UDC 004.94
DOI https://doi.org/10.32782/2663-5941/2023.5/30

***Omechenko V.V.***
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"

***Rolik O.I.***
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"

# INTEGRATION OF PROACTIVE AND REACTIVE APPROACHES TO SCALING IN KUBERNETES

*This article is devoted to developing a method for integrating proactive and reactive approaches to automate scaling computing resources in a Kubernetes cluster. A proactive approach allows you to scale cluster resources in advance and release them only after passing the peak load, which reduces the risk of service quality deterioration and significantly reduces unprofitable resource reservation. However, the main drawback of this method is the inability to adapt to abnormal loads directly during operation. Although the reactive approach is less effective in typical situations, it continues functioning in the standard mode under abnormal loads. Therefore, a hybrid approach – which includes reactive and proactive components – will effectively manage resources under typical loads and continue functioning in abnormal conditions. This paper considers a solution for automatic horizontal scaling, which includes a developed proactive component in combination with an existing solution for reactive horizontal scaling in Kubernetes, namely the Horizontal Pod Autoscaler. This work describes problems that may arise when developing hybrid solutions, in particular, the problem of determining the moment of switching between components, and possible options for their solution are given. Experiments are conducted in the article to verify the developed solution. The first experiment shows the ability to move to proactive management if the required accuracy of predictions is obtained. Also, this experiment allows you to compare the proactive and reactive approaches with each other in the context of service quality and unprofitable resource reservation. The second experiment demonstrates the ability to detect abnormal workload and switch to the reactive component. The speed of reaction to anomalies and the overall impact on service quality indicators are also evaluated.*

***Key words:*** *dynamic management of computing resources, proactive scaling, reactive scaling, Kubernetes, Horizontal Pod Autoscaler.*

**Formulation of the problem.** Proactive scaling methods are more effective than reactive scaling methods in the context of managing computing resources in information systems for several reasons [1]. First, this approach allows you to scale applications in advance during peak loads by analyzing historical data, which significantly reduces the risk of deterioration in quality of service (QoS). Secondly, this approach significantly reduces overprovisioning due to a more accurate calculation of the required amount of computing resources and advance planning of downscaling. When using the reactive approach, upward scaling is performed when the load peak occurs [2]. Given the time required to deploy and initialize the application, this can lead to a temporary critical drop in service quality, including complete failure. However, the proactive approach, unlike the reactive one, is not self-sufficient and has several prerequisites for effective operation.

Any proactive scaling solution is based on time-series analysis methods that identify seasonality, trends, recurring patterns, and other features of the target application's load. The presence of anomalies in historical data, which can be caused by data loss, network problems, or denial of service, can significantly affect the accuracy of predictions. In addition, a proactive scaler cannot perform its functions when an abnormal load occurs directly during operation. One possible solution to this problem is to use reactive scaling in situations where the proactive approach is ineffective. When integrating these two approaches, several tasks need to be solved. In particular, it is necessary to define an indicator that will signal that the proactive component is ineffective at some point in time, and it is necessary to apply the reactive one [3]. It is also necessary to define an indicator for reverse switching from reactive to proactive. In addition, it is necessary to ensure that the reactive component is activated quickly to reduce the duration of QoS degradation.

This paper focuses on the problem of combining reactive and proactive approaches to

scaling. Nowadays, Kubernetes provides all the necessary tools and capabilities for automating resource management processes and is the de facto orchestration standard, so potential solutions to this problem are explicitly considered for the Kubernetes environment**. The purpose of this paper** is to improve the characteristics of proactive and reactive scaling solutions by integrating them into a single subsystem of computing resource management.

**Analysis of recent research and publications.** In [4], the authors present the CloudScale solution for managing computing resources for multi-tenant cloud systems. This solution includes both proactive and reactive components that can work together or separately in the respective modes of operation. The predictive component is based on fast Fourier transforms to find similar load patterns, after which Markov chains are applied to identify the current state of the application and the required transition. The reactive component, in turn, calculates the error between the current amount of reserved resources and the required amount, based on which the amount of allocated resources is adjusted. Thus, the reactive component performs a corrective function. On different datasets, these components in pairs have shown better results than each of them separately.

In another paper [5], the authors propose another option for integrating the reactive and proactive approach, namely, scaling up is reactive, and scaling down is proactive. The reactive component constantly analyzes the current metrics of the speed of response to requests. If these metrics violate the Service Level Agreement (SLA), upward scaling occurs. The proactive component, which is based on a regression model, prevents the premature release of reserved resources. The results of experiments conducted on synthetic data demonstrate the ability of this solution to adapt to simple load patterns.

A similar approach for horizontal scaling is proposed in [6, 7]. In this paper, two independent controllers are used. The reactive controller is responsible for scaling up, and the proactive controller is responsible for scaling down. However, in these works, the authors conclude that if there is a significant period of application initialization, the reactive controller loses its effectiveness, and the proactive approach shows much better results.

The works on hybrid scaling in Kubernetes are currently not widespread [8]. Therefore, this topic requires more research.

**Outline of the main material of the study.** The developed solution for managing computing resources includes reactive and proactive components. To simplify the experiments and presentation of the results, only CPU time management and horizontal scaling are considered in this paper.

The application developed in this paper solves the problem of horizontal scaling of applications in Kubernetes, which is to manage the number of replicas of a selected application to ensure the required level of QoS metrics and minimize wasted resources. Some application $X$, which at any time t has a given number of replicas $R_t – X_1...X_{Rt}$ that process requests. The total amount of CPU time required to process requests at any given time t is described by the load function $W_{CPU}(t)$. When scaling horizontally, the application $X$ has constant requests $C_X$ for CPU time. In this case, at any time t, the following equality must be satisfied to minimize unprofitable reservation of CPU time and maintain the required level of QoS:

$$R_t * C_X = W_{CPU}(t). \quad (1)$$

In this work, the proactive approach is used as the main one since, provided that high accuracy predictions are obtained, it allows the application to scale up prematurely to ensure the required level of service quality, as well as to release the reserved resources only after the peak loads are over. However, in cases where the accuracy of the predictions is not high, the proactive component significantly loses its effectiveness. The reactive component should be activated if the accuracy of the predictions is critically low and, accordingly, transfer control to the proactive component when the required accuracy is reached. An essential characteristic of both components is the ability to work in the observation mode – to collect and process metrics and calculate the required number of replicas but not apply the obtained values to the application. For example, if the accuracy of predictions drops, this will allow you to quickly switch to reactive management and continue processing requests as usual.

One of the main problems studied in this paper is determining when to transfer control between components. While there are no prerequisites for a reactive approach, a proactive approach depends on the accuracy of the predictions made. If anomalies appear in the load pattern, the prediction algorithm may lose accuracy. An indicator of the need to switch from a proactive to a reactive approach can be [9]:

1. Underestimation or significant overestimation of the number of application replicas over several iterations which allows you to quickly identify the difference between the actual workload and the forecast. Verification over several iterations, although it increases the response time, allows you to be sure of the final result.

2. Low accuracy of the prediction model obtained by splitting historical data into training and evaluation parts. In order to quickly detect an abnormal situation, this approach requires constant retraining and re-evaluation of the model, which can require significant computing resources and time.

3. Drop in QoS metrics, such as response time. The difficulty of this approach lies in finding a metric that unambiguously identifies problems with resource allocation.

In this paper, we use the first approach because it is easy to implement and allows us to quickly identify and fix the problem of dynamic allocation of computing resources.

When a given solution is initialized to automate the scaling of an application, previous resource utilization metrics may not be available. In this case, due to the impossibility of applying a proactive approach, the management is transferred to the reactive component.

The proactive component is based on the Prophet prediction algorithm. Prophet [10] is a time series prediction library developed by Facebook. The main goal of the development was to create a simple, transparent, and understandable model-generation algorithm that would allow us to obtain reliable predictions quickly.

This algorithm is based on an additive regression model that has several components:

$$y(t) = g(t) + s(t) + h(t) + e(t), \quad (2)$$

where $g(t)$ is the trend component, $s(t)$ is the seasonal component, $h(t)$ is the anomalous component, and $e(t)$ is the error function. In addition to the additive regression model, Prophet also uses a Fourier transform.

Historical data on resource utilization is obtained by querying the Prometheus server. To get resource utilization metrics, in particular CPU, the query of the form *sum(rate(container_cpu_usage_seconds_total{container !=""}[1m])) by (pod)* is used. To get the currently specified requests, the Kubernetes API is used, namely the *sum(spec.container[].requests.cpu)* attribute in the deployment manifest [11].

The advantages of this model include the ability to work with a variety of time series, the ability to work efficiently with large data sets and missing data, and flexibility in customization. This algorithm has the ability to detect seasonality, trends, and anomalies in time series automatically.

When developing a solution for horizontal scaling, an additional parameter must be taken into account, namely the application initialization time. This value can be determined automatically, but since it will have a significant impact on the scaler's performance, it should be set by the user. For example, when initializing a web application, it is necessary to establish connections to other services and databases. Also, Kubernetes needs additional time to place a pod on a node, obtain an image, and perform readiness checks. Therefore, this solution allows you to set the *initializationTime* parameter, which the proactive component uses to plan upward scaling.

For reactive scaling, HPA is used – a built-in solution for automating horizontal scaling in Kubernetes [12]. HPA receives system metrics of resource utilization from the metrics-server. The main setting *targetAverageValue* or *targetAverageUtilization* is responsible for the desired load per application replica. HPA calculates the current load value as the average value among all the pods of the target deployment in the Running status. The controller continuously monitors the status of the pods and the current average load, calculating the desired number of replicas with *desiredReplicas = ceil[currentReplicas * ( currentMetricValue / targetMetricValue )]*. If the actual and desired values differ, the application is scaled. HPAs may have a cooldown period to prevent rapid, oscillating scaling decisions. During this period, no further scaling decisions are made. Also, this component allows you to configure the minimum and maximum values of the number of replicas in order to avoid over-scaling or under-scaling.

To minimize the switching time from the proactive to the reactive component, HPA operates in the observation mode, which is achieved by setting the selectPolicy parameter to disabled for the *scaleDown* and *scaleUp* control policies.

While the reactive component is running, the proactive component constantly collects historical usage data and calculates the optimal number of replicas. If, within a certain number of iterations, which can be configured, the calculated values correspond to the actual needs, then control is transferred to the proactive component. Accordingly, if the provided values of the number of replicas are insufficient or significantly exceed the required number (>50%), the reverse operation is performed.

**Experiments.** To evaluate the quality of the developed solution, two experiments were conducted on a minikube cluster [13], including one node with 12 processor cores and 16 GB of RAM. All applications are hosted on the same physical machine.

The workload is generated using locust, which allows you to send requests according to a predefined scenario [14]. The test application can be scaled both horizontally and vertically. For each received request, some CPU-intensive task is performed.

The test application has a query limit of 200 millicores. In addition, when scaling reactive, setting limits higher than requests is essential. On the one hand, this may affect other applications on the same node, but it is a general practice not to load nodes with more than 80–90% of the maximum capacity of computational resources. Setting limits allows you to better estimate the number of application replicas required in cases with a rapidly growing load. Otherwise, the application will be significantly limited in the use of CPU time, which will significantly degrade QoS and interfere with the assessment of the required number of application replicas. In the case of limitations on setting limits, it is necessary to take into account an additional metric that shows the throttling of the application, which is not always correct. In this experiment, the limits were set at 250 millicores.

To approximate real-world applications, the test application requires a 10-second initialization interval. This is necessary for image download, volume attachment, and initialization of the main application components. Since minikube is used for testing, all the necessary data is contained on one machine, so there are no network delays.

To improve the representativeness of the experiments, we also further reduced the global period for collecting system metrics by setting the *metricResolution* value to 15 seconds as kubelets work with same frequency. Also, the *scaleUp* and *scaleDown* scaling policies in HPA were edited to reduce the reaction time to changes in workloads.

The developed solution is launched in the first experiment without any previous history. The load period is 5 minutes and varies from 20 to 100 requests per second. Figure 1 shows the results of the experiment. In the beginning, resource management is performed by the reactive component, which is why there are high response times at the peak moments. After three periods of seasonal load, the proactive component is able to accurately calculate the required number of replicas and takes over. Starting from the 800th second of the experiment, the response time at the moment of peaks onset decreases significantly, which corresponds

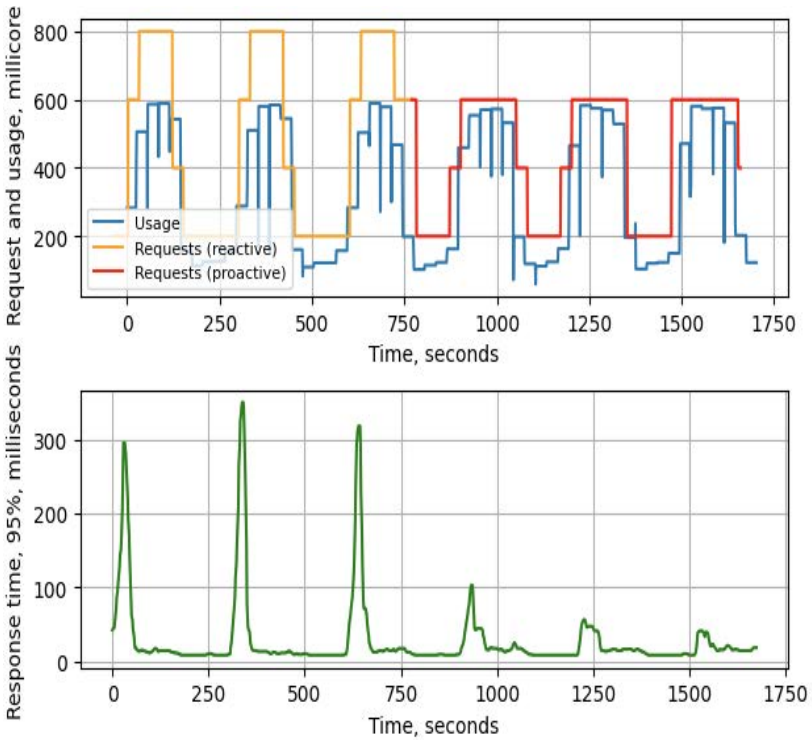


**Fig. 1. The example of the transition from reactive to proactive management**

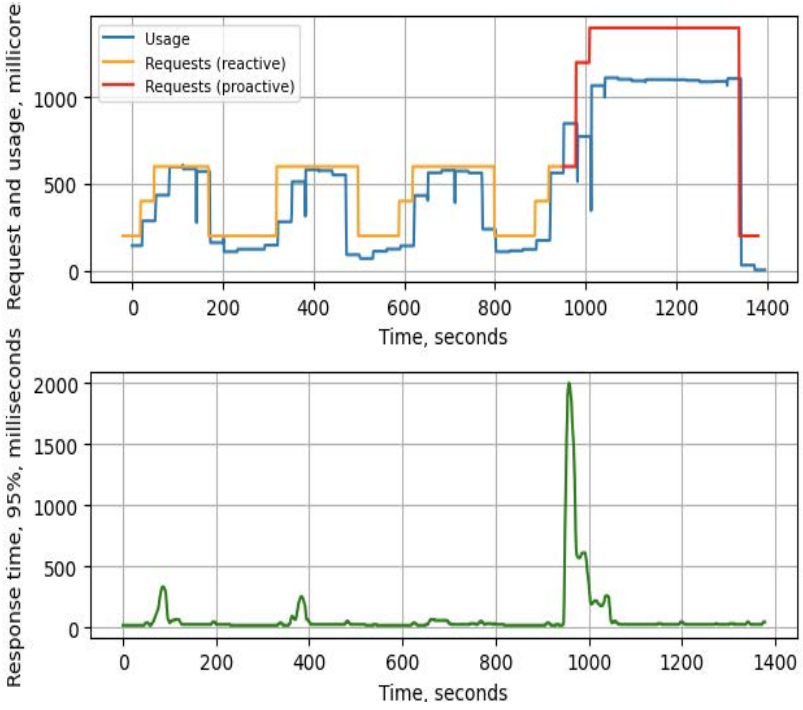


**Fig. 2. An example of switching from proactive to reactive control with an atypical load pattern**

to the time when control is transferred to the proactive component.

It's worth noting that the 95th percentile response time with proactive scaling is 210% better than with reactive scaling. In addition, proactive scaling used 11% less CPU time.

The second experiment demonstrates the ability to reverse the transition. Initially, the control is performed by the reactive component, but after the appearance of an atypical load, the control is transferred to the reactive component. The results are shown in Figure 2.

At about 950 seconds, an atypical load arrives, causing a critical drop in QoS. After several iterations of the proactive component, which missed the required number of replicas, a decision is made to transfer control to the reactive component. After that, scaling up to 7 pods immediately occurs, which should be enough to process all received requests. The response time, in this case, is about 20 seconds. At this interval, this atypical situation led to a significant increase in the 95th percentile of response time – from 80 milliseconds to 173. However, given that such abnormal situations are generally rare, the impact on the overall response time is not significant.

**Conclusions.** In this paper, we have demonstrated a solution for hybrid scaling in Kubernetes. Although the performance of the proactive approach is much better – 210% less response time and 11% less unprofitable reservation of computing resources – it cannot perform its functions correctly in atypical situations, as demonstrated in the second experiment. A method for identifying the need to switch between components was proposed. In future work, this approach can be applied for vertical scaling.

**Bibliography:**

1. Omelchenko V. V., Rolik O.I. Automation of resource management In information systems based on reactive vertical scaling. Адаптивні системи автоматичного управління. 2022. №41. P. 65–78.
2. Omelchenko V. V., Rolik O. I. Workloads prediction methods for proactive resource scaling in Kubernetes. III International Scientific Symposium "Intelligent Solutions" (IntSol-2023). 2023.
3. Straesser M., Grohmann J., von Kistowski J., Eismann S., Bauer A. i Kounev S. Why is it not solved yet. Proceedings of the 2022 ACM/SPEC on International Conference on Performance Engineering. 2022.
4. Shen Z., Subbiah S., Gu X., Wilkes J., Cloudscale: Elastic resource scaling for multi-tenant cloud systems. Proceedings of the 2nd ACM Symposium on Cloud Computing. 2011.
5. Iqbal W., Dailey M. N., Carrera D., i Janecek P. Adaptive resource provisioning for read intensive multi-tier applications in the cloud. Future Generation Computer Systems. № 27. 2011. P. 871–879.
6. Ali-Eldin A., Kihl M., Tordsson J., Elmroth E. Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control. Proceedings of the 3rd workshop on Scientific Cloud Computing Date - ScienceCloud '12. 2012. New York. P. 31.
7. Ali-Eldin A., Tordsson J., Elmroth E. (2012) An adaptive hybrid elasticity controller for cloud infrastructures. Network Operations and Management Symposium (NOMS). 2012. P. 204–212.
8. 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. 2014. Вип. 12. №4. P. 559–592.
9. Qu C., Calheiros R. N., Buyya R. Auto-Scaling Web Applications in Clouds. ACM Computing Surveys. 2018. Вип. 51. № 4. P. 1–33.
10. Taylor S. J., Letham B. Forecasting at scale. PeerJ. URL: https://doi.org/10.7287/peerj.preprints.3190v2 (date visited: 25.09.2023).
11. Deployment Controllers. Kubernetes Documentation. URL: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/. (date visited: 25.09.2023).
12. Horizontal Pod Autoscaler. Kubernetes Documentation. URL: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/. (date visited: 25.09.2023).
13. Minikube Documentation. URL: https://minikube.sigs.k8s.io/docs/. (date visited: 25.09.2023).
14. Locust. Locust GitHub Repository. URL: https://github.com/locustio/locust. (date visited: 25.09.2023).

**Омельченко В.В., Ролік О.І. ІНТЕГРАЦІЯ ПРОАКТИВНОГО І РЕАКТИВНОГО ПІДХОДІВ ДО МАСШТАБУВАННЯ В KUBERNETES**

*Робота присвячена розробці методу інтеграції проактивного і реактивного підходів для автоматизація процесу масштабування обчислювальних ресурсів у кластері Kubernetes. Проактивний підхід дозволяє завчасно масштабувати ресурси кластеру і звільняти їх лише після проходження піку навантаження, що зменшує ризик погіршення якості обслуговування і значно зменшує збиткове резервування ресурсів. Проте, головним недоліком такого методу є неможливість адаптації до аномальних навантажень безпосередньо під час роботи. Хоча реактивний підхід і є менш ефективним*

*в типових ситуаціях, при аномальних навантаженнях продовжує функціонувати в стандартному режимі. Тому гібридний підхід – який включає реактивну і проактивну складові – дозволить ефективно розподіляти ресурси при типових навантаженнях і продовжувати функціонування в аномальних умовах. В роботі розглядається рішення для автоматичного горизонтально масштабування, яке включає розроблений проактивний компонент в поєднанні з існуючим рішенням для реактивного горизонтального масштабування в Kubernetes, а саме Horizontal Pod Autoscaler. В даній роботі проаналізовано проблеми, які можуть виникнути при розробці гібридних рішень, зокрема, проблема визначення моменту перемикання між компонентами та наводяться можливі варіанти їх вирішення. В статті проводяться експерименти для перевірки розробленого рішення. Перше дослідження показує здатність переходити до проактивного управління, якщо необхідна точність передбачень отримана. Також даний експеримент дозволяє порівняти проактивний і реактивний підхід між собою в контексті якості обслуговування та збиткового резервування ресурсів. Друге дослідження демонструє здатність визначати аномальне навантаження і вмикати реактивний компонент. Також оцінюється швидкість реагування на виникнення аномалій і загальний вплив на показники якості обслуговування.*

***Ключові слова:*** *динамічне управління обчислювальними ресурсами, проактивне масштабування, реактивне масштабування, Kubernetes, Horizontal Pod Autoscaler.*