AutoScale: Adaptive QoS-Aware Container-based Cloud Applications Scheduling Framework

Yao Sun[1*], Lun Meng[2] and Yunkui Song[3]
1 School of Software Engineering, Jinling Institute of Technology, Nanjing 211169, China
2 College of public administration, Hohai university, Nanjing 210098, China
3 Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
*Corresponding author: Lun Meng (m_l_01@163.com)

Received July 6, 2018; revised August 29, 2018; accepted November 18, 2018; published June 30, 2019

Abstract

Container technologies are widely used in infrastructures to deploy and manage applications in cloud computing environment. As containers are light-weight software, the cluster of cloud applications can easily scale up or down to provide Internet-based services. Container-based applications can well deal with fluctuate workloads by dynamically adjusting physical resources. Current works of scheduling applications often construct applications' performance models with collected historical training data, but these works with static models cannot self-adjust physical resources to meet the dynamic requirements of cloud computing. Thus, we propose a self-adaptive automatic container scheduling framework AutoScale for cloud applications, which uses a feedback-based approach to adjust physical resources by extending, contracting and migrating containers. First, a queue-based performance model for cloud applications is proposed to correlate performance and workloads. Second, a fuzzy Kalman filter is used to adjust the performance model's parameters to accurately predict applications' response time. Third, extension, contraction and migration strategies based on predicted response time are designed to schedule containers at runtime. Furthermore, we have implemented a framework AutoScale with container scheduling strategies. By comparing with current approaches in an experiment environment deployed with typical applications, we observe that AutoScale has advantages in predicting response time, and scheduling containers to guarantee that response time keeps stable in fluctuant workloads.

Keywords: Kalman filter; Fuzzy logic; Cloud applications; Resource scheduling; Performance management

This work was supported by the National Key R&D Program of China (grant 2018YFC0831300), Ministry of Education of Humanities and Social Science Research (grant 17YJCZH156 and grant 15YJCZH117), the National Social Science Foundation of China (grant 16CXW027), and Fundamental Research Fund for the Central Universities (grant 2016B30314).

http://doi.org/10.3837/tiis.2019.06.003 **ISSN : 1976-7277**

1. Introduction

Nowadays, various applications are deployed on cloud computing to provide online services, and thus guaranteeing these applications' Quality of Service (QoS) is important. Isolated independent software components are composed to build cloud applications. These components often communicate with others using standard protocols, e.g., RPC, SOAP, RESTFul. Scheduling cloud applications online in the granularity of a component is necessary, because an application's various components have different requirements of physical resources. Containers (e.g., Docker) are widely adopted as cloud applications' basic service infrastructures. Stand-alone lightweight containers include required codes and resources in executable software packages, which isolate applications from their deployment environment and avoid the interferences of different applications on the same platform. So, applications deployed in containers easily run on different platforms regardless of deployment environment. Administrators can efficiently start or stop containers with low operating latency and performance overhead, so it is easy to adjust containers' resources to deal with fluctuate workloads in cloud computing. Current approaches construct an application's performance model, estimate the resource requirements of the application, and then adjust the resources of virtual machines or physical hosts. However, system administrators modeling an application's performance ought to be expert in the application. Furthermore, because the cloud computing environment (e.g., workloads, deployment) is changing overtime, setting a performance model's parameters obtained from historical monitoring data is difficult.

To address above issues, we propose a self-adaptive automatic container scheduling framework AutoScale for cloud applications, which uses a feedback-based approach to adjust physical resources by extending, contracting and migrating containers. First, a queue-based performance model for cloud applications is proposed to correlate performance and workloads. Second, a fuzzy Kalman filter is used to adjust the performance model's parameters to accurately predict applications' response time. Third, extension, contraction and migration strategies based on predicted response time are designed to schedule containers at runtime. Finally, we schedule resources by extending, contracting and migrating containers to achieve required response time. We list our contributions as follows:

- Compared with current methods using domain knowledge, AutoScale adopts a Jackson queueing network to model applications' performance automatically. AutoScale correlates performance and workloads, and then predict various applications' response time without human intervention.
- Compared with existing static performance models, AutoScale adopts a fuzzy Kalman filter to predicts response time, which adaptively adjusts applications' parameters to deal with fluctuant workloads. Thus, AutoScale is suitable for dynamic cloud computing environment.
- We have implemented a container scheduling framework AutoScale with our approach, and validate it in predicting response time and scheduling containers by conducting extensive experiments with typical applications.

We organize the rest of our paper as follows. The recent related works are analyzed and reviewed in Section 2. The Jackson queueing network-based performance model is proposed in Section 3. AutoScale to automatically extend, contract and migrate containers is designed in Section 4. We conduct extensive experiments with typical applications to validate our approach in Section 5, and conclude this paper in Section 6.

2. Related Work

Current approaches construct applications' performance models with admission control, fuzzy logic and machine learning. Lama et al. correlate resources and performance, and then estimate required resources in the constraints of QoS [8]. Cao et al. predict the maximum number of concurrent workloads in the constraint of available resources, and then deny extensive requests with admission control [9]. Cherkasova et al. construct a performance model incorporating throughput and workloads, and then only accept suitable requests in the condition of limited resources [10]. Robertsson et al. predict performance with a linear model presenting the relationship between resources and performance, and then allocate required resources to guarantee QoS [11]. Xu et al. target at virtualized environment by taking VM's performance overhead into the performance model, which adopt reinforcement learning to manage a cloud's resources [12]. Karlsson et al. model the resource requirements of an application by isolating an application from surrounding environment, and then provide resources for the application adaptively [13]. Lama et al. adopt machine learning technologies to analyze interference among VMs, and then online provision physical resources for applications [14]. Alizadeh et al. calculate a performance model's parameters according to collected historical monitoring data instances from networks, and then dynamically adjust network resources [15]. Thant et al. optimize scientific workflows in IaaS to minimize VM deployment span, cost and failure [16]. Some recent works also focus on scheduling containers. Alsched using a queue-based model defines the utility function of allocating resources to tasks, and then makes a scheduling decision based on the calculated utility [18]. Li et al. used a two-stage method to design an online scheduling mechanism and offline reconfiguration mechanism [19]. Paragon uses a multi-queueing model-based method to categorize heterogeneous resources and applications, analyzes their different resource requirements, and then calculates their utilities automatically [20]. Existing works use historical collected monitoring data to construct applications' performance models and calculate their parameters, so are not suitable for cloud computing, where models and parameters are changing with dynamic workloads. Thus, we online model applications' performance with a Jackson queueing, and then dynamically adopt a fuzzy Kalman filter to optimize the model's performance. Our performance model requiring no historical data instances rapidly converges without human experience. Our approach efficiently schedules physical resources for applications deployed in containers to achieve desired response time, which can well deal with sudden workloads.

3. Resource Provision Approach

Response time is often adopted to measure the QoS of a cloud application. Since an application utilizes allocated physical resources to process requests, the resources and workloads of an application decide the response time. In this section, a Jackson queueing network is adopted as a performance model to correlate response time with workloads. Then, a Kalman filter is proposed to estimate the constructed performance model's parameters. Furthermore, to improve prediction accuracy, a fuzzy logic method is adopted to online adjust the parameters of the performance model. Finally, to achieve desired response time, this section proposes scheduling strategies based on predicted response time to extend, contract and migrate containers at runtime.

3.1 Application Performance Model

An application's performance model correlating response time with workloads is used to predict response time. The procedure that a cloud application processes requests can be characterized as a Jackson queueing network as follows [1]:

- An open-loop Jackson queueing network represents a cloud application with correlated application components;
- The independent nodes of a Jackson queueing network represent the application components of a cloud application;
- The edges of a Jackson queueing network represent data transmission with a message bus among application components;
- A node representing an application component processes a request event; the processed event is sent to the next node representing an application component; the event leaving the network represents a response sent back.
-

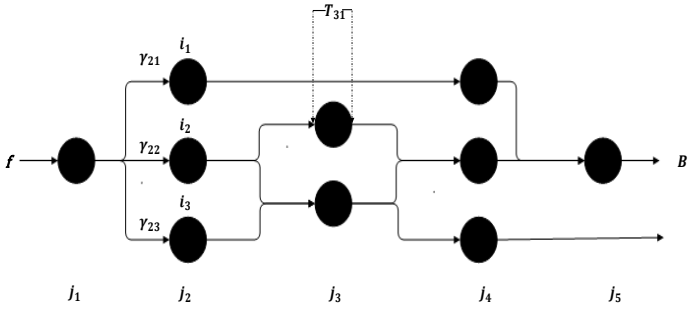


Fig. 1. Cloud applications' queuing network model

The workflow of processing requests is described as follows: a cloud application accepts a request from a customer; the cloud application capsules the arrival request as an event; many components of the cloud application process the event sequentially; the cloud application sends back a response event to the customer. A cloud application distributes requests to many instances of an application component with a round robin strategy. **Fig. 1** shows that a Jackson queueing network models a component-based cloud application. In the figure, f represents the workflow of processing a request sent from a customer; j_i represents application component j's i^{th} component instance; application components j_1, j_2, …, j_n process a request sequentially, where n is the number of application components; application component instance j_k ($1 \leq k \leq m$) represents component j's k^{th} instance, where m is the number of component j's instances.

According to the Jackson queueing network-based performance model, this paper predicts an application's response time in the condition of allocated physical resources. Application component j's resource preference (rp_j) presents the required intensive physical resource (e.g., disk I/O, CPU). First, rp_j's resource utilization is calculated as:

$$u_j = \sum_i (u_{0j} + \tau_j \times \gamma_{ji} \times T_{ji}) \ , \ (0 \leq u_j \leq 1) \tag{1}$$

where r_{ji} is application component instance j_i's request number per second, and the concurrency number varies in Poisson distribution; T_{ji} is j_i's execution time; τ_j is a correlation factor between resource utilization and request concurrency; u_{0j} is rp_j's idle resource usage, when an application component accepting no request is idle. The execution time of processing a request flow f is calculated as:

$$B = d + \sum_j \frac{T_j}{1\text{-}u_j} \quad , \tag{2}$$

where T_j is component j's execution time, and d is request flow f's transmission time.

An application's u_{0j} and r_{ji} are obtained by collecting monitoring data instances, and τ_j is easily estimated with domain knowledge, but parameters T_{ji} and d are difficult to be calculated.

3.2 Response Time Prediction

T_{ji} and d in formula (2) is necessary to predict response time, and thus this subsection introduce the method of calculating them. A Kalman filter is a linear forecasting method to predict the future value based on previous values, which recursively rectifies the prediction model by comparing the current monitored value with predicted value based on the prediction model [2]. Since a Kalman filter efficiently online evolves the prediction model without significant performance overhead, this paper adopts it to deal with fluctuant workloads in cloud computing. Thus, T_{ji} and d are predicted with a Kalman filter as follows:

$$X_{k+1} = AX_k + W_k, \tag{3}$$

$$Z_k = H_k X_k + V_k, \tag{4}$$

where $Z_k = (T_j, d)^T \forall j$ is an observed matrix recording the monitored execution time of components; X_k is a predicted matrix recording the predicted execution time of components; $H_k = (u_j, u_{0j}, \gamma_{ji}, B)^T \forall i$ records response time, resources usage and concurrent requests; $W_k \sim N(0,W)$ fitting for Gaussian distribution is a white noise excitation covariance matrix; $V_k \sim N(0,V)$ fitting for Gaussian distribution is a white noise measurement covariance matrix [3]. This paper online adjusts W_k and V_k, which change with deployment as follows:

$$W_k = TW, \tag{5}$$

$$V_k = UV, \tag{6}$$

where T and U are adjusting factors.

The Kalman filter model adjusts the predicted matrix as follows:

(1) Update X initialized with $w_{k-1} = 0$:

$$\hat{X}_k^- = A\hat{X}_{k-1}; \tag{7}$$

(2) Update covariance matrix P_k^-:

$$P_k^- = AP_{k-1}A^T + W_k; \tag{8}$$

(3) Calculate Kalman gain:

$$K_k = P_k^- H_k^T \left(H_k P_k^- H_k^T + V_k\right)^{-1}; \tag{9}$$

(4) Rectify X:

$$\hat{X}_k = \hat{X}_k^- + K_k\left(z_k - H_k\hat{X}_k^-\right); \tag{10}$$

(5) Rectify P_k:

$$P_k = (I - K_k H_k)P_k^-. \tag{11}$$

The predicted value is iteratively rectified with a Kalman filter that only utilizes the current predicted value and the current monitored value in each iteration regardless of historical data instances.

3.3 Performance Model Adjustment

The above response time prediction based on Kalman filter has the following issues:

(1) The definition of a state transformation matrix has effect on the precision of predicting response time. However, it is difficult to define a precise transformation matrix, because an application component's workloads vary in irregular nonlinearity;

(2) A typical Kalman filter sets each monitored data instance with the same weight, so cannot train and update a precise model with increasing monitored values, which is not suitable for dynamic workloads in cloud computing environment;

(3) Feedback-based methods are required to online adjust the excitation matrix and the noise matrix of the typical Kalman filter to process irregular changing values.

To solve the above problems of the typical Kalman filter, this paper decreases the white noise residual error to online adjust the Kalman filter's parameters. Formula (3) and (4) adopt fuzzy logic to rectify the Kalman filter by adjusting matrix *W* and *V*. The residual error means the degree that the monitored value deviates from the predicted value:

$$r = Z(k) - \overline{Z}(k), \tag{12}$$

When the residual error's variance exceeds the set threshold, the performance model is online rectified. This paper calculates the following residual error variance:

$$P(r) = A(H_k P_k H_k^T + W)H_k^T + V, \tag{13}$$

T and *U* in formula (5) and (6) are adjusted with a fuzzy logic function combining fuzzy rules and a subordinating degree function. This paper adopts a Sugeno fuzzy logic model [17] (i.e., TS) that are effective in nonlinearity to adjust the Kalman filter's parameters. This paper defines the following fuzzy logic rules R^i:

$$R^i: if\ x_1\ is\ A_1^i\ and\ x_2\ is\ A_2^i\ then\ u_i = p_1^i x_1 + p_2^i x_2 + r^i\ 1 \le i \le n \tag{14}$$

where the i^{th} fuzzy logic rule is R^i; the j^{th} 's input is x_j; the i^{th} rule's output is u_i; x_j's weight is p_j^i; r^i is a constant; *n* is the number of fuzzy logic rules.

x_j is an input of rule R_i, u_i is the output of rule R_i, and then we calculate the overall result as:

$$U = \sum_i^m u_i w_i \tag{15}$$

where w_i is u_i's weight, *m* is the number of fuzzy logic rules.

We adopt formula (12) to calculate residual errors' mean and variance, where *n* is the number of data instances in a period.

$$\text{avg} = \frac{1}{n}\sum r_i \tag{16}$$

$$\text{cov} = \frac{1}{n}\sum r_i r_i^T \tag{17}$$

We calculate monitored residual errors' variance with formula (17) and predict residual errors' variance with formula (13), and then compare them. When the monitored mean is much bigger than zero, and the monitored variance is much bigger than the predicted variance, we adjust the noise matrix to improve the Kalman filter's precision. To construct a fuzzy logic model, we input residual errors' mean and variance, and then output noise matrixes *W* and *V*'s parameters *U* and *T*. **Table 1** shows the following defined fuzzy logic rules as follows:

(1) when the status is "Z", U and T keep stable;

(2) when the status is "S", U decreases and T increases;

(3) when the status is "L", U increases and T decreases;

(4) when the status is "M", both U and T increase.

Many optimal linearity equations are designed with the data instances obtained from a series of experiments. Two examples are listed as follows:

$$T = P(r) \times 0.3 + 0.8,\ \ U = -P(r) \times 0.2 + 1.9, \tag{18}$$

where residual errors' mean is equal to "0" and residual errors' variance is low.

$$T = -P(r) \times 0.5 + 0.6, \quad U = P(r) \times 0.1 + 1.4, \tag{19}$$

where residual errors' mean is low and residual errors' variance is high.

To improve the Kalman filter's precision, we adjust the Kalman filter's parameters with many set rules trained with collected data instances from extensive experiments. Furthermore, various scenarios and applications can also adopt our approach with generated specific rules.

Table 1. Rules of fuzzy logic

		Residual Errors' Mean		
		0	Low	High
Residual Error's Variance	0	**S**	**Z**	**Z**
	Low	**S**	**Z**	**L**
	High	**L**	**L**	**M**

4. AutoScale: Design and Implementation

4.1 Container Scheduling Strategy

A container scheduling strategy is proposed to guarantee cloud applications' desired response time in this subsection. The detailed container scheduling algorithm is described in **Table 2**. We initiate a Jackson Network Queueing model (JNQ) to predict response time under specific workloads (Line 1); initiate an Extended Kalman Filter (EKF) with parameters that are the maximum resource utilization of a host, the minimum resource utilization of a host, and the application's maximum response time (Line 2); initialize an Fuzzy Logic Adaptive Controller (FLAC) to adjust EKF model's parameters (Line 3); define migration actions to migrate containers, an expansion action function to expand containers, and a contraction action function to contract containers (Line 4). Then, we online adjust the performance model and predict response time in period (Line 5) as follows: adjusting the parameters of EKF with FLAC (Line 6); adjusting the parameters of the JNQ with EKF (Line 7); using the JNQ to predict response time (Line 8). The scheduling strategies of migration, expansion and contraction are described as follows:

- Migration (Line 9): the scheduler stores a source container with the state of an application as an image, shuts down the source container and releases its resources, allocates resources for a container in a target physical machine with sufficient resources and starts the container image, when a container's resource usage is less than the pre-defined upper threshold.
- Expansion (Line 10): the scheduler allocates physical resources for a container, and then starts it in a chosen target physical machine, when a container's resource usage is more than the pre-defined upper threshold.
- Contraction (Line 11): the scheduler stops and releases some instances to reduce the number of instances, when an application's many instances' resource utilization is much lower than the predefined expected value.

Table 2. Container Scheduling Algorithm

Container Scheduling Algorithm
Input: Maximum resource utilization (*MaxR*), Minimum resource utilization (*MinR*), Response time threshold (*RT*)
Output: Actions of migrating, expanding and contracting containers
Initiate Jackson network queueing model: JNQ ()
Initiate an extended Kalman filter: EKF(*X*(*0*), *Z*(*0*));
Initiate a fuzzy logic adaptive controller: FLAC(*P(r)*);
Define scheduling function Migrate (), Expand (), Contract ();
While (a period)
{
Update the parameters of EKF with FLAC in formula (18) and (19): (*U*, *T*) = FLAC(*P*(*r*));
Update the parameters of JNQ with EKF in formula (3) and (4): EKF(*X(i)*, *Z(i)*);
Predict response time with JNQ in formula (1) and (2): *B* = *JNQ ()*;
If $\sum u_i$ > *MaxR* && B < *RT*:
Then Migrate ();
If B > *RT*:
Then Expand ();
If u_i < MinR:
Then Contract ().
}

4.2. Framework Implementation

As shown in **Fig. 2**, the workflow of our approach is described as follows: a user chooses or customizes a container deployment template; the application deployer verifies the template and sets a configuration for the container and application; the collector monitors each container with an agent deployed in each slave; the scheduler makes a scheduling plan based on collected monitoring data; the executor deploys container instances with applications in suitable slaves. The application deployer analyzes the correlations between components with automatic deployment tools (e.g., Puppet); records the analyzed results in a configuration file that decides the components' dependencies and boot sequence; initializes a Jackson network queueing to model applications. The container scheduler deploys agents in container instances to collect the monitoring data of slave and containers, e.g., response time, resource utilization. According to monitored data instances, the scheduler adjusts the Jackson network queueing model with a Kalman filter and a fuzzy logic controller. The scheduler guarantees the performance (i.e., response time) of cloud applications by migrating, extending or contracting containers. Containers without allocating and restricting resources in advance can dynamically apply and release resources on demand at runtime. Thus, containers can improve the resource utilization of their located host. However, they perhaps have the serious problem of resource competition. For example, when the memory utilization of many containers in a host suddenly increase simultaneously, the failure of "Out of Memory Kill" happens, and then some containers are forcibly ceased. So, we restrict the resource utilization of every container to grantee the lower limits of a container's resources.

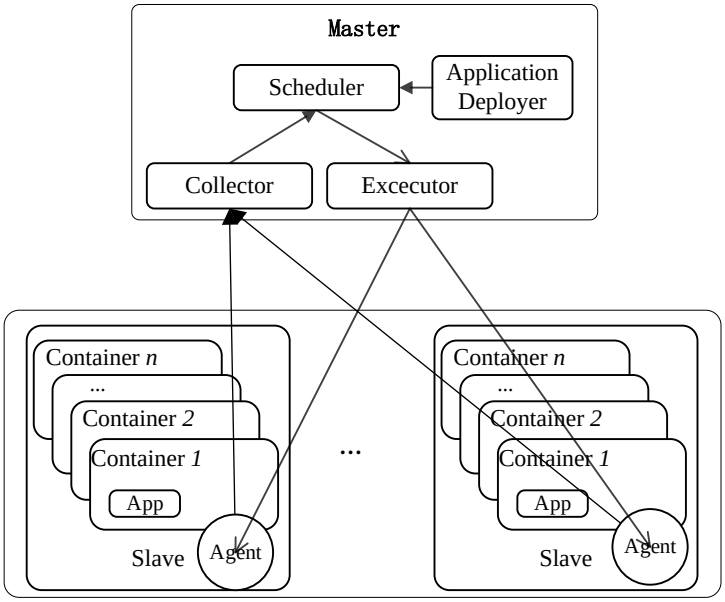


Fig. 2. Container Scheduling Framework

5. Experiment

5.1 Deployment Environment

Fig. 3 shows our experimental environment deployed with the cluster of eight hosts; a gigabit ethernet network connects these hosts to construct a cluster; each cloud application is deployed in a container with CenOS 7 and Docker 1.7; each host with Intel Core i7 CPU 3.4GHz and an 8G memory deploys many applications in dockers. The deployment environment includes JMeter deployed on a workload generator node, Nginx deployed on a load balancer node, a Master node to manage four Slaver nodes, and MySQL deployed on a database node. The cloud application Web Serving of a cloud-based application benchmark suite Cloudsuite[1] is deployed on four slaver nodes.

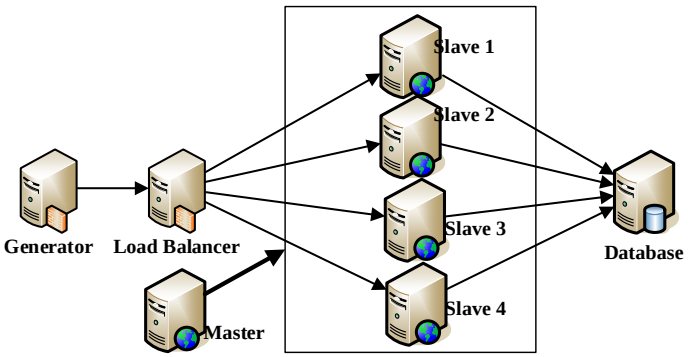


Fig. 3. Deployment environment

[1] http://cloudsuite.ch//pages/benchmarks/webserving/

5.2 Predicting Response Time

5.2.1 Simulating Workloads with Trends

This subsection simulates workloads with increasing concurrency, and then monitors response time in period. The response time corresponding to concurrent requests is predicted. Our approach is compared with existing typical methods in predicting response time to validate the effect. The compared existing methods are introduced and validated as follows.

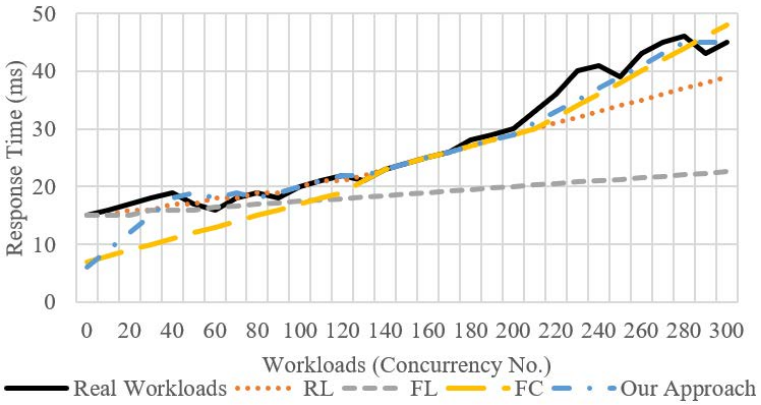


Fig. 4. Comparison in workloads with trends

(1) RL: Reinforcement Learning

A reinforcement learning method is adopted to construct a performance model and adjust the parameters of the model [4]. **Fig. 4** shows the results of experiments in fluctuate workloads, where the performance model slowly converges during from the 150th second to the 200th second and from the 230th second to the 270th second. Furthermore, the reinforcement learning method requires a training dataset with a large scale of data instances, so the method cannot be applied in fluctuant workloads.

(2) FL: Fuzzy Logic

A neural fuzzy controller self-constructs its performance model's structure, and online adjusts the parameters with online learning, which is designed to predict response time [5]. Since the parameters of a fuzzy controller have significant effect on a performance model's accuracy and these parameters vary with workloads, it is difficult to set suitable parameters and online adjust these parameters, which requires domain knowledge and continuous analysis. **Fig. 4** shows that the monitored response time significantly deviates the predicted response time during from the 50th second to the 100th second and from the 150th to the 300th second. It is because that this method does not online adjust parameters to adapt to changing workloads.

(3) FC: Feedback Control

A feedback-based control technology is adopted to design feedback-based loops [6]. This method can improve running applications' stability, and train operating rules with low computation complexity. **Fig. 4** shows that the method cannot accurately predict response time at beginning, because a long initialization period is required to construct a feedback controller.

(4) AutoScale: Our Approach

Fig. 4 shows that the error rate of our approach is less than 5%, and the errors of our approach always occur at beginning, because our approach ought to train the performance model in the initialization period. Note that our approach achieves less than 0.5% error rate during from the 200th second to the 250th second, when the simulated workloads significantly

increase. Thus, from the experimental results, we can see that our approach predicts response time with a high accuracy in fluctuant workloads.

5.2.2 Simulating Workloads with Periodicity

Periodic workloads are simulated in a time-of-day pattern, and then we compare our approach with RAMA [7] applied for periodic workloads in predicting response time. **Fig. 5** and **Fig. 6** show that the error rate of our approach is lower than that of RAMA at beginning, and then the error rate of our approach decreases after initialization. The error rate of RAMA is about above 10%, but that of our approach is about 5%. The error rate of our approach is much lower than that of RAMA in fluctuant workloads, because RAMA is not suitable for dynamic workloads. Thus, the accuracy of our approach in predicting response time is much better than that of RAMA in periodic workloads. To reduce so many errors at beginning, our approach decreases the sampling frequency to increase the size of the training dataset during initialization.

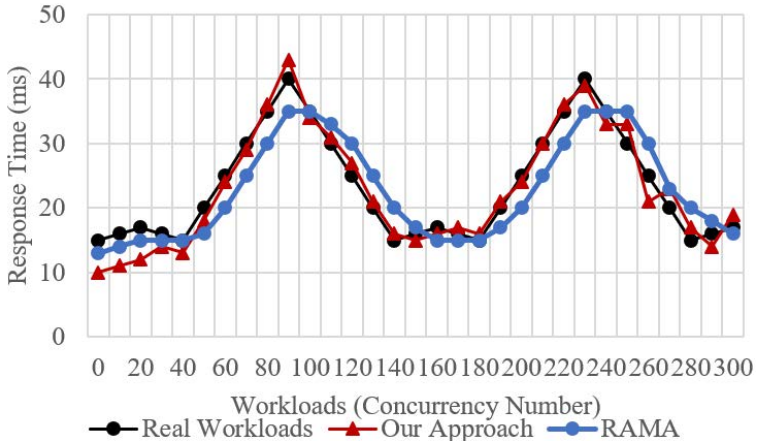


Fig. 5. Comparison in workloads with periodicity

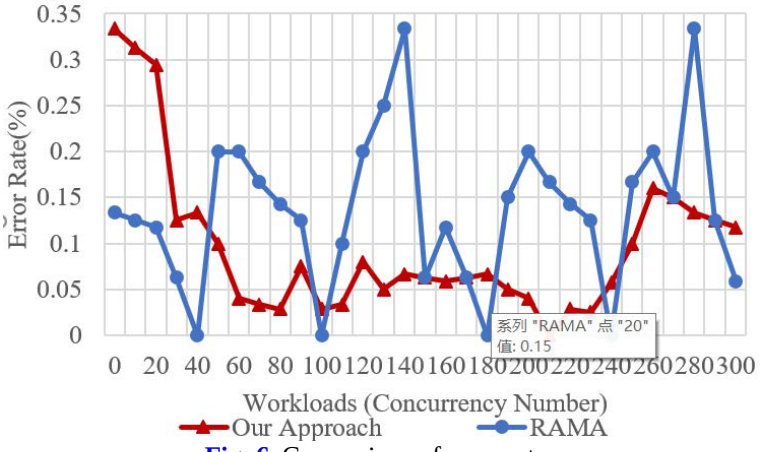


Fig. 6. Comparison of error rates

5.3 Scheduling Effect

JMeter simulates cyclical workloads, stable workloads, rising workloads and decreasing workloads sequentially. We compare our approach with a typical admission control-based scheduling method [10] in the effect of scheduling resources. The admission controller cannot

adapt to fluctuant workloads, because it takes a long period to analyze suitable parameters. Six services are deployed on node Slave 1 and six services are deployed on node Slave 2 in the initialization period. **Fig. 7** shows the experimental results, which are analyzed as follows:
(1) Stable workloads (0-200th sec.): when 30 concurrent requests per second are simulated, the Kalman filter is initialized;
(2) Dynamic workloads (200-300th sec.): when 30-60 concurrent requests per second are simulated, the response time keeps in a narrow scope as three services on Slave 1 are migrated to Slave 3 and Slave 4.
(3) Stable workloads (300-400th sec.): when 50 concurrent requests per second are simulated, the response time keeps stable.
(4) Rising workloads (400-700th sec.): when 50-100 concurrent requests per second are simulated, the response time decreases as services on Slave 1 and Slave 2 are expanded to other nodes.
(5) Decreasing workloads (700-800th sec.): when 100-30 concurrent requests per second are simulated, some services are contracted to fewer nodes.

The response time of our approach keeps stable in about 35 micro seconds, so our approach is much better than admission control-based scheduling method in dealing with fluctuant workloads.

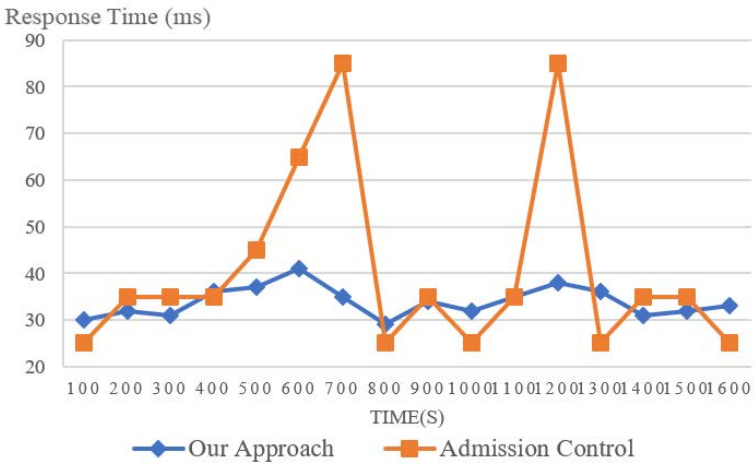


Fig. 7. Comparison of scheduling effect

6. Conclusion

Containers are widely adopted as infrastructures to support cloud applications providing Internet-based services. We propose a container-based scheduling approach to guarantee cloud applications' performance in cloud computing environment with dynamic workloads. A performance model based on a Jackson queueing network is proposed to correlate performance and workloads. A fuzzy Kalman filter is adopted to adjust the performance model's parameters to accurately predict applications' response time. According to the constructed model and calculated parameters, we extend, contract and migrate containers to achieve desired response time at runtime. Furthermore, this paper has implemented AutoScale with scheduling strategies, deployed typical applications and validated it by simulating fluctuant workloads. By comparing with current works, AutoScale achieves much more

accurate results in predicting performance, and is effective in scheduling containers to guarantee that the response time keeps stable under fluctuant workloads.

References

[1] Shanthikumar J G, Buzacott J A., "Open queueing network models of dynamic job shops," *International Journal of Production Research*, 19(3): 255-266, 1981. Article (CrossRef Link)

[2] Kalman R E., "A New Approach to Linear Filtering and Prediction Problems," *Transaction of the ASME-Journal of Basic Engineering*, 82:35 -45, 1960. Article (CrossRef Link)

[3] Sinopoli B, Schenato L, Franceschetti M, et al., "Kalman filtering with intermittent observations," *IEEE Transactions on Automatic Control*, 49(9): 1453-1464, 2004. Article(CrossRef Link)

[4] Martinez J F, Ipek E., "Dynamic multicore resource management: A machine learning approach," *IEEE Micro*, 29(5): 8-17, 2009. Article (CrossRef Link)

[5] Lama P, Zhou X., "Autonomic provisioning with self-adaptive neural fuzzy control for end-to-end delay guarantee," *IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, 151-160, 2010. Article (CrossRef Link)

[6] Chengyang Lu, Lu Y, Abdelzaher T F, et al., "Feedback control architecture and design methodology for service delay guarantees in web servers," *IEEE Transactions on Parallel and Distributed Systems*, 17(9): 1014-1027, 2006. Article (CrossRef Link)

[7] Lama P, Guo Y, Zhou X., "Autonomic performance and power control for co-located Web applications on virtualized servers," *IEEE/ACM 21st International Symposium on Quality of Service (IWQoS)*, 1-10, 2013. Article (CrossRef Link)

[8] Lama P, Zhou X., "Efficient server provisioning with control for end-to-end response time guarantee on multitier clusters," *IEEE Transactions on Parallel and Distributed Systems*, 23(1): 78-86, 2012. Article (CrossRef Link)

[9] Cao J, Zhang W, Tan W., "Dynamic control of data streaming and processing in a virtualized environment," *IEEE Transactions on Automation Science and Engineering*, 9(2): 365-376, 2012. Article (CrossRef Link)

[10] Cherkasova L, Phaal P., "Session-based admission control: A mechanism for peak load management of commercial web sites," *IEEE Transactions on Computers*, 51(6): 669-685, 2002. Article (CrossRef Link)

[11] Robertsson A, Wittenmark B, Kihl M, et al., "Design and evaluation of load control in web server systems," in *Proc. of IEEE American Control Conference*, 3: 1980-1985, 2004. Article (CrossRef Link)

[12] Xu C Z, Rao J, Bu X., "URL: A unified reinforcement learning approach for autonomic cloud management," *Journal of Parallel and Distributed Computing*, 72(2): 95-105, 2012. Article (CrossRef Link)

[13] Karlsson M, Karamanolis C, Zhu X., "Triage: Performance isolation and differentiation for storage systems," *IEEE International Workshop on Quality of Service*, IWQOS 2004: 67-74, 2004. Article (CrossRef Link)

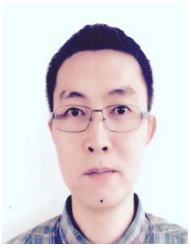
[14] Lama P, Zhou X., "Autonomic provisioning with self-adaptive neural fuzzy control for percentile-based delay guarantee," *ACM Transactions on Autonomous and Adaptive Systems*, 8(2): 9, 2013. Article (CrossRef Link)

[15] Alizadeh M., "Fast and Smart Network Resource Management for Datacenters and Beyond," in *Proc. of the 13th International Conference on emerging Networking EXperiments and Technologies*, 12-21, 2017. Article (CrossRef Link)

[16] Takagi T, Sugeno M., "Fuzzy identification of systems and its applications to modeling and control," *IEEE Transactions on Systems, Man and Cybernetics*, (1): 116-132, 1985. Article (CrossRef Link)
[17] Adriyendi, "Fuzzy Logic using Tsukamoto Model and Sugeno Model on Prediction Cost," *International Journal of Intelligent Systems & Applications*, 6(2), 2018. Article (CrossRef Link)
[18] Tumano VA, Cipar J, Kozuch MA, Ganger GR., "Alsched: Algebraic scheduling of mixed workloads in heterogeneous clouds," in *Proc. of the 3rd ACM Symp. on Cloud Computing*, ACM Press, 2012. Article (CrossRef Link)
[19] Li Z, Zhang Y, Zhao Y, Peng Y, Li D., "Best Effort Task Scheduling for Data Parallel Jobs," in *Proc. of ACM SIGCOMM Conference*, ACM Press, 555-556, 2016. Article (CrossRef Link)
[20] Delimitrou C, Kozyrakis C., "Paragon: QoS-Aware scheduling for heterogeneous datacenters," *ACM Transactions on Computer Systems*, 31(4): 77-88, 2013. Article (CrossRef Link)

Yao Sun is a lecturer in the Jinling Institute of Technology in China. He received the PhD degree in Computer Software and Theory from the Institute of Software, Chinese Academy of Sciences in 2016, and MS degree in Computer Architecture from Northeast Normal University of China in 2009. His research interests include data engineering, distributed systems, and intelligent systems.

Lun Meng is a lecturer in the Hohai University in China. He received the PhD degree in Tsinghua University in 2013, and MS degree in Computer Architecture in Northeast Normal University of China in 2009. His research interests include public opinion analysis, news communication, and autonomic computing for cloud computing systems.



Yunkui Song is an assistant professor in the Institute of Software, Chinese Academy of Sciences in China. He received the MS degree in Computer Software and Theory from the Institute of Software Chinese, Academy of Sciences in 2009. His research interests include application performance management, software reliability, and autonomic computing for cloud computing systems.