International Journal of Innovative Research in Engineering and Management (IJIREM)
ISSN (Online): 2350-0557, Volume-10, Issue-6, December 2023
https://doi.org/10.55524/ijirem.2023.10.6.7
Article ID IJIR2481, Pages 49-54
www.ijirem.org

Kubernetes and Docker Load Balancing: State-of-the-Art Techniques and Challenges

Indrani Vasireddy[1], G.Ramya[2], and Prathima Kandi[3]

[1] Associate Professor, Department of Computer Science and Engineering, Geethanjali College of Engineering, Hyderabad, India
[2, 3] Assistant Professor, Department of Computer Science and Engineering, Geethanjali College of Engineering, Hyderabad, India

ABSTRACT - In the ever-evolving landscape of container orchestration, Kubernetes stands out as a leading platform, empowering organizations to deploy, scale, and manage containerized applications seamlessly. This survey paper explores the critical domain of load balancing within Kubernetes, investigating state-of-the-art techniques and the associated challenges. Container-based virtualization has revolutionized cloud computing, and Kubernetes, as a key orchestrator, plays a central role in optimizing resource allocation, scalability, and application performance. Load balancing, a fundamental aspect of distributed systems, becomes paramount in ensuring efficient utilization of resources and maintaining high availability. The study focuses on contemporary methods for achieving effect-ive load balancing on containers, with a specific examination of Docker Swarm and Kubernetes—prominent systems for container deployment and management. The paper illustrates how Docker Swarm and Kubernetes can leverage load bal-ancing techniques to optimize traffic distribution. Load balancing algorithms are introduced and implemented in both Docker and Kubernetes, and their outcomes are systematically compared. The paper concludes by highlighting why Kuber-netes is often the preferred choice over Docker Swarm for load balancing pur-poses.This paper provides a comprehensive overview of the current state-of-the-art techniques employed in Kubernetes load balancing. Challenges inherent to Docker load balancing are addressed, encompassing is-sues related to the dynamic nature of containerized workloads, varying application demands, and the need for real-time adaptability. The survey also explores the role of load balancing in enhancing the scalability and overall performance of applications within Kubernetes clusters. In conclusion, this survey consolidates the current knowledge on Docker and Kubernetes load balancing, offering a state-of-the-art analysis while identifying challenges that pave the way for future research and advancements in the realm of container orchestration and distributed systems.

KEYWORDS- Load Balancing, Docker Swarm, Kubernetes, Containers.

I. INTRODUCTION

Cloud computing revolutionizes IT infrastructure by providing on-demand access to a myriad of resources, including computing power, storage, databases, analytics, and intelligence, over the internet. This paradigm [2] shift allows services to be consumed on a pay-per-use basis, featuring auto-scalability and eliminating the need for dedicated server spaces. The traditional complexities of hardware and software maintenance are alleviated, as cloud computing negates the requirement for in-house experts.

Security is bolstered in the cloud environment, and software updates occur seamlessly. Data accessibility and sharing transcend local storage limitations, enabling users to save and retrieve information effortlessly over the internet. In contrast to classical deployment scenarios, where applications contend with hardware dependencies, operating system specifications, and potential conflicts between software libraries, cloud computing introduces a transformative solution known as the dependency matrix. The dependency matrix plays a crucial role in streamlining the development, packaging, deployment, and scaling of applications, ensuring ease and independence. The architectural foundation involves the integration of hardware and operating systems, complemented by a container engine like Docker. This container engine serves as a vital component, facilitating the creation of containers that encapsulate libraries and dependencies. As a result, these containers can seamlessly traverse different machines, abstracting away concerns about underlying dependencies. Each container encapsulates everything an application requires to run, thereby overcoming challenges related to conflicting library versions and dependencies. This paper delves into the intricacies of this container-based approach, exploring its impact on application development, deployment, and scalability in cloud computing environments.

Before the advent of Docker and containerization, companies typically deployed software applications on either bare metal servers or virtual machines. Docker, as an open-source platform, revolutionized this landscape by enabling users to package applications and their dependencies into Docker containers for seamless development and deployment. Containerization, as facilitated by Docker, encapsulates all necessary dependencies, including frameworks and libraries,

ensuring the efficient and error-free execution of applications across diverse computing environments. The key advantages of Docker containers include their lightweight nature, making them easily transferable and executable across different computer environments, irrespective of the host operating system or configurations.

Docker containers offer benefits such as minimal space occupancy and faster ap-plication execution. They operate in isolation, sharing the operating system kernel with other containers. The security of containers is robust, thanks to container policies that ensure a highly secure Docker infrastructure. The extended Docker API offers robust techniques for creating and scaling services, implementing health checks, load balancing, traffic distribution, and more. Services in Docker are a set of containers akin to Docker compose but with enhanced features. Docker Swarm serves as a Layer 4 TCP load balancer. In this context, we have established three Swarm nodes, comprising two worker nodes and one manager node. The manager node hosts the Swarm commands, managing load balancing, distribution, scaling, DNS service, booking, and discovery.

The Docker Swarm LB operates on all nodes, efficiently handling balance requests across any of the container/hosts within the node. In the absence of NGINX or NGINX Plus, Docker Swarm LB manages incoming customer demands within the swarm net-work. NGINX Plus boasts several advanced features that position it as an ideal load balancer for arrays of upstream servers: Load balancing and session selection: Enhances load balancing across worker processes and employs session persistence methods to recognize and honor application sessions. HTTP health checking and server slow start: Utilizes asynchronous synthetic transactions to verify the proper functioning of each up-stream server and employs an agile "slow start" approach to reintroduce servers when they recover. Live traffic monitoring: Provides immediate reporting of activity and performance. Dynamically configured upstream server groups: Offers a tool for facilitating common upstream service tasks, such as the secure and temporary removal of a server.

In the context of running a micro-services architecture comprising numerous micro-services, each potentially having varying instances[1]—ranging from one to thousands—managing the scalability, load distribution, health monitoring, and other operational aspects can be an intricate task. Kubernetes, however, emerges as a robust solution to address these challenges. Kubernetes[14] facilitates the deployment and management of diverse applications on multiple cloud platforms such as AWS, Azure, and Google Cloud. It provides the ability to create container images for applications developed in Python, Java, Node.js, and other languages, enabling seamless deployment within a Kubernetes environment.

Within Kubernetes, various services play crucial roles, including Cluster IP service, Headless service, NodePort service, and Load Balancer. The Load Balancer service, offered by Kubernetes, stands out as a vital component[15] that efficiently distributes in-coming web traffic across multiple backend servers. This distribution allows applic-ations to dynamically scale in response to demand, enhance availability, and optim-ize server capacity utilization. As traffic[10] increases, surpassing the capacity of a single server, horizontal scaling by adding more servers becomes imperative. Load balancers play a pivotal role in deciding which server should handle incoming requests when mul-tiple servers are available. A well-designed load balancer maximizes system capacity and minimizes request fulfillment time by efficiently distributing incoming traffic. Various strategies, such as Round Robin, Least Connections, and Consistent Hashing, are employed by load balancers[12] to distribute traffic effectively. Round Robin assigns servers in a repeating sequence, ensuring the equitable utilization of servers. Least Connections directs traffic to the server currently handling the fewest requests, while Consistent Hashing ensures consistent assignment based on IP address or URL, akin to database sharding.

This paper is organized as follows: Section 2 presents the related work, Section 3 presents the Discussions, Section 4 presents the Challenges Section 5 Future enhancement, and Section 6 conclusion.

II. RELATED RESEARCH

The realm of Kubernetes[3] scheduling algorithms has emerged as a central focus for both researchers and practitioners, garnering considerable attention in recent years. This review meticulously navigates through the intricate landscape of scheduling algorithms within Kubernetes, exploring a diverse range of theories, methodologies, and insights gleaned from prior studies. With the contemporary[4] surge in the adoption of micro-services in software system design, load balancing has become a pivotal element in micro-services architecture, exerting a profound impact on resource utilization and the overall efficiency of service functionality. The literature emphatically underscores the crucial requirement for adept scheduling of workloads in Kubernetes environments.

In the study conducted by Toka et al. [11], their research reveals that the proposed scheme significantly enhances system performance during periods of rapid load pressure, exhibiting a decrease in operational cluster instability compared to the default auto scaler. Li et al. [16] present two dynamic scheduling algorithms, Balanced-Disk-IO-Priority (BDI) and Balanced-CPU-Disk-IO-Priority (BCDI). BDI aims to enhance disk I/O balance between nodes, while BCDI addresses the issue of load imbalance between CPU and disk I/O on a single node.

Ismail, Bukhary Ikhwan, et al. [20] conduct an evaluation of Docker as an edge computing technology. Their assessment, based on deployment and termination, resource and service management, fault tolerance, and caching, highlights Docker's advantages in rapid and efficient deployment, low overhead, and overall good performance, positioning it as one of the best technologies for edge computing platforms. Cito, Jürgen Ferme, Vincenzo C. Gall, Harald (2016) et al. [17] explore the utilization of Docker containers to enhance reproducibility in software and web engineering research. Emphasizing Docker's role in supporting research artifact reproducibility, the study underscores its significance in the current virtualization landscape, particularly in the deployment and production of web applications.

Ahmed et al. [19] delve into the deployment of Docker containers in a heterogeneous cluster using a dynamic scheduling framework for Kubernetes named KubCG. The platform optimizes container deployment by considering Kubernetes Pod timelines and prior data on container execution, providing insights for efficient resource utilization in CPU and GPU resources. Weizheng Ren et al. [18] propose enhancements to existing load balancing algorithms in the Dynamic Balance Strategy of High Concurrent Web Cluster Based on Docker Container. They aim to make load balancing a dynamic technique dependent on traffic, comparing Docker and Kubernetes architecture and ad-dressing drawbacks in the existing NGINX load balancing strategy. The study calculates real-time performance weight ratios of the cluster by weight.

Docker and Kubernetes, as open-source tools, have revolutionized the creation, deployment, and management of applications across diverse platforms, particularly in the realms of micro-services and cloud-based services. These technologies play a pivotal role in various domains of information technology, offering versatile applications. Load balancing, a crucial aspect of their functionality, remains a focus of research, with studies exploring scheduling strategies, enhancing security, and optimizing dynamic load balancing[13] techniques in cloud platforms. Our research paper aims to provide a com-prehensive analysis of the dynamic load balancing capabilities offered by Docker and Kubernetes. We delve into various load balancing techniques, outlining their ideal use cases and comparing them against alternative methods.

In the context of State machine replication in containers managed by Kubernetes [7], the authors propose the integration of coordination services in Kubernetes to con-trol container size and enable automatic state replication. They introduce the DORADO protocol and evaluate its performance through preliminary tests, detailing the execution environment and experimental setup. The conclusion outlines challenges and future prospects for the protocol. In the paper on a decentralized system for load balancing of containerized micro-services in the Cloud [2], the authors present a decentralized or-chestration system for load balancing in containerized microservices. They discuss container limitations for load balancing and introduce a swarm-like algorithm for container migration. The paper concludes with preliminary experimental results and summary remarks. The literature survey covers static algorithms such as Load Balancing Min-Min Algorithm, Load Balancing Min-Max Algorithm, and Round Robin Load Balancing Algorithm. The study evaluates these techniques based on parameters like fairness, throughput, fault tolerance, overhead, performance, response time, and resource utilization.

III. DISCUSSIONS

In the literature review section, a thorough examination has been presented, encompassing load balancing in Docker and four sub-categories within the realm of Kubernetes scheduling. It is imperative to provide a concise discussion on the categorized literature review that is outlined in this section. The proliferation of public cloud vendors has prompted the inclusion of Containers as a Service (CaaS) in their offerings. This surge in popularity is attributed to Docker, a software enabling Linux containers to op-erate independently within an isolated environment on a host. The orchestration and allocation approaches depend on the specific software in use. The central aim of the study was to examine how container execution evolves over time. Two dynamic alloc-ation algorithms were employed and compared against the default Docker algorithm. The efficiency of these algorithms is contingent upon the workload's weight and scales proportionally with the increasing number of nodes in the Cloud.

In the context of a Portable Load Balancer for Kubernetes Cluster, Linux contain-ers have garnered favor due to their lightweight and portable characteristics. Presently, numerous web services[6] are deployed as clusters of containers. The paper focuses on Kubernetes Clusters, but it's noted that Kubernetes relies on load balancing provided by cloud providers. To address this, the authors proposed a portable load balancer applicable in any environment, facilitating the migration of web services. This solution leveraged the Linux kernel's Internet Protocol Virtual Server (IPVS), resulting in an enhanced portable web service without compromising performance.

load balancing[5] in docker can be achieved using NGINX. NGINX is a versatile web server employed as a reverse proxy, HTTP cache, and load balancer. Engineered for high concurrency and minimal memory usage, it adopts an asynchronous approach, executing requests through a single thread instead of creating a new process for each web request. In the NGINX architecture, a master process oversees multiple worker processes. The master effectively manages the workers, which handle the actual processing of the server. The asynchronous nature of NGINX ensures that each received request can be executed by a worker concurrently, without impeding other requests. Notably, there are two types of load balancers: the open-source NGINX and NGINX Plus. NGINX Plus, being the commercial variant, introduces critical application features not present in the native Swarm load balancer.

Within the domain of multi-objective optimization-based scheduling in Kubernetes, numerous research studies have been undertaken to optimize diverse objectives. These

objectives include minimizing energy consumption and cost, maximizing resource utilization, and meeting application performance requirements. Various optimization techniques, such as genetic algorithms, particle swarm optimization, and ant colony optimization, are employed in these studies. Additionally, some research incorporates ma-chine learning-based approaches to predict workload patterns and inform scheduling decisions. Challenges persist in this area, including [9] the multi-objective nature of the problem, which poses a significant challenge in finding optimal solutions that balance conflicting objectives. The dynamic nature of the cloud environment also necessitates real-time adaptation of scheduling decisions to changing conditions. Despite these challenges, research in multi-objective optimization-based scheduling in Kubernetes shows great potential for achieving efficient and effective resource management, with further work needed to

address challenges and validate approaches in real-world scenarios.

Conversely, AI-based scheduling in Kubernetes has become a prominent area of research in recent years. Numerous studies propose different approaches to optimize scheduling decisions using machine learning and other AI techniques. Notable achievements include the development of scheduling algorithms capable of handling complex workloads in dynamic environments. These algorithms consider factors such as re-source availability, task dependencies, and application requirements to make optimal scheduling decisions. Reinforcement learning-based scheduling algorithms, which ad-apt to changing workload patterns and learn from experience, as well as deep learning-based approaches that capture complex patterns in workload data, have been proposed. While these studies demonstrate that AI-based[8] scheduling improves efficiency and performance in Kubernetes clusters, challenges remain. The lack of real-world datasets for training and evaluating AI-based scheduling algorithms is a significant challenge, and there is a trade-off between accuracy and computational complexity. Future research in this area could focus on developing more efficient and scalable AI-based scheduling algorithms capable of handling large-scale, real-world workloads.

Finally, autoscaling-enabled scheduling emerges as a nascent research area aiming to optimize resource utilization and enhance application performance by combining autoscaling and scheduling techniques. Finally, autoscaling-enabled scheduling emerges as a nascent research area aiming to optimize resource utilization and enhance application performance by combining autoscaling and scheduling techniques. Several recent studies indicate significant improvements in resource utilization and application performance through autoscaling-enabled scheduling, reducing resource wastage and improving response times. How-ever, challenges persist, particularly in designing effective autoscaling-enabled scheduling algorithms that can adapt to dynamic workload changes while maintaining application performance. Practical implementation in real-world scenarios also requires fur-ther research, as existing studies often occur in controlled experimental settings. Challenges in algorithm design, standardization, and practical implementation need to be addressed, urging future research to focus on developing more effective and practical autoscaling-enabled scheduling techniques. How-ever, challenges persist, particularly in designing effective autoscaling-enabled schedul-ing algorithms that can adapt to dynamic workload changes while maintaining applic-ation performance. Practical implementation in real-world scenarios also requires further research, as existing studies often occur in controlled experimental settings. Challenges in algorithm design, standardization, and practical implementation need to be addressed, urging future research to focus on developing more effective and practical autoscaling-enabled scheduling techniques.

IV. CHALLENGES

Through this research, a notable observation emerges regarding different load balancing techniques: a lack of clear understanding regarding why specific algorithms outperform others and how each can be enhanced. Despite researchers attempting to implement or improve load balancing algorithms, a comprehensive transformation in load balancing within container environments has not been achieved. Given the evolving nature of containers, considerable research remains, and the importance of load balancing is growing. As we progress, emphasizing load balancing becomes crucial for enhancing system performance and reducing carbon emissions. Our aim is to identify algorithms suitable for various purposes and implement these techniques while comparing their performance. Presently, there is widespread acknowledgment that, concerning container orchestration, Kubernetes outperforms Docker Swarm. Kubernetes's scalability, portability, and self-healing attributes contribute to its preference over Docker Swarm. Kubernetes's longer existence and extensive documentation add to its popularity. Our investigation seeks to uncover why Kubernetes is the preferred choice for implementing load balancing.

The research papers employ diverse algorithms to enhance Kubernetes scheduling, tested across various platforms and environments. The survey thoroughly analyzes the current literature, forming a taxonomy to not only assess the current state-of-the-art but also identify challenges and future directions. Areas identified for potential future re-search include the growing need for advanced computation optimization techniques as Kubernetes popularity rises, with a focus on sophisticated algorithms, potentially incorporating AI or machine learning. Integration with emerging technologies like serverless computing could enhance resource usage. Testing and implementation to reveal limitations of current learning algorithms for scheduling, emphasizing tooling and automation improvement, along with the development of testing frameworks. Continuous refinement of Kubernetes' implementation and development process is crucial, including comprehensive testing and validation strategies. The future of testing and implementation in Kubernetes involves ongoing innovation, collaboration, and a commitment to driving the platform forward.

V. FUTURE RESEARCH

Before the advent of container technologies, deploying applications was a time-consuming manual process, consuming significant company resources. The introduction of container technologies, particularly Docker and Kubernetes, revolutionized and standardized the deployment process, making it more efficient.

In the current landscape, load balancing for containerized applications comes in various forms tailored to different use cases. Docker has significantly simplified the capacity to scale, providing built-in features for service discovery and load balancing. Developers now spend less time creating these support functions independently and more time focusing on their applications. Docker automates tasks such as setting DNS for service discovery and adding applications to the load balancer pool when scaling is required, enabling organizations to deploy highly available and scalable applications in a shorter timeframe.

The realm of Kubernetes resource management predominantly relies on optimization modeling frameworks and heuristic-based algorithms. A promising avenue for future research involves enhancing and proposing novel resource management algorithms. This prospect directs future investigations towards overcoming the challenges associated with handling intricate and dynamic workloads across distributed and heterogeneous environments within Kubernetes. Research efforts may delve into the development of more intricate algorithms and techniques addressing workload placement, resource allocation, and load balancing. Additionally, there exists an opportunity to explore innovative approaches to containerization and virtualization. The integration of emerging technologies such as edge computing and 5G networks may further unlock avenues for achieving more efficient and scalable resource management within Kubernetes.

The majority of research efforts in Kubernetes scheduling have predominantly centered around evaluations conducted on small clusters. However, an intriguing future research direction involves scaling up the cluster sizes for algorithmic evaluation. Despite Kubernetes demonstrating effectiveness in managing clusters comprising several thousand nodes, there arises a crucial need to assess its performance in even larger cluster sizes. This evaluation encompasses scrutinizing the scalability of the Kubernetes scheduler, pinpointing potential bottlenecks, and proposing viable solutions to address them. Furthermore, it becomes imperative to evaluate the repercussions of larger cluster sizes on application performance and resource utilization. Exploring this avenue of research holds the potential to unveil more efficient scheduling algorithms and enhanced management strategies tailored for large-scale Kubernetes deployments.

Despite these advancements, existing methods still face efficiency challenges, and the scalability of new algorithms is limited for widespread use. The increasing adop-tion of containers makes load balancing an essential requirement, necessitating further research into efficient implementations

VI. CONCLUSIONS

Scaling and reaping the benefits of Docker has never been easier. Docker comes equipped with built-in service discovery and load balancing features, alleviating developers from the task of creating these supporting functionalities themselves. Instead of dedicating time to manual API calls for setting DNS in service discovery, Docker automates the process. When scaling an application becomes necessary, Docker seamlessly integrates it into the load balancer pool. By leveraging these features, organizations can deploy highly available and adaptable applications in a significantly reduced timeframe.In sum-mary, the survey on Kubernetes scheduling offers a comprehensive examination of the present landscape in this field. It delves into the objectives, methodologies, algorithms, experiments, and outcomes of diverse research endeavors within this domain. The sur-vey underscores the significance of scheduling in Kubernetes and underscores the ne-cessity for proficient and effective scheduling algorithms. Despite the advancements made, the experimental results indicate opportunities for enhancements in this realm, emphasizing the need for ongoing efforts to devise novel algorithms and enhance exist-ing ones. In conclusion, the survey contributes valuable insights into the existing status of Kubernetes scheduling and directs attention toward promising avenues for future research.

CONFLICTS OF INTEREST

The authors declare that they have no conflicts of interest.

REFERENCES

[1] Chang, C., Yang, S., Yeh, E., Lin, P. & Jeng, J. A Kubernetes-based monitoring platform for dynamic cloud resource provisioning. GLOBECOM 2017-2017 IEEE Global Communica-tions Conference. pp. 1-6 (2017)

[2] Zhong Z, Buyya R (2020) A Cost-Efficient Container Orchestration Strategy in Kubernetes-Based Cloud Computing Infrastructures with Heterogeneous Resources. ACM Trans Internet Technol 20(2):1–24

[3] Kim SH, Kim T (2023) Local scheduling in kubeedge-based edge computing environment. Sensors 23(3):1522

[4] Peng Y, Bao Y, Chen Y, Wu C, Guo C (2018) Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters. Proceedings of the 13th EuroSys Conference, EuroSys

[5] Mao H, Schwarzkopf M, Venkatakrishnan SB, Meng Z, Alizadeh M (2019) Learning schedul-ing algorithms for data processing clusters. SIGCOMM Conference of the ACM Special In-terest Group on Data Communication. pp 270–288

[6] Chaudhary S, Ramjee R, Sivathanu M, Kwatra N, Viswanatha S (2020) Balancing effi-ciency and fairness in heterogeneous GPU clusters for deep learning. Proceedings of the 15th European Conference on Computer Systems, EuroSys

[7] Kubernetes: Available: http://kubernetes.io/.

[8] Taherizadeh S, Stankovski V (2019) Dynamic multi-level auto-scaling rules for containerized applications. Computer J 62(2):174–197

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

[10] Jain, N., Mohan, V., Singhai, A., Chatterjee, D. & Daly, D. Kubernetes Load-balancing and related network functions using P4. Proceedings Of The Symposium On Architectures For Networking And Communications Systems. pp. 133-135 (2021)

[11] Toka L, Dobreff G, Fodor B, Sonkoly B (2021) Machine Learning-Based Scaling Manage-ment for Kubernetes Edge Clusters. IEEE Trans Netw Serv Manage 18(1):958–972

[12] Masne, S., Wankar, R., Raghavendra Rao, C. & Agarwal, A. Seamless provision of cloud services using peer-to-peer (p2p) architecture. Distributed Computing And Internet Techno-logy: 8th International Conference, ICDCIT 2012, Bhubaneswar, India, February 2-4, 2012. Proceedings 8. pp. 257-258 (2012)

[13] Kim SH, Kim T. Local Scheduling in KubeEdge-Based Edge Computing Environment. Sensors (Basel). 2023 Jan 30;23(3):1522. doi: 10.3390/s23031522. PMID: 36772562; PM-CID: PMC9921110.

[14] Wankar, Rajeev. (2008). Grid Computing with Globus: An Overview and Research Chal-lenges. International Journal of Computer Science Applications.

[15] Vasireddy, Indrani, Rajeev Wankar, and Raghavendra Rao Chillarige. "Recreation of a Sub-pod for a Killed Pod with Optimized Containers in Kubernetes." International

Conference on Expert Clouds and Applications. Singapore: Springer Nature Singapore, 2022.
[16] Li D, Wei Y, Zeng B (2020) A Dynamic I/O Sensing Scheduling Scheme in Kubernetes. ACM International Conference Proceeding Series. Pp 14–19
[17] Cito, Jürgen & Ferme, Vincenzo & C. Gall, Harald. (2016). Using Docker Containers to Improve Reproducibility in Software and Web Engineering Research. 609-612. 10.1007/978-3-319-38791-8_5
[18] Dynamic Balance Strategy of High Concurrent Web Cluster Based on Docker Container. Weizheng Ren et al 2018 IOP Conf. Ser.: Mater. Sci. Eng. 466 012011
[19] El Haj Ahmed G, Gil-Castiñeira F, Costa-Montenegro E (2021) KubCG: A dynamic Kubernetes scheduler for heterogeneous clusters. Software Pract Experience 51(2):213–234
[20] Ismail, Bukhary Ikhwan, et al. "Evaluation of Docker as edge computing platform." 2015 IEEE Conference on Open Systems (ICOS). IEEE, 2015.