UDCV 004.042

D. Gutman, O. Syrota

PROACTIVE AUTOMATIC UP-SCALING FOR KUBERNETES

Abstract: Container management systems are widely used in cloud computing. The leader of the market is Kubernetes. There no ability to configure a proactive scaling for your service using Kubernetes. The research presents an autoscaling technique based on a proactive approach to scale services in Kubernetes.

Keywords: Autoscaling, cloud computing, Kubernetes, proactive scaling, reactive scaling.

Introduction

Cloud computing and containers are getting popular nowadays. More businesses use cloud providers instead of dedicated manageable machines to host applications. This allows saving costs on hardware and on professionals who support its functionality [1]. In addition, cloud computing supports easy scalability and it allows avoiding underprovisioning or overprovisioning of resources [2].

Because the workload on services is not constant, much attention paid to autoscaling - the process of automatically scaling application instances with increasing workload. Effective autoscaling improves the quality of service (other words minimize response time for customers). Automatic scaling of resources based on workload change reduces provisioning costs and helps clients to achieve performance objectives. The cloud provider, on the other hand, should attempt to consolidate load onto highly utilized physical machines, in order to reduce wasted power consumption [3].

Autoscaling can be divide into two types - reactive and proactive. With a reactive approach, scaling is based on past workload data. On the opposite, with a proactive approach scaling decision based on the forecasted workload. Workload forecasting is made with the help of time series prediction methods [4,5] or machine learning, as presented in [6,7].

Reactive scaling has the following disadvantage: It takes time to launch a new instance, so when the scaling decision made it will take some time until the new instance will be available. During this period the QOS (quality of service, average response time) is decreased. Proactive scaling solves this problem (assuming an effective load prediction system) because resources allocated in advance, see Fig. 2.

A container is a standard unit of software that packages up a code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings [8].

Containers have made a big impact to cloud computing, making easy for developers to deploy apps. Now it is an industry standard of delivering apps. Applications mostly developed

using a microservice technique. In that case, one microservice is one Docker container. Large applications have many containers, and it would be difficult to manage every single one of them manually.

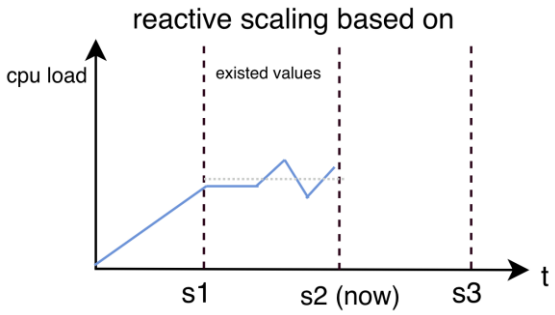

Figure 1. Reactive scaling

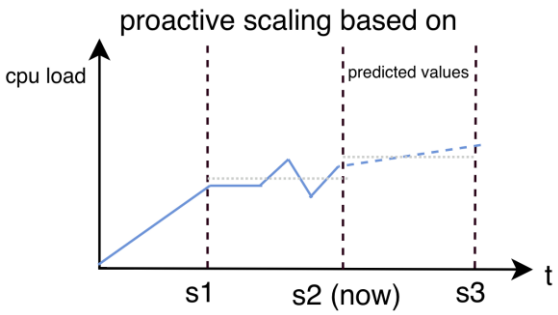

Figure 2. Proactive scaling

To build a big system with a thousand containers, the container management (or orchestration) systems created. Some examples: Kubernetes, Docker Swarm and Apache Mesos. The most popular container management system is Kubernetes, developed by Google Inc. It allows automating the deployment, scaling, and management of containerized applications.

Kubernetes widely used as an important part of the cloud computing infrastructure [9]. AWS gives the ability to fully manage Kubernetes cluster using EC2 or to run it without needing to provision using Amazon Elastic Container Service [10], Google Cloud Platform has Kubernetes Engine, that allows to get up and running with Kubernetes in no time, by completely eliminating the need to install, manage, and operate your own Kubernetes clusters. [11].

Currently, Kubernetes supports only manual scaling or reactive autoscaling [12]. Search for articles on the implementation of proactive scaling for Kubernetes gave no results.

This paper presents a method of creating a proactive scaling system for the Kubernetes. The efficiency of the proposed proactive scaling will be measured and compared to the standard reactive method. The average response time of a scalable service will be measured and used as a performance metric.

Mathematical model for proactive scaling

To implement a proactive scaling system the workload prediction method is required.

Methods to predict time series from statistics are well suited: Exponential Smoothing (ES) and Double Exponential Smoothing (DES). DES method is used for forecasting in proactive autoscaling in [13, 14] and it showed its effectiveness. The time series may have the following characteristics: trend and seasonality. The task of autoscaling related to the application workload trend. Seasonality not covered in this paper.

In the ES method, the prediction of the following time series values performed using the formula [15]:

$$F_t = \alpha X_t + (1 - \alpha) F_{t-1} \quad (1)$$

where t - forecast period,

F_t - Forecasted value,

F_{t-1}- Previous forecasted value,

X_t- Actual value for the previous period,

α - Data smoothing factor [0, 1].

Simple exponential smoothing does not do well when there is a trend in the data, which is inconvenient [16]. In such situations better to use DES method, which is the recursive application of an exponential filter twice. DES method takes into account the trend of the time series and has two formulas [17]:

$$s_t = \alpha x_t + (1-\alpha)(s_{t-1}+b_{t-1}) \quad (2)$$

$$b_t = \beta(s_t - s_{t-1}) + (1-\beta)b_{t-1} \quad (3)$$

where t - forecast period,

α- Data smoothing factor [0, 1],

β- Trend smoothing factor [0, 1],

b - Trend in moment t,

s - Prediction in moment t

x_t- Actual value for the previous period.

To forecast beyond x_t:

$$F_t = s_t + mb_t \quad (4)$$

Both forecasting methods (ES, DES) used in this paper.

Proactive automatic up-scaling for Kubernetes

Proposed proactive scaling system parts are:

- Kubernetes cluster
- Time series database
- Scaler service

The proactive scaling system architecture shown in Fig. 3.

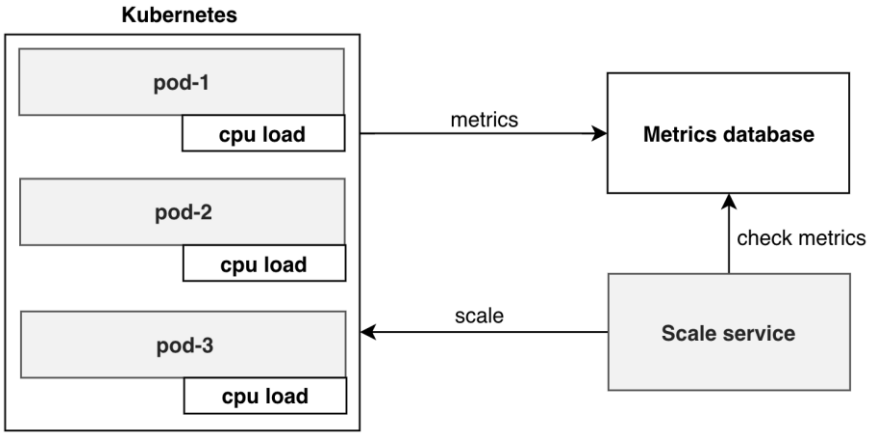

Figure 3. Proactive Scaling System Architecture for Kubernetes

 ISSN 1560-8956

A pod is the smallest entity in a Kubernetes, and it is a one Docker container usually. In that case, it is a containerized target application, which will be scaled.

Each Kubernetes pod writes data about CPU workload to the time series database during a periodic task. The scaling service also has a periodic task that performs the following logic:

1. Take the last N records in the database for each pod.
2. Predict the workload using ES/DES method and data from the database.
3. Scale if mean predicted workload is more than a threshold.

Proactive scaling system for Kubernetes implemented using the Java programming language and Spring Framework 5. Database InfluxDB chosen for storing time series. All source code is available on GitHub [18]. Scaler settings are:

- The interval for recording CPU load data for pods is 10 seconds.
- The interval of the periodic scaling task is 30 seconds.
- Prediction time is 1 minute.
- Prediction based on workload records for the last 2 minutes.
- Scaling threshold is 80%

Experiments

Experiments performed using Minikube, an application for running the Kubernetes cluster on a single host [19]. It runs using a virtual machine called VirtualBox 5. Virtual machine settings are two CPU x 2.0 GHz, 4 GB RAM.

JMeter 5.0 is a tool to load test functional behavior and measure performance developed by the Apache Software Foundation [20]. It used to test application performance during different scenarios.

Each experiment repeated 5 times, and average results are present in tables. Experiments are modeling increase of application workload. The first experiment is modeling smooth increasing of a workload; the second is about rapid increasing.

In both experiments α is equals to 0.75 in ES and DES methods, β is equals to 0.5.

1. Experiment #1: Small trend

In this experiment, the workload is increasing slowly.

- Experiment time: 10 min 30 sec.
- The number of requests per second: 4-125.
- The maximum number of pods: 3.

Table 1

Comparison of the results of experiment №1

Parameter	Reactive	Proactive (ES)	Proactive (DES)
Requests processed	32429	32445 (+0.05%)	32275 (-0.48%)
Average request time	79.33	76.67 (-3.36%)	70.33 (-11.34%)
Median request time	37.67	31.67 (-15.93%)	31.67 (-15.93%)
Throughput (r/s)	51.47	51.50 (+0.05%)	51.23 (-0.47%)

A proactive approach to scaling can improve the average response time by 11.34% in a load scenario with a small trend. DES again is more efficient than the default for Kubernetes reactive approach and the proactive scaling using ES.

2. Experiment #2: Large trend

In this experiment, the workload is increasing rapidly.

- Experiment time: 7 min 30 sec.
- The number of requests per second: 10-100.
- The maximum number of pods: 3.

A proactive scaling could improve the average response time up to 14.75% in a high-trend scenario. Processed request count, median request time and throughput are a little bit better with DES method.

Table 2

Comparison of the results of experiment №2

Parameter	Reactive	Proactive (ES)	Proactive (DES)
Requests processed	21937	22559 (+2.83%)	23054 (+5.09%)
Average request time	61.00	52.00 (-14.75%)	52.00 (-14.75%)
Median request time	28.33	23.67 (-16.47%)	26.33 (-7.06%)
Throughput (r/s)	48.74	50.13 (+2.83%)	51.23 (+4.36%)

 ISSN 1560-8956

Conclusion

The experiment performed by two scenarios: with small (4.1) and large (4.2) trends. In each scenario experiment first performed with reactive autoscaling, then with proactive autoscaling using the ES model of time series analysis, and then with proactive autoscaling using DES model. By comparing results from various scenarios and prediction models, we can conclude that workload forecasting minimizes the average response time of application. There is also noticeable that the DES model is more efficient than the ES model in both scenarios. The reason is that the DES method takes into account the trend of the time series. Also regarding experiments, we can make the next conclusions:

1. Reactive scaling is more efficient in scenarios with a small workload trend. The reason is that smooth increasing of workload makes possible for proactive autoscaler to work on time.

2. Proactive scaling is more efficient in the scenarios with a rapid workload increase.

Directions for future research:

- Search for parameter values and for DES, to archive the best average response time.
- Comparison of the effectiveness of different methods for predicting the workload.
- Hybrid scaling system (reactive and proactive) for Kubernetes.
- Proactive scaling that will take into account communications between microservices.
- Technologically fit solution into Kubernetes infrastructure for easy plugging it as Kubernetes extension.
- Technologically add support for Prometheus timeseries database which is widely used to collect metrics and monitor microservices.

REFERENCES

1. *Armbrust, M. Fox, A, Griffith, R. Joseph, D. A. Katz, R. Konwinski* Above the clouds / A Berkeley View of cloud computing. University of California, Berkeley, February 2009, pp. 6-7.

2. *Armbrust, M. Fox, A, Griffith, R. Joseph, D. A. Katz, R. Konwinski* Above the clouds / A Berkeley View of cloud computing. University of California, Berkeley, February 2009, pp. 10-12.

3. *Tighe, Michael & Bauer, Michael* Integrating Cloud Application Autoscaling with Dynamic VM Allocation / in Proceedings of 14th IEEE/IFIP Network Operations and Management Symposium: Management in a Software Defined World, Krakow, Poland, May 2014.

4. *Kee Kim, Wei Wang, Yanjun Qi, and Marty Humphrey* Empirical Evaluation of Workload Forecasting Techniques for Predictive Cloud Resource Scaling / in Proceedings of 9th International Conference on Cloud Computing, San Francisco, CA, USA, June 2016, pp 8-9.

5. *N. Roy, A. Dubey and A. Gokhale* Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting / in Proceedings of 4th International Conference on Cloud Computing, Washington, DC, USA, July 2011, pp. 500-507.

6. AWS News blog, New - Predictive Scaling for EC2, Powered by Machine Learning. [Online]. Available: https://aws.amazon.com/blogs/aws/new-predictive-scaling-for-ec2-powered-by-machine-learning

7. *Wajahat, Muhammad & Gandhi, Anshul & Karve, Alexei & Kochut, Andrzej* Using machine learning for black-box autoscaling / in Proceedings of 7th International Green and Sustainable Computing Conference, Hangzhou, China, November 2016, pp. 1-8.

8. What is a Container? [Online]. Available: https://www.docker.com/resources/what-container

9. *David Bernstein* Containers and Cloud: From LXC to Docker to Kubernetes / IEEE Cloud Computing, Vol. 1, Issue 3, pp. 81-84, Sep. 2014.

10. Kubernetes on AWS. [Online]. Available: https://aws.amazon.com/kubernetes

11. Kubernetes Engine [Online]. Available: https://cloud.google.com/kubernetes-engine

12. Autoscaling in Kubernetes. [Online]. Available: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale

13. *Aslanpour, Mohammad Sadegh and Seyed Ebrahim Dashti* Proactive Auto-Scaling Algorithm (PASA) for Cloud Application / International Journal of Grid and High Performance Computing, Vol. 9, Issue 3, pp. 1-16, 2017.

14. *K. Kanagala and K. Sekaran* An approach for dynamic scaling of resources in enterprise cloud / in Proceedings of IEEE 5th International Conference on Cloud Computing Technology and Science, Vol. 2, Bristol, UK, Dec 2013, pp. 345–348.

15. *Brown, Robert Goodell* Smoothing Forecasting and Prediction of Discrete Time Series / 1963, Englewood Cliffs, NJ: Prentice-Hall, USA, pp. 99-104.

16. NIST/SEMATECH e-Handbook of Statistical Methods, 6.4.3.3. Double Exponential Smoothing [Online]. Available: https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc433.htm

17. *Brown, Robert Goodell* Smoothing Forecasting and Prediction of Discrete Time Series / 1963, Englewood Cliffs, NJ: Prentice-Hall, USA, pp. 128-132.

18. Proactive autoscaler for Kubernetes [Online]. Available: https://github.com/dimamon/proactive-scaler

19. Minikube project on GitHub [Online]. Available: https://github.com/kubernetes/minikube

20. Apache JMeter [Online]. Available: https://jmeter.apache.org