Enabling Dynamic Rescheduling in Kubernetes Environments with Kubernetes Scheduling Extension (KSE)

Pedro Moritz de Carvalho Neto[1], **Márcio Castro**[1], **Frank Siqueira**[1]

[1]Universidade Federal de Santa Catarina (UFSC)
Florianópolis/SC

pedro.moritz@posgrad.ufsc.br, {marcio.castro,frank.siqueira}@ufsc.br

Abstract. *An inefficient distribution of pods over nodes in a Kubernetes cluster environment may lead to a suboptimal scenario called node unbalancing. Problems related to scalability, reliability, availability, power consumption and use of resources may arise from this scenario. This study proposes an extension named Kubernetes Scheduling Extension (KSE) that allows the implementation of different node balancing algorithms in Kubernetes.*

1. Introduction

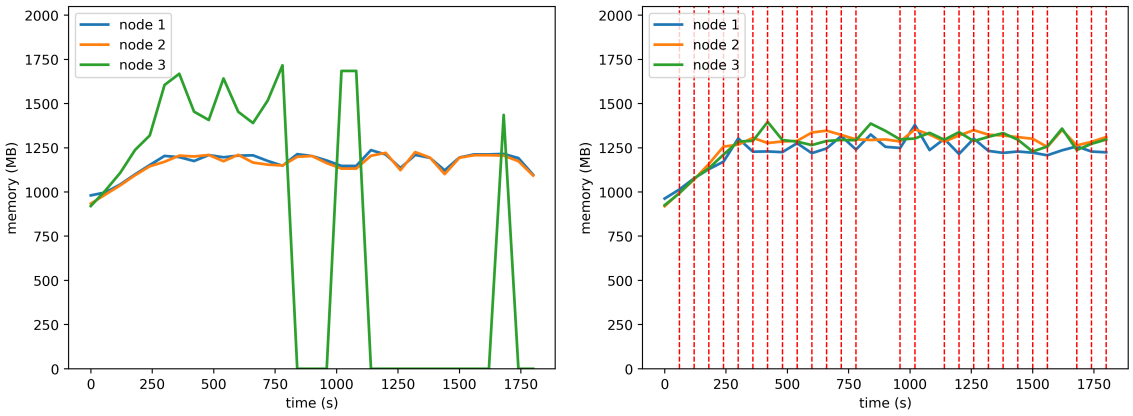
The use of virtualization technologies has shown to be a valuable resource for Cloud Computing and has been intensely used in private and public clouds to improve resource optimization and availability. In this context, the use of container-based virtualization allows dynamic and scalable scenarios, more flexible than the standard Virtual Machine approach. Container technology has enabled a revolution in the Cloud Computing paradigm by creating an abstraction layer that encapsulates and isolates the applications into the cluster [Buyya et al. 2018]. In order to fully extract the versatility of containers, an orchestrator needs to be chosen. The orchestrator is a set of tools responsible for the provisioning, deployment, networking, scaling, availability, and also for the life cycle management of containers. Kubernetes, developed by Google in 2008 and handed over to the Cloud Native Computing Foundation in 2014, is one of the most popular among container orchestrators [CNCF 2022]. Kubernetes provides containerized applications in atomic structures called *pods*, which are executed on the nodes of the cluster. It has been offered as a managed service by the leading public cloud providers but has also been widely used on self-managed clusters.

2. Target Problem and Proposed Approach

The use of resources in a Kubernetes cluster is dynamic, changing in time according to clients demands. It means that applications running in cluster nodes may consume different levels of resources with a high potential to create unbalanced scenarios, exhausting and/or underusing compute resources during the pods' life cycle. Kubernetes offers APIs as a way to extend its default scheduler, known as *kube-scheduler*, enabling a great opportunity to solve problems related to unbalanced nodes. In this work, we propose an extension to the default Kubernetes scheduler allowing the implementation of dynamic rescheduling algorithms, taking into consideration different criteria in an integrated and flexible way. This extension, named *Kubernetes Scheduling Extension* (KSE), is used to implement a classical greedy strategy to mitigate some problems related to unbalanced nodes in a Kubernetes environment.

3. Preliminary Experiments and Conclusions

A preliminary scenario was built to compare the outcomes of the default Kubernetes scheduler (*kube-scheduler*) with a scheduler that periodically balances the nodes based



(a) kube-scheduler

(b) KSE + GreedyLB

Figure 1. Kube-scheduler vs. KSE+GreedyLB (3 worker nodes and 6 pods).

on the amount of memory consumed by the pods. This node balancer, named *GreedyLB*, was implemented using the proposed extension (KSE) and does the following steps periodically: first, it sorts pods in decreasing order of memory consumption; then it iteratively maps an unassigned pod that consumes the most memory to the node that has the least memory consumption.

The experiments were executed on a Kubernetes cluster composed of 4 nodes (master + 3 worker nodes) implemented with Minikube[1]. Two pods were initially deployed on each worker node. A dummy application was implemented in order to simulate real world memory-consuming applications and was deployed on each pod as a containerized application. The workload for the experiments was created with a load testing tool, *Grafana k6*[2]. In order to provide a node imbalance situation, the requests were assigned to the pods using a normal distribution algorithm. Since *kube-scheduler* is not able to actively reschedule pods during their lifetime, the application will consume resources until pod/node limits are reached. After pods/nodes collapse, it is not even possible to gather metrics from Kubernetes agents, so the memory metrics readings fall to 0, as shown in Figure 1a. The KSE, on the other hand, was able to evict and reschedule pods according to the *GreedyLB* strategy (Figure 1b). The red dotted lines show when at least one pod was evicted and rescheduled in order to achieve node balance. As a consequence, KSE was also able to prevent node failure and application downtime.

As future work, we intend to carry out a broad range of experiments with different node balancing algorithms and workload distributions (considering both CPU and memory) in order to evaluate the proposed approach.

References

Buyya, R., Srirama, S. N., Casale, G., Calheiros, R., Simmhan, Y., Varghese, B., Gelenbe, E., Javadi, B., Vaquero, L. M., Netto, M. A., et al. (2018). A manifesto for future generation cloud computing: Research directions for the next decade. *ACM computing surveys (CSUR)*, 51(5):1–38.

CNCF (2022). Cloud native computing foundation annual survey 2022 - the year cloud native became the new normal.

[1]https://minikube.sigs.k8s.io/docs/
[2]https://k6.io/