UDC 004.94

V. Omelchenko, O. Rolik

AUTOMATION OF RESOURCE MANAGEMENT IN INFORMATION SYSTEMS BASED ON REACTIVE VERTICAL SCALING

Abstract: The rapid development of information systems creates new challenges for developers. The problem of minimizing the use of IT infrastructure resources while ensuring the agreed level of service is one of the critical ones in the existing conditions. The article is devoted to the research of existing automation methods of resource management and QoS indicators. Capabilities of managing computing resources in Kubernetes and in general are analyzed. The work provides a detailed analysis of reactive and proactive approaches, their advantages and disadvantages. Considerable attention is paid to vertical scaling, in particular to the open-source Vertical Pod Autoscaler solution. A series of experiments was performed to analyze the effectiveness of this approach in various conditions. Based on the results, the appropriate use cases for using VPA were determined. In addition, the work proposes a hybrid approach, which includes a reactive and proactive component, which allows you to use the advantages of both methods.

Keywords: IT infrastructure, resource management, QoS, quality of service, vertical scaling, horizontal scaling, Kubernetes, Vertical Pod Autoscaler

Introduction

Nowadays, many companies prefer to place their information resources in cloud platforms instead of creating and supporting their own physical IT infrastructure. This approach has numerous advantages, including the possibility of easy and flexible scaling, which is a critically important attribute of modern information systems. However, the cost of using IT services provided by cloud service providers is constantly increasing. Therefore, the problem of minimizing financial costs arises when companies try to reduce their expenses on information services without reducing the quality of these services. At the same time, the high cost of computing resources and the maintenance of data centers (DC) force cloud service providers to use IT infrastructure management systems (IMS) [1], which allow efficient use of IT infrastructure resources.

A large number of information technologies, models and management methods for the allocating and reallocating of computing resources are used in modern IMS, including management of the creation of virtual machines (VMs), operational management of VM resources, their migration without downtimes and degradation of the quality of the services they provide [2]. In addition, IMS constantly solves the problem of dense filling of physical servers or cluster nodes so that the servers released as a result of such optimization can be put into standby mode or turned off. An equally important task to be solved by the IMS is the management of the quality of services provided by the IT infrastructure [1]. The IMS should maintain the quality of services at a stable level agreed with users, while spending the minimum amount of IT infrastructure resources.

The evolution of models and methods of managing IT infrastructure resources and the provided quality of services has gone through a number of stages. At the first stage, models and methods that were developed in scientific institutions were used in experimental IMSs [1]. At the second stage, IMSs were built on the basis of proprietary information technologies, which were created by R&D divisions of companies and had a closed nature. Currently, specialized open-source tools or universal systems are widely used in IMSs. Due to the popularity of the containerization paradigm today, a large number of orchestrators such as Kubernetes, ECS, Nomad have emerged and are widely used to manage containerized applications. These solutions allow us to manage QoS (quality of service) by maintaining it at the agreed level and with minimal use of DC resources [3]. Analysis of management methods and algorithms used by orchestrators allows us to determine the expediency of using these methods and algorithms or the necessity of their modernization.

One of the main approaches to manage the quality of services provided by containerized applications is vertical and horizontal scaling. Autoscaling allows you to adjust the level of QoS and the amount of reserved computing resources. In addition, containers should be placed as densely as possible for the most effective utilization of computing resources [4].

The purpose of this work is to explore the efficiency of scaling algorithms provided by orchestrators when solving the problems of managing QoS.

Analysis of Kubernetes cluster capabilities in resource management of containerized applications

Kubernetes is an open source platform for managing workloads and applications. It is one of the most universal and wide-spread solutions to manage containerized applications. The functionality of this system is very flexible and allows you to automate the processes of load balancing, as well as the deployment and scaling of applications, manage data stores and access permissions for them, and solve many other tasks that should be solved by IMS. Kubernetes manages clusters of Linux containers hosted on a group of virtual or physical machines as a single system. Each cluster node has special software for interaction with other system components.

Resources $R = \{R_j\}, \quad j = \overline{1, M},$ managed by Kubernetes are distributed among M clusters.

The structural unit of each *j-th*, $j=\overline{1,M}$, cluster is a node. Each node of the *j-th* cluster has a certain amount of its own resources $\{r_{ij}\}$, where r_{ij} is the amount of resources of the *i-th*, node belonging to the *j-th*, $j=\overline{1,M}$, of the cluster, and *N* is the number of nodes in the *j-th*, $j=\overline{1,M}$, cluster.

Each node is characterized by a set of parameters that together determine its power. For simplicity, we will assume that each node is characterized by two parameters (α_{ij},β_{ij}), where α_{ij},β_{ij} are, respectively, the processor capacity and RAM capacity of the *i-th*, $i=\overline{1,N}$, node of the *j-th*, $j=\overline{1,M}$, cluster.

Applications from the set of $A_j=\{a_{kj}\}$, $k=\overline{1,L_j}$, are deployed on the nodes of the *j-th* cluster, where L_j is the number of containerized applications from the set A_j. Each instance of the application a_{kj}, $k=\overline{1,L_j}$, $j=\overline{1,M}$, from the set A_j has requirements for the volumes of computing resources (α_{kj},β_{kj}), where α_{kj},β_{kj} – respectively, the requirements for processor capacity and RAM capacity of the *k-th* application $k=\overline{1,L_j}$, from the set A_j. Such minimum necessary requirements for computing resources in Kubernetes are called requests.

Placing an application from the set A_j on a node of the *j-th* cluster is allowed to be done only when the node resources are sufficient for the correct functioning of the application. Conditions are described by the following formula:

$$\alpha_{kj}\le\alpha_{ij},\beta_{kj}\le\beta_{ij},k=\overline{1,L_j},j=\overline{1,M},i=\overline{1,N}, \qquad (1)$$

where α_{kj}, $k=\overline{1,L_j}$, $j=\overline{1,M}$, are requirements for a processor capacity of the application a_{kj}; β_{kj}, $k=\overline{1,L_j}$, $j=\overline{1,M}$, – requirements for the RAM capacity of the application a_{kj}; α_{ij}, $i=\overline{1,N}$, $j=\overline{1,M}$, is the processor capacity of the *i-th*, $i=\overline{1,N}$, node of the *j-th*, $j=\overline{1,M}$, cluster on which the application a_{kj} is placed, $k=\overline{1,L_j}$, $j=\overline{1,M}$; β_{ij}, $i=\overline{1,N}$, $j=\overline{1,M}$, is the RAM capacity of the *i-th*, $i=\overline{1,N}$, node of the *j-th*, $j=\overline{1,M}$, cluster, on to which is placed the application a_{kj}, $k=\overline{1,L_j}$, $j=\overline{1,M}$.

The Kubernetes functionality allows you to deploy applications from the set A_j by monitoring the fulfillment of conditions (1).

Limits can be set for applications of the set A_j. If the *j-th*, $j=\overline{1,M}$, node running the containerized instance a_{kj}, $k=\overline{1,L_j}$, $j=\overline{1,M}$, of the application from the set A_j has an excess of available resources compared to the request of the application a_{kj} for resources, the container can use more resources than specified in the request [5]. However, the application a_{kj}, $k=\overline{1,L_j}$, $j=\overline{1,M}$, cannot use more resources than specified in the limits configuration.

Management of the level of quality of services provided by containerized applications from the set A_j in the IMS is carried out by automating the scaling of the resources provided to the applications, taking into account the limits set for the applications from the set A_j, resources r_{ij}, $i=\overline{1,N}$, $j=\overline{1,M}$, nodes of the *j-th* cluster and resources R_j, $j=\overline{1,M}$, of the cluster.

Scaling in conditions of insufficient computing resources can lead to denial of service to applications from the set A_j.

Features of autoscaling of IT infrastructure resources

Each application a_{kj}, $k=1,\ldots,L_j,$, $j=\overline{1,M}$, from the set A_j has requirements for the volumes of computing resources (α_{kj},β_{kj}). These volumes are determined by the target values Q_{bk} of the service quality level parameters. Where Q_{bk}, $b=\overline{1,D}$, $k=\overline{1,L_j}$, is the target value of the quality indicator of service *b*-th, $b=\overline{1,D}$, provided by the containerized application a_{kj}, $k=\overline{1,L_j}$, and *D* is the number of quality indicators.

In the IMS during QoS management, the actual values q_{bkj} of the service quality level parameters by the application a_{kj}, $k=\overline{1,L_j}$, $j=\overline{1,M}$, are measured. Where q_{bkj}, $b=\overline{1,D}$, $k=\overline{1,L_j}$, $j=\overline{1,M}$, is the actual value of *b*-th, $b=\overline{1,D}$, the service quality indicator provided by the containerized application a_{kj}, $k=\overline{1,L_j}$, $j=\overline{1,M}$, and *D* is the number of quality indicators. The IMS performs QoS management in such a way that the conditions are met

$$q_{bkj} \le Q_{bk}, b=\overline{1,D}, k=\overline{1,L_j}, j=\overline{1,M}. \quad (2)$$

The number of volumes of resources (α_{kj},β_{kj}) provided to the application for the performance of target indicators Q_{bk}, $b=\overline{1,D}$, quality is determined under certain conditions. For example, the response time of the application a_{kj} for a user request, measured at the output of the IT infrastructure, which also depends on the number of user requests per unit of time. When changing the operating conditions of the application a_{kj}, for example, with a significant increase in the number of requests, the actual values q_{bkj}, $b=\overline{1,D}$, of the QoS indicators exceed the target Q_{bk}, $b=\overline{1,D}$, $k=\overline{1,L_j}$, and condition (2) ceases to be fulfilled. In order for the QoS indicators to return to the target values under the new operating conditions of the application a_{kj}, i.e. to make condition (2) fulfilled, additional $(\Delta\alpha_{kj}, \Delta\beta_{kj})$ computing resources must be added to the container of the a_{kj} application. Adding additional resources $(\Delta\alpha_{kj}, \Delta\beta_{kj})$ is performed by autoscaling of node and cluster resources.

During the process of developing autoscaling methods that will be used in IMS for QoS management two main problems must be solved. The first problem is that it is necessary to determine when and under what conditions it is necessary to scale the resources for the applications from the set A_j. The second problem is to estimate the minimum amount of additional resources $(\Delta\alpha_{kj}, \Delta\beta_{kj})$ that must be provided to applications a_{kj} from the set A_j, for which condition (2) is not fulfilled will not be fulfilled in the near future in order not to violate QoS requirements.

Autoscaling methods can be divided into two large groups – reactive and proactive.

Reactive methods are based on the constant measurement of the actual values of indicators q_{bkj}, $b = \overline{1, D}$, $k = \overline{1, L_j}$, $j = \overline{1, M}$, and monitoring the fulfillment of condition (2). If condition (2) is not fulfilled, reactive methods provide additional resources $(\Delta\alpha_{kj}, \Delta\beta_{kj})$ during vertical scaling of applications a_{kj} from the set A_j. Another option is horizontal scaling – deploying new instances of applications for which condition (2) is not fulfilled on additional nodes. If the percentage of resource usage decreases, then reverse scaling is performed. As a result, the number of additional resources $(\Delta\alpha_{kj}, \Delta\beta_{kj})$ provided to the application a_{kj} decreases to zero in the future.

Proactive methods perform QoS management in advance without waiting for condition (2) to stop being fulfilled. For this, the dynamics of changes in the actual values of indicators q_{bkj}, $b = \overline{1, D}$, $k = \overline{1, L_j}$, $j = \overline{1, M}$, the quality of functioning of all applications from the set A_j are monitored. If the dynamics is negative and QoS requirements will be violated in the near future, then additional resources $(\Delta\alpha_{kj}, \Delta\beta_{kj})$ are provided in advance, without waiting for condition (2) to stop being fulfilled. In addition, the algorithm analyzes historical data to determine the amount of additional resources $(\Delta\alpha_{kj}, \Delta\beta_{kj})$, which should be added for containers in case of non-fulfillment of conditions (2).

The proactive method, in contrast to the reactive one, allows to improve QoS management indicators, since there is no delay before scaling under typical loads. In addition, the proactive method can accurately estimate the required amount of computing resources at any time, which allows you to scale the application down faster than when using the reactive method [6]. The main disadvantage of the proactive method is its complexity. In particular, algorithms based on time series analysis and neural networks are used to implement this approach. Also, this method works well only when there are periodic overloads that can be predicted.

If it is necessary to minimize the costs of resources by reducing the volumes of reserved resources, it is advisable to use the combined method of autoscaling, when the reactive component is used to determine the moment when condition (2) ceases to be fulfilled, and the components of the proactive method are used when estimating the minimum necessary amount of additional resources $(\Delta\alpha_{kj}, \Delta\beta_{kj})$ based on the analysis of historical data. For example, this approach was used in Vertical Pod Autoscaler (VPA) of the Kubernetes platform. The combined approach can be used in QoS management only when a short-term drop in quality of service indicators or denial of service to users for some time is acceptable.

Analysis of the autoscaling algorithm in Vertical Pod Autoscaler

For effective proactive QoS management and taking into account the specifics of the operation of each application from the set A_j, the IMS must constantly receive information about the actual values of the indicators q_{bkj}, $b=\overline{1,D}$, $k=\overline{1,L_j}$, $j=\overline{1,M}$, of the quality of services, the current value of the number of user requests to each instance of the application a_{kj}, $k=\overline{1,L_j}$, $j=\overline{1,M}$, the available use of resource volumes $(\alpha^*_{kij}, \beta^*_{kij})$ of applications from the set A_j. Here α^*_{kij} and β^*_{kij} are the actual usage, respectively, of the processor capacity and RAM capacity of the *i-th*, $i=\overline{1,N}$, by the *j-th* node, $j=\overline{1,M}$, of the cluster on which the application a_{kj}, $k=\overline{1,L_j}$, $j=\overline{1,M}$, is run.

The autoscaling algorithm used in VPA calculates the needs for computing resources based on the analysis of historical data of the values $(\alpha^*_{kij}(t), \beta^*_{kij}(t))$ – the use of processor time $\alpha^*_{kij}(t)$, and of memory $\beta^*_{kij}(t)$ for the previous periods of the application a_{kj}, $k=\overline{1,L_j}$, $j=\overline{1,M}$, which is installed on the *i-th*, $i=\overline{1,N}$, node of the *j-th*, $j=\overline{1,M}$, cluster. During operation, the VPA algorithm recalculates $(\alpha_{kij}, \beta_{kij})$ – target values, $(\alpha^-_{kij}, \beta^-_{kij})$ – lower and $(\alpha^+_{kij}, \beta^+_{kij})$ – upper thresholds of resource utilization container of the application a_{kij}, which is run on the *i-th* node *j-th* cluster.

The target values $(\alpha_{kij}, \beta_{kij})$ are directly applied to the configuration of the container's resources.

The lower bound $(\alpha^-_{kij}, \beta^-_{kij})$ determines the volume of resources at which it is not guaranteed that the application a_{kij} has enough resources for full operation. If the current indicators of resource utilization are less than this threshold, then VPA initiates the reconfiguration of the container with a different resource configuration in order to reduce unprofitable resource reservation.

The upper bound $(\alpha^{+}_{kij}, \beta^{+}_{kij})$ is an indicator of unprofitable resource reservation. Resources reserved above this threshold are guaranteed not to be used by an application from the set A_j, when viewed relative to historical data. Similarly to the lower threshold $(\alpha^{-}_{kij}, \beta^{-}_{kij})$, if the current resource usage indicators $(\alpha^{*}_{kij}, \beta^{*}_{kij})$ of application a_{kj} are higher than this threshold, then the VPA initiates the reconfiguration of the container with increased resources from in order to improve the stability of the application.

In VPA, resources requests and limits are calculated as a percentile relative to previously obtained values of the usage of resource volumes $(\alpha^{+}_{kij}, \beta^{+}_{kij})$.

The lower threshold $(\alpha^{-}_{kij}, \beta^{-}_{kij})$ is set at the level of 0.5. This means that at least 50% of the time the a_{kj} application had enough resources to provide quality service.

The target value $(\alpha_{kij}, \beta_{kij})$ is set at the level of 0.9, which means that with the amount of resources equal to $(\alpha_{kij}, \beta_{kij})$, 90% of the time the application a_{kj} provided quality service.

The upper threshold $(\alpha^{*}_{kij}, \beta^{*}_{kij})$ is set at the level of 0.95. That is, quality service was provided by the a_{kj} application 95% of the time.

This approach can be justified for the processor time a_{kj}, because in the case of a lack of this resource, the application a_{kj} continues to work, but more slowly, so a slight degradation of QoS indicators is possible. However, in the case of the RAM capacity resource β_{kij}, the approach based on percentile calculation can lead to unacceptable critical consequences, which will significantly affect the efficiency of service provision. The lack of RAM β_{kij} for the application a_{kj} will lead to a denial of service due to the OOM error, and if the current error is not reacted to in time, then to systematic interruptions in the provision of services by the application a_{kj}. Therefore, it is worth increasing the limits on resource requests to prevent denials of service.

The VPA algorithm gives preference to more recent data when calculating recommended resource threshold values. All values $(\alpha^{*}_{kij}(t), \beta^{*}_{kij}(t))$ of resource consumption are divided into intervals, and each interval has its own weight depending on recency when calculating a specific percentile. The weight function has the following form:

$$W(t) = 2^{(\frac{t-t_0}{h})}, \tag{3}$$

where t is the end time of the period, t_0 is the time of the beginning of the accumulation of the history of values $(\alpha^*_{kij}(t),\beta^*_{kij}(t))$ is the length of the intervals of dividing historical data.

For each interval, the weight factor (3) adjusts the time of resource usage relative to other intervals:

$$H = \sum_{i=1,\overline{M}} G_i \cdot W(t_i), \tag{4}$$

where *H* is the final set of historical data, G_i is the interval of historical data, t_i is the end time of the period of the *i-th* group.

In Fig. 1 shows the visualization of historical data on the use of some resource with $(\alpha^*_{kij}(t),\beta^*_{kij}(t))$. Time is divided into three intervals.

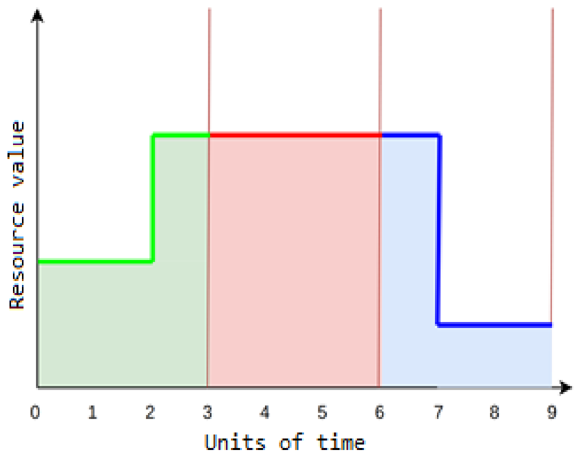

Figure 1. Raw historical data of a resource usage

After applying the weights, the historical data from the last interval has the most significant impact on the percentile calculation, and the first interval has the least impact, as shown in Fig. 2.

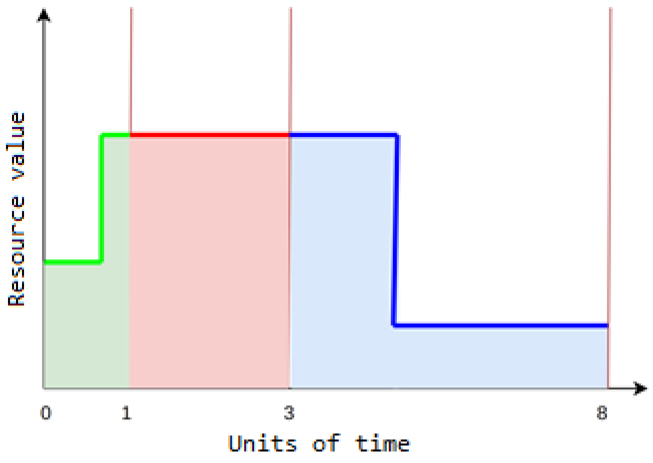

Figure 2. Transformed historical data of a resource usage

This approach has a significant drawback – in the case when the periodicity of the load is significantly higher than the duration of the period of division into intervals, then the weighting factors out the influence of, for example, past peak loads. This will cause the containers to restart repeatedly.

The efficiency of processing historical data $\alpha^*_{kij}(t)$ of the processor time usage is affected by throttling – limiting the use of the processor by some application a_{kj} in case of exceeding the set limits on resources. It is used by the operating system during distributing processor time between all applications installed on the same physical server in proportion to requests for resources.

Based on the analysis of historical data $\alpha^*_{kij}(t)$ of the processor time usage, it is not possible to clearly establish that the operation of the application is limited by the throttling mechanism, and, therefore, it is not possible to calculate the amount of processor time that is additionally required for full operation. Historical data $\alpha^*_{kij}(t)$ in this case indicates 100% use of processor time.

One possible solution to this problem is the collection and use of historical data of throttling. However, it is quite difficult to accurately estimate the required amount of processing time that should be provided by the application a_{kj} based on throttling historical data.

VPA uses the following approach to solve this problem. The upper limit is calculated as the 95th percentile of historical data. In the event that application a_{kj} needs more processor time a_{kij}, then there is a 5% margin, the use of which will affect the given 95th percentile in future recommendation calculations. Moreover, it is possible to set limits on processor time a_{kij}, greater than requests, which will make throttling more predictable and allow to use of more processor time if possible.

Study of the operation of the VPA algorithm

To determine the optimal values of resource allocation VPA algorithm relies on the analysis of actual historical data $(\alpha^*_{kij}(t), \beta^*_{kij}(t))$ of processing time $\alpha^*_{kij}(t)$ and memory $\beta^*_{kij}(t)$ produced by the application a_{kj}, $k=\overline{1,L_j}$, $j=\overline{1,M}$, which is deployed on the *i-th*, $i=\overline{1,N}$, node *of the j-th*, $j=\overline{1,M}$, cluster. At the same time, other Kubernetes work mechanisms are taken into account, especially how requests and limits work. This mechanism can limit the use of the CPU for the application, which can interfere with the calculation of the necessary amount of this resource for the full operation of the application in some cases. In this article, this mechanism is investigated experimentally.

The GKE cluster (Google Kubernetes Engine), which includes seven virtual machines of type e2-highcpu-4 (2 vCPU, 4GB). This type of machine allows you to fully perform CPUoriented tasks. The application under test is a web server that performs calculations on each request in order to load the CPU. Specialized software Locust is used for load testing, an instance of which is placed in the test cluster in order to minimize the impact of the network on the test results. The resources limits compared to the request is 30%. The *decay-half-life* period is equal to the loading period.

The operation of the algorithm is studied without edge cases – a monotonically increasing or decreasing periodic load. The load increases by 5% with each subsequent period. From the analysis of Fig. 3. it can be seen that in this case this algorithm responds in time to a gradual increase in load and provides more resources for the application.

QoS metrics – in this case, request response time – are not degraded, as the experiment has only a slight overrun of CPU requests that matches the limit values. However, if the node did not have free resources, a degradation in QoS would be possible.

Studies of the operation of the reactive VPA algorithm with the presence of periodic short-term high loads showed the same results.

The developers of the VPA algorithm tried to eliminate some of the shortcomings of the reactive approach, but this creates new problems. So, for example, one of the optimizations is designed to level atypical instantaneous high loads in addition by introducing a coefficient for the calculated recommended values. Consider a situation where the 99th percentile of application load is within 100 millicores, and the 100th percentile is 500 millicores. You can allocate 100% load for the application, but this will lead to the reservation of unprofitable resources in more than 99% of the entire working time, so it makes sense to ignore such a load, which will not lead to critical consequences, since the processor time for processing requests at moments of atypical load will be used from the next period.

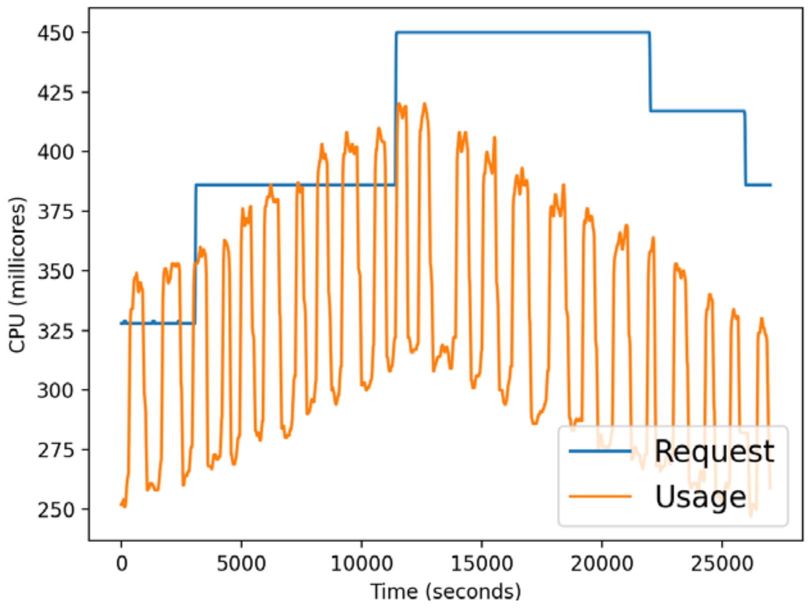

Figure 3. Results of VPA work

However, this can be a problem in the case when short-term high loads are a feature of the application and the usage value of n is much higher than the average, and the time of such loads is much shorter than the time of the minimum load. As can be seen from the analysis of Fig. 4 in such a case, the 95th percentile limit is much larger than in the case of a balanced load, which leads to a significant degradation of the quality of service indicators at such times.

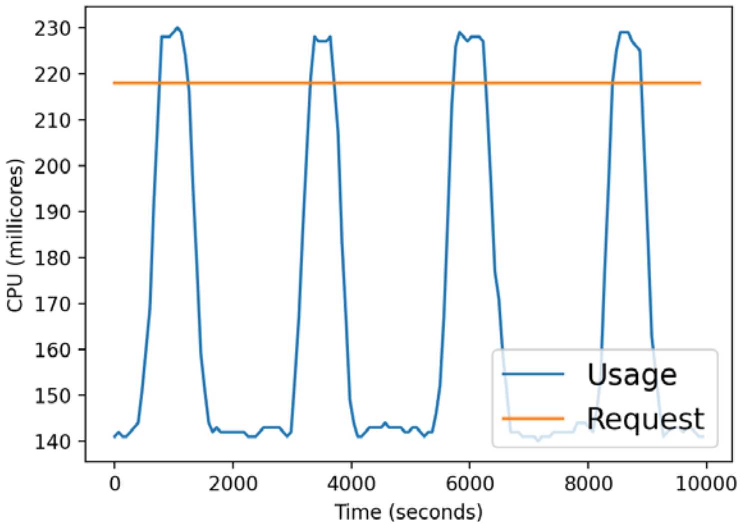

Figure 4. Operation of VPA in the presence of short-term high loads

Picture Fig. 5. shows an example of the test application, in which the load varies from 150 requests per second to 300. The peak load time is 25% of the period. Limiting the 95th percentile in this case leads to a drop in the quality of service over a fairly long period of work.

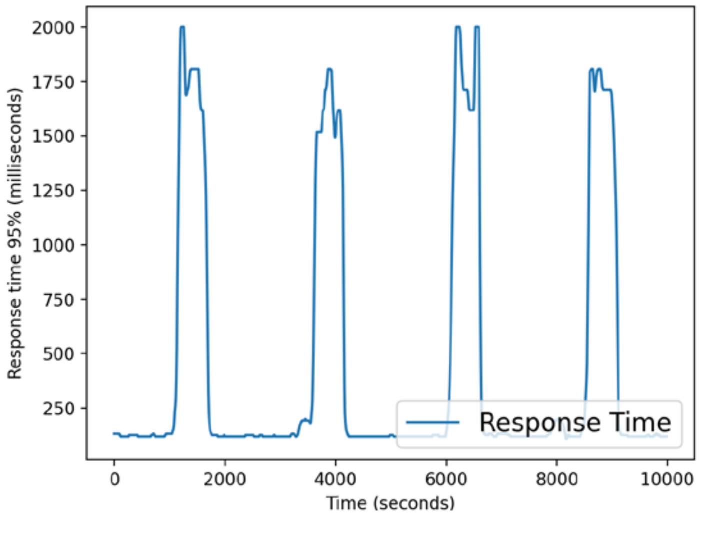

Figure 5. Response time – 95th percentile

This experiment shows that with this type of load, it is better not to use this implementation of the reactive algorithm at all or to give preference to proactive algorithms, which are more flexible in general. In particular, a proactive approach will allow you to prepare for such a load by pre-scaling the application if necessary. This is not a problem of reactive algorithms in general, but of the implementation analyzed in this article.

The research of the operation of the VPA algorithm under complex periodic loading made it possible to obtain the following results.

In this experiment, VPA has a decay-half-life equivalent to a shorter periodicity. The maximum load is 300 requests per second, and the minimum is 200 requests. In addition, as can be seen from Fig. 6 for some time there is no maximum load – this is a kind of simulation of "weekdays off".

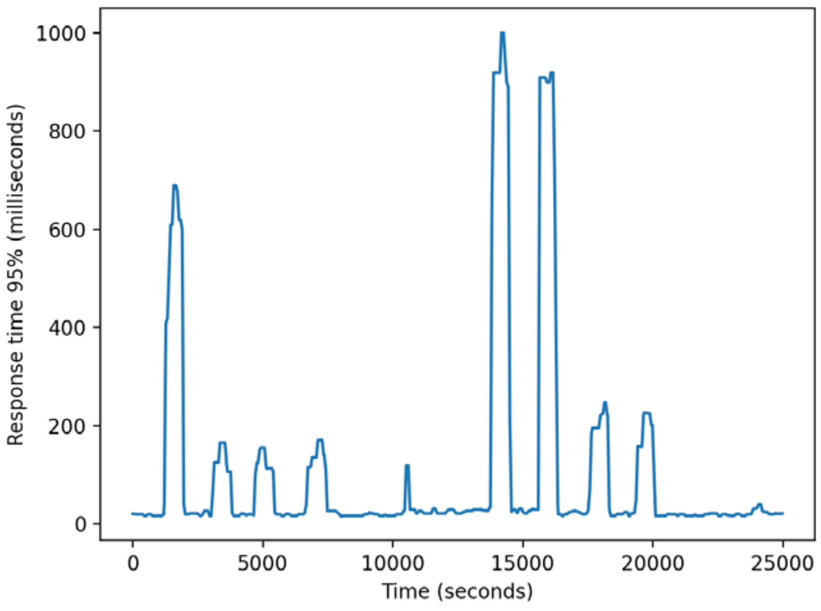

Figure 6. Reponse time – 95th percentile

In the picture Fig. 7. it can be seen that the reactive algorithm reacts in the expected way, especially, in periods of absence of high loads, it reduces requests for processing time. On the one hand, this is good, because in this way the algorithm minimizes unprofitable reservation, on the other hand, it leads to a critical drop of QoS during the period of high loads. This is clearly visible in the second period depicted in the graph, when the application does not get enough CPU time for a while.

Also from the comparison of Fig. 6 and Fig. 7 a long delay can be observed between the reduction of the load to the minimum level and the VPA response to it. This means that the resource has not been used for a long time. This is due to the fact that in this approach it is difficult to assess when it is worth reducing the number of allocated resources when load is decreasing.

In such cases, the reactive algorithm is an inappropriate solution, as it leads to systematic violations of service quality indicators.

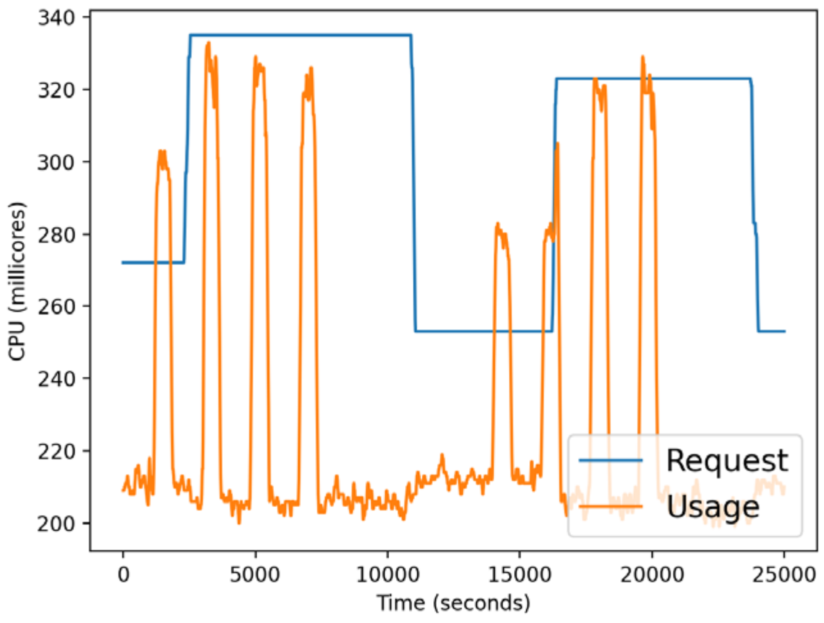

Figure 7. Results of VPA work

Conclusions

The paper analyzes approaches to automatic scaling – reactive and proactive, which are implemented in Kubernetes in Vertical Pod Autoscaler. On the basis of the conducted analysis and practical experiments, it was concluded that reactive autoscaling negatively affects the operation of the application under periodic load, in particular, when the load on the application increases, and leads to degradation of QoS indicators. In addition, there is a significant delay between the load drop and the reduction of the allocated resource in order to exclude premature reallocations at moments of temporary load drop due to external reasons. Therefore, this approach should not be used for scaling in conditions of short-term load fluctuations. This approach to managing the quality of services provided by critical IT infrastructure should be avoided.

The reactive approach is suitable for static or monotonically increasing or decreasing loads, or when the load is not periodic and its value cannot be estimated in the future. Also, this approach can be used as part of a complex proactive approach when scaling IT infrastructure resources.

REFERENCES

1. Ролик А. И. Управление корпоративной ИТ-инфраструктурой / А.И. Ролик, С.Ф. Теленик, М.В. Ясочка // К.: Наукова думка, 2018. – 576 с.

2. Rolik O., Kolesnik V., Halushko D. (2018). IT Service Quality Management Based on Fuzzy Logic. in. Proc. International Scientific-Practical Conference on Problems of Infocommunications Science and Technology, PIC S and T 2018, October 9-12, Kharkiv. – IEEE, 2018. – pp. 604–608.

3. Rolik O., Bodaniuk M., Kolesnik V., and Samotyy, V. (2017). The Algorithm for Sequential Analysis of Variants for Distribution of Virtual Machines in Data Center. FedCSIS., – Czes Republic, 3-6 Sept. 2017, pp. 183–187.

4. Rolik O., Telenyk S., Zharikov E., and Samotyy V.Dynamic Virtual Machine Allocation Based on Adaptive Genetic Algorithm.in Proc. CLOUD COMPUTING 2017: The Eighth International Conference on Cloud Computing, GRIDs, and Virtualization, February 19-23, 2017, Athens, Greece, IARIA, 2017, pp. 108-114.

5. Resource Management for Pods and Containers. Kubernetes, kubernetes.io/docs/concepts/configuration/manage-resources-containers. Accessed 23 Nov. 2022.

6. Chenhao Qu, Rodrigo N. Calheiros and Rajkumar Buyya. 2018. Auto-Scaling Web Applications in Clouds: A Taxonomy and Survey. ACM Comput. Surv. 51, 4, Article 73 (July 2019), 33 pages. https://doi.org/10.1145/3148149.

7. Martin Straesser, Johannes Grohmann, Jóakim von Kistowski, Simon Eismann, André Bauer, and Samuel Kounev. 2022. Why Is It Not Solved Yet? Challenges for Production-Ready Autoscaling (Author Preprint). In Proceedings of the 2022 ACM/SPEC International Conference on Performance Engineering (ICPE ’22), April 9–13, 2022, Bejing, China. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3489525.3511680

8. Yahya Al-Dhuraibi, Fawaz Paraiso, Nabil Djarallah, and Philippe Merle. Autonomic vertical elasticity of docker containers with elasticdocker. In 2017 IEEE 10th international conference on cloud computing (CLOUD), pages 472–479. IEEE, 2017.

9. Laura R. Moore, Kathryn Bean, and Tariq Ellahi. 2013. Transforming reactive auto-scaling into proactive auto-scaling. In Proceedings of the 3rd International Workshop on Cloud Data and Platforms (CloudDP '13). Association for Computing Machinery, New York, NY, USA, 7–12. https://doi.org/10.1145/2460756.2460758.