Demonstrating the Benefits of Service-Aware Pod Autoscaling with Shared Resources

Demonstrating the Benefits of Service-Aware Pod Autoscaling with Shared Resources

Federico Tonini, Carlos Natalino, Lena Wosinska and Paolo Monti
Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden
{tonini,carlos.natalino,wosinska,mpaolo}@chalmers.se

***Abstract*—Service providers can leverage shared resources to reduce the overall amount of required resources while keeping acceptable Quality of Service (QoS) levels. Kubernetes (K8s) provides a Horizontal Pod Autoscaling (HPA) mechanism that allows to automatically adjust the number of Pods to closely follow the user demand variations over time. To properly leverage shared resources with HPA, service providers need to limit the use of dedicated resources and overprovisioning. However, in the case of traffic spikes, there may not be enough resources to satisfy the demand. The HPA, which relies on resource usage to drive the scaling, is unaware of how many requests could not be served with the required QoS. This might result in an underestimation of the number of required Pods to be added, leading to additional QoS degradation. This demonstration showcases the effectiveness of a new Pod autoscaling mechanism (i.e., Service Aware Pod Autoscaling (SAPA)) that relies on user request measurements from the service load balancer to better estimate the number of required Pods. SAPA allows selecting the amount of Pod resources (dedicated and shared) in a simple way. We demonstrate the benefits of SAPA by comparing it to a K8s cluster based on the traditional HPA in terms of resource usage and service latency.**

***Index Terms*—Cloud native services, QoS, service degradation, Pod autoscaling, Kubernetes, Shared resources.**

I. Introduction

Containers and container orchestration platforms changed the way services are provided. Compared to Virtual Machines (VMs), they offer easier management of distributed applications, reduced overhead, reduced start/stop time, and more effective utilization of compute resources [1], [2]. Kubernetes (K8s) [3] is a container orchestration platform that allows autonomous scaling of Pods (i.e., collections of containers) to closely follow user demand by means of a Horizontal Pod Autoscaling (HPA) mechanism [4]. K8s provides two ways of assigning resources: resource *request* and *limit* [5]. For each Pod, service providers can specify the assignment of dedicated resources by setting the *request* amount, while shared resources are defined by the *limit*. Dedicated resources can be accessed at any time during service operation, and are paid for regardless of their use. On the other hand, shared resources are in contention with other Pods running on the same machine and are used only when dedicated resources are not enough to satisfy the demand. Since they are subject to contention, they are paid for only when accessed.

Service providers can rely on shared resources to reduce cost and degradation [6]. To do so, the over-provisioning of dedicated resources should be kept to a minimum, encouraging the use of shared resources. However, in the case of sudden traffic spikes, resources might not be enough to satisfy the whole demand. In this case, degradation is experienced due to the time it takes for K8s to detect the lack of resources and adjust the number of Pod replicas (usually referred to as *scaling delay*). In this situation, the HPA is unaware of the demand that was not satisfied, hence may under-dimension the required (or desired) Pods. To overcome this issue, dedicated resources are usually over-provisioned, preventing the use of shared resources. Therefore, a better scaling mechanism is needed to fully exploit the benefits of shared resources.

Different strategies have been proposed to improve the HPA mechanism. In [7], the authors experimentally assess the performance of different Artificial Intelligence (AI)-based prediction models in reducing HPA over-provisioning while providing some level of Quality of Service (QoS). The works in [8], [9] introduce workload prediction techniques to anticipate traffic bursts and adjust the number of replicas accordingly. Experimental results show reduced service response time and resource usage compared to the conventional HPA. The work in [10] relies on Reinforcement Learning (RL) to automate the scaling mechanism while continuously learning and adapting the resources to the environment. All the aforementioned works improve the scaling process substantially. However, these strategies focus on scaling dedicated resources and are not designed to work with shared resources. The HPA can be used in the presence of shared resources, but its performance is limited by the (sometimes) incorrect evaluation of the required replicas. Therefore, there is a need for a new strategy to overcome the HPA issues while leveraging shared resources.

In this experimental demonstration, we will show how service-related information can be used to improve K8s scaling in the presence of shared resources. More specifically, we propose to use a novel scaling mechanism called Service Aware Pod Autoscaling (SAPA) that takes as input the user requests for a better estimation of the required Pod replicas [11]. Knowledge about the demand is crucial to avoid the underestimation intrinsic of HPA, allowing to keep dedicated

This work was supported by EUREKA cluster CELTIC-NEXT projects AI-NET-ANIARA and AI-NET-PROTECT funded by VINNOVA.

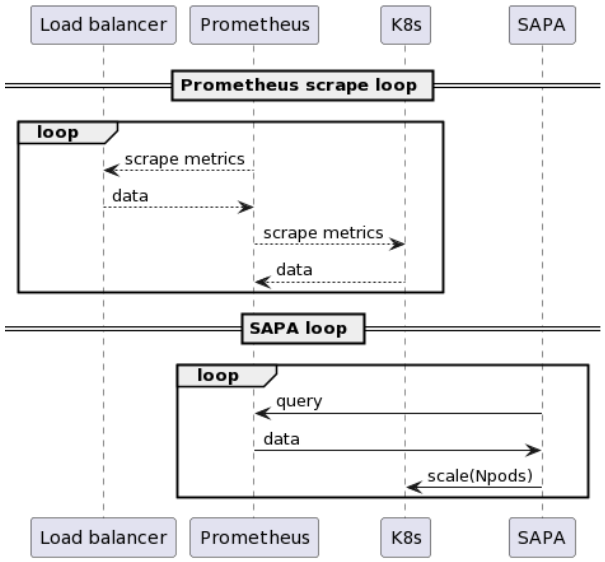


Fig. 1. The communication between the different components in the demonstrator.

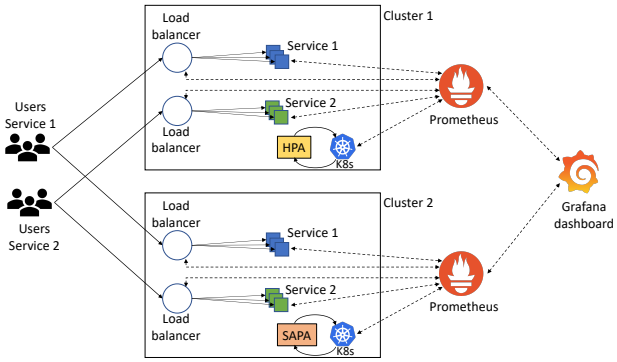


Fig. 2. Demo setup.

resource over-dimensioning to a minimum while leveraging shared resources. An initial assessment of the benefits introduced by SAPA is presented in [11], by means of a custom Python-based simulator. In this demo, we experimentally prove these benefits. More specifically, we will demonstrate the effectiveness of SAPA by comparing the operations of two K8s clusters, each one with different autoscaling mechanisms, one driven by the K8s HPA and the other driven by the SAPA. Different metrics (i.e., service delay and CPU usage) will be collected and shown through a custom Grafana dashboard [12] to illustrate the benefits of SAPA.

II. Design of the Service Aware Pod Autoscaling (SAPA) Mechanism

In this section, we briefly present the architectural design of the proposed SAPA mechanism for efficient Pod autoscaling. The architecture is based on the following components:

- a K8s cluster handling the Pods utilized by the service. More specifically, K8s is used to scale up to down the number of Pods performing the service, a process that consists in either creating new Pod replicas or terminating them.
- A load balancer, one for each service, collecting user requests and forwarding them to different Pod replicas.
- Prometheus [13], a metrics collector that stores data to be used for statistics. Examples of these data are the amount of user requests per unit of time and the number of running Pod replicas.
- The SAPA, which is in charge of computing the desired number of Pod replicas.

Periodically, the SAPA component leverages the aggregated number of requests over a period of time (N_{req}) to compute the desired number of Pod replicas (N_{Pods}) with the following formula:

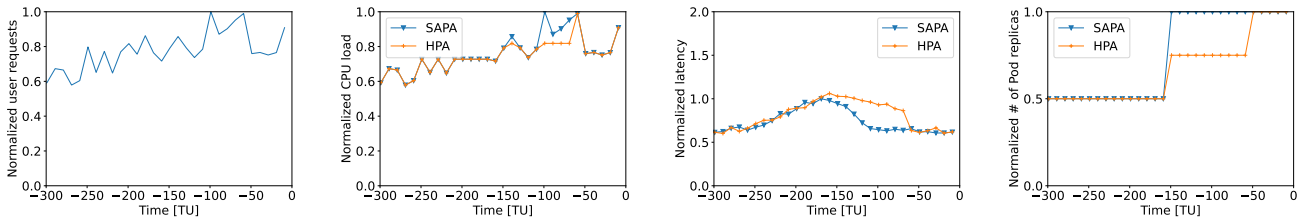
$$N_{Pods} = \left\lceil \alpha \cdot \frac{N_{req}}{M_{req}} \right\rceil, \quad (1)$$

where M_{req} represents the maximum supported request rate per Pod without degrading the QoS. The ratio N_{req}/M_{req} is the actual number of replicas that are required to satisfy the demand by relying only on the Pod request resources. By tuning the value of α, N_{Pods} can be under-dimensioned or over-dimensioned, changing the share between the Pod request and limit resources to be used.

The demonstration workflow of SAPA is depicted in Fig. 1. There are two main loops executed in parallel that consist of an exchange of messages among the different components of the architecture. In the first loop, Prometheus periodically (e.g., every 5s) collects and stores metrics, e.g., number of user requests arrived at the load balancer, number of Pod replicas, and resource usage. The second loop is usually executed within a larger period (e.g., 30 s). In this loop, the SAPA component retrieves the value of the user requests from Prometheus and the number of running Pods. Then, SAPA computes the desired number of replicas (N_{Pods}) using (1) and evaluates the need for scaling by comparing it with the number of running replicas. If the two are different, SAPA triggers a scaling procedure to update the number of Pod replicas. This is done by specifying the new number of replicas to K8s. Finally, K8s creates or terminates Pods to reach the desired value received from the SAPA.

III. Demo Setup

The demo will be executed remotely on a workstation running at Chalmers' premises. Figure 2 shows the main building blocks of our demo. We will consider two K8s clusters with the same amount of resources. One of them will run the traditional HPA, while the other will run SAPA. Two services will be considered and will be deployed in both K8s clusters. Two independent traffic profiles, one per service, will be synthetically generated and varied over time, mimicking user request fluctuations. We will execute the requests of each traffic profile twice, targeting both K8s clusters. To have a



(a) (b) (c) (d)

Fig. 3. Mock of the dashboard plots showing hypothetical normalized values. Time is expressed in Time Units (TUs).

fair comparison, all the Pods will be configured to use shared CPU resources. Each cluster will be monitored by an instance of Prometheus, which will collect and monitor the following Key Performance Indicators (KPIs): user requests per unit of time, response time, number of Pod replicas and desired Pods, CPU usage per Pod, CPU usage per node of the cluster, and total CPU usage.

Figure 3 depicts an example of a custom dashboard for one service in a hypothetical scenario, where values have been normalized to their maximum. The aforementioned KPIs (collected by Prometheus) for the different clusters will be shown using plots updated in real-time. The number of user requests for one service is varied over time (see Fig. 3(a)). The CPU load on the Pods (Fig. 3(b)) in the two clusters varies according to the user demand, reaching a plateau once the replicas are not enough to satisfy the demand (around time -250 [TU]). Consequently, the service latency starts increasing (Fig. 3(c)). At this point, the HPA and SAPA adjust the number of replicas, which are updated after the scaling delay (Fig. 3(d)). The service latency start decreasing. However, the HPA is not aware of the entity of the user demand, and performs a wrong estimation of the number of replicas, which reaches the correct value only around time instant 50 [TU]. This translates into higher service latency for HPA compared to SAPA. The experienced latency depends on the number of replicas. This value can be over-dimensioned to reduce the overall latency. In the HPA, this can be done by reducing the scaling threshold. For the SAPA, the value of α can be increased. This, in turn, reduces the use of shared resources, increasing the overall CPU to be allocated to the service, which is proportional to the number of Pods.

During the demonstration, the user will be able to define the user load in the system by changing the profile of the number of requests per second. To perform the demo on-site, a laptop connected to a big screen will show the custom Grafana dashboard [12]. A stable Internet connection will also be required.

IV. Conclusion

In this experimental demonstration, we showcase how to overcome the challenge of providing resource efficiency and QoS with shared resources for containerized microservices based on K8s. Through our demonstration, we show the implementation of SAPA, a Pod autoscaling mechanism that relies on user traffic data to drive the scaling. Compared to the traditional HPA, the proposed mechanism allows for better exploitation of shared resources for containerized services in cloud native environments. As a future direction, the SAPA can be enhanced with predictions to anticipate user request variations. An intelligent strategy can also be defined to tune the value of α over time according to the availability of shared resources and latency performance.

References

[1] E. Casalicchio and S. Iannucci, "The state-of-the-art in container technologies: Application, orchestration and security," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 17, p. e5668, 2020, e5668 cpe.5668. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5668

[2] "IBM - The Benefits of Containerization and What It Means for You," https://www.ibm.com/cloud/blog/the-benefits-of-containerization-and-what-it-means-for-you.

[3] "Kubernetes," https://kubernetes.io/.

[4] "Kubernetes Horizontal Pod Autoscaler," https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/.

[5] "Kubernetes Resources," https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/.

[6] F. Tonini, C. Natalino, D. A. Temesgene, Z. Ghebretensaé, L. Wosinska, and P. Monti, "Benefits of Pod dimensioning with best-effort resources in bare metal cloud native deployments," *IEEE Networking Letters*, pp. 1–5, 2023.

[7] L. Toka, G. Dobreff, B. Fodor, and B. Sonkoly, "Adaptive AI-based auto-scaling for Kubernetes," in *IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, 2020, pp. 599–608.

[8] D.-D. Vu, M.-N. Tran, and Y. Kim, "Predictive hybrid autoscaling for containerized applications," *IEEE Access*, vol. 10, pp. 109 768–109 778, 2022.

[9] M. Abdullah, W. Iqbal, J. L. Berral, J. Polo, and D. Carrera, "Burst-aware predictive autoscaling for containerized microservices," *IEEE Transactions on Services Computing*, vol. 15, no. 3, pp. 1448–1460, 2022.

[10] F. Rossi, M. Nardelli, and V. Cardellini, "Horizontal and vertical scaling of container-based applications using reinforcement learning," in *IEEE International Conference on Cloud Computing (CLOUD)*, 2019, pp. 329–338.

[11] F. Tonini, C. Natalino, D. A. Temesgene, Z. Ghebretensaé, L. Wosinska, and P. Monti, "A service-aware autoscaling strategy for container orchestration platforms with soft resource isolation," in *2023 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, 2023, pp. 1–6.

[12] "Grafana," https://grafana.com/.

[13] "Prometheus," https://prometheus.io/.