UDC 004.94

PROACTIVE HORIZONTAL SCALING METHOD FOR KUBERNETES

Rolik O. I. – Dr. Sc., Professor, Head of the Department of Information Systems and Technologies, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine.

Omelchenko V. V. – Postgraduate student of the Department of Information Systems and Technologies, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine.

ABSTRACT

Context. The problem of minimizing redundant resource reservation while maintaining QoS at an agreed level is crucial for modern information systems. Modern information systems can include a large number of applications, each of which uses computing resources and has its own unique features, which require a high level of automation to increase the efficiency of computing resource management processes.

Objective. The purpose of this paper is to ensure the quality of IT services at an agreed level in the face of significant dynamics of user requests by developing and using a method of proactive automatic application scaling in Kubernetes.

Method. This paper proposes a proactive horizontal scaling method based on the Prophet time series prediction algorithm. Prometheus metrics storage is used as a data source for training and validating forecasting models. Based on the historical metrics, a model is trained to predict the future utilization of computation resources using Prophet. The obtained time series is validated and used to calculate the required number of application replicas, considering deployment delays.

Results. The experiments have shown the effectiveness of the proposed proactive automated application scaling method in comparison with existing solutions based on the reactive approach in the selected scenarios. This method made it possible to reduce the reservation of computing resources by 47% without loss of service quality compared to the configuration without scaling.

Conclusions. A method for automating the horizontal scaling of applications in Kubernetes is proposed. Although the experiments have shown the effectiveness of this solution, this method can be significantly improved. In particular, it is necessary to consider the possibility of integrating a reactive component for atypical load patterns.

KEYWORDS: dynamic resource provisioning, Kubernetes, autoscaling, horizontal scaling, proactive scaling, Prophet, Horizontal Pod Autoscaler.

ABBREVIATIONS

CPU is central processor unit;
HPA is horizontal pod autoscaler;
HTTP is hypertext transfer protocol;
LSTM is namely long-term short-term memory;
MAPE is mean average percentage error;
QoS is quality of service;
RAM is random access memory.

NOMENCLATURE

C_x is a x-th type of computing resource, such as CPU time or RAM;

R_{Cx} is a deployment request for computing resource C_x;

$I_{MAX}(t)$ is a a function of the number of deployment instances with regard to all requested computational resources;

$I_{Cx}(t)$ is a a function of the number of deployment instances in the context of the computing resource C_x;

$W_{Cx}(t)$ is a a function of the total workload (actual usage) of the resource C_x;

D_{UP} is an upscaling delay for a deployment;

D_{DOWN} is a downscaling delay for a deployment.

INTRODUCTION

The emergence and use of orchestrators such as Kubernetes, Nomad, EC2, and others for managing IT infrastructure resources has dramatically simplified many aspects of deploying and managing computing resources in cloud environments. These tools take developers to the next level of abstraction with new challenges, including compute resource management. Under-provisioning of computing resources can lead to a deterioration in QoS. Meanwhile, over-provisioning wastes both computing and financial resources that could be used to solve other computing tasks. Balancing these two aspects is the task of dynamic resource allocation [1]. Modern information systems can include thousands of different applications, each with its own unique features and resource requirements, which makes this task much more difficult. The reactive scaling approach, when resources are added after QoS constraints are violated, is an essential part of any IT infrastructure management system. Nevertheless, the reactive approach has a number of disadvantages associated with the irrational use of computing resources and systematic violation of QoS requirements. These problems are solved by using proactive management methods, when the amounts of computing resources required for the operation of applications in accordance with the defined QoS constraints are managed in advance, taking into account the dynamics of changes in the values of QoS indicators [2].

The object of study is the management of computing resources in information systems to maintain the quality of services provided by applications deployed in the IT infrastructure at an agreed level.

The subject of study is a method of proactive horizontal scaling of computing resources allocated to applications.

The purpose of the work is to develop a method and technical solution for automating proactive horizontal scaling in the Kubernetes environment to maintain QoS at an agreed level. This solution should be universal,

DOI 10.15588/1607-3274-2024-1-20

namely, without the need for significant manual configuration, without the requirement for significant prior knowledge of the features of the applications and their resource requirements, and with the ability to adapt to the features of each application in an automatic mode.

1 PROBLEM STATEMENT

Suppose that for some application, the R_{Cx} requests for computing resources are set to constant values. This application can be horizontally scalable, and the number of application instances is determined by the function $I(t)$.

Then, to minimize the use of the computing resource C_x while maintaining a given level of service quality, the following equality must be satisfied at any time t:

$$I_{C_x}(t) = ceil\left(\frac{W_{C_x}(t)}{R_{C_x}}\right).$$

Since for the application to work correctly, it must be provided with all types of necessary computing resources, we obtain the following equality:

$$I_{MAX}(t) = ceil\left(\max\left\{\frac{W_{C_x}(t)}{R_{C_x}} : x = 1...X\right\}\right).$$

In real-world IT infrastructures, there is always a delay between determining the need to allocate additional resources after a decrease in QoS and the actual allocation of additional computing resources or application instances. Accordingly, to ensure the correct operation of the application and, therefore, ensure the specified QoS indicators, it is necessary to take into account this delay in D_{up} scaling up. However, when scaling down, resources should not be reduced in advance:

$$I_{MAX}(t) = ceil\left(\begin{cases} \max\left\{\dfrac{W_{C_x}(t+D_{up})}{R_{C_x}} : x = 1..X\right\}, W_{C_x}(t) \le W_{C_x}(t+D_{up}), \\ \max\left\{\dfrac{W_{C_x}(t+D_{down})}{R_{C_x}} : x = 1..X\right\}, W_{C_x}(t) > W_{C_x}(t+D_{down}) \end{cases}\right).$$

Accordingly, to obtain accurate values for the number of instances, it is necessary to obtain accurate predictions for the workload function $W_{Cx}(t)$, which includes collecting metrics, aggregating and processing them, evaluating the accuracy of the resulting prediction models, and selecting the best one. In addition, it is necessary to decide when and how to apply the obtained $I_{MAX}(t)$ values.

2 REVIEW OF THE LITERATURE

Proactive scaling methods can be divided into the following groups: threshold-based rules, reinforcement learning, queuing theory, control theory, and time series analysis [3]. The methods discussed in this paper are based on time series analysis, so this section compares the methods of this group.

One of the concepts for predictive scaling is based on finding the most similar load pattern in the past and extrapolating it to the current state. For example, in the work [4], the authors propose a solution in which the historical time series is analyzed for patterns and, based on them, the most similar load pattern to the current one is searched for. The identified patterns may differ in scale, but the correlation between the elements of the identified pattern and the current pattern should be similar. The resulting patterns are interpolated using weighted interpolation. The most similar patterns will have the highest weight in the resulting time series. The main idea is to find the most similar load situation in the past and adapt it to the current load. Among the advantages of this method is the ability to predict non-trivial time series, in which there is no seasonality, but typical load segments can occur at random moments. The paper also compares it with other methods such as RightScale, linear regression, and autoregression. On different data, the accuracy rates were both significantly better than the alternatives and significantly worse, depending on the testing application and experimental conditions.

The next paper considers predictive scaling based on ARMA/ARIMA [5]. The authors propose an approach in which there are several levels of confidence that are selected depending on the application features. The authors also evaluate the accuracy of the resulting model and assess the impact on service quality indicators. In particular, the best obtained accuracy of MAPE is 0.09, which is approximately equal to the accuracy level of such algorithms as Prophet [6] and GreyKite [7]. In addition, in the worst-case scenario, the share of rejected requests was 5%, which is acceptable. The data used to evaluate the accuracy were taken from open sources, namely historical load data for Wikipedia. It is worth noting that the historical data does not contain complex seasonality, so the evaluation is based on a simple time series. The article also notes that finding the coefficients p and q, which specify the order of the ARIMA model, requires quite significant computing resources for this approach.

In the paper [8], the authors proposed a two-component PRESS design for predictive scaling. The first component, signature-driven, is used for load patterns that contain examples of repeating loads. Fast Fourier Transform is used to process this type of pattern, and Pearson's criterion is used to compare the similarity of current and past examples. If the criterion does not provide the required similarity index, the second component, state-driven, is used. This component is based on Markov chains, where all possible variants are evenly distributed among a given number of baskets. After that, a graph of possible states and a probability matrix are built. The authors suggest that this solution can be easily scaled and is suitable for massive systems.

In [9], neural networks, namely LSTM are used to manage the quality of services. LSTM networks are a particular type of recurrent neural networks capable of learning long-term dependencies. The main feature of LSTM networks is their ability to find and store information over long sequences or periods of time, which makes LSTM networks good at solving time series forecasting tasks. In addition, this algorithm is supplemented with Reinforcement Learning to obtain more accurate predictions. The proposed solution is tested on NASA datasets and shows better accuracy compared to linear approaches and LSTM-based approaches without augmentation.

The solution proposed in this paper does not require significant computing resources for its operation, which

DOI 10.15588/1607-3274-2024-1-20

allows it to handle a large number of loads in a cluster. It also supports working with time series containing complex seasonality, trends, and anomalies. The developed solution is integrated into the Kubernetes ecosystem, which allows to test its effectiveness in a real environment and compare it with existing solutions.

3 MATERIALS AND METHODS

Modern IT infrastructure management systems use various platforms for orchestrating containerized applications. The most common orchestration platform is Kubernetes, which provides tools for flexible management of computing resources. A Kubernetes cluster operates on nodes, each of which has its own set amount of computing resources $\{C_1, C_2, \ldots, Cn\}$, including CPU time and memory. Applications in the form of deployments contain information about requests $\{R_{C1}, R_{C2}, \ldots, R_{Cx}\}$ for the resources required for their correct operation. The specification of each deployment also contains the required number of application instances. Kubernetes places application instances in the form of pods on the cluster nodes in such a way that the sum of all requests of the placed instances does not exceed the total volume of the nodes. This ensures that each instance of the application will have enough resources to operate and ensure the specified QoS indicators. Suppose an instance uses more of a specific C_x resource than specified in the R_{Cx} specification. In that case, it will either crash or be artificially slowed down, depending on the type of resource.

Horizontal scaling is the adaptation of the number of deployment instances depending on the current demand for computing resources. By increasing the number of instances, the overall capacity to process web requests or tasks from the queue increases. Adding a new instance of an application can require significant D_{up} time, which includes network latency, image look-up and downloading, scaling the Kubernetes cluster itself, and application initialization. A proactive approach to QoS management, with predictions for the workload functions $W_{Cx}(t)$, has the ability to scale deployments in advance to maintain the required QoS level.

Without losing the meaning of the research and to simplify the experiments and presentation of the results, this paper considers only the management of the CPU time of computing resources. In general, this solution has the ability to work with any metric, such as memory and network bandwidth.

Figure 1 shows a structural simplified diagram of the integration of the proposed solution into a Kubernetes cluster. This diagram shows a user sending a request to an application hosted in the cluster. A header-based network proxy redirects the request to the target application's balancer, which distributes all incoming requests among the deployment instances. When a new application instance is added, this balancer ensures that it is included in the load balancing processing without additional network settings.

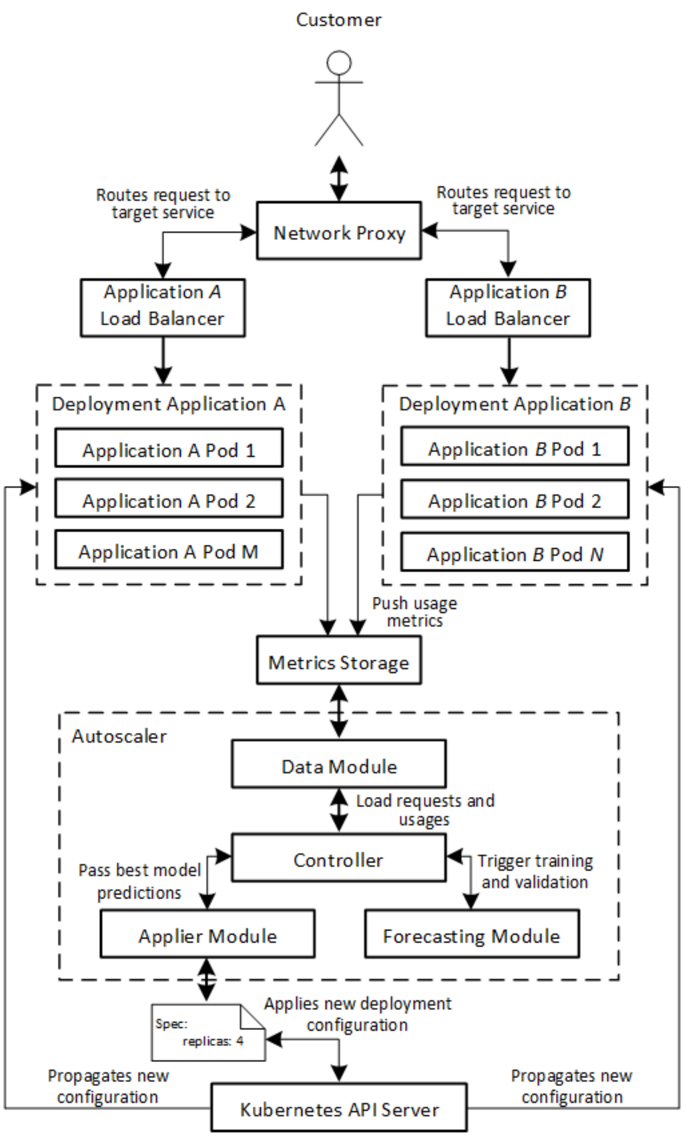

Figure 1 – Structure diagram of integration in Kubernetes

Processing a request requires the use of computing resources, such as CPU time, memory, or disk. At regular intervals, a monitoring system in Kubernetes, such as Prometheus, collects data on the total usage of computing resources from the pods. After aggregation, this information is stored in a special storage – metrics storage – for time series and is available for analysis.

The solution proposed in this paper relies on Prometheus as a source of historical data. Prometheus is a de facto standard in Kubernetes and offers long-term data storage and a specialized language for querying and aggregating data.

Having historical data about the application workload, it is necessary to calculate future values of the workload in order to set the required number of application replicas. Therefore, a crucial part of the proposed solution is a prediction algorithm. The modern generation of prediction algorithms, such as Prophet, Greykite, and TBATS, are accurate and capable of detecting seasonality, trends, and anomalies in an automatic mode. For this work, Prophet

DOI 10.15588/1607-3274-2024-1-20

was chosen, but any other method from the list can be used in the future.

Prophet is a time series prediction library developed by Facebook [7]. The main goal of the development was to create a simple, transparent, and understandable model generation algorithm that would simplify getting reliable predictions quickly. Prophet provides convenient tools for analyzing time-series and cross-validating the resulting model and has a user-friendly API.

The Forecasting module is responsible for the work with predictions and provides a unified API regardless of which prediction algorithms are used. This module requires the history of computing resource usage for previous periods of operation as input. In the current implementation, historical data is divided into training and validation data. On the training data, a set of models is trained, each with different input parameters. Then, the accuracy of the predictions is checked on the validation data. First, this allows us to choose the model with the best parameters and, accordingly, with the best accuracy. Secondly, this approach allows us to assess the accuracy of the final model as a whole and the appropriateness of its application.

When the proposed solution is initialized to automate the scaling of the target application, there may be no previous usage metrics. In this case, the proposed solution sets the number of `defaultReplicas` replicas set by the administrator until a reliable prediction can be obtained. Reliability of resource utilization prediction is calculated using the mean absolute percentage error. If the model error is less than the confidence parameter, then the proposed solution, relying on the obtained values, sets the calculated optimal number of replicas by changing the replicas field in the Kubernetes deployment manifest [10].

Historical data is obtained through requests to the Prometheus server, which is the data module's responsibility. To get historical CPU usage metrics, the query of the form `sum(rate(container_cpu_usage_total {container!=""}[60s])) by (pod)` is used. To get the current specified requests, the Kubernetes API is used, namely the `spec.container[0].requests.cpu` field in the deployment manifest. The selected Prophet prediction algorithm is capable of automatically detecting seasonality, trends, and anomalies in time series, so it does not require additional configuration. However, the administrator can specify base seasonalities with dedicated configuration parameters to improve the accuracy of predictions. The model is trained and evaluated with a specified `trainEvaluatePeriod` period. When the model's accuracy drops, this solution sets the number of replicas to `defaultReplicas` until the required accuracy is obtained. Internal or external load anomalies, incorrect operation of the monitoring subsystem, or network problems in the cluster can cause accuracy drops. This approach ensures the correct operation of the application until it is possible to get accurate predictions of resource utilization again.

When designing a horizontal scaling solution, it is necessary to take into account that applications need some time to initialize. For example, in the case of a Redis database, the application needs to read the last snapshot into memory and establish connections to other cluster components. In addition, after the command to increase the number of application replicas, it takes some time to initialize the pod, namely, to deploy it on an available node, download the image, and connect the volumes. Therefore, the proposed solution has an additional parameter, `applicationTimeToStart`, to accurately calculate the moment when it is necessary to increase the number of application replicas. In addition, the process of determining the `applicationTimeToStart` parameter for an application in a cluster can be automated.

Accordingly, the proposed solution checks the need to scale the application at short intervals. When scaling up, the Applier Module component checks the predicted utilization values at the `currentTime + applicationTimeToStart` time to set the required number of replicas in advance and not affect the QoS performance. When scaling down, only the value at the current time is checked so as not to reduce the number of replicas ahead of time. Therefore, we have a simplified formula for calculating the required number of replicas based on the forecast – `max(forecastedUsage[currentTime + timeToStart], actualUsage[currentTime])`.

This solution is placed in a Kubernetes cluster as a deployment and has direct access to the Prometheus server and the Kubernetes API using the Data Module.

4 EXPERIMENTS

The proposed solution is compared with two other configurations for resource management.

The first configuration uses HPA to compare a proactive and reactive approach. In order for the comparison to be valid, it is necessary to minimize the delays related to collecting system metrics.

The second configuration is to provide the test application with the required amount of resources and compare whether the developed solution affects the QoS indicators.

To test the resulting solution and compare it with other configurations, we use a Kubernetes cluster based on minikube [11]. This cluster includes a single node that has six processor cores with a clock frequency of 3.6GHz and 16 gigabytes of RAM. Any network communication does not go beyond a single machine. This cluster contains an installed Prometheus for monitoring using the kube-prometheus stack.

A scenario with a periodic load was chosen for testing. The load period is 300 seconds. The total minimum load is 100 millicores or 100m, the maximum is 550m.

Load testing is performed using the specialized locust utility [12] for sending HTTP requests, an instance of which is also deployed in the cluster. The load pattern of requests consists of periodic oscillations. The oscillation period is 300s, the minimum request frequency is ten requests per second, and the maximum is 50.

The selected test application can be scaled both horizontally and vertically. The test application is a web server that performs some CPU-intensive work for each request. The number of application replicas or the amount of allocated resources does not affect the application's performance. The application is initialized for a specified

DOI 10.15588/1607-3274-2024-1-20

time before becoming available for requests and before the readiness probe is considered successful. In our experiments, this time is 5s.

In this work, scaling is performed only for the CPU resource. The request for this resource for the test application is 200m. If this value is exceeded, trotting will be applied to the application, which may affect the test results. A limit of 250m was also set for this resource. The initial number of replicas is five.

HPA is a built-in solution for automating horizontal scaling in Kubernetes. HPA is a representative of the reactive approach, so it does not contain any prediction algorithms. The concept is to maintain the value of the average load per instance – `targetAverageUtilization` – set by the administrator by adjusting the number of replicas. The main difference is that the decision to scale is made based on the current values of computing resource utilization.

Since the reactive approach is highly dependent on delays in obtaining current data, it is necessary to minimize this impact on the results of the experiment. HPA uses metrics-server as a data source, so the resolution setting for metrics-server is set to the lowest possible value – 15s. Also, the sync-period, cpu-initialization-period, initial-readiness-delay settings were set to the minimum value to speed up the HPA's response time to changes. In addition, the limits for downscaling and upscaling rates were removed in the behavior scaling policies.

5 RESULTS

Figure 2 shows the results of the test scenario without using automatic scaling. This pair of graphs contains data on the actual total utilization, CPU time reservation, and the resulting request processing time. In this case, the application has enough instances to process the received requests. At the maximum load, the total actual CPU time usage is 570m. At the same time, the response time at peak times increases from 12 to 90 milliseconds, which is actually a QoS indicator in this experiment.

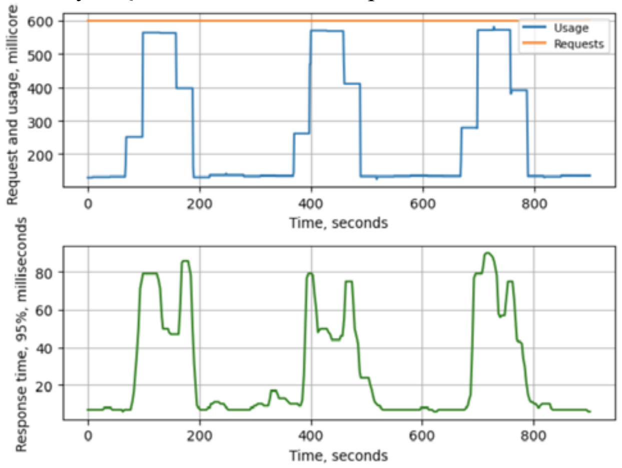

Figure 2 – No-autoscaling result

Figure 3 shows the results of testing the developed solution. This graph shows that the response time of the test application is similar to the first experiment – from 12 to 90 milliseconds so that this behavior can be interpreted as a feature of the application. Upscaling occurred ahead of time, and downscaling occurred after the peak load was passed.

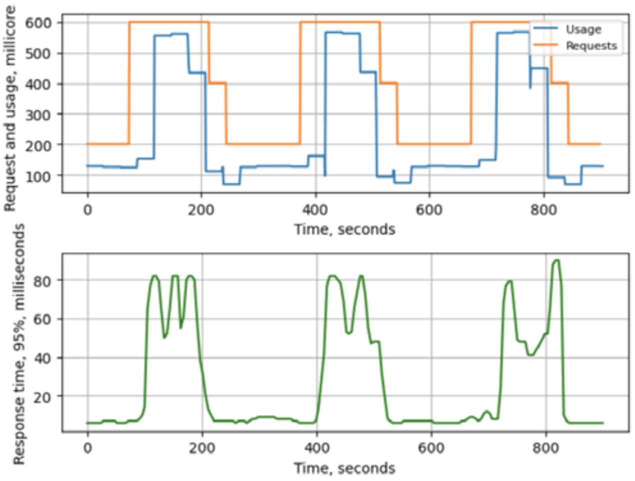

Figure 3 – Proactive scaling results

Figure 4 shows the results of HPA testing with the configuration described earlier. It should be noted that the metric for reserved resources is the sum of the requests of all pods that are in the Ready status. Therefore, in this figure, you can see that there is a slight delay between the increase in actual resource utilization and the resource provided. Because of this, there is a short-term deterioration in QoS at the time of this delay, namely an increase in response time from 90 in previous experiments to 1700 milliseconds.

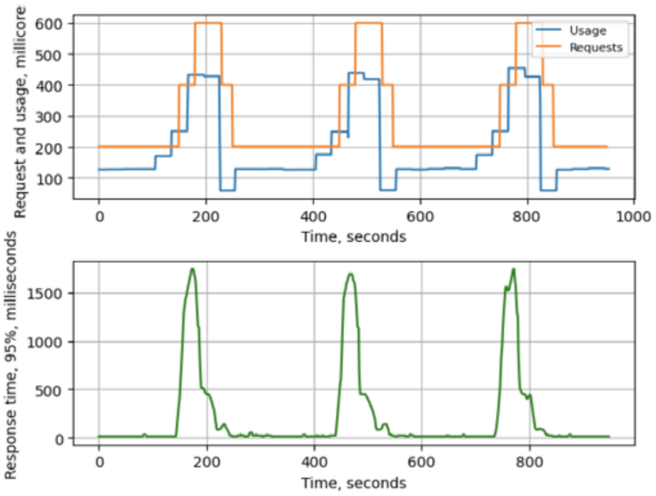

Figure 4 – Reactive scaling results

Figure 5 compares response times for all three configurations.

© Rolik O. I., Omelchenko V. V., 2024
DOI 10.15588/1607-3274-2024-1-20

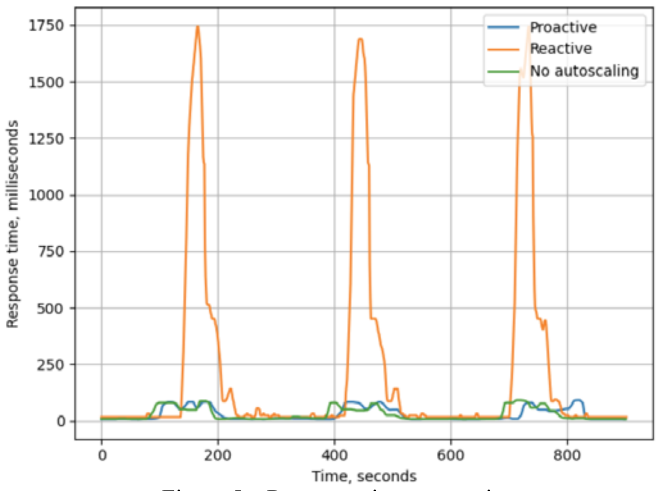

Figure 5 – Response time comparison

Table 1 shows the average response time and the 95th and 99th percentiles. In particular, on average, all response times for the proactive and reactive configurations differ by a factor of 8 in favor of the former but reserve 26% more resources. Also, the response times for this solution and management without autoscaling are similar, but the resource reservation is reduced by 47%. It is worth noting that this experiment's conditions aim to demonstrate the advantage of the proactive approach under fast-growing load patterns, which is a significant problem with the reactive approach.

Table 1 – Response time comparison

Approach	**Response time**		
	Average, ms	***95%, ms***	***99%, ms***
Proactive	23	87	190
No autoscaling	22	88	188
Reactive	160	770	1350

6 DISCUSSION

The results obtained indicate the efficiency and effectiveness of the developed solution. However, the experiments were conducted for one type of load – processor time. Future studies should also take into account more complex patterns, such as those containing several seasonalities or trends.

Also, this solution has not been tested for another essential computing resource, memory. This type has its own specifics of resource allocation since if the set requests or limits are exceeded, the application may crash. In addition, unlike CPU time, some part of the memory is used to store the code and initial data for work regardless of the load. This means that the calculation of the total amount of memory during horizontal scaling should include the constant component described above.

In the above architecture, the D_{UP} and D_{DOWN} delays are set by the user, but these parameters can also be determined from historical data.

Vertical and hybrid scaling can also be built based on this architecture since the main components – data modules, prediction, and application – are similar.

CONCLUSIONS

In this paper, we have developed a solution for predictive horizontal scaling in Kubernetes. The obtained experimental results allow us to conclude that in the selected scenarios, the developed architecture allows to significantly reduce the reservation of computing resources while maintaining a high level of QoS compared to HPA. That is, the proposed method uses computing resources more efficiently.

The scientific novelty. We have proposed a relatively simple architecture for horizontal scaling in Kubernetes, which can be easily adapted to different types of loads or types of scaling. In addition, the use of new time series prediction methods for processing workloads was proposed.

The practical orientation of the study. The main part of this research is the development of an automated subsystem for horizontal scaling in Kubernetes. The resulting solution is a ready-to-use component that is fully integrated into the orchestrator ecosystem.

Prospects for further research. In future research, it makes sense to consider integrating the proposed method with a reactive component. In addition, this architecture can be used for vertical and hybrid scaling.

REFERENCES

1. Rolik O. I., Telenik S. F., Yasochka M. V. Upravlinnya korporativnoyu infrastrukturoyu. Kyiv, Naukova Dumka, 2018, 576 p.
2. Omelchenko V. V., Rolik O. I. Automation of resource management in information systems based on reactive vertical scaling, *Adaptive systems of automatic control*, 2022 Vol. 2, No. 4, pp. 65–78. DOI: 10.20535/1560-8956.41.2022.271344.
3. Lorido-Botran T., Miguel-Alonso J., Lozano J. A. A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments, *Journal of Grid Computing: Springer Science and Business Media LLC,* 2014,Vol. 12, No. 4, pp. 559–592. DOI: 10.1007/s10723-014-9314-7.
4. Caron E., Desprez F., Muresa A. Pattern Matching Based Forecast of Non-periodic Repetitive Behavior for Cloud Clients, *Journal of Grid Computing: Springer Science and Business Media LLC*, 2011, Vol. 9, No. 1, pp. 49–64. DOI: 10.1007/s10723-010-9178-4.
5. Calheiros R. N., Masoumi E., Ranjan R., Buyya R. Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS, *IEEE Transactions on Cloud Computing: Institute of Electrical and Electronics Engineers,* 2015, Vol. 3, No. 4, pp. 449–458. DOI: 10.1109/tcc.2014.2350475.
6. Hosseini R., Chen A., Yang K., Patra S. Greykite: Deploying Flexible Forecasting at Scale at LinkedIn, *arXiv.* 2022. DOI: 10.48550/ARXIV.2207.07788.
7. Taylor S. J., Letham B. Forecasting as scale, *PeerJ,* 2017. DOI: 10.48550/ARXIV.2111.15397.
8. Zhenhuan G., Xiaohui G., Wilkes J. PRESS: PRedictive Elastic ReSource Scaling for cloud systems, *2010 International Conference on Network and Service Management. IEEE,* 2010. DOI: 10.1109/cnsm.2010.5691343.
9. Zhong J., Duan S., Li Q. Auto-Scaling Cloud Resources using LSTM and Reinforcement Learning to Guarantee Service-Level Agreements and Reduce Resource Costs,

DOI 10.15588/1607-3274-2024-1-20

Journal of Physics: Conference Series, 2019, Vol. 1237, No. 2, pp. 22–33. DOI: 10.1088/1742-6596/1237/2/022033.
10. "Deployment Controllers" Kubernetes Documentation. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/.
11. "Minikube Documentation," Minikube Documentation. [Online]. Available: https://minikube.sigs.k8s.io/docs/.
12. Locust, "Locust GitHub Repository," GitHub. [Online]. Available: https://github.com/locustio/locust.

Received 07.12.2023.
Accepted 12.02.2024.

УДК 004.94

МЕТОД ПРОАКТИВНОГО ГОРИЗОНТАЛЬНОГО МАСШТАБУВАННЯ В KUBERNETES

Ролік О. І. – д-р техн. наук, професор, завідувач кафедри інформаційних систем та технологій, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна.

Омельченко В. В. – аспірант кафедри інформаційних систем та технологій, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна.

АНОТАЦІЯ

Актуальність. Інформаційні системи можуть включати велику множину додатків, кожен з яких використовує обчислювальні ресурси та має свої унікальні особливості роботи, що вимагає високого рівня автоматизації задля підвищення ефективності виконання процесів управління обчислювальними ресурсами. Проблема мінімізації збиткового резервування ресурсів при підтриманні показників QoS на узгодженому рівні є важливою для сучасних інформаційних систем.

Мета роботи. Метою даної роботи є забезпечення якості IT-сервісів на узгодженому рівні в умовах суттєвої динаміці запитів користувачів шляхом розробки та використання методу проактивного автоматичного масштабування додатків в Kubernetes.

Метод. В даній роботі пропонується метод проактивного горизонтального масштабування на основі алгоритму передбачення часових рядів Prophet. Як джерело даних пропонується використовувати сховище метрик Prometheus. На основі історичних метрик використання обчислювальних ресурсів отримується модель для передбачення майбутніх об'ємів використання за допомогою Prophet. Отримані значення валідуються, після чого застосовуються для обрахунку необхідної кількості реплік додатку з врахуванням затримок розгортання подів.

Результати. Проведені досліди показали ефективність запропонованого методу для проактивного автоматичного масштабування додатків у порівнянні з існуючими рішеннями з використанням реактивного методу в обраних сценаріях. Даний метод дозволив зменшити резервування обчислювальних ресурсів на 47% без втрати в якості обслуговування у порівнянні з конфігурацією без масштабування.

Висновки. Запропоновано метод автоматизації горизонтального масштабування додатків в Kubernetes. Хоча проведені досліди показали ефективність даного рішення, даний метод може бути значно доповнений. Зокрема, необхідно розглянути можливість інтеграції реактивної складової для нетипових шаблонів навантаження.

КЛЮЧОВІ СЛОВА: динамічне виділення ресурсів, Kubernetes, автомасштабування, горизонтальне масштабування, проактивне масштабування, Prophet, Horizontal Pod Autoscaler.

ЛІТЕРАТУРА

1. Ролік О. І. Управління корпоративною інфраструктурою / О. І. Ролік, С. Ф. Теленик, М. В. Ясочка. – Київ, Наукова Думка, 2018. – 576 с.
2. Omelchenko V. V. Automation of resource management in information systems based on reactive vertical scaling / V. V. Omelchenko, O. I. Rolik // Adaptive systems of automatic control. – 2022 – Vol. 2, No. 4. – P. 65–78. DOI: 10.20535/1560-8956.41.2022.271344.
3. Lorido-Botran T. A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments / T. Lorido-Botran, J. Miguel-Alonso, J. A. Lozano // Journal of Grid Computing: Springer Science and Business Media LLC. – 2014. – Vol. 12, No. 4. – P. 559–592. DOI: 10.1007/s10723-014-9314-7.
4. Caron E. Pattern Matching Based Forecast of Non-periodic Repetitive Behavior for Cloud Clients / E. Caron, F. Desprez, A. Muresa // Journal of Grid Computing: Springer Science and Business Media LLC – 2011. – Vol. 9, No. 1. – P. 49–64. DOI: 10.1007/s10723-010-9178-4.
5. Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS / [R. N. Calheiros, E. Masoumi, R. Ranjan, R. Buyya] // IEEE Transactions on Cloud Computing: Institute of Electrical and Electronics Engineers. – 2015. – Vol. 3, No. 4. – P. 449–458. DOI: 10.1109/tcc.2014.2350475.
6. Greykite: Deploying Flexible Forecasting at Scale at LinkedIn / [R. Hosseini, A. Chen, K. Yang, S. Patra] // arXiv. – 2022. DOI: 10.48550/ARXIV.2207.07788.
7. Taylor S. J. Forecasting as scale / S. J. Taylor, B. Letham // PeerJ. – 2017. DOI: 10.48550/ARXIV.2111.15397.
8. Zhenhuan G. PRESS: PRedictive Elastic ReSource Scaling for cloud systems / G. Zhenhuan, G. Xiaohui, J. Wilkes // 2010 International Conference on Network and Service Management. IEEE. – 2010. DOI: 10.1109/cnsm.2010.5691343.
9. Zhong J. Auto-Scaling Cloud Resources using LSTM and Reinforcement Learning to Guarantee Service-Level Agreements and Reduce Resource Costs / J. Zhong, S. Duan, Q. Li // Journal of Physics: Conference Series. – 2019 – Vol. 1237, No. 2. – P. 22–33. DOI: 10.1088/1742-6596/1237/2/022033.
10. "Deployment Controllers" Kubernetes Documentation. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/.
11. "Minikube Documentation," Minikube Documentation. [Online]. Available: https://minikube.sigs.k8s.io/docs/.
12. Locust, "Locust GitHub Repository," GitHub. [Online]. Available: https://github.com/locustio/locust.

DOI 10.15588/1607-3274-2024-1-20