Kubernetes in Edge and Cloud Computing: A Comparative Study of K3s, K0s, MicroK8s, and K8s

Preprint · March 2025

DOI: 10.13140/RG.2.2.19852.22403

8 authors, including:

Manolis Skoularikis
University of Western Macedonia
2 PUBLICATIONS **0** CITATIONS
SEE PROFILE

Athanasios Liatifis
University of Western Macedonia
20 PUBLICATIONS **299** CITATIONS
SEE PROFILE

Dimitrios Pliatsios
University of Western Macedonia
29 PUBLICATIONS **720** CITATIONS
SEE PROFILE

Vasileios Argyriou
Kingston University
280 PUBLICATIONS **4,269** CITATIONS
SEE PROFILE

Kubernetes in Edge and Cloud Computing: A Comparative Study of K3s, K0s, MicroK8s, and K8s

Manolis Skoularikis*, Athanasios Liatifis*, Dimitrios Pliatsios*, Vasileios Argyriou†,
Evangelos Markakis¶ Thomas Lagkas‡, Georgios Th. Papadopoulos§, Panagiotis Sargiannidis*
*Department of Electrical and Computer Engineering,
University of Western Macedonia, Kozani 50100, Greece
emails: m.skoularikis@uowm.gr, aliatifis@uowm.gr, dpliatsios@uowm.gr, psarigiannidis@uowm.gr
†Faculty of Engineering, Computing and the Environment,
Kingston University, Kingston upon Thames, United Kingdom
email: vasileios.argyriou@kingston.ac.uk
¶Department of Electrical & Computer Engineering,
Hellenic Mediterranean University, Heraklion, Greece
email: emarkakis@hmu.gr
‡Department of Informatics,
Democritus University of Thrace, Kavala, Greece
email: tlagkas@cs.duth.gr
§Department of Informatics and Telematics,
Harokopio University of Athens, Athens, Greece
email: g.th.papadopoulos@hua.gr

***Abstract*—The increasing use of containerized applications characterizes the modern era. These applications are executed within isolated packages of code, known as containers, which include libraries, binaries, configuration files, and all necessary components to run the application on any operating system, packaged into a lightweight executable file. The use of containerized applications is growing exponentially, creating a need for more efficient, scalable, and secure management of these applications. The solution to this issue was provided by Kubernetes, an open-source technology initially developed by Google and later adopted by the Cloud Native Computing Foundation (CNCF). Kubernetes automates the deployment, scaling, and management of containerized applications and has become the de-facto container orchestration framework in cloud-native environments due to its robust architecture, flexibility, and extensive ecosystem of available tools. Moreover, thanks to its extensible API Kubernetes can easily integrate security solutions. This work focuses on comparing the resource consumption and performance of four Kubernetes distributions, namely K3s, K0s, MicroK8s and K8s. The comparison results highlight that K3s and MicroK8s are suitable for lightweight deployments and resource-constrained environments, K0s is ideal for single-node environments and K8s is well suited for large-scale environments.**

***Index Terms*—cloud computing, computing continuum, edge computing, Kubernetes, resource orchestration, fog computing**

I. Introduction

The advent of the Cloud has induced several changes related to application deployment, management and development. Traditionally applications were created as a large monolithic binary file often using proprietary protocols and custom communication interfaces. However, the strict requirements of modern applications in terms of new features delivery, response time and scalability make monolithic development approaches obsolete. Modern applications are shipped as containerized applications, focusing on portability scalability and high availability of services. This is achieved by splitting the application into several containers and orchestrating them through container orchestration engines like Kubernetes (K8s).

The core objective of any container orchestration system, like K8s, is to efficiently manage workloads by identifying the most appropriate node to deploy a container. This process involves several actions, such as excluding non-eligible nodes, evaluating the remaining nodes' states, and attempting to balance the number of containers per node based on predefined criteria.

Although Kubernetes has become the de-facto orchestration engine for containerized applications, it remains a complex tool built on multiple abstraction layers [1]. Its extensive plugin capabilities, which integrate various services and functionalities, further contribute to this complexity. Many Kubernetes distributions have emerged since the initial version of kubernetes (K8s), focusing on simplifying the deployment and onboarding of new nodes process while remaining compliant with the Kubernetes specification.

Recent advances in edge computing and the growing de-

mand for efficient edge-to-cloud continuum approaches are pushing organizations towards distributing their applications to multi-cluster environments and heterogeneous infrastructures [2]–[4]. Nonetheless, the decentralization of workloads across multi-cluster environments introduces new security challenges, including inter-cluster authentication, data integrity, and secure communication across heterogeneous infrastructures [5]. To mitigate the lack of computing resources and minimize the deployment time of a new cluster in edge sites without sacrificing the flexibility and advantages modern orchestrating platforms offer, organizations deploy lighter Kubernetes distributions like K3s or K0s. However, the need to effortlessly deploy applications across the edge-to-cloud continuum remains [6].

This work constitutes a comparative study of modern open-source Kubernetes distributions focusing on their ability to handle computationally heavy workloads. The goal is to identify practical differences between the distributions and limitations in terms of performance during heavy workload scenarios. In more detail, the contributions of this work are as follows:

- An overview of the Kubernetes architecture and its main components is provided.
- Four Kubernetes distributions, namely K8S, K3s, MicroK8s, and K0s, are presented and described.
- The Grafana k6 tool is used to assess the performance of the distributions in terms of CPU and memory utilization under various configuration parameters.

The rest of the paper is structured as follows: Section II presents the Kubernetes architecture and related literature, Section III presents Kubernetes architecture and the architectural differences of three Kubernetes distributions, including the vanilla distribution. Section IV provides details related to testbed infrastructure setup, configuration of clusters and stress testing tools. Finally, Section V concludes our findings and suggests future activities.

II. Related Work

Container orchestration, especially with Kubernetes, is increasingly recognized for its role in effectively automating and scaling applications. However, there is still a lack of in-depth research that examines its performance and efficiency across various distributions, emphasizing the need for more focused studies in this area.

The authors of [7] presented an analysis of container orchestration tools, including Kubernetes, Docker Swarm, Apache Mesos, and Cattle. They evaluated them based on performance metrics such as provisioning time, scalability, and failover mechanisms. The experiments highlighted the strengths and limitations of each tool, indicating Kubernetes's suitability for complex application deployments, while simpler orchestrators are suitable for more straightforward scenarios. The insights offered by this study aim to assist IT managers in selecting the most appropriate orchestration tool.

The authors of [8] provided a comparative study of the performance and security of three Kubernetes distributions: K8s, K3s, and K0s. Their analysis covers aspects such as scalability, latency, and resource consumption under various workloads, along with security assessments. Results indicate that K0s generally perform best in terms of pod creation throughput and scalability, while K3s face scalability limitations. Kubernetes demonstrates fewer security vulnerabilities compared to the other distributions, making it the most robust option for secure deployments. This study aims to aid practitioners in selecting the appropriate Kubernetes distribution for different resource-constrained and security-sensitive environments.

In [9], Koziolek and Eskandani compared four lightweight Kubernetes distributions, namely MicroK8s, K3s, K0s, and MicroShift examining their performance in terms of resource usage, control plane throughput, and data plane latency under stress. Results show that K3s and K0s perform best in control plane operations, while MicroShift leads in data plane throughput. Each distribution presents trade-offs between performance, usability, and security, providing insights for developers in selecting the most suitable lightweight Kubernetes solution for edge and IoT environments.

Böhm and Wirtz in [10] provided a comparative analysis of Kubernetes distributions such as MicroK8s and K3s against standard Kubernetes, focusing on their resource consumption and operational efficiency during typical lifecycle events, such as starting, stopping, and adding nodes. Results indicate that while K3s offers performance advantages in terms of resource utilization and speed for certain operations, MicroK8s incurs higher resource usage and time consumption across most lifecycle steps. This study offers insights into the suitability of lightweight Kubernetes options for resource-constrained applications, such as IoT and edge computing environments.

In [11], Vasireddy *et al.* present a comprehensive review of load-balancing solutions in Kubernetes, an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. As modern distributed systems increasingly rely on efficient resource utilization, Kubernetes plays a critical role in dynamically adjusting workload distribution among nodes to optimize performance, scalability, and availability. The research examines various load-balancing strategies, including algorithms based on resource awareness. Key challenges, such as fluctuating resource demands and the dynamic nature of cloud-native applications, are discussed along with trade-offs concerning response time, performance, and scalability. The review also explores advancements in Kubernetes load balancing for edge computing, providing insights for future improvements in resource orchestration. Through this analysis, the paper consolidates existing knowledge and outlines directions for further research on enhancing resource utilization and load balancing in Kubernetes environments.

III. Modern Container Orchestrators

Kubernetes is built on a distributed, modular architecture focusing on the scalability and fault tolerance of distributed applications through a declarative model [12]. A set of nodes are joined together to form a cluster, where workloads are distributed and coordinated. The most fundamental unit of

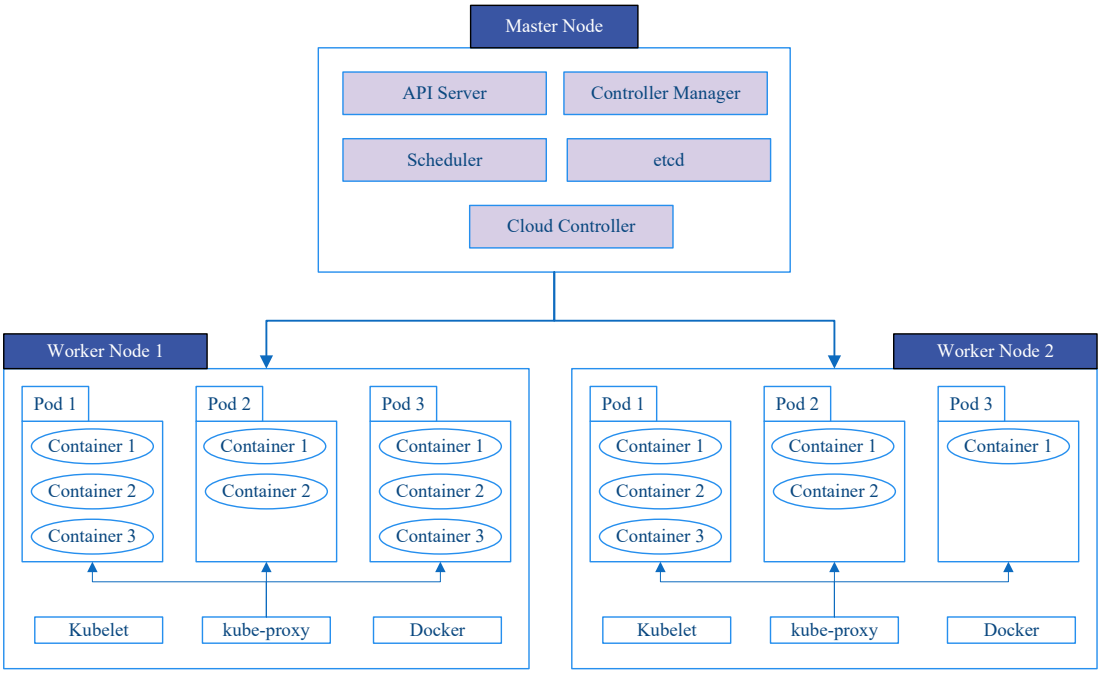

Fig. 1. Kubernetes Architecture

Kubernetes is the Pod, representing a single instance of a running service. A Pod includes one or more tightly coupled containers that cooperatively offer a service. A Kubernetes cluster deploys Pods on nodes and interconnects them through the Container Network Interface (CNI). The CNI is responsible for IP Address Management, routing between Pods and translation of virtual IP addresses to Pod addresses. To realise this, the Kubernetes specification was designed in a modular and extensible manner. Figure 1 describes the architecture of a Kubernetes cluster and consists of the following components:

- Kubelet: a process that runs on each cluster node and ensures containers inside Pods are running.
- kube-scheduler: responsible for assigning new Pod to the most appropriate node.
- kube-controller-manager: controller processes that constantly monitor the cluster and make changes to bring the cluster to the desired state.
- kube-api-server: the main interface between the cluster and third-party applications.
- cloud-controller-manager: a component that links the cluster with external cloud providers.
- etcd: a consistent and distributed data store where cluster state is stored.
- kube-proxy: a networking component responsible for the resolution of virtual IPs to actual Pod IP addresses. This component is often replaced by CNI implementations that offer more features.

Since the initial release of Kubernetes, several distributions have emerged, many are open-source while others are offered as services by large cloud providers [13]. This section provides an overview of the Kubernetes distributions being analyzed, outlining their characteristics and functionalities.

A. K8s

K8s (also known as vanilla Kubernetes) is the version of Kubernetes distributed by the CNCF and contains the core components and basic functionalities. It is the most up-to-date version of Kubernetes compared to other distributions. It consists of the control plane, which includes the API server, scheduler, etcd, and controller manager and the worker nodes, which contain the kubelet, container runtime, and kube-proxy. It can be installed in various environments, whether local or large-scale. Additionally, it allows some components to be installed externally to the cluster, with communication maintained, providing flexibility and customization depending on the needs of the environment and the administrator [14].

B. K3s

K3s is a lightweight container orchestrator developed by Rancher Labs and certified by the CNCF. It is distributed as a single binary, containing only the core components of Kubernetes: kube-apiserver, kube-scheduler, kubelet, kube-controller-manager, and kube-proxy, which the administrator can replace with alternatives of their choice. It uses SQLite instead of etcd for cluster data storage but also supports etcd, MySQL, and PostgreSQL for larger-scale clusters. Networking is achieved through a CNI plugin, Flannel. K3s includes Traefik as an ingress controller, allowing HTTP and HTTPS traffic from outside the cluster to be routed to services. Its architecture is relatively simple. It consists of a server node, which holds the control plane components and storage, and an agent node. Both nodes contain kubelet, container runtime,

and CNI. Finally, each of the server and agent components is contained in a single process [15].

C. MicroK8s

MicroK8s was developed by Canonical and is a lightweight container orchestrator compatible with Kubernetes. It includes all core Kubernetes components such as Kubernetes API server, scheduler, controller manager, kubelet, container runtime (containerd), and kube-proxy allowing for quick startup and low resource consumption. These components and services are packaged in a snap package, which is a bundle of an application and its dependencies that can be installed and run across various Linux distributions, enabling administrators to install MicroK8s with a single command. One of its key features is that it allows administrators to enable or disable components as needed, thereby reducing resource usage by only running essential elements in the cluster. MicroK8s supports multiple nodes with a single command and manages the distribution of pods across nodes, maintaining control at the master node level [16]. Finally, networking is achieved using a CNI plugin, Calico.

D. K0s

K0s is an open-source, single binary Kubernetes distribution that contains all the components needed to create a Kubernetes Cluster. It was developed by Mirantis and can be deployed from bare metal servers to edge devices. K0s reduces the complexity with a lightweight installation process while maintaining full compatibility with Kubernetes APIs. Multiple Kubernetes components are combined into one executable without the requirement of external dependencies, making deployment easy and efficient. It is optimized for low-resource environments such as IoT and edge devices but also supports multi-node clusters for larger environments. Includes native support for high availability, allowing the creation of highly resilient Kubernetes clusters. Lastly, K0s supports custom networking solutions, allowing users the flexibility of choosing their preferred CNI plugin [17].

IV. Evaluation Testbed and Results

A. Grafana k6

Grafana k6 [18] is an open-source load testing tool designed to be an efficient and developer-friendly approach to performance testing. The test scripts are written in JavaScript enabling developers to identify and resolve issues early in the development cycle before they reach production.

Initially launched by a startup k6 in 2016 and later acquired by Grafana Labs in 2021. The aim of the project is to provide teams with versatile and extensible load-testing tools for building and monitoring reliable applications. Grafana k6 also offers more testing tools, aside from the standard ones, such as browser testing and fault injection testing to address a wider range of performance testing needs.

The script that was used defined a stress-testing scenario simulating user activity on the Sink endpoint through HTTP protocol. There were three stages, the ramp-up stage where the test gradually increased the number of virtual users from 1 to 8000 over three minutes, assessing how the system handles a growing number of requests. This is followed by the sustained load stage, which maintains those 8000 users for five minutes to test the system's stability under consistent traffic. Finally, the ramp-down stage reduced the user count to zero over a period of three minutes, evaluating the system's ability to recover as traffic decreases. The rationale behind the test is to send HTTP GET requests to the endpoint, with a one-second delay between them to mimic real-world user behaviour.

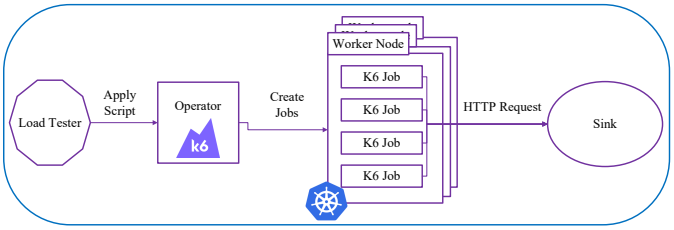

Fig. 2. Grafana k6

B. Testbed Infrastructure

The infrastructure on which the topologies for the three resource orchestrator distributions compatible with Kubernetes were based is as follows:

- It contains 64 Intel(R) Xeon(R) 2.10 GHz processors, ensuring that the cluster nodes will have sufficient resources to run the pods and handle heavy workloads.
- The Cluster's memory is 252GB, sufficient to support multiple Virtual Machines simultaneously.
- It contains 24 TB of storage, enough to store a large data load without the risk of running out of space.

Due to the above, it is ensured that no bottleneck will occur during the execution and management of a large workload.

The evaluation parameters are summarized in Table I. Each cluster has three nodes in total of which one is the control plane and all three are eligible for allocation of workloads, whereas each node is allocated 4 CPU cores, 8 GB of memory, and 150 GB of storage. Each node has Ubuntu 22.04 with kernel version 5.15.0 installed and Kubernetes version 1.31 with contained 1.7.22 as the container runtime environment. Finally, four Kubernetes distributions are evaluated, namely K3s, K0s, K8s, and MicroK8s.

C. Results

The comparison results are illustrated in Figures 3, 4, 5 and 6, and depict the cluster-average CPU and memory consumption over time and the peak CPU and memory utilisation per node. The cluster-average results depict how each Kubernetes distribution handled the workload as a whole, whereas the node-peak figures depict peak CPU and memory utilization for each node within the cluster.

Fig. 3 illustrates the average CPU resource utilization of all four Kubernetes distributions. Three out of four distributions demonstrated similar behaviour with minor variations in terms

TABLE I
EVALUATION PARAMETERS

Parameter	Value
Nodes in Cluster	3
Cluster Worker Nodes	3
Cluster Control Plane Nodes	1
CPU Cores per Node	4
Memory per Node	8 GB
Storage Capacity per Node	150 GB
Operating System	Ubuntu 22.04
Kernel Version	5.15.0
Containerd Version	1.7.22
Kubernetes Version	1.31
Kubernetes Distribution	K3s, K0s, K8s, MicroK8s

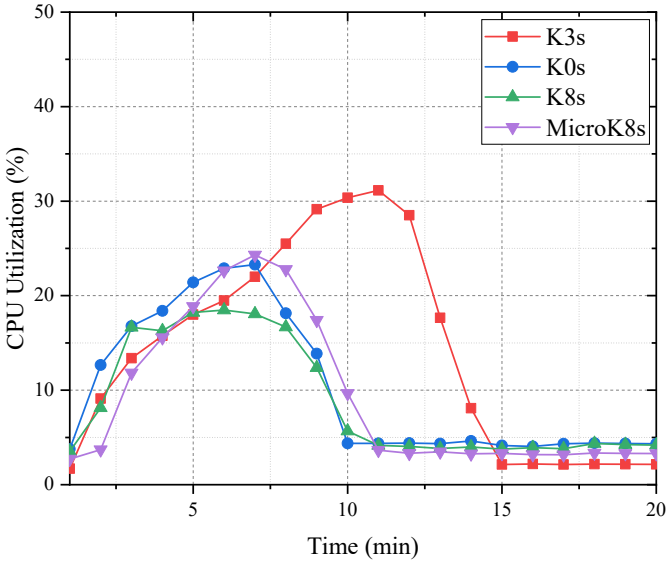

Fig. 3. Kubernetes distributions total cluster CPU Utilization

of ramp-up stage finilization and peak CPU consumption. The K3s distribution ramp-up stage was the slowest and demonstrated the highest peak cluster CPU utilization. In terms of node peak CPU utilization the K3s and MicroK8s demonstrated the least variation of CPU utilization (i.e. they distributed the workload evenly), while K0s and K8s distributions scheduled the workload on a single node (Node 1 and Node 3 respectively). It is noteworthy to mention that K8s distribution had the least peak cluster CPU utilization, while K3s distribution demonstrated the highest peak cluster utilization.

The total cluster memory utilization of all distributions is depicted in Fig. 4. K0s and K8s distribution demonstrate similar behaviour in terms of memory utilization over time, whereas MicroK8s achieved the least total memory utilization. The K3s demonstrated marginal changes in terms of memory

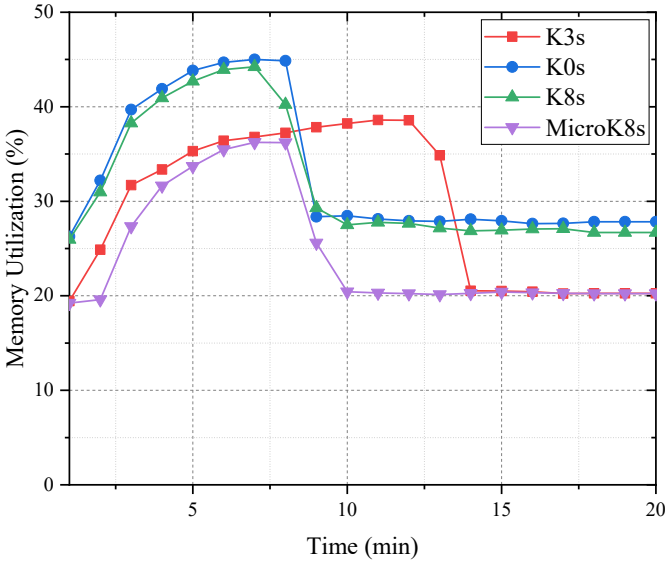

Fig. 4. Kubernetes distributions total cluster Memory Utilization

consumption. This however impacted the total duration of the stress test. It is important to highlight that K0s and K8s consume more memory, even in an idle state, compared to MicroK8s and K3s.

Finally, the peak node memory consumption is shown in Fig. 6. The results show that one node consumed considerably more memory than the others in three out of four distributions. One could argue, that this is attributed to a node acting as a control plane and worker node, however, this is not true since K8s Node 3 is a worker node. K3s distribution achieved uniform memory utilization across nodes.

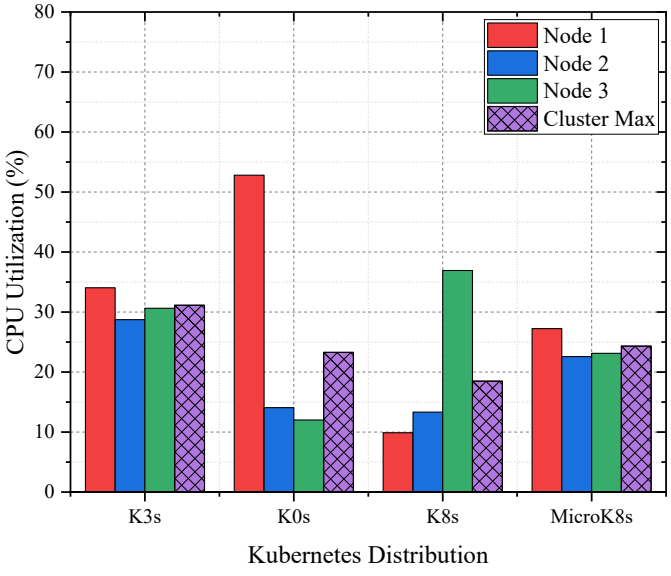

Fig. 5. Kubernetes distributions nodes and cluster peak CPU utilization

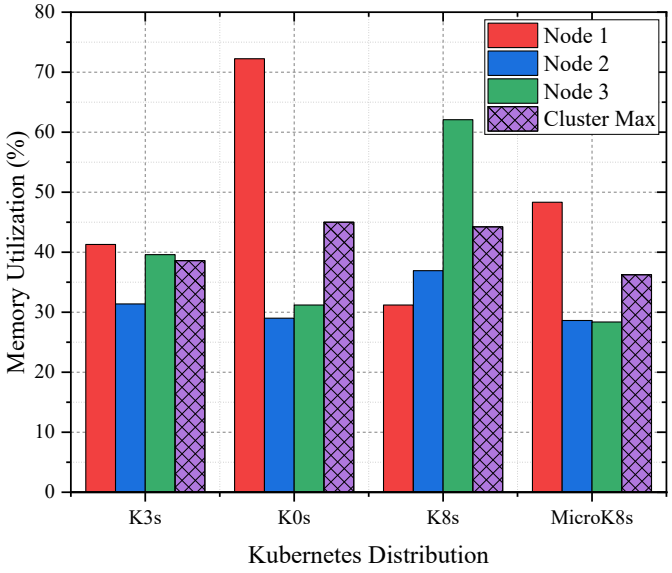

Fig. 6. Kubernetes distributions nodes and cluster peak memory utilization

V. Conclusion

This analysis examined the CPU and Memory usage of four Kubernetes distributions, namely K3s, K0s, K8s, and MicroK8s. Compared how each handled workload distribution, resource efficiency and scalability and aimed to provide insights into their performance and suitability for different use cases.

The results revealed the strengths and the trade-offs of each distribution. To begin with, K3s and MicroK8s had a balanced workload distribution across nodes but higher resource usage, with MicroK8s being more efficient in resource management, making them ideal for environments with limited resources, such as IoT and edge computing. K0s was efficiently using resources but relied heavily on a single node for the workload, as it was not substantial enough to distribute across its nodes. K8s demonstrated robust performance overall and it primarily utilized a single node due to its ability to efficiently manage lighter workloads, however in the case of heavier workloads it would have effectively used all available nodes, making it suitable for large-scale environments.

Our findings indicate that K3s and MicroK8s are ideal solutions for edge deployments with constant computational loads, whereas K0s and vanilla can manage rapid computational spikes more efficiently. This makes K0s a good candidate for both edge and core sites. Finally, K8s is ideal for large deployments. In the future, we aim to extend our work to interconnected cluster scenarios and study how multiple distributions impact the overall performance of distributed applications.

Acknowledgment

This work has received funding from the European Union's Horizon Europe research innovation action programme under grant agreement No. 101135423 – ENACT.

References

[1] C. Carrión, "Kubernetes scheduling: Taxonomy, ongoing issues and challenges," *ACM Comput. Surv.*, vol. 55, no. 7, Dec. 2022.

[2] A. Liatifis, D. Pliatsios, P. Radoglou-Grammatikis, T. Lagkas, V. Vitsas, N. Katertsidis, I. Moscholios, S. Goudos, and P. Sarigiannidis, *Edge Networking Technology Drivers for Next-generation Internet of Things in the TERMINET Project*. River Publishers, 2024.

[3] M. Mahbub and R. M. Shubair, "Contemporary advances in multi-access edge computing: A survey of fundamentals, architecture, technologies, deployment cases, security, challenges, and directions," *Journal of Network and Computer Applications*, vol. 219, p. 103726, 2023.

[4] S. Böhm and G. Wirtz, "Cloud-edge orchestration for smart cities: A review of kubernetes-based orchestration architectures," *EAI Endorsed Transactions on Smart Cities*, vol. 6, no. 18, p. e2, May 2022.

[5] S. I. Shamim, "Mitigating security attacks in kubernetes manifests for security best practices violation," in *Proceedings of the 29th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, 2021, pp. 1689–1690.

[6] A. Nizamis, J. Neises, D. Ospina, U. Wajid, C. I. V. López, P. Trakadas, K. Votis, O. Lazaro, S. Nechifor, and C. Palau, "Enact - a framework for adaptive scheduling and deployments of data intensive workloads on energy efficient edge to cloud continuum," in *2024 IEEE International Conference on Engineering, Technology, and Innovation (ICE/ITMC)*, 2024, pp. 1–9.

[7] I. M. A. Jawarneh, P. Bellavista, F. Bosi, L. Foschini, G. Martuscelli, R. Montanari, and A. Palopoli, "Container orchestration engines: A thorough functional and performance comparison," in *IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.

[8] P. Ascensão, L. F. Neto, K. Velasquez, and D. P. Abreu, "Assessing kubernetes distributions: A comparative study," in *2024 IEEE 22nd Mediterranean Electrotechnical Conference (MELECON)*. IEEE, 2024, pp. 832–837.

[9] H. Koziolek and N. Eskandani, "Lightweight kubernetes distributions: A performance comparison of microk8s, k3s, k0s, and microshift," in *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering*, 2023, pp. 17–29.

[10] S. Böhm and G. Wirtz, "Profiling lightweight container platforms: Microk8s and k3s in comparison to kubernetes." in *13th European Workshop on Services and their Composition (ZEUS 2021)*, Feb. 2021, pp. 65–73.

[11] I. Vasireddy, P. Kandi, and S. Gandu, "Efficient resource utilization in kubernetes: A review of load balancing solutions," *International Journal of Innovative Research in Engineering & Management*, vol. 10, no. 6, pp. 44–48, 2023.

[12] A. E. Nocentino and B. Weissman, *Kubernetes Architecture*. Berkeley, CA: Apress, 2021, pp. 53–70.

[13] H. Aqasizade, E. Ataie, and M. Bastam, "Kubernetes in action: Exploring the performance of kubernetes distributions in the cloud," 2024. [Online]. Available: https://arxiv.org/abs/2403.01429

[14] "Production-Grade Container Orchestration — kubernetes.io." [Online]. Available: https://kubernetes.io/

[15] "K3s — k3s.io." [Online]. Available: https://k3s.io/

[16] "MicroK8s - Zero-ops Kubernetes for developers, edge and IoT — MicroK8s — microk8s.io." [Online]. Available: https://microk8s.io/

[17] "Documentation — docs.k0sproject.io." [Online]. Available: https://docs.k0sproject.io/stable/

[18] "Grafana k6s." [Online]. Available: https://k6.io