Research on Resource Prediction Model Based on Kubernetes Container Auto-scaling Technology

UNITED THROUGH SCIENCE & TECHNOLOGY
The Electrochemical Society
Advancing solid state & electrochemical science & technology
248th
ECS Meeting
Chicago, IL
October 12-16, 2025
Hilton Chicago
Science + Technology + YOU!
SUBMIT ABSTRACTS by March 28, 2025
SUBMIT NOW

Research on Resource Prediction Model Based on Kubernetes Container Auto-scaling Technology

Anqi Zhao[1], Qiang Huang[1,2*], Yiting Huang[1], Lin Zou[1], Zhengxi Chen[1] and Jianghang Song[1]

[1] College of Information Engineering, Sichuan Agricultural University, Yaan, Sichuan, 625014, China

[2] The Lab of Agricultural Information Engineering, Sichuan Key Laboratory, 625000 Yaan, China

[*]19623556@qq.com

Abstract. Cloud computing provides a new way for computing resource acquisition and brings about changes in software development deployment. Docker is a lightweight alternative technology of virtualization, which is simple, convenient and practical when developing and deploying applications. Kubernetes is an open source container management system based on Docker container technology. This paper studies the existing auto-scaling strategy of Kubernetes and proposes an auto-scaling optimization strategy which can solve the response delay problem in the expansion phase. This strategy uses a combination of empirical modal decomposition and ARIMA models to predict the load of Pods and adjust the number of Pods in advance according to the prediction result. Our experiment proves that the strategy can achieve the purpose of capacity expansion before the peak load and reduce the application request response time.

1. Introduction

Docker is an open source application container engine [1], which is simple, convenient and practical when developing and deploying applications. Kubernetes is an open source container management system based on Docker container technology, which can deploy applications across multiple server hosts and schedule or scale these containers on the cluster. It can avoid the phenomenon that the service is stopped due to server downtime, physical disk damage, etc., ensuring the reliability and continuity of the service.

2. Problem analysis of Kubernetes instance automatic scaling policy

2.1. Research on auto-scaling strategy

In Kubernetes' existing automatic scaling service process, the Horizontal Pod Autoscaler needs to first define the relevant parameters to ensure that the Horizontal Pod Autoscaler works according to the target and then it checks the load status of Pod through the monitoring tool and get the CPU usage and CPU allocation of the replica set. Then we calculate the CPU utilization of the replica set.

We compare the CPU utilization with a preset target value to calculate the expected number of replicas. The specific calculation formula is shown in equation (1).

$$\mathrm{E}=\mathrm{ceil}\left(\frac{\mathrm{CU}}{\mathrm{TU}}*\mathrm{CR}\right) \tag{1}$$

E is the expected number of replicas, *CU* is the CPU usage of the replica set, *TU* is the target value, and *CR* is the current number of replicas. Then adjust the number of Pod replicas automatically through the Replication Controller or Deployment. When the load is too large, it will create a new Pod to offload which can reduce the waiting time for service requests and reduce users' request response time.

2.2. Problem analysis

The initialization of Pod can be divided into four phases. The phases are shown in Table 1.

Table 1. Composition of Pod initialization time

Phase	Time	Process
1	*t1*	Trigger HPA to expand and calculate the expected number of replicas and send the results to the Replication Controller.
2	*t2*	The Replication Controller receives the expected number of replicas and compares the expected value with the actual number of current replicas to determine whether expansion should be performed.
3	*t3*	The scheduling module checks that there is a newly created Pod and dispatches it to the appropriate node in the cluster according to the scheduling policy.
4	*t4*	Kubelet implement Pod creation process includes downloading images from Docker Registry, launching and initializing application containers, etc.

Therefore, the initialization time of Pod is calculated as in equation (2).

$$\mathrm{t_{init}}=\sum_{i=1}^{4}\mathrm{t_i} \tag{2}$$

It takes t_{init} time from the HPA expansion triggered by the load peak at *s1* to completing the expansion at *s2*, which will cause a delay in response to t_{init} time. During this time, a large number of user requests will accumulate and it will lead to the phenomenon that service requests queued waiting. The response delay increases the user's request response time, resulting in a drop in service quality.

Up to now, many scholars home and abroad have studied how to optimize auto-scaling performance. Al-Haidari F studied the effect of the threshold setting on the automatic scaling performance [3], but he did not take the adaptability to resource changes into account. Barrett E uses reinforcement learning theory to develop a scaling strategy [4], but it takes a lot of time to summarize the experience and has poor adaptability to sudden load. Jingqi Yang proposed a load prediction method based on linear regression algorithm and string-matching algorithm [5]. This method considers the adaptability to resource changes, but does not take the complexity of the load itself into account so that it also has certain limitation.

This paper proposes an optimized capacity expansion strategy for the response delay caused by the time required for Pod initialization. The strategy adopts the load prediction method combining empirical mode decomposition and ARIMA, which considers the adaptability to resource changes and the complexity of the load itself. The experiment shows that compared with traditional methods, the accuracy of prediction can be improved, which is of great value for improving auto-scaling performance and reducing user response time.

3. Prediction model principle and method

Since the historical load data of Pod replica set is different because of users' access characteristics, the choice of the prediction method must take into account the stability of the load data, and the complexity, randomness and non-stationary nature of the local load data. And because predictive expansion also consumes a certain amount of time and resources, it is not appropriate to use an algorithm that is too complex. The empirical mode decomposition method excels in the non-stationary signal smoothing process. The ARIMA model is considered to be the most advanced and effective analysis method in the time series analysis mode. Therefore, this paper proposes a prediction model that combines empirical modal decomposition with ARIMA model to predict the load of Pod.

3.1. Empirical Mode Decomposition (EMD)

Empirical mode decomposition is a signal analysis method that can smooth non-stationary signals [6]. The method performs signal decomposition according to the time scale characteristics of the data itself, and decomposes the complex signal into a finite number of Intrinsic Mode Functions (IMF). It has obvious advantages in dealing with non-stationary and nonlinear data.

Its modeling is as follows [7]:

(1) For the original signal $x(t)$, find the maximum and minimum values near each point on the signal curve and use the 3 times spline function interpolation to draw the curves of the maximum value point and the minimum value point respectively, forming an upper envelope $xmax(t)$, a lower envelope $xmin(t)$.

(2) Find the average package route $m(t)$. The calculation formula is shown in equation (3).

$$m(t)=xmax(x)+xmin(t) \tag{3}$$

(3) Subtract $m(t)$ from the original signal $x(t)$ to get $h(t)$

$$h(t)=x(t)-m(t) \tag{4}$$

(4) Determining whether $h(t)$ is a determined intrinsic mode function IMF:

① number of extreme points (including maximum and minimum values) and cross-zero number modulus is less than or equal to 1.

②The average envelope $m(t)$ is approximately 0 at any time.

(5) If $h(t)$ satisfies the above two conditions, $h_1(t)$ is an IMF, denoted as $c_1(t)$, otherwise repeat steps (1)-(3) until meet the conditions. This is repeated until the last sequence $r(t)$ can no longer be decomposed, and finally the original signal decomposition is represented by the IMF component and a residual R.

$$x(t)=\sum_{i=1}^{n} c_i(t)+r(t) \tag{5}$$

In the above formula, $c_i(t)$ is the IMF obtained by EMD decomposition, and $r(t)$ is the trend residual of the signal.

3.2. ARIMA model

The ARIMA model, also known as the differential autoregressive moving average model, can be applied to random line time series prediction. It is considered to be the most advanced and effective analysis method in time series analysis. [8].

ARIMA (p, d, q) model means that the highest order of the autocorrelation coefficient after the d-order difference of the time series is p, and the highest order of the moving smoothing coefficient is q. Its mathematical model [9] is shown in equation (6).

$$\nabla^d y_t=\sum_{i=1}^{p} \emptyset_i \nabla^d \omega_{t-i}+\varphi_t+\sum_{j=1}^{q} \theta_j \nabla^d \delta_{t-j} \tag{6}$$

In the formula: ω_t is the resource demand at any time t of the time series. $\emptyset_i$ is the parameter of the autoregressive model. i is the natural number N. θ_j is the parameter of the moving average model. j is the natural number N and φ_t is the white noise with a mean of 0. The sequence, δ_t, is the residual sequence value at any time t of the time series, and ∇^d is used as a differential processing of the data sequence [10].

The forecasting steps are as follows:

(1) Input resource usage sequence $\omega_t,\omega_{t+1},\ldots,\omega_{t+n}$;

(2) Determine whether the resource sequence satisfies the condition of the stationary nature, if it is satisfied, go to (3), otherwise the sequence continued (2) after the difference;

(3) Average the resource sequence, and use the processed result to calculate the autocorrelation function $f(t)$ and the partial correlation function $z(t)$;

(4) The obtained autocorrelation function and partial autocorrelation function are identified by ARMA model, and the parameters of the model function are estimated by using known resource sequences;

(5) Test validity for the model: if the model is valid, the model is used to predict the future resource usage value, and if it is invalid, jump to (3) recalculate.

3.3. Establish EMD-ARIMA prediction model

Aiming at the response delay of Kubernetes in container auto-scaling, this paper proposes a method for forecasting future load based on historical load data of Pod replica set. The method first obtains the CPU utilization of the replica set as the load data through the monitoring tool Prometheus, and then decomposes the load data into a finite eigenmode function (IMF) by empirical mode decomposition. Then, the ARIMA model is constructed separately for each component, and finally the sub-sequences are superimposed to obtain the predicted value [11]. Then, the predicted load result is used to calculate the number of replicas.

The forecasting model flow is shown in Figure 1.

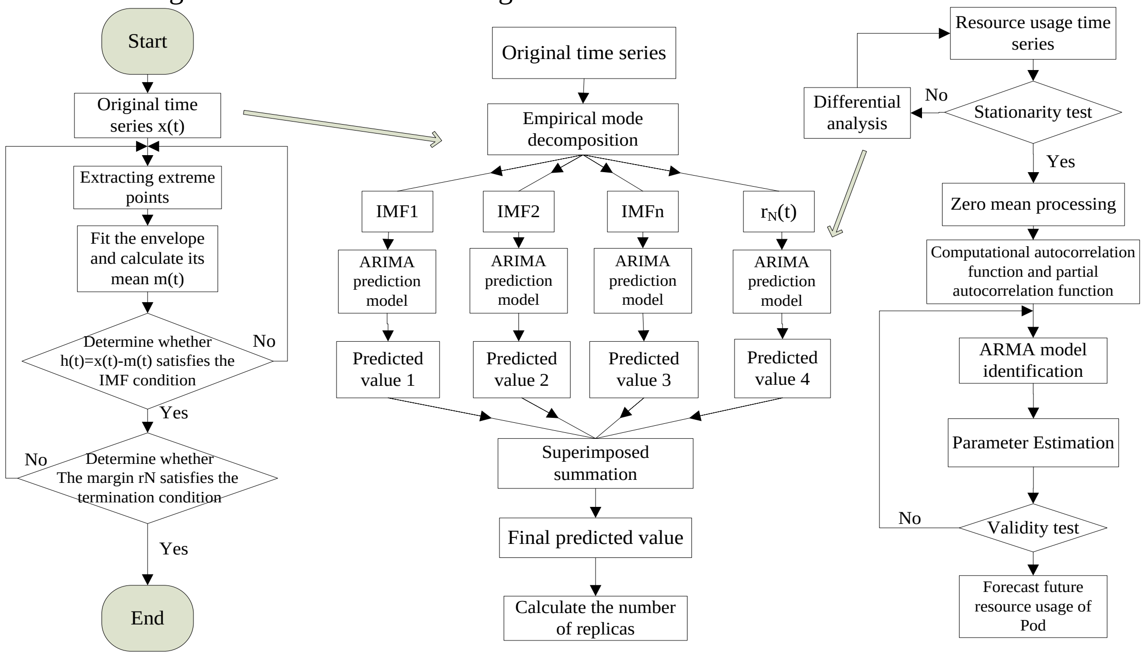
Start
Original time series x(t)
Extracting extreme points
Fit the envelope and calculate its mean m(t)
Determine whether h(t)=x(t)-m(t) satisfies the IMF condition
No
Yes
No
Determine whether The margin rN satisfies the termination condition
Yes
End
Original time series
Empirical mode decomposition
IMF1
IMF2
IMFn
$r_N(t)$
ARIMA prediction model
ARIMA prediction model
ARIMA prediction model
ARIMA prediction model
Predicted value 1
Predicted value 2
Predicted value 3
Predicted value 4
Superimposed summation
Final predicted value
Calculate the number of replicas
Resource usage time series
Differential analysis
No
Stationarity test
Yes
Zero mean processing
Computational autocorrelation function and partial autocorrelation function
ARMA model identification
Parameter Estimation
No
Validity test
Forecast future resource usage of Pod

Figure 1. EMD-ARIMA forecasting model flow chart

4. Experimental protocol

We prepare a test application, make it into a docker image, and use Pod mode to deploy it on the experimental cluster. We validate the auto-scaling optimization algorithm proposed in this paper by comparing with Kubernetes' existing responsive scaling strategy. In this experiment, two Kubernetes experimental clusters need to be set up as a control group, cluster 1 is built using Kubernetes 1.12, and cluster 2 is recompiled and built using the improved Kubernetes source package.

Then we use Apache JMeter as the load generator to test the scaling optimization strategy mentioned in this paper. The optimization model is tested by changing the thread group size, startup time, request interval, and application port accessed by the load generator to design different loads.

We use the EMD-ARIMA model and the ARIMA model to predict the load data of Pod. The results are shown in Figure 2.

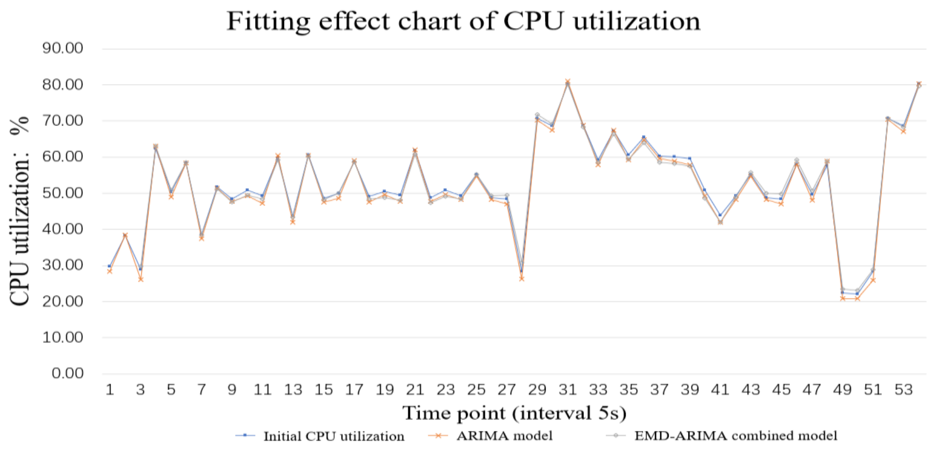

Figure 2. Fitting effect diagram of replica set CPU utilization

We use absolute mean error (MAE), absolute mean percentage error (MAPE) and root mean square error (RMSE) as the indicators to analyze and compare the prediction accuracy of each model. The results are shown in Figure 3.

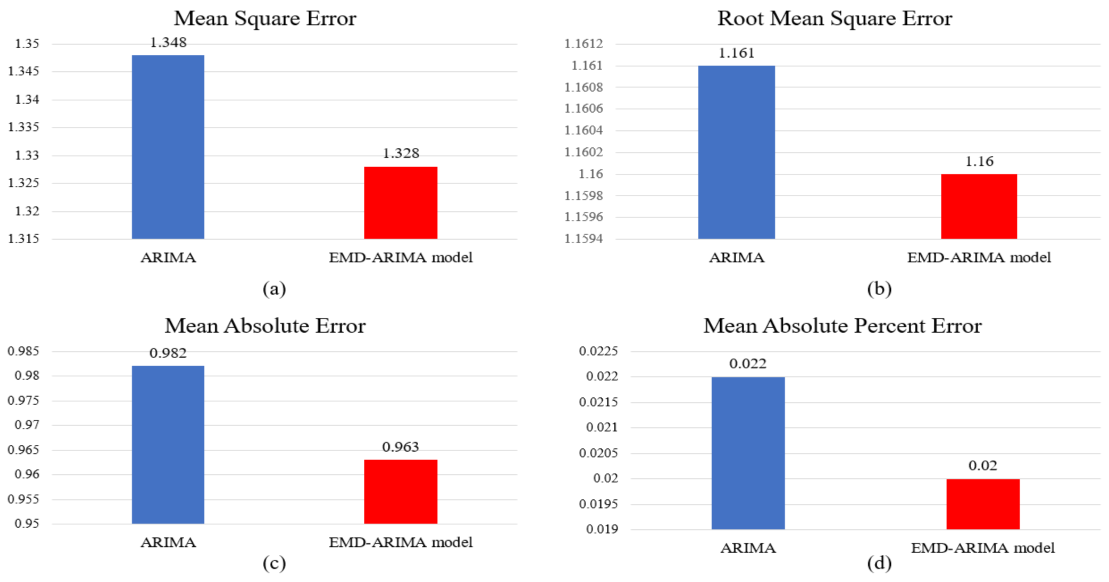

Figure 3. Comparison of prediction accuracy

It can be seen that after the original data is denoised by the EMD denoising algorithm in signal processing, using the ARIMA algorithm to predict the components decomposed by the EMD algorithm can improve the accuracy of the prediction, which proves the effectiveness of the prediction model.

Then we compare the improved Kubernetes cluster with the existing version of the Kubernetes cluster to test whether the improved HPA can start capacity expansion before the peak load and reduce the average response time of the user to verify the effectiveness of the optimization strategy.

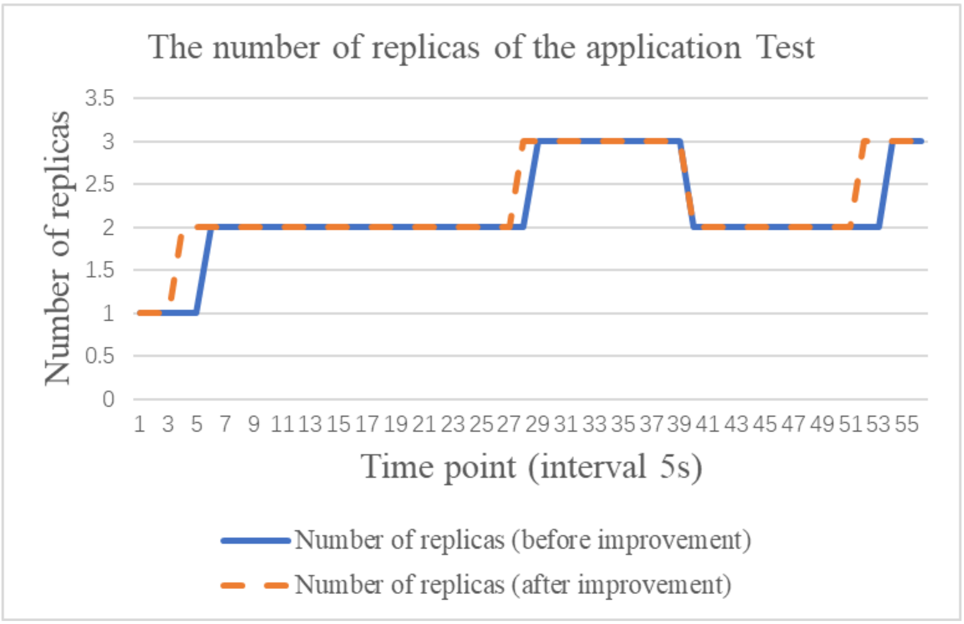

Figure 4. Change in the number of instances of the application Test

Three load peaks in total were designed in this experiment. On the improved Kubernetes cluster, before each peak, the instance scaling module will gradually increase the number of instances of the test application, and increase the number of instances of the application to three. After the peak load, the Kubernetes shrinks. The capacity strategy gradually reduces the number of instances, which proves the feasibility of the optimization model.

5. Conclusion

Aiming at the response delay of the existing Kubernetes scaling strategy in the expansion phase, this paper uses the combination of empirical mode decomposition and ARIMA model to predict the load of Pod application and adjust the number of Pod in advance according to the results. The experiment proves that the optimization strategy can effectively reduce the response time of the user. However, in real life, Pod load data will exhibit different load characteristics according to different services. Further research on business characteristics can help improve the accuracy of prediction.

Acknowledgments

This research was supported by Research Interests Training Program of Sichuan Agricultural University, Innovation Training Program of Sichuan Agricultural University and the Lab of Agricultural Information Engineering, Sichuan Key Laboratory.

References

[1] Wu Zhixue. Development and trend of cloud computing virtualization technology [J]. Computer Applications, 2017, 37 (04): 915-923. (in Chinese).
[2] Yang Mao. Research on the automatic telescopic technology of containers based on Kubernetes [D]. Xi'an University of Posts and Telecommunications, 2018. (in Chinese).
[3] Al-Haidari F, Sqalli M, Salah K. Impact of cpu utilization thresholds and scaling size on autoscaling cloud resources[C]//2013 IEEE 5th International Conference on Cloud Computing Technology and Science. IEEE, 2013, 2: 256-261.
[4] Barrett E, Howley E, Duggan J. Applying reinforcement learning towards automating resource allocation and application scalability in the cloud[J]. Concurrency and Computation: Practice and Experience, 2013, 25(12): 1656-1674.
[5] Yang Jingqi. Research on scalability of cloud business platform [D]. Beijing University of Posts and Telecommunications, 2014. (in Chinese).

[6] Huang Norden E., Shen Zheng, Long Steven R. The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis[J]. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, 1998, 454(1971).
[7] Du Wei Peng, Wang Yi, Shen Ying. Application of Empirical Mode Decomposition (EMD) in Deformation Data Analysis [J]. Inner Mongolia Science and Technology and Economy, 2019 (01): 103-105. (in Chinese).
[8] Yang Pengfei. Research and implementation of resource dynamic scheduling based on Kubernetes [D]. Zhejiang University, 2017. (in Chinese).
[9] Fan Chao, Guo Yafei, Cao Peige. Prediction of Grain Yield by GM (1,1)-ARIMA Combined Model Based on Wavelet Transform[J]. Jiangsu Agricultural Sciences,2019,47(01): 221-224. (in Chinese).
[10] Xie Wenzhou, Sun Yanxia. Research on Resource Prediction Model Based on Kubernetes Load Characteristics[J]. Network Security Technology and Applications, 2018(04): 27-28. (in Chinese).
[11] Yu Yingwei, Xu Dayu, Shou Guozhong, Wang Peixin. Prediction of Temperature and Humidity Based on Empirical Mode Decomposition and Wavelet Neural Network[J]. Jiangsu Agricultural Sciences, 2019, 47(01): 211-216. (in Chinese).