A Survey of Autoscaling in Kubernetes

Minh-Ngoc Tran
School of Electronic Engineering
Soongsil University
Seoul, Korea
mipearlska1307@dcn.ssu.ac.kr

Dinh-Dai Vu
School of Electronic Engineering
Soongsil University
Seoul, Korea
daivd@dcn.ssu.ac.kr

Younghan Kim
School of Electronic Engineering
Soongsil University
Seoul, Korea
younghak@ssu.ac.kr

2022 Thirteenth International Conference on Ubiquitous and Future Networks (ICUFN) | 978-1-6654-8550-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICUFN55119.2022.9829572

***Abstract*— Autoscaling is the vital feature of cloud infrastructure to acquire or allocate computing resources on-demand, which allows users to automatically scale the resources provisioned to the applications without human action under a fluctuating workload to optimize the resource cost while satisfying the Quality of Service (QoS) requirements. Kubernetes (K8s), the most prevalent container orchestration, provides built-in autoscalers to deal with the scaling problem in terms of vertical and horizontal at container level but still has some limitations. In this paper, we survey the state of the art of existing approaches to solve the problem of container autoscaling in Kubernetes, their main characteristics as well as their current issues. Based on the analysis, new future directions that can be explored are proposed.**

Keywords— autoscaling, Kubernetes

I. INTRODUCTION

Kubernetes is currently the most popular container orchestrator used by service providers that offer full lifecycle management of containerized applications. To guarantee deployed containerized applications' performance and service-level agreement (SLA) overtime against dynamic user workloads or resources availability, autoscaling is a vital feature. This feature dynamically acquires or releases container resources to meet applications' QoS demand. Kubernetes fulfills this requirement by providing two auto-scaler features: Horizontal Pod Autoscaling and Vertical Pod Autoscaling (we denoted them in this paper as default K8s HPA and default K8s VPA), however, these default features have slow adaptation performances against dynamic workloads.

To improve default K8s autoscaling limitation, there are various enhanced autoscaling approaches have been proposed with diverse characteristics and use-cases, such as different target application architecture, horizontal or vertical based scaler, reactive or proactive operation, adoption or not of machine learning techniques, etc. To the best of our knowledge, there is a lack of studies that put together and analyze the existing autoscaling works for Kubernetes. The main goal of this paper is to present a comprehensive discussion on the current state-of-the-art research on container-based application autoscaling in Kubernetes considering the differences in their types and characteristics. We also mention the issues and challenges of the current works and propose several new directions for future research.

II. THE CURRENT RESEARCH STATE OF AUTOSCALING IN KUBERNETES

The general process of autoscaling in Kubernetes includes three steps: monitoring resources/workload metrics, analyzing monitored data, and deciding on suitable scaling methods. Proposed autoscaling solutions focus on improving the default K8s autoscaling feature in one or all of these steps. Besides, they also target different types of applications.

In this section, we categorize existing approaches for autoscaling based on four different aspects: application architectures (architecture of the scaling target application), methods (how are the containers scaled), timing (when and how to trigger scaling decision), and indicators (which metrics was monitored to make scaling decisions).

A. Application Architectures

1) Monolithic Architecture: Monolithic applications have all or most of their functionalities combined and packaged within a single process. These applications are normally deployed as a single Kubernetes container. Because of its simple implementation, several autoscaling works use this kind of application for evaluation while focusing their main contributions on optimizing autoscaling method or timing. Notable used monolithic applications are simple HTTP websites [1-4], and CPU-intensive/non-intensive applications [5] (manually created by adjusting the number of mathematical functions).

2) Microservice-based Architecture: Microservice-based applications, in contrast, consist of multiple standalone services that interact with each other. Autoscaling approaches targeting this kind of application need to consider the dependency between services. However, these services are independently deployable and scalable. Works of [6-9] propose autoscaling solutions based on analyzing resource demand for each independent service. Yu et al. [10], and Coulson et al. [11], on the other hand, analyze microservices dependency to determine which service to be scaled. Choi et al. [12] consider the tail latency of the whole microservice chain to pre-scale services and reduce provisioning time.

B. Methods

1) Horizontal Scaling: In Kubernetes environment, Horizontal scaling refers to increasing or decreasing the number of replicas of the same pod to share the load. The default k8s HPA depends on manually setting up some threshold values such as CPU utilization, the minimum and the maximum number of pods. Some researchers try to provide a better mechanism by considering additional metrics such as traffic characteristics [1, 13] or response latency [12]. Besides, Some studies aim to custom HPA with machine learning or heuristic analysis [4, 9-12].

2) Vertical Scaling: The vertical scaling method refers to increasing or decreasing the assigned resources for containers. This method requires terminating current containers and then redeploying them with new assigned resources. Because of this limitation, vertical scaling receives less attention from researchers. RUBAS in [14], and ELASTICDOCKER in [8] address this limitation by utilizing the container checkpoint technique CRIU [15] for saving the

state of the containers that need scaling before terminating them.

3) Hybrid Scaling: This method combines both horizontal and vertical scaling. Current approaches [3, 7, 16] perform hybrid scaling in a cascading way: using the vertical scaling method to determine the optimal required resources for containers first, then using the horizontal one to dynamically changes the number of container instances.

C. Timing

1) Reactive Scaling: In the reactive method, the system monitors current workload traffic [10, 13] or resources usage [2, 3, 6, 8, 14, 17]. If the workload or resource demand reaches a pre-defined threshold, the system will then calculate and determine the suitable autoscaling decision.

2) Proactive Scaling: Proactive autoscaling uses sophisticated techniques to predict future demands to arrange resource provisioning. It helps to decide to scale up or down according to a predetermined forecast. In recent years, artificial intelligence and machine learning have become prevalent and contributed to building a proactive autoscaler. Machine learning models can learn from past scaling decisions and workload behavior to generate scaling decisions ahead of time, and the accuracy of these models depends on the large of the achieved dataset. During our review, the most common machine learning methods for container scaling were based on regression [9, 18, 19], deep neural networks [1, 11, 12], and reinforcement learning [5, 7]. In some works such as [4, 20], multiple deep neural networks are used at the same time with the best performer will be used for triggering autoscaling decisions. Overall, with the advantage of pre-scaling applications before real issues happen(resources overloading, burst workload period, etc.), proactive scaling methods are better solutions than reactive scaling methods and will continue to be the dominant research direction in the future.

D. Indicators

The actions of auto-scalers are based on performance indicators of the application obtained in the monitoring phase. These indicators are produced and monitored at different levels of the system hierarchy from low-level metrics at the physical or hypervisor level to high-level metrics at the application level. Table 1 shows the common metrics for autoscaling.

TABLE I. COMMON METRICS USED IN KUBERNETES AUTO-SCALING

Metrics	Usage
CPU, RAM, Network I/O, Disk	Common metrics usually provided by container
Response Time	Metric type is largely used. It is part of SLA
Number of Requests	Mostly seen in horizontal scaling
Custom Metrics	Add application knowledge to the model and can improve accuracy

1) Low-Level Metrics: The simplest solution is to use the utilization of physical resources as indicators and scale resources horizontally or vertically to maintain the overall utilization within a pre-defined upper and lower bound. Typical metrics are CPU and memory usage/utilization [2, 5, 7, 8, 14, 16, 21]. Industry systems also widely adopt this approach [20]. If the auto-scaler of this kind only supports horizontal scaling, it can be utilized by both cloud providers and service providers.

2) High-Level Metrics: They are performance indicators observed at the application layer. Typical metrics are traffic or workload rate [1, 4. 9, 11-13, 18, 19], request-response latency [7, 12, 16], SLA [10]. Only auto-scalers deployed by service providers can utilize these metrics as they are not visible to cloud providers. Different from the previous metrics, they cannot directly trigger scaling actions but can be used to support making efficient scaling plans. In addition, these metrics are not straightforward to measure.

III. CURRENT AUTO-SCALING APPROACHES ISSUES

In this part, we discuss the issues of current autoscaling approaches based on their method kind.

With the horizontal scaling approaches, although they can guarantee SLA, they cannot optimize resource utilization. Horizontal scaling methods only change the number of pod instances while keeping the size of the pod (assigned CPU and memory resources) unchanged. Hence, during the low-request period from users, not all assigned resources for each pod are used (low pod utilization), especially for some resource-intensive applications. Multiple low pod utilization will lead to low cluster utilization, which turns out to be costly for service providers.

With the vertical scaling approaches, in contrast, they have the limitation of satisfying the quality of service (QoS) although they can optimize resource utilization. First, the vertical scaling methods only work with system usage metrics which are not good enough scaling indicators for application performance [7]. Second, vertical scaling methods require service restarting. This is not applicable for some state-dependent and long booting time services such as database or message broker. Although applying the container checkpointing technique in [8, 14] is a workaround solution, it still costs a huge amount of system resources to checkpoint the container state if vertical scaling frequently happens in fluctuated traffic scenarios. Finally, increasing pod assigned resources sometimes only raise pod performance to a saturation point as studied in [3], which is not efficient.

Hybrid scaling methods, which combine both the horizontal and vertical scaling methods, can solve these methods' standalone problems. However, current hybrid approaches are executed in a cascading way which means using vertical scaling to find the suitable resources for the first deployed pod first, then applying horizontal scaling afterward. However, because all pods in a Kubernetes deployment have the same assigned resources, the decided amount of resources found by the vertical scaling process will be applied to all pods instantiated by the horizontal scaling process. This might lead to low pod utilization for the newly created instances in fluctuated traffic cases. For example, when the current number of instances is not enough to serve the incoming traffic, one new instance is horizontally scaled up but this instance only serves a few requests that the old instances cannot serve.

IV. Future Direction Discussion

Based on the survey and analysis, we point out some potential directions for future research

1) Runtime modifying pod resources feature for vertical scaling: A method that allows Kubernetes container resources to be modified at runtime without terminating and restarting should be considered to utilize the benefit of vertical scaling.

2) Dynamic hybrid autoscaling: Horizontal and vertical autoscaling should be organized and dynamically applied at the same time because of the limitation of the cascading process as analyzed in the above part

3) Hybrid autoscaling for microservice-based applications: There are limited works [7, 20] on hybrid autoscaling for microservice-based architecture. Current works have only addressed the problem of which component in a single microservice chain to scale. Dynamically deciding which scaling method (horizontal/vertical) to use on a component, multiple service chain consideration, the long tail latency of a whole service chain instantiation duration when scaling up consideration are some potential directions

4) Infrastructure-level autoscaling: Service providers do not always have enough resources in their current clusters for scaling up their service instances. Most of the current research focus on pod-level autoscaling, node-level and cluster-level autoscaling need to be considered

V. Conclusion

Many research works have targeted Kubernetes autoscaling problem and many auto-scalers with diverse characteristics and designs to improve the default autoscaling features have been proposed recently. In this paper, we surveyed the development of auto-scaling techniques for container applications in Kubernetes in terms of target application architecture, scaling method, scaling timing, and scaling indicators. We identified current research issues and discussed some promising future directions.

Acknowledgment

This work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grants funded by the Korea government (MSIT) (No. 2020-0-00946, Development of Fast and Automatic Service recovery and Transition software in Hybrid Cloud Environment)

References

[1] M. Imdoukh, I. Ahmad, "Machine learning-based auto-scaling for containerized applications", in Neural Computing and Applications, Vol.32, 2020, pp. 9745-9760.

[2] Nguyen, T. T., Yeom, Y. J., Kim, T., "Horizontal pod autoscaling in Kubernetes for elastic container orchestration", Sensors, Vol. 20, No. 16, 2020, p. 4621.

[3] Balla, D., Simon, C., & Maliosz, M.. "Adaptive scaling of Kubernetes pods". In NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium, Apr. 2020, pp. 1-5.

[4] Toka, L., Dobreff, G., Fodor, B., & Sonkoly, B. "Machine learning-based scaling management for kubernetes edge clusters". IEEE Transactions on Network and Service Management, Vol. 18, No. 1, 2021, pp. 958-972.

[5] Rossi, F., "Auto-scaling Policies to Adapt the Application Deployment in Kubernetes". In ZEUS, 2020, pp. 30-38.

[6] A. Khaleq, I. Ra, "Agnostic approach for microservices autoscaling in cloud applications", In Proceddings of the 2019 International Conference on Computational Science and Computational Intelligence, Dec. 2019, pp. 1411-1415.

[7] A. Khaleq, I. Ra, "Intelligent Autoscaling of Microservices in the Cloud for Real-Time Applications". IEEE Access, Vol. 9, Feb. 2021, pp. 35464-35476.

[8] Al-Dhuraibi, Y. and Paraiso, F. "Autonomic Elasticity of Docker Containers with ELASTICDOCKER", In Proceedings of the 10th International Conference on Cloud Computing (CLOUD), 2017, pp. 472–479.

[9] Rudrabhatla, C. K, "A Quantitative Approach for Estimating the Scaling Thresholds and Step Policies in a Distributed Microservice Architecture". IEEE Access, Vol. 8, 2020, pp. 180246-180254.

[10] G. Yu, P. Chen, Z. Zheng., "Microscaler: Cost-effective Scaling for Microservice Applications in the Cloud with an Online Learning Approach". IEEE Transaction on Cloud Computing (Early Access), 2020, pp. 1-1.

[11] Coulson, N. C., Sotiriadis, S., & Bessis, N. "Adaptive microservice scaling for elastic applications". IEEE Internet of Things Journal, Vol. 7, No. 5, 2020, pp. 4195-4202.

[12] B. Choi, J. Park, C. Lee, D. Han., "pHPA: A Proactive Autoscaling Framework for Microservice Chain". In APNet 2021: 5th Asia-Pacific Workshop on Networking (APNet 2021), 2021, pp. 65-71.

[13] L. Phuc, L-A. Phan, T. Kim., "Traffic-Aware Horizontal Pod Autoscaler in Kubernetes-Based Edge Computing Infrastructure". IEEE Access, Vol. 10, 2022, pp. 18966-18977.

[14] Rattihalli, G., Govindaraju, M., Lu, H., & Tiwari, D., "Exploring potential for non-disruptive vertical auto scaling and resource estimation in kubernetes". In 2019 IEEE 12th International Conference on Cloud Computing (CLOUD), 2019, pp. pp. 33-40.

[15] CRIU – Checkpoint/Restore In Userspace, Available: https://criu.org/Main_Page

[16] L. Baresi, D. Hu, G. Quattrocchi, L. Terracciano., "KOSMOS: Vertical and Horizontal Resource Autoscaling for Kubernetes". In International Conference on Service-Oriented Computing - ICSOC 2021, 2021, pp. 821-829.

[17] W-S. Zheng, L-H. Yen, "Auto-scaling in Kubernetes-Based Fog Computing Platform". In New Trends in Computer Technologies and Applications. ICS 2018, 2019, pp. 338-345.

[18] H. Zhao, H. Lim, M. Hanif, C. Lee., "Predictive Container Auto-Scaling for Cloud-Native Applications". In 2019 International Conference on Information and Communication Technology Convergence (ICTC), 2019, pp. 1280-1282.

[19] D-H. Luong, H-T. Thieu, A. Outtagarts, Y. Ghamri-Doudane., "Predictive Autoscaling Orchestration for Cloud-native Telecom Microservices". In 2018 IEEE 5G World Forum (5GWF), 2018, pp. 153-158.

[20] Rzadca, K et al., "Autopilot: workload autoscaling at Google". In Proceedings of the Fifteenth European Conference on Computer Systems, 2020, pp. 1-16.

[21] B. Thurgood, R. Lennon "Cloud Computing With Kubernetes Cluster Elastic Scaling". In ICFNDS '19: Proceedings of the 3rd International Conference on Future Networks and Distributed Systems, 2019, pp. 1-7.