambda$ **then**
12 $\quad u_k \leftarrow \frac{W_s(N_k, \lambda, \mu) + W_s(N_k - 1, \lambda, \mu)}{2}$;
13 **else**
14 $\quad u_k \leftarrow W_s(N_k, \lambda, \mu) + \epsilon$
15 **return** N_k

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. Then QLP is invoked, if the real queuing length L_r is larger than α (e.g., 10) times of the theory queuing length $L_s(N_k, \lambda, \mu)$ or y_k is larger than β (e.g., 1.2) times of the largest stable response time $W_s(N, \lambda, \mu)$. 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 4: Inverse queuing model based feedback control (FeedBack_InverseQM)

input : λ, L_r, y_k
1 Initialize $\alpha \leftarrow 10$, $\beta \leftarrow 1.2$;
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 $\quad N_k \leftarrow$ QLP(λ, μ, W_{sla}, N_{k-1}, L_r);
6 **else**
7 $\quad m_k^{'}, N_k \leftarrow$ ARML(s, λ, m_{k-1}, W_{sla});
8 $\quad$ **if** $m_k^{'} \neq null$ **then**
9 $\quad\quad u_k \leftarrow$ Equation (16);
10 $\quad\quad N_k \leftarrow$ InverseQM(λ, μ, 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 existing algorithms on a real Kubernetes cluster which locates on four physical machines with the configuration of 6∼12

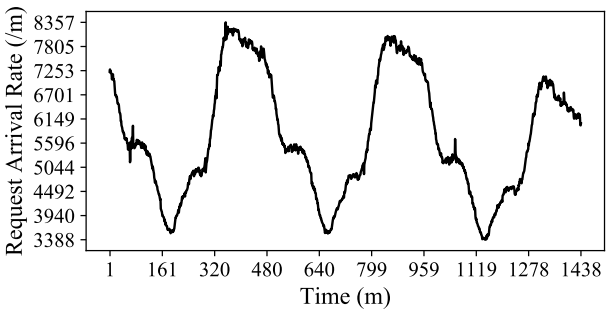

Fig. 4. Request arrival rates of applied Wikipedia access traces

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. 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 [41]. The connection timeout time is set to 10 seconds and the allowed maximum number of connections is 500 per container. The user access traces of Wikipedia [43] as shown in Figure 4 with common peaks and valleys of Web systems are used to generate requests through JMeter [44].

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 of requests obtained from logs of nginx-ingress-controllers every minute and the total consumed CUs are two adopted 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.

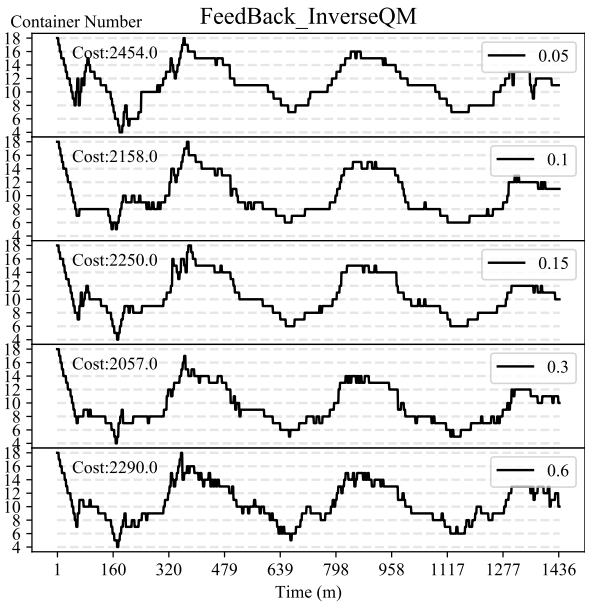

Fig. 5. Container numbers of FeedBack_InverseQM with different K_I

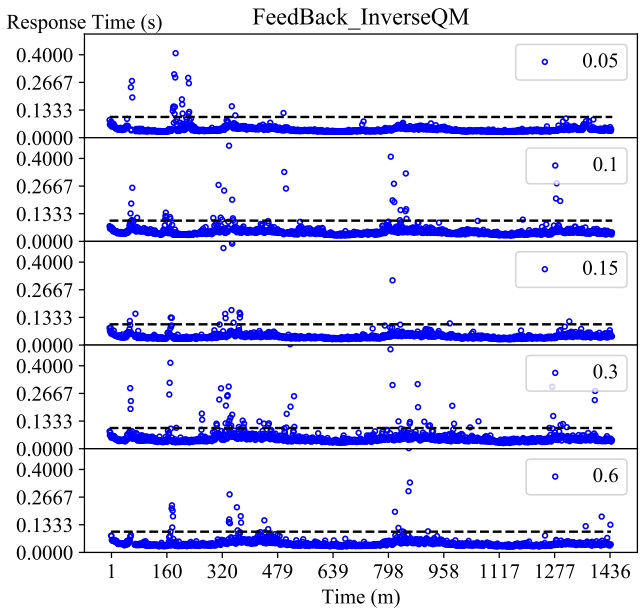

Fig. 6. Response times of FeedBack_InverseQM with different K_I

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 [22], 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. Figures 5 and 6 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.

According to the performance tuning results of [30], poles of Feedback_QMDL and Feedback_QMCA are set to

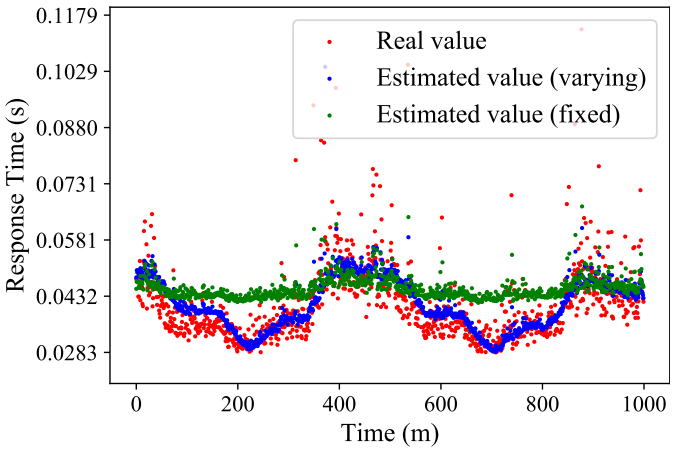

Fig. 7. Real response times and estimated values of profiled performance models using fixed and varying processing rates, respectively

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

be 0.9 and 0, respectively. The pole of Feedback_InverseP is set to be 0.95 consistent with [19]. Parameters $\mu_b = 7.771$, $c = 1574.510$ and $a = 0.215$ of the varying-processing-rate-based performance model were 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 $c = 0$. Figure 7 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).

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 with the second lowest cost (2250 CUs) in total.

Figure 8 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. Firstly, ARML of FeedBack_InverseQM is helpful to improve the accuracy of output errors. Because different arrival rates have different stable working points, samples are collected by ARML to study reference response times for diverse arrival rates leading to some fluctuations in the initial stage of Figure 8. Studied reference response times are shown in Figure 9 which increases the accuracy of output errors. Secondly, AOM of FeedBack_InverseQM is able to improve control stability. FeedBack_InverseQM is sensitive to output errors

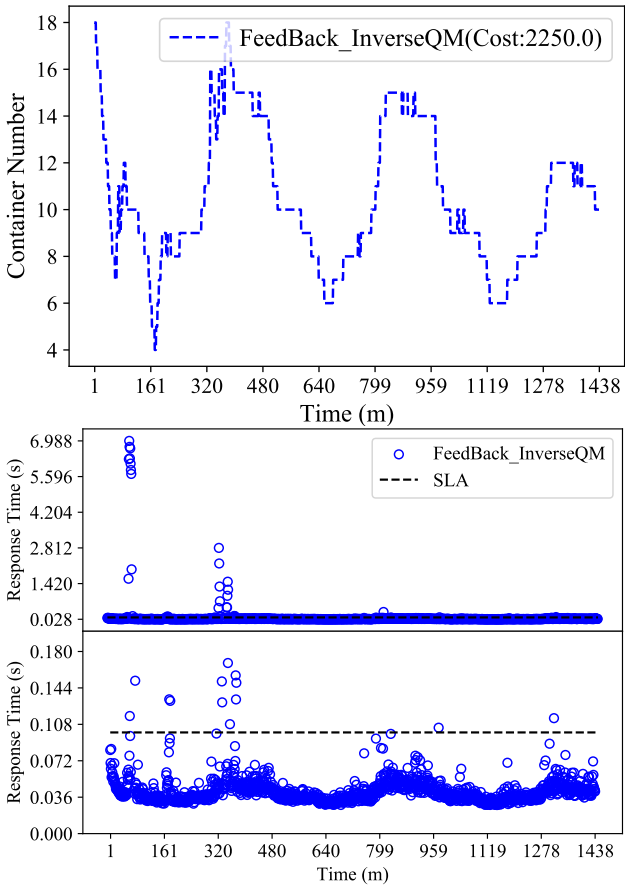

Fig. 8. Container numbers and response times of FeedBack_InverseQM

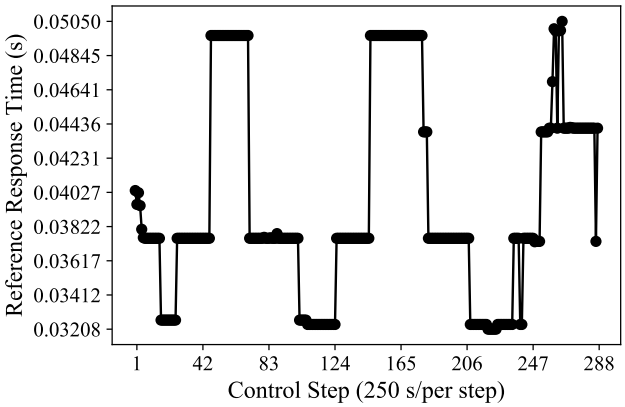

Fig. 9. Reference response times of control steps with mature models

when u_{k-1} is near relatively flatten parts of the queuing model as shown in Figure 3. 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 because of arrival rate differences and inaccuracy of queuing models. Sometimes, the original output errors generated by the reference model are so large that it makes u_k fluctuate drastically. Figure 10 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 fierce fluctuations. Both ARML and AOM make

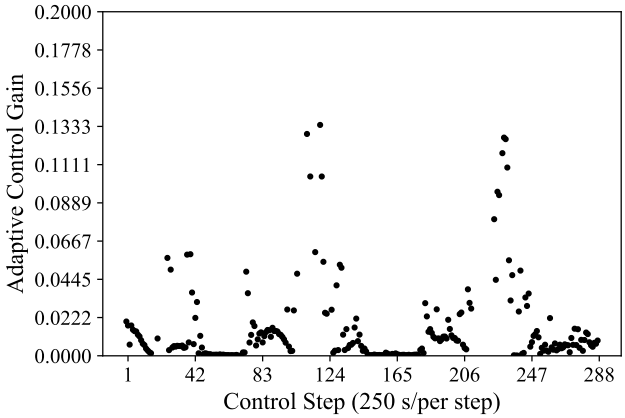

Fig. 10. Adaptive control gains ($K_I \times K_a$) of different control steps

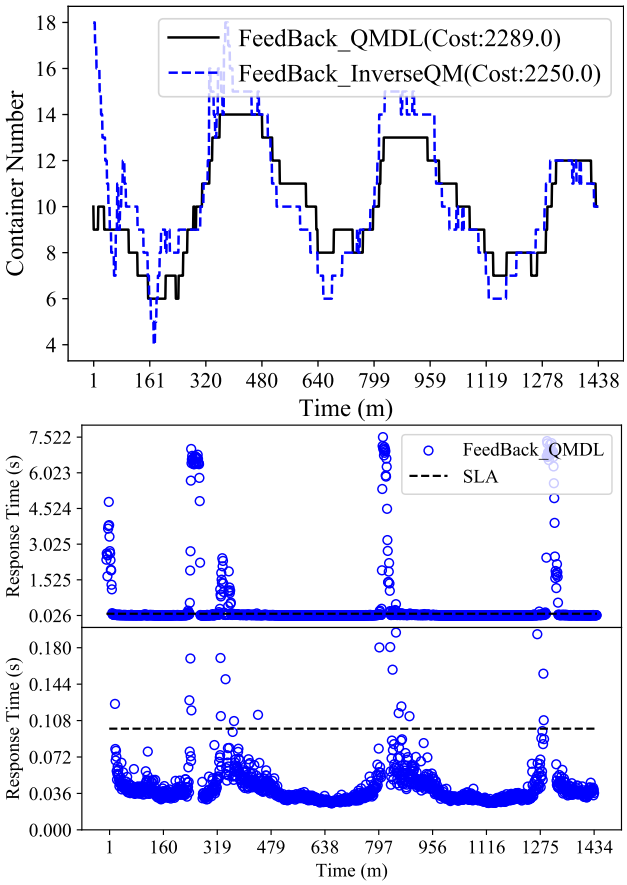

Fig. 11. Container numbers and response times of FeedBack_QMDL

FeedBack_InverseQM obtain the most stable performance as shown in Figure 8 except the initial sampling stage.

Figure 11 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 container number 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

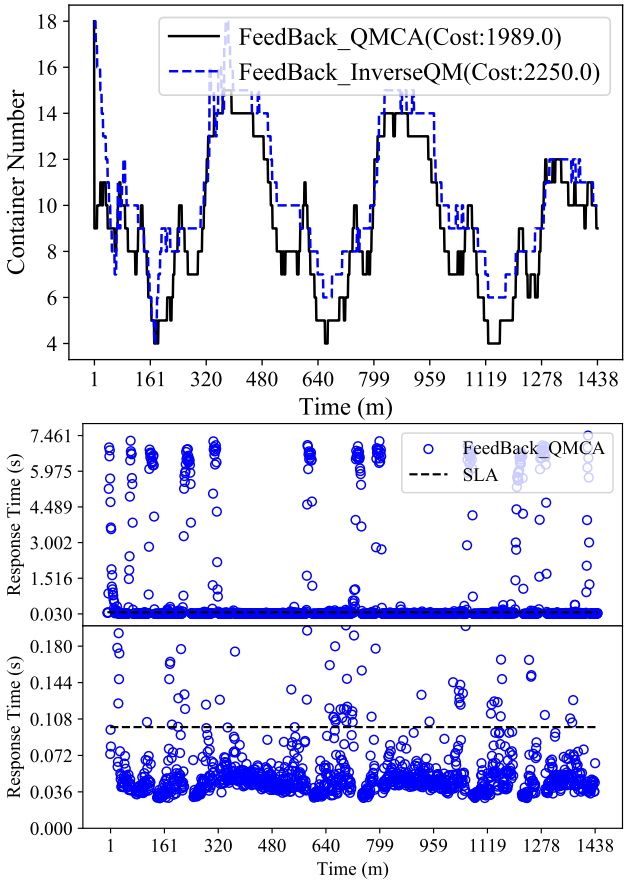

Fig. 12. Container numbers and response times of FeedBack_QMCA

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.

Figure 12 demonstrates that FeedBack_QMCA is very likely to allocate or deallocate excess containers leading to frequent large fluctuations (larger overshoots) 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.

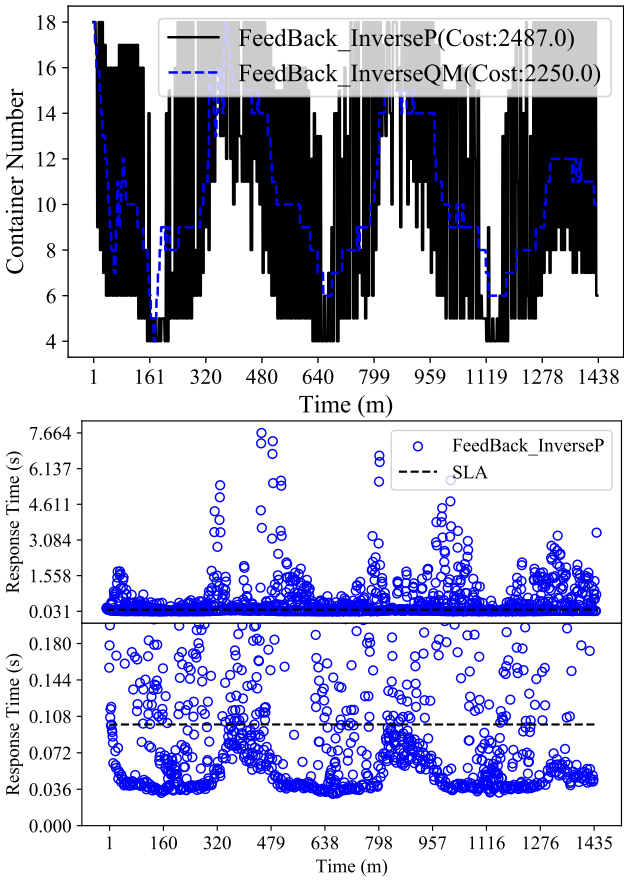

Fig. 13. Container numbers and response times of FeedBack_InverseP

Figure 13 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 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 Conclusions 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 % and obtains the second lowest cost. Designing container auto-scaling algorithms considering geography distribution of Cloud, Fog and Edge resources is promising future work.

Acknowledgements

This work is supported by the National Natural Science Foundation of China (Grant No.61972202), the Fundamental Research Funds for the Central Universities (No.30919011235) and China Scholarship Council. This work is mainly accomplished during the visiting of the first author in the CLOUDS Laboratory of the University of Melbourne.

References

[1] R. Buyya and S. N. Srirama, *Fog and edge computing: principles and paradigms*. John Wiley & Sons, 2019.

[2] A. Hegde, R. Ghosh, T. Mukherjee, and V. Sharma, "Scope: A decision system for large scale container provisioning management," in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. IEEE, 2016, pp. 220–227.

[3] "Kubernetes: Production-grade container orchestration," Kubernetes Web site, Kubernetes, 2020, https://kubernetes.io/.

[4] "KubeEdge, a kubernetes native edge computing framework," 2020, 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 Transactions on Internet Technology (TOIT)*, 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 *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*. ACM, 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 Transactions on Services Computing*, vol. 12, no. 5, pp. 712–725, 2018.

[8] U. Altaf, G. Jayaputera, J. Li, D. Marques, D. Meggyesy, S. Sarwar, S. Sharma, W. Voorsluys, R. Sinnott, A. Novak *et al.*, "Auto-scaling a defence application across the cloud using docker and kubernetes," in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion*. IEEE, 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 *2012 IEEE 26th International Conference on Advanced Information Networking and Applications*. IEEE, 2012, pp. 15–22.

[10] J. Jiang, J. Lu, G. Zhang, and G. Long, "Optimal cloud resource auto-scaling for web applications," in *13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing. IEEE, May 2013, pp. 58–65.

[11] G. Huang, S. Wang, M. Zhang, Y. Li, Z. Qian, Y. Chen, and S. Zhang, "Auto scaling virtual machines for web applications with queueing theory," in *2016 3rd International Conference on Systems and Informatics*. IEEE, Nov 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 *2006 IEEE International Conference on Autonomic Computing*. IEEE, June 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," *The Journal of Supercomputing*, vol. 69, no. 1, pp. 492–507, Jul 2014.

[14] Chenyang Lu, Ying Lu, T. F. Abdelzaher, J. A. Stankovic, and Sang Hyuk Son, "Feedback control architecture and design methodology for service delay guarantees in web servers," *IEEE Transactions on Parallel and Distributed Systems*, 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 *2008 10th IEEE International Conference on High Performance Computing and Communications*. IEEE, Sep. 2008, pp. 328–334.

[16] Y. Hu, G. Dai, A. Gao, and W. Pan, "A self-tuning control for web qos," in *2009 International Conference on Information Engineering and Computer Science*. IEEE, Dec 2009, pp. 1–4.
[17] Lui Sha, Xue Liu, Ying Lu, and T. Abdelzaher, "Queueing model based network server performance control," in *23rd IEEE Real-Time Systems Symposium*. IEEE, Dec 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, March 2008.
[19] L. Baresi, S. Guinea, A. Leva, and G. Quattrocchi, "A discrete-time feedback controller for containerized cloud applications," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2016, pp. 217–228.
[20] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, "Elasticity in cloud computing: State of the art and research challenges," *IEEE Transactions on Services Computing*, vol. 11, no. 2, pp. 430–447, March 2018.
[21] S. Shevtsov, M. Berekmeri, D. Weyns, and M. Maggio, "Control-theoretical software adaptation: A systematic literature review," *IEEE Transactions on Software Engineering*, vol. 44, no. 8, pp. 784–810, Aug 2018.
[22] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback control of computing systems*. Hoboken, NJ: John Wiley and Sons, Inc., 2004.
[23] T. Patikirikorala, A. Colman, J. Han, and L. Wang, "A multi-model framework to implement self-managing control systems for qos management," in *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 2011, pp. 218–227.
[24] L. F. Bittencourt, M. M. Lopes, I. Petri, and O. F. Rana, "Towards virtual machine migration in fog computing," in *2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*. IEEE, 2015, pp. 1–8.
[25] M. Zakarya, L. Gillam, H. Ali, I. Rahman, K. Salah, R. Khan, O. Rana, and R. Buyya, "epcaware: A game-based, energy, performance and cost efficient resource management technique for multi-access edge computing," *IEEE Transactions on Services Computing*, 2020, in press. [Online]. Available: https://doi.org/10.1109/TSC.2020.3005347
[26] X. Dutreilh, A. Moreau, J. Malenfant, N. Rivierre, and I. Truck, "From data center resource allocation to control theory and back," in *2010 IEEE 3rd International Conference on Cloud Computing*, July 2010, pp. 410–417.
[27] H. Li and S. Venugopal, "Using reinforcement learning for controlling an elastic web application hosting platform," in *Proceedings of the 8th ACM International Conference on Autonomic Computing*. ACM, 2011, pp. 205–208.
[28] E. Barrett, E. Howley, and J. Duggan, "Applying reinforcement learning towards automating resource allocation and application scalability in the cloud," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 12, pp. 1656–1674, 2013.
[29] X. Wang, Z. Du, Y. Chen, S. Li, D. Lan, G. Wang, and Y. Chen, "An autonomic provisioning framework for outsourcing data center based on virtual appliances," *Cluster Computing*, vol. 11, no. 3, pp. 229–245, 2008.
[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 and Computation: Practice and Experience*, p. e5926, July 2020, in press. [Online]. Available: https://doi.org/10.1002/cpe.5926
[31] R. Tolosana-Calasanz, J. Diaz-Montes, O. F. Rana, and M. Parashar, "Feedback-control queueing theory-based resource management for streaming applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1061–1075, April 2017.
[32] 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 communications*, vol. 24, no. 3, pp. 48–56, 2017.
[33] A. Chung, J. W. Park, and G. R. Ganger, "Stratus: Cost-aware container scheduling in the public cloud," in *Proceedings of the ACM Symposium on Cloud Computing*, 2018, pp. 121–134.
[34] M. Imdoukh, I. Ahmad, and M. G. Alfailakawi, "Machine learning-based auto-scaling for containerized applications," *Neural Computing and Applications*, vol. 32, pp. 9745–9760, 2020.
[35] C. Guerrero, I. Lera, and C. Juiz, "Genetic algorithm for multi-objective optimization of container allocation in cloud architecture," *Journal of Grid Computing*, vol. 16, no. 1, pp. 113–135, 2018.
[36] M. Abdullah, W. Iqbal, and F. Bukhari, "Containers vs virtual machines for auto-scaling multi-tier applications under dynamically increasing workloads," in *International Conference on Intelligent Technologies and Applications*. Springer, 2018, pp. 153–167.
[37] G. Santos, H. Paulino, and T. Vardasca, "Qoe-aware auto-scaling of heterogeneous containerized services (and its application to health services)," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 2020, pp. 242–249.
[38] M. Litoiu, M. Mihaescu, D. Ionescu, and B. Solomon, "Scalable adaptive web services," in *Proceedings of the 2Nd International Workshop on Systems Development in SOA Environments*. ACM, 2008, pp. 47–52.
[39] N. M. Calcavecchia, O. Biran, E. Hadad, and Y. Moatti, "Vm placement strategies for cloud scenarios," in *2012 IEEE Fifth International Conference on Cloud Computing*. IEEE, 2012, pp. 852–859.
[40] B. Xu, Z. Peng, F. Xiao, A. M. Gates, and J.-P. Yu, "Dynamic deployment of virtual machines in cloud computing using multi-objective optimization," *Soft computing*, vol. 19, no. 8, pp. 2265–2273, 2015.
[41] "Nginx ingress controller," NGINX Ingress Controller Web site, 2020, https://kubernetes.github.io/ingress-nginx/.
[42] "Kubernetes java client," Kubernetes Java Client Interface Web site, 2020, https://github.com/kubernetes-client/java.
[43] "Wikipedia access traces," WikiBench Web site, WikiBench, 2020, http://www.wikibench.eu/?page_id=60.
[44] "Apache jmeter: Workload generator," Apache JMeter Web site, Apache, 2020, https://jmeter.apache.org/.

Zhicheng Cai received his Ph.D. degree in Computer Science and Engineering from Southeast University, China, in 2015. He is an associate professor at Nanjing University of Science and Technology, China. He is also a visiting scholar of the University of Melbourne from Sep 2019 to Sep 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 *IEEE Transactions on Services Computing*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Automation Science and Engineering*, *Future Generation Computer Systems*, *journal of Grid Computing*, *Concurrency and Computation: Practice and Experience* and at conferences such as *ICSOC*, *ICPADS*, *ISPA*, *ICA3PP*, *SMC*, *CBD* and *CASE*.

Rajkumar Buyya is a Redmond Barry Distinguished Professor and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He has authored over 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, 100,700+ citations). Microsoft Academic Search Index ranked Dr. Buyya 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 Dr. Buyya 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, a Fellow of IEEE, 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 Co-Editorin-Chief of Journal of Software: Practice and Experience. www.buyya.com.