Availability optimization of consistency and availability-based micro-service systems through elastic scheduling of container resources

Qinghui Ren, Shenglin Li*, Bo Song, Chen Chen

Department of Information Engineering, Army Logistics University of PLA, Chongqing 401331, China

1936423516@qq.com

ABSTRACT. *The availability of consistency (C) and availability (A)-based micro-service systems is low when both consistency and partition tolerance (P) are satisfied. Considering the low resource occupation and fast supply of containers, this paper puts forward an approach to optimize the availability of CP micro-service systems based on the elastic scheduling of container resources, and sets up a prediction model of response time using the cascade queuing system. Then, the author determined whether to relax, restrict or maintain the container resource in light of the conformity of the response time. Finally, the proposed optimization approach was verified through experiments. The results show that a 2~3s-long adaptation period is needed for the approach under abrupt load changes, and the response time can be accurately predicted to ensure the system availability in the other cases.*

RÉSUMÉ. *La disponibilité des systèmes de micro-service basés sur la cohérence (C) et la disponibilité (A) est faible lorsque la cohérence et la tolérance de partition (P) sont satisfaites. Compte tenu de la faible occupation des ressources et de l'approvisionnement rapide de conteneurs, cet article propose une approche visant à optimiser la disponibilité des systèmes de micro-service CP basés sur la planification élastique des ressources de conteneur et établit un modèle de prévision du temps de réponse à l'aide du système de mise en file d'attente en cascade. Ensuite, l'auteur a déterminé s'il fallait assouplir, restreindre ou maintenir la ressource du conteneur compte tenu de la conformité du temps de réponse. Enfin, l'approche d'optimisation proposée a été vérifiée par des expériences. Les résultats montrent qu'une période d'adaptation de 2 à 3 secondes est nécessaire pour l'approche lors de changements brusques de la charge et que le temps de réponse peut être prédit avec précision pour garantir la disponibilité du système dans les autres cas.*

KEYWORDS: *consistency (C), availability (A), partition tolerance (P), micro-service system, container, prediction model, elastic scheduling.*

MOTS-CLÉS: *cohérence (C), disponibilité (A), tolérance de partition (P), système de micro-service, conteneur, modèle de prévision, planification élastique.*

DOI:10.3166/ISI.23.6.45-60 © 2018 Lavoisier

1. Introduction

Micro-service system has a typical distributed structure, whose consistency (C), availability (A), and partition tolerance (P) cannot be satisfied simultaneously (Thönes, 2015; Newman, 2015). In general, the availability and partition tolerance are prioritized in commerce platforms and similar service systems requiring high availability (Souri *et al.*, 2014), while consistency and partition tolerance are highlighted in military field and other service systems requiring high data consistency (Wei *et al*., 2017). Thus, the former systems are referred to as the AP systems while the latter as the CP systems. In the CP systems, the availability should be maximized without sacrificing the consistency.

Concerning the trade-off between availability and consistency, the BASE (Basically Available, Soft state and Eventually consistent) theory holds that the two properties can be weighted according to demand rather than have an either-or relationship (Jiang *et al*., 2012). Inspired by the BASE theory, the quorum algorithm only needs a small number of read-write instances to achieve strong consistency, thereby improving the system availability (Raja and Prabhu, 2017). In addition, some scholars have enhanced the availability of containers deployed with micro-services by elastic provision of resources, in light of the low resource occupation and fast supply of the containers (Hao *et al*., 2017). For instance, Cherkasova established a loss model of session and throughput to estimate resource demand and realize precise scheduling (Cherkasova and Phaal, 2002). Based on performance isolation, Karlsson analysed the resource requirements of various applications, and adaptively supplied resources for containers (Karlsson *et al*., 2009).

The above studies show that elastic resource scheduling can optimize the availability of micro-service systems. However, there is no report on how to improve the system availability while ensuring strong consistency. To make up for this gap, this paper proposes a method to effectively enhance the availability of micro-service systems without scarifying consistency.

The remainder of this paper is organized as follows: Section 2 describes the general framework of our approach; Section 3 details the prediction model of response time and the scheduling algorithm of container resources; Section 4 verifies the proposed approach through experiment; Section 5 wraps up this paper with conclusions.

2. Approach overview

If service k has $Q_N(k)$ instances, the traditional method to maintain strong consistency requires that writing operations must be successful for all the $Q_N(k)$ instances, such that any instance can be read to get consistent and up-to-date data. The quorum algorithm allows the writing operations to be completed only $Q_W(k)$ ($Q_W(k) \leq Q_N(k)$) times, at the cost of reading data from $Q_R(k)$ ($Q_R(k) \leq Q_N(k)$) instances. Thus, the strong data consistency can be guaranteed as long as $Q_W(k) + Q_R(k) > Q_N(k)$. This method is equivalent to transferring part of the writing operation overhead to the reading operation. Thus, the values of $Q_W(k)$ and $Q_R(k)$ are related to the system efficiency in reading and writing operations.

In existing systems, the values of $Q_W(k)$ and $Q_R(k)$ are typically configured in advance and not changed unless an instance fails or new instances need to be added. For example, Dynamo uses the “322” mode, in which $Q_W(k) = Q_R(k) = 2$ if $Q_N(k) = 3$, and adopts the balance strategy under uncertain reading and writing frequencies (Decandia *et al*., 2007). In actual systems, however, one operation may be more intensive than the other, making it hard to improve system performance through the “322” mode.

In addition, it is common to deploy multiple containers simultaneously on the same physical machine. Despite being isolated from each other, the containers use the same CPU and memory at the bottom level. The resulting unfair resource usage and resource competition may make services unavailable.

To solve these two problems, our approach is implemented in the following steps (Figure 1).

(1) Data collection: Collect such data as the number of service instances, CPU usage and memory usage.

(2) Model calculation: Import the collected data into the prediction model and initialize the model.

(3) Forwarding rule adjustment: Adjust the values of $Q_W(k)$ and $Q_R(\mathrm{k})$ according to the current frequencies of reading and writing operations.

(4) Time prediction: Predict the response time based on the model, and judge by the response time whether container resources need elastic resource scheduling

(5) Resource scheduling: Execute the scheduling instruction and return to Step (1), forming a close loop approach.

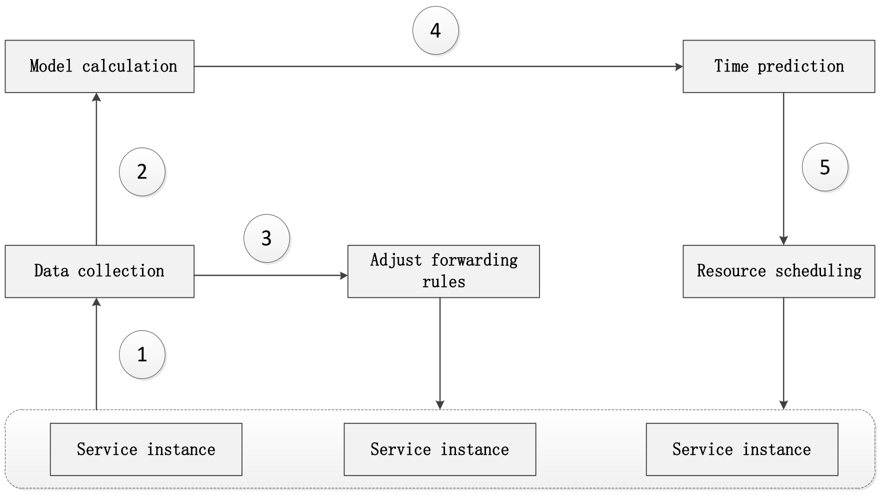

Figure 1. General framework of our approach

In summary, our approach for optimizing system availability combines the forwarding rule adjustment and container resource scheduling. The forwarding rule adjustment, known for its high speed, low cost and optimization degree, enables the system the respond the requests with fewer instances while ensuring strong consistency. The container resource scheduling is featured by slow speed, high cost and optimization degree. The integrated approach both ensures the availability of the system, and reduces the response cost without significantly reducing the availability.

3. System availability optimization

3.1. Request processing model

The cascade queuing system was adopted to build the request processing model for the following reasons:

(1) In the micro-service structure, the micro-service nodes are highly decoupled and the demand of each node is satisfied by multiple independent service centres at randomized service time.

(2) The micro-service nodes communicate with each other through messages to satisfy the demand of requests being processed in multiple service centres.

(3) When a request is processed in the service centre, it can choose to jump to the next node or leave the system.

For a service with multiple instances, the request will jump to the next node or respond to the user only if all the processing results of $Q(k), (Q(k) \leq Q_N(k))$ nodes are returned. The $Q(k)$ is an adjustable variable: $Q(k) = Q_W(k)$ for writing operations and $Q(k) = Q_R(k)$ for reading operations.

Let f be the user request stream and $\mathrm{k_i}$ be the i–th service visited by the request $(1 \leq i \leq m)$. Then, the relationship between f, $k_i$and $Q(k_i)$ can be described as: When a request f arrives at k_i, $Q(k_i)$ instances should be processed: either jump to the service $\mathrm{k_{i+1}}$ or leave the system.

The response time to the request can be expressed as:

$$T_r = d + \sum_{i=1}^{m} T_p(k_i) \tag{1}$$

where T_r is the response time of the request; d is the network transmission time of the request; $T_p(k_i)$ is the total processing time of k_i; $\sum_{i=1}^{m} T_p(k_i)$ is the total processing time of all services.

Assuming that the request processing times of micro-service nodes are independent of each other and exponentially distributed, the density function of the request processing times can be expressed as:

$$f(x) = \begin{cases} \frac{1}{\theta(k_i)[1-\mu(k_i)]} e^{-x/\theta(k_i)[1-\mu(k_i)]} & , \ x \geq 0 \\ 0 & , \ x < 0 \end{cases} \tag{2}$$

where $1/\{\theta(k_i)[1-\mu(k_i)]\}$ is the mean processing time of multiple instances of k_i; $\theta(k_i)$ is the parameter of exponential distribution with a resource utilization rate of 0; $\mu(k_i)$ is the preferred resource utilization rate of $\mathrm{k_i}$. The preferred resource is defined as the resource with the highest utilization rate in the computer.

Let $\theta[\mu(k_i)] = \theta(k_i)[1-\mu(k_i)]$. Then, the distribution function can be written as:

$$F(x) = \begin{cases} 1 - e^{-x/\theta[\mu(k_i)]} & , \quad x \geq 0 \\ 0 & , \quad x < 0 \end{cases} \tag{3}$$

Thus, the total processing time of k_i is:

$$T_p(k_i) = E[X_{Q(k_i)}] \tag{4}$$

where $X_{Q(k_i)}$ is the $Q(k_i)$ –th smallest node among the $Q_N(k_i)$ nodes. Thus, we have:

$$F_{Q(k_i)}(x) = P\big(X_1 \leq x, X_2 \leq x, \ldots, X_{Q(k_i)} \leq x, X_{Q(k_i)+1} \geq x, \ldots, X_{Q_N(k_i)} \geq x\big) = [F(x)]^{Q(k_i)}[1-F(x)]^{Q_N(k_i)-Q(k_i)} \tag{5}$$

According to equation (5), the mathematical expectation $E[X_{Q(k_i)}]$ can be obtained on Matlab before deriving the mean processing time $T_p(k_i)$. When $Q_N(k_i)$ is fixed, the size of $E[X_{Q(k_i)}]$ depends on $\mu(k_i)$, $\theta(k_i)$ and $Q(k_i)$.

Combining equations (2) and (5), the response can be obtained as:

$$\mathrm{T_r} = \mathrm{d} + \sum_{\mathrm{i}=1}^{\mathrm{m}} \mathrm{E}[\mathrm{X_{Q(k_i)}}] \tag{6}$$

Note that, during the calculation of $\mathrm{T_r}$, $\mu(k_i)$ and m can be obtained by monitoring, $\mathrm{Q(k_i)}$ can be determined by message forwarding rules, while d and $Q(k_i)$, which are difficult to monitor, need to be computed according to existing parameters when the system is running.

3.2. Prediction of response time based on Kalman filtering

Kalman filtering, originally designed to estimate the optimal state of linear systems, has been modified into extended Kalman filtering, traceless Kalman filtering and many other filtering algorithms with wide application and high accuracy in the past decades (Chen *et al.*, 2017; Degue and Ny, 2018). Here, Kalman filtering is employed to predict the response time of request.

Based on the response time of the previous sampling period and the measured parameter values in the current period, the state equation and the observation equation can be established as:

$$\begin{cases} X(t) = Y(t-1) + W(t) \\ Z(t) = FH(t) + U(t) \end{cases} \tag{7}$$

where $X(t)$ is the initial predicted response time at time t; $Y(t-1)$ is the final predicted response time at time $t-1$; $W(t)$ is the process noise; $Z(t)$ is the calculated response time at time t; F is the rule to convert parameters to the calculated response time; $W(t)$ is the measurement noise; $H(t)$ is a multidimensional vector of parameters; $H(t)$ can be described as:

$$H(t) = (d, m, \theta(k_i), Q(k_i), \mu(k_i))^T \tag{8}$$

$W(t)$ and $U(t)$ are white Gaussian noises, whose mean value is usually zero (Cai *et al.*, 2012). However, the accuracy of these noises may be greatly affected under abrupt load changes and inaccurate statistics of noise model. In light of this, the values of $W(t)$ and $U(t)$ were adjusted adaptively according to the error, aiming to correct the model error and make the Kalman filter adapt to the time-varying system.

Thus, the noise model is modified as:

$$\begin{cases} W(t) = \alpha(t) W_0 \\ U(t) = \beta(t) U_0 \end{cases} \tag{9}$$

where $W(t)$ and $U(t)$ are initial noises; $\alpha(t)$ and $\beta(t)$ are time-varying adjustment parameters. Besides determining the initial parameters, the adjustment rules of $\alpha(t)$ and $\beta(t)$ should be specified as:

$$\begin{cases} r_1 = [\tilde{Y}(t) - X(t)]/\tilde{Y}(t) \\ r_2 = [\tilde{Y}(t) - Z(t)]/\tilde{Y}(t) \end{cases} \tag{10}$$

where $\tilde{Y}(t)$ is the true response time at time t; r_1 is the deviation between the predicted value and the true value; r_2 is the deviation between the true value and the calculated value. The value of r_1 is negatively correlated with the system stability, and the proximity of r_2 to zero is positively correlated with the model accuracy.

Obviously, when $r_1 \approx r_2 \approx 0$, the prediction deviation is so small that it is not necessary to adjust the values of $\alpha(t)$ and $\beta(t)$. However, when r_1 and r_2 deviate far from the zero point, the values of $\alpha(t)$ and $\beta(t)$ should be adjusted such that the predicted value is close to the true value. Table 1 shows the proposed adjustment rules of $\alpha(t)$ and $\beta(t)$.

Table 1. Adjustment rules of $\alpha(t)$ and $\beta(t)$

r_2 \ r_1	Small	Zero	Large
Small	α(t) ↓, β(t) ↓	β(t) ↓	α(t) ↑, β(t) ↓
Zero	α(t) ↓	--	α(t) ↑
Large	α(t) ↓, β(t) ↑	β(t) ↑	α(t) ↑, β(t) ↑

In Table 1, the symbol ↑ represents an increase, and the symbol ↓ indicates a decrease. Small, Zero and Large represent the following three cases: far less than 0, equal to 0 and far greater than 0.

The three cases should be classified by the system's tolerance of the prediction error. The classification rules are as follows: when r_1 and r_2 fall in $(-0.1\sim0.1)$, i.e. the predicted or calculated value deviates from the true value by less than 10%, the deviation can be considered as in the Zero state; otherwise, the predicted or calculated value deviates from the true value by more than 10%, the deviation can be considered as in the Small or Large state, which calls for the adjustment of $\alpha(t)$ or $\beta(t)$. The adjustment range of $\alpha(t)$ or $\beta(t)$ depends on the degree of deviation between the predicted value and the calculated value. After the adjustment, the predicted value and the calculated values should return to the Zero state.

After predicting the response time $Y(t)$ at time t can be predicted, the following steps should be implemented:

(1) Let $Y(t-1)$ and $\tilde{Y}(t-1)$ be the predicted and true response times at time $t-1$, respectively. Then, the deviation is $[\tilde{Y}(t-1)-Y(t-1)]$ and the covariance is $P(t-1)=[\tilde{Y}(t-1)-Y(t-1)]^2$.

(2) Taking $Y(t-1)$ as the temporary predicted value X, the deviation can be obtained as $P(t|t-1)=P(t-1)+W(t)$. Then, the Kalman gain $K=P(t|t-1)/[P(t|t-1)+U(t)]$ can be derived. Finally, the final predicted value $Y(t)=X(t)+K[Z(t)-X(t)]$ can be obtained based on the calculated value $Z(t)$.

(3) If a request arrives at the system between $t\sim t+1$, update $P(t)=[\tilde{Y}(t)-Y(t)]^2$, otherwise make $P(t)=P(t-1)$. Then, predict the $Y(t+1)$ for the next period based on $Y(t)$ and $P(t)$. In this way, the Kalman filter can continuously predict the response time of the next period by recursion.

In the above calculation method, the values of d and $\theta(k_i)$ must be calculated first according to the measured parameters before determining the two initial values $Y(0)$ and $P(0)$ of the Kalman filter and the initial values of W_0 and U_0 can be determined. Thus, it takes a certain time for the filter to reach convergence.

3.3. Scheduling algorithm

As shown in Figure 2, the external requests are distributed to the container via routing services, and processed by independent services in the container. The routing services are initiated when the number of processed instances reaches the minimum number for reading or writing. Meanwhile, the monitoring service continuously collects system data, container resource utilization and other data, and transfers the collected data to the scheduler. Next, the scheduler will firstly analyse the data and adjust the forwarding rules periodically, then calculate the response time of the request by the prediction model and determine whether to relax, restrict or maintain the container resource, and finally send the scheduling signal to the executor. The executor mainly performs two functions: obtain the resource utilization rate of the containers through *docker stats* command and submit the data to monitoring service; modify the minimum number instances for reading or writing and control the resource quotas of each container in light of the scheduling signal. The above process hinges on the correct implementation of the scheduler.

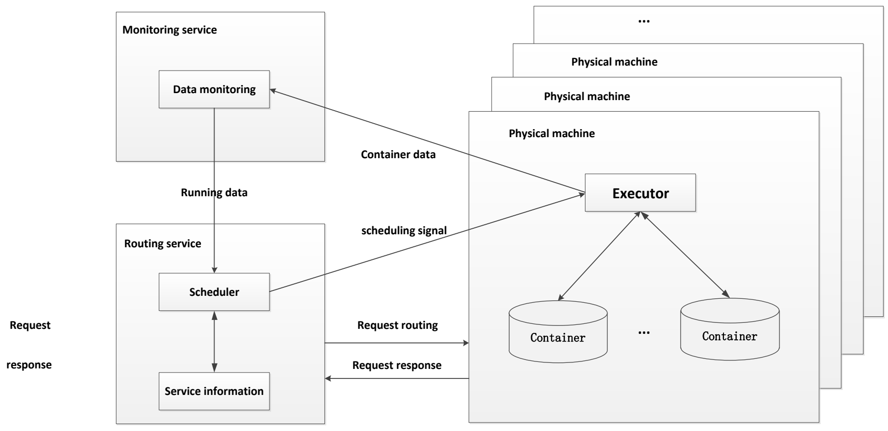

Figure 2. Implementation framework of our optimization approach

Scheduling algorithm

```
1  global var readNum, writeNum, α, β;
2  const var T_MAX_LIMIT, T_MIN_LIMIT,T_REFRESH_CYCLE;
3  define func Adjusted_value();
4  define func Expand(Service);
5  define func Contract(Service);
6  Input: $H(t) = (d, m, \theta(k_i), Q(k_i), \mu(k_i))^T$;
7  var α,β=$W_0$,$U_0$;
8  (α,β)=Adap(X(i),Y(i),Z(i));
9  var Y(i+1)=KF(Y(i),P(i));
10 During last TIME_REFRESH_CYCLE
   11 if $|readNum - writeNum| \gg 0$: Adjusted_value();
12 end if
13 if Y(i+1)< TIME_MIN_LIMIT: Contract(Service);
14 end if
15 if Y(i+1)> TIME_MAX_LIMIT: Expand(Service);
16 end if
```

Line 1 defines the parameter variables α and β in Kalman filtering and the number of instances for reading and writing as global variables. Line 2 defines the maximum response time, the minimum response time, and the refresh cycle. Lines 3~5 define the adjustment function, as well as the expansion function and restriction function of container resources. Lines 6~9 set out the rules to predict the response time of the next request. Lines 10~12 specify the rules to judge whether the adjustment function is executed. Lines 13~14 lay down the rules to judge whether the restriction function is executed. Lines 15~16 provide the rules to judge whether the expansion function is executed.

4. Experimental simulation and results analysis

4.1. Experimental environment and initial configuration

The initial configuration of the experimental environment is shown in Table 2. Two computers with the same configuration were selected for the experiment. One computer was equipped with Linux and Docker 1.7 for deploying the inventory management system, and the other was equipped with Windows and JMeter stress testing tool for testing the availability of the said system (Boettiger, 2015; You *et al.*, 2015).

Table 2. Experimental configuration

Hardware	CPU	Intel® Core™ i7-4790 CPU @ 3.60GHz
	Memory	8.00 GB
Operating system	Linux	Ubuntu-16.04
	Windows	Windows 7
Software	Container	Docker 1.7
	Testing tool	JMeter

Three service instances were deployed in the experimental environment using the “322” mode. The container runtime accounts for 10% of the CPU working time (the *cpu-period*: 1,000,000; the *cpu-quota*: 100,000). The upper limit of the container memory was configured as 512MB by setting *-m* to 512M .

4.2. Experimental design

The experiment mainly consists of the comparison between predicted and true response times and the verification of our approach to optimize system availability.

4.2.1. Comparison between predicted and true response times

The external requests were generated by JMeter, and the parameters were read continuously during the system operation. Calculated by Matlab, the predicted response time was contrasted against the true response time to verify the accuracy of the prediction model. The experimental load generated by JMeter simulated such five cases as steady load, steady load increase, abrupt load increase, steady load decrease and abrupt load decrease. The response time comparison was carried out in the five cases and the results were analysed in details.

4.2.2. Verification of availability optimization approach

The scheduling program was imported to the inventory management system. Then, the availability optimization approach was verified by the observed fluctuation range of the response time under the five cases. The approach applicability was obtained in various application environments by simulating different loads and types of requests.

4.3. Comparative experiment of prediction model

The applicability of the prediction model was verified in three cases: the number of requests is relatively steady (Case 1); the number of request changes steadily (Case 2); the number of request changes abruptly (Case 3).

Case 1: Simulated by JMeter, the number of writing operations equals that of reading operations. Ten requests were simulated per second in the 100s-long experiment. The predicted value was compared with the mean value of all requests in the next second. The predicted values are shown in Figure 3 below.

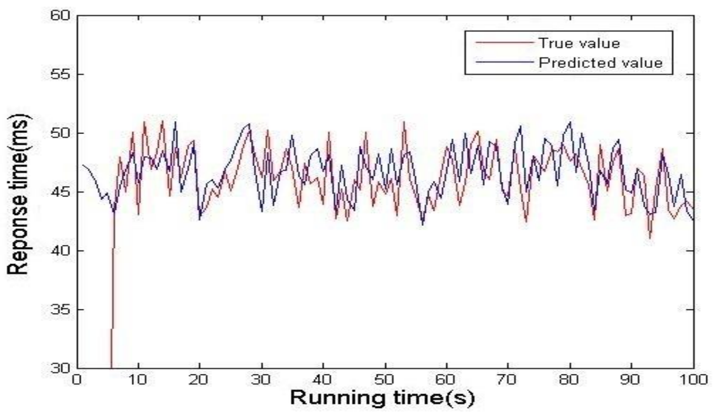

Figure 3. Predicted values under steady load

As shown in Figure 3, the prediction model converged in 5s at the beginning, and the prediction was very effective after convergence. Thus, the prediction model can work accurately when the number of requests is relatively steady.

Case 2: Simulated by JMeter, the number of writing operations equals that of reading operations. The total number of requests increased or decreased steadily throughout the experiment. Ten requests were simulated per second in 0~20s, twenty per second in 21~30s, thirty per second in 31~40s, forty per second in 41~50s, fifty per second in 51~60s, forty per second in 61~70s, thirty per second in 71~80s, twenty per second in 81~90s, and ten per second in 91~100s. The experiment lasts for 100s. The predicted value was compared with the mean value of all requests in the next second. The predicted values are shown in Figure 4 below.

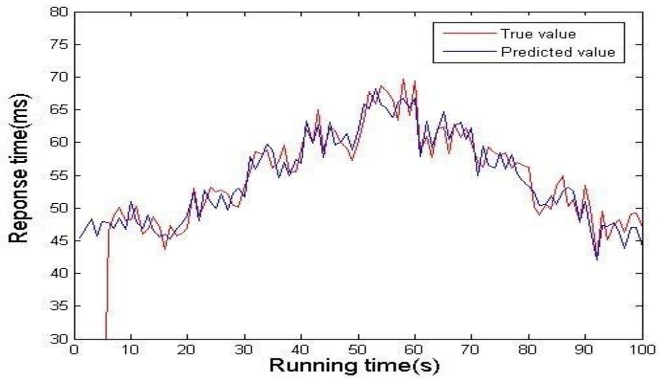

Figure 4. Predicted values under steady load changes

As shown in Figure 4, the predicted values were close to the true values except for the convergence period. This means the prediction model enjoys high accuracy despite the steady changes in the number of requests.

Case 3: Simulated by JMeter, the number of writing operations equals that of reading operations. The total number of requests increased or decreased abruptly throughout the experiment. Ten requests were simulated per second in 0~30s, fifty per second in 31~70s, and ten per second in 71~100s. The experiment lasts for 100s. The predicted value was compared with the mean value of all requests in the next second. The predicted values are shown in Figure 5 below.

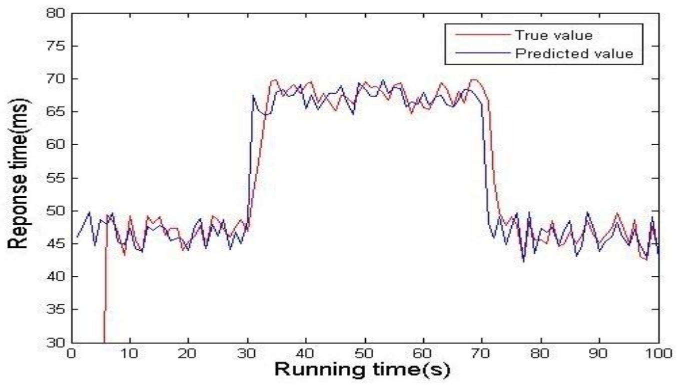

Figure 5. Predicted values under abrupt load changes

As shown in Figure 5, the predicted results were relatively accurate in the steady load period. Two 2~3s-long inaccurate periods appeared at the abrupt increase of the number of requests at 30s and the abrupt decrease at 70s, respectively. After that, the predicted values returned to the relatively accurate state. The results indicate that the prediction model needs 2 to 3 seconds to adapt to abrupt load changes.

To sum up, the prediction accuracy is high under a relatively steady load or steadily changing load, but a 2~3s-long adaptation period is needed under abrupt load changes. This is because the prediction model depends on the predicted value of the previous period and the calculated value of the current period. Proper inclination to the calculated value can ensure the prediction accuracy at small load changes, but the predicted value of the previous period may seriously affect the prediction accuracy under instantaneous sharp load changes. Hence, the response time of our model is highly accurate unless under extreme conditions like frequent, abrupt load changes.

4.4. Verification of availability optimization approach

The availability optimization approach was experimentally verified under the writing-reading request ratio of 4:1 and the steady range of response time of 40~50ms. The experiment covers three cases: the number of requests is relatively steady at a low value (Case 1); the number of request changes steadily (Case 2); the number of request changes abruptly (Case 3).

4.4.1. Availability in Case 1

It can be seen from Figure 3 that most of the response times fell between 40 and 50ms (Schafer *et al*., 2016) when the load was stabilized at 10 requests per second. Hence, the experimental condition in Figure 3 was applied to verify the relationship between the number of instances for reading and writing operations and response time. The experimental results are displayed in Figure 6 below.

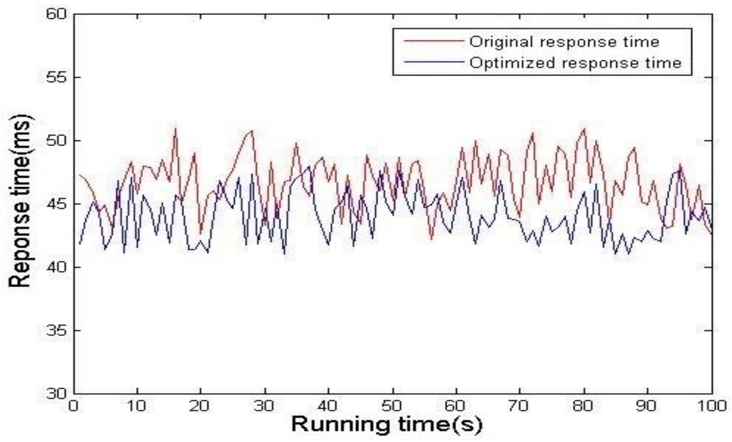

Figure 6. Availability under steady load

As shown in Figure 6, the response time of the optimization approach was slightly shorter than the original value. Since the response time always fell between 40 and 50ms, there was no scheduling of container resources. It can be concluded that fewer instances for reading and writing operations can slightly improve the system availability.

4.4.2. Availability in Case 2

It can be seen from Figure 4 that the response time increased or decreased gradually with the steadily changing load. Under this condition, the experimental results after applying the optimization approach are presented in Figure 7 below.

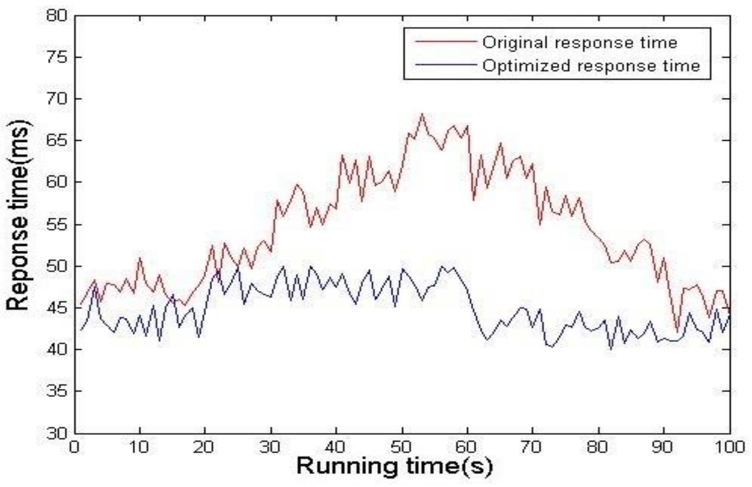

Figure 7. Availability optimization under steady load changes

As shown in Figure 7, the response time of the optimization approach was much shorter than the original value and maintained between 40 and 50ms. This is attributable to the high prediction accuracy. When the response time at the next time is predicted to exceed 50ms, more resources will be allocated to the container to reduce the response time.

4.4.3. Availability in Case 3

It can be seen from Figure 5 that the response time increased or decreased dramatically with the abrupt changing load. When the load was too great, the response time exceeded 50ms. Under this condition, the experimental results after applying the optimization approach are shown in Figure 8 below.

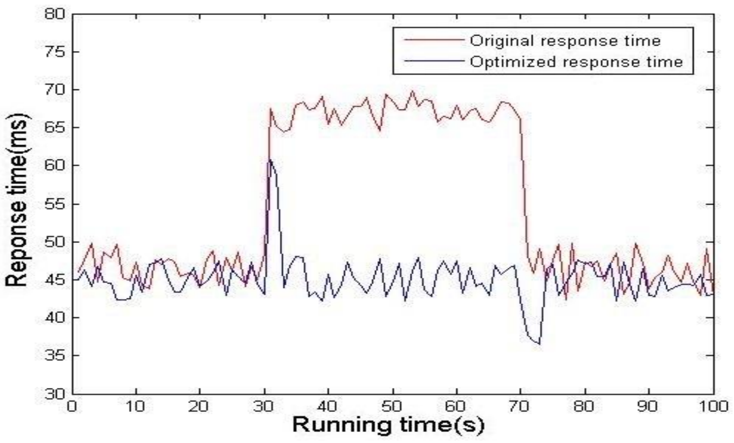

Figure 8. Availability optimization under abrupt load changes

As shown in Figure 8, the response time of the optimization approach was much shorter than the original value under steady, high load. However, the response time exceeded 50ms at 30s, owing to the prediction inaccuracy. The increasing amount of resources provided to the container could not reduce the response time to less than 50ms. Thus, the response time was reduced slightly but still above 50ms in 2~3s. Meanwhile, the response time was below 40ms at 70s, also the result of prediction in accuracy. The response time stayed below 40ms for several seconds before returning to the steady state.

In summary, the fewer number of instances for reading and writing operations can slightly improve the system availability. The system availability mainly depends on the elastic scheduling of container resources. Under steady load or steadily changing load, the response time can be predicted accurately, and the system availability can be optimized effectively. Under abrupt load changes, a 2~3s-long convergence period is needed to predict the response time. In this case, the availability optimization approach cannot always guarantee the response time within a certain range. Of course, the proposed approach can optimize the system availability in normal conditions.

5. Conclusions

This paper ensures the strong data consistency by the quorum algorithm, establishes a prediction model based on cascaded queuing system, and predicted the response time of requests based on Kalman filtering. After judging whether the response time is in default, the author put forward an approach to optimize the availability of the CP micro-service system by adjusting the forwarding rules and container resource scheduling according to the compliance of the response time. Finally, the proposed optimization approach was verified through experiments. The results show that a 2~3s-long adaptation period is needed for the approach under abrupt load changes, and the response time can be accurately predicted to ensure the system availability in the other cases.

References

Boettiger C. (2015). An introduction to Docker for reproducible research. *Acm Sigops Operating Systems Review*, Vol. 49, No. 1, pp. 71-79. https://doi.org/10.1145/2723872.2723882

Cai J. F., Chan R. H., Nikolova M. (2012). Two-phase approach for deblurring images corrupted by impulse plus Gaussian noise. *Inverse Problems & Imaging*, Vol. 2, No. 2, pp. 187-204.

Chen B., Liu X., Zhao H., Principe J. C. (2017). Maximum correntropy Kalman filter. *Automatica,* Vol. 76, pp. 70-77. http://dx.doi.org/10.1016/j.automatica.2016.10.004

Cherkasova L., Phaal P. (2002). Session-based admission control: A mechanism for peak load management of commercial web sites. *IEEE Transactions on Computers,* Vol. 51, No. 6, pp. 669-685. https://doi.org/10.1109/TC.2002.1009151

Decandia G., Hastorun D., Jampani M., Kakulapati G., Lakshman A., Pilchin A., Sivasubramanian S., Vosshall P., Vogels W. (2007). Dynamo: amazon's highly available key-value store. *ACM Sigops Sym-posium on Operating Systems Principles. ACM*, pp. 205-220. https://doi.org/10.1145/1323293.1294281

Degue K. H., Ny J. L. (2018). On differentially private Kalman filtering. *IEEE Global Conference on Signal and Information Processing. IEEE*, pp. 487-491. https://doi.org/10.1109/GlobalSIP.2017.8308690

Hao T. Y., Wu H., Wu G. Q., Zhang W. B. (2017). Elastic resource provisioning approach for container in micro-service architecture. *Journal of Computer Research and Development*, Vol. 54, No. 3, pp. 597-608. Http://dx.chinadoi.cn/10.7544/issn1000-1239.2017.20151043

Jiang W. Y., Bin L. I., Ling L. (2012). Research on data consistency and concurrency optimization of distributed system. *Computer Engineering*, Vol. 38, No. 4, pp. 260-262. http://www.ecice06.com/EN/10.3969/j.issn.1000-3428.2012.04.085

Karlsson M., Karamanolis C., Zhu X. (2009). Triage: Performance isolation and differentiation for storage systems. *Twelfth IEEE International Workshop on Quality of Service. IEEE*, pp. 67-74. https://doi.org/10.1109/IWQOS.2004.1309358

Newman S. (2015). Building microservices (First Edition). *USA: O'Reilly Media, Inc,* pp. 2-3.

Raja J. K., Prabhu V. (2017). An integrated software system for enterprise performance management. *International Journal of Management & Decision Making*, Vol. 8, No. 1, pp. 89-113.

Schafer D. R., Weiss A., Tariq M. A., Andrikopoulos V., Säez S., Krawczyk L., Rothermel K. (2016). HAWKS: A system for highly available executions of workflows. *IEEE International Conference on Services Computing. IEEE*, pp. 130-137. https://doi.org/10.1109/SCC.2016.24

Souri A., Pashazadeh S., Navin A. H. (2014). Consistency of data replication protocols in database systems: A review. *International Journal on Information Theory*, Vol. 3, No. 4, pp. 19-32.

Thönes J. (2015). Microservices. *IEEE Software*, Vol. 32, No. 1, pp. 116-116.

Wei H., Huang Y., Lu J. (2017). Probabilistically-atomic 2-atomicity: enabling almost strong consistency in distributed storage systems. *IEEE Transactions on Computers*, Vol. 66, No. 3, pp. 502-514. http://dx.doi.org/10.1109/TC.2016.2601322

You J., Zhang L., Wang H., Sun Y. (2015). JMeter-based aging simulation of computing system. *International Conference on Computer, Mechatronics, Control and Electronic Engineering. IEEE*, pp. 282-285. http://dx.doi.org/10.1109/CMCE.2010.5609969