A Cloud-Based Container Microservices: A Review on Load-Balancing and Auto-Scaling Issues

Shamsuddeen Rabiu [a,1,*], Chan Hauh Yong [a,2], Sharifah Mashita Syed Mohamad [a,3]

[a] School of Computer Sciences, University Sains Malaysia,11800 USM, Pulau Pinang Malaysia.
[1] shamsrabiu@student.usm.my; [2] hychan@usm.my; [3] mashita@usm.my
* corresponding author

ARTICLE INFO

Article history
Received March 20, 2022
Revised August 5, 2022
Accepted October 22, 2022

Keywords
Microservice
Container
cloud-based
load balancing
auto-scaling.

ABSTRACT

Microservices are being used by businesses to split monolithic software into a set of small services whose instances run independently in containers. Load balancing and auto-scaling are important to cloud features for cloud-based container microservices because they control the number of resources available. The current issues concerning load balancing and auto-scaling techniques in Cloud-based container microservices were investigated in this paper. Server overloaded, service failure and traffic spikes were the key challenges faced during the microservices communication phase, making it difficult to provide better Quality of Service (QoS) to users. The aim is to critically investigate the addressed issues related to Load balancing and Auto-scaling in Cloud-based Container Microservices (CBCM) in order to enhance performance for better QoS to the users.

1. Introduction

Over the last decade, microservices have become the design style of choice for scalable, evolving cloud-based applications. For a well-designed microservice architecture with improved QoS, a solid understanding of relevant quality attributes is required.

A well-designed microservice architecture with improved QoS is dependent on a thorough understanding of the quality attributes involved [1]. With the emergence of some complex business scenarios, many enterprise business requirements have also increased dramatically, and the existing monolithic architecture often cannot meet the needs. Because of the increase in business volume, the system functions become increasingly complex. As the number of user usage increases, data also increases, server response will become slower and slower, and the user experience becomes terrible. The original system architecture will gradually become chaotic and fragile [2].

Developers are increasingly turning to microservices to address the issue of monolithic development. Containers have recently become popular for deploying microservices across geographically distant clouds. Virtual machines (VMs) have been replaced with containers, which are a lighter version of virtual machines. They're gaining a lot of traction in the industry because they're much lighter than VMs. They're simple to download and put into action [3, 4].

Auto-scaling is essential in the cloud computing environment for lowering cloud operating costs and improving QoS. The goal of this study was to discover load balancing and auto-scaling solutions that could dynamically change resource allocation to cloud services based on incoming workloads [5]. This paper provides reviews on the issues of load balancing and auto-scaling in cloud-based container

microservice, the microservices cloud-based system, container microservice, and the primary objective/issues of load balancing and auto-scaling in microservices. The paper is motivated to investigate current issues surrounding load balancing and auto-scaling techniques in cloud-based container microservices to overcome the problem of server overload, traffic spikes, and service failure and improve the performance of its services for better QoS to users. The remainder of the paper is organized as follows: Section 2 discusses related work, Section 3 discusses the result of the research work, and Section 4 discusses the conclusion and future work.

1.1 Microservice Architecture

Microservice architecture has emerged as a lightweight subset of Service Oriented Architecture (SOA) that firms like Amazon use to avoid monolithic application challenges while reaping some of the SOA design benefits [6]. Microservices have gained significant traction in the business world in recent years. This architecture can be thought of as a refined and simplified version of SOA [7].

According to [9], "a microservice architectural style is a means of designing a single app as a collection of small services, each running in its process and communicating via lightweight mechanisms, most frequently a Hypertext Transfer Protocol (HTTP) resource Application Programming Interface (API)." As a result, the architecture can be viewed as a group of modest services that interact via common communication channels to achieve the goals of users [10]. It communicates with one another to provide users with the required functionality [8].

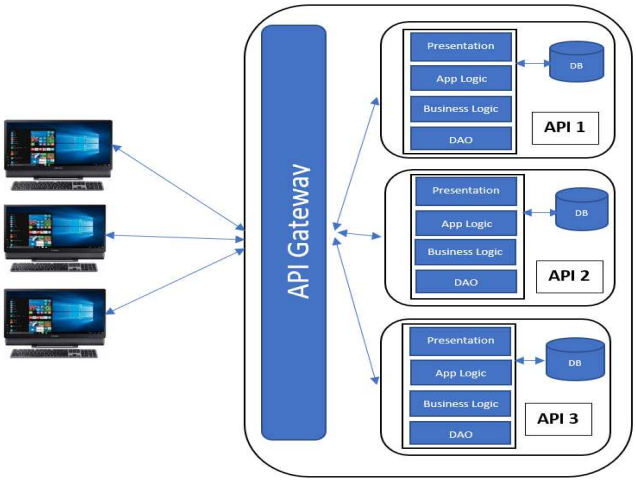

Figure 1. Microservice Architecture [23]

The application is broken down into a series of discrete services that can be developed, deployed, and run separately. In general, each microservice type will install multiple instances, split the workload across multiple servers or data centers, and connect the network to share the load [11]. Microservices fundamentally alter many current cloud system assumptions, presenting both benefits and problems when it comes to improving service quality and usage [12]. As per the principles of Microservice Architecture design, every service needs to be independent of the other.

1.2 Cloud-based Microservices

Cloud computing, when properly built, is a model that allows businesses to create corporate applications that can expand their computer resources on demand. Companies that use Infrastructure as a Service (IaaS) or Platform as a Service (PaaS) solutions for their applications encounter some issues when attempting to use cloud computing features such as auto-scaling, continuous delivery, rapid deployments, high availability, and dynamic monitoring. The majority of companies that attempted to migrate an application to an IaaS/PaaS solution used a monolithic application [6]. Cloud computing is an emerging computing model, according to the authors of [14].

Computing services have become a major subject of Information Technology for academic and industrial study in recent years [16]. Processing microservices in a cloud environment with minimal processing time and cost while efficiently utilizing computing resources is a difficult task [17]. Different challenges in the deployment and continuous integration of microservices were initially evaluated in [18], and an automated method was then presented and designed to address these challenges utilizing various parameters such as response time, throughput, deployment time, and so on.

1.3 Cloud-Based Container Microservices

A container microservices cloud-based application is made up of a group of small, self-contained services that connect with one another using a lightweight mechanism [19]. It is becoming more popular as companies migrate their infrastructures to the cloud [9]. Companies face a variety of challenges when deploying their applications on the cloud due to their unique requirements. The main challenge is the scalability of additional program functionalities based on their needs [18]. As a result of the evolution of cloud container technology, several companies have developed their applications as microservices, breaking down monolithic architecture into discrete services that run independently in containers [20]. When paired with microservices-style architecture, promising container technologies like Docker offer considerable agility in designing and implementing applications. Several essential deployment technologies, including container-based virtualization and container orchestration solutions, have also come on board as a result of the designing of microservice architecture [21].

Containers are an excellent choice for deploying microservices. They can be launched in seconds and swiftly redeployed in the event of failure or migration, and they can scale to meet demand. Containers, as depicted in the diagram below, operate as applications (microservices) within the operating system. The operating system runs on top of the hardware, and each operating system may have multiple containers, each of which runs the application [23].

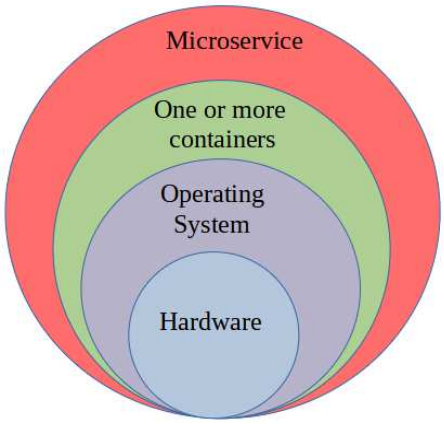

Figure 2. A containers microservice layer architecture [23].

Based on the nature of the research, load balancing and auto-scaling are the primary objective constraints that would be used to obtain the optimal values for workload distribution, response time, and scalability for cloud-based container microservices to avoid server overloaded, traffic spikes, service failure, and to scale up/out the number of available instances based on the loads, for better QoS to users.

1.4 Load Balancing vs Auto-Scaling in Microservices

Load balancing and auto-scaling are the critical features in clouds, responsible for adjusting the number of available resources to meet QoS demand. By balancing the load across multiple resources, load balancing can achieve optimal resource usage, maximum throughput, maximum response time,

and minimize overload [24]. The load balancer starts and stops any virtual machine in the cloud. The load balancing technique, in conjunction with the auto-scaling feature, makes it simple to automatically increase or decrease backend capacity in response to traffic spikes [25]. Consequently, there is a need to optimize load balancing and auto-scaling strategies to configured and address cloud-based container microservices problems. One of the most critical features of cloud computing is load balancing. It's a method that uniformly distributes the dynamic local workload among all nodes in the cloud, avoiding situations when some nodes are overburdened. Others, on the other hand, are idling or doing little work. It aids in achieving high user satisfaction as well as a high resource utilization ratio, hence enhancing overall system performance and resource consumption [26]. Load balancing, by implementing fair-over, aids in the continuation of services when one or more components of a microservice fail. Simultaneously, in cloud computing systems, the auto-scaling technique enables on-demand resource availability based on specified workloads. Leading providers such as Amazon Web Capabilities (AWS) Lambda [13] have launched services to address concerns about microservice challenges (AWS).

1.5 Load Balancing in Microservices

For the development of a business unit, microservices architecture relies heavily on load balancing [27]. Load balancing, also known as a server farm or server pool, is the act of distributing newly received network traffic across servers in real-time or in batches. It's a method for distributing the load across Server VMs in order to maximize resource usage, reduce response time, and avoid burden [16]. When server resources aren't evenly distributed, some get overused while others sit idle, reducing cluster performance. To address this issue, load balancing techniques are employed to ensure that all activities are treated equally, hence boosting processing capacity and service quality at the same time [27]. This process of sharing can be done on a large scale or according to a set of rules. Rules include Round Robin and Least Connections [28]. Load balancing refers to the distribution of workload among microservices and is only relevant in the context of horizontal scalability [29]. Load balancing is one of the most significant and major challenges in the cloud; because the cloud uses so many VMs, manually allocating resources in the cloud is challenging [16]. Load balancing is a key element of microservices architecture for the building of a business unit [27]. Because API calls are used to communicate between the services, numerous instances and loads must be maintained across all accessible instances.

Load balancing alone minimizes the response time and distributes workloads uniformly across servers. Still, during low/high services activity, there is a need to automatically scale up/out the services infrastructure during low/high services activity. This problem could be auto-scaled based on the capacity of the services so that instances can be increased or decreased dynamically based on loads. To allow optimal workload distribution, the technology is integrated into the microservice deployment architecture and set with appropriate load balancing algorithms [30].

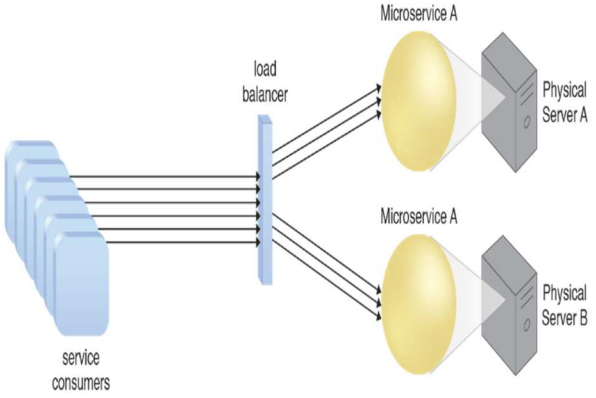

Figure 3. Load balancing in Microservices [30]

As shown above, Physical Server B has a redundant copy of Microservice A. To guarantee that the workloads are distributed evenly, the load balancer intercepts service consumer requests and distributes them to both Physical Server A and B. The primary function of load balancing in microservices is to distribute the workload across the entire servers. It's a technique for shifting a load, task, or process from an overloaded server or data center to an underloaded server or data center [16]. The Load Balancer was able to balance the load by distributing it over two or more Cloud Servers. It keeps the load balanced, ensuring that no resource is overloaded. As a result, scalability, continuous delivery, and operational efficiency increasingly require isolation. It's also generated interest in microservices-based architecture, which let us split up a monolith and build, deploy, execute, scale, and manage services separately [31]. Apart from microservices' agility and scalability, service providers must properly orchestrate hundreds of microservices. In order to do so, a critical issue must be addressed: how to effectively distribute loads through microservices [32]. Load balancing algorithms are designed for this reason. There are several ways to load balancing algorithms in the literature, and these algorithms can be classified into two types. The server-side load balancing approach and the client-side load balancing approach [33], suggested that load balancing brings up a mechanism that can assist improve system throughput and efficiency by maximizing resource use. Its purpose is to disperse the load across available resources to enhance throughput while minimizing response time. It also aids in the improvement of performance and the effective utilization of resources.

Several load balancing algorithms have been enlisted by [34] and compared with their proposed load balancing approach. Job scheduling optimization algorithms, Weights Fixed Load Balancing Algorithms, and Server Dynamic Allocation Weight Algorithms were used, as well as traditional Round Robin and Random Algorithms. Load balancing is classified into two types: server-side load balancing and client-side load balancing:

1.5.1 Server-side load balancing

Server-side load balancing is monolithic. It is valid for both the client and the server. It receives network and application traffic and distributes it among numerous backend servers using a variety of methods. The client requests are distributed to the server through the middle component [36]. A load distributor is placed in front of the servers and distributes traffic to them so that they can perform the primary task equally or following predefined guidelines [28].

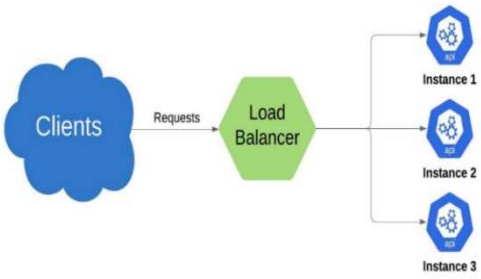

Figure 4. Server-side load balancing across multiple servers [28].

The process of deploying service instances across different servers and then setting a load balancer in front of them is known as server-side load balancing. Most of the time, it's a hardware load balancer. All incoming request traffic is routed through this load balancer, which serves as a middle component. It then uses an algorithm to determine which server a particular request should be sent to [37].

1.5.2 Client-side load balancing

In order to deliver requests, the client maintains a list of server IPs. The client chooses an IP address from the list at random and sends the request to the server [36]. The load balancing is handled by the client in client-side load balancing. In this circumstance, the client API should then be aware of all

hardcoded or service registry instances of server API addresses. Bottlenecks and single points of failure can be avoided using this strategy. There is no need to know anything about the server API other than the registered name if service discovery is used because the server registry mechanism would provide all of that information [28].

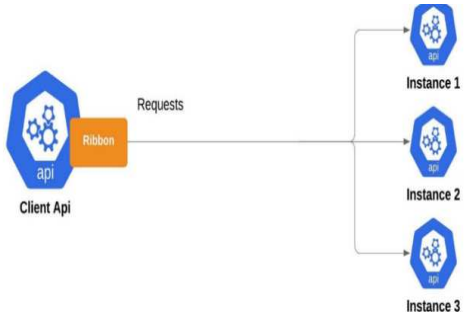

Figure 5. Client-side load balancing across multiple servers [28]

The service's instances are distributed across different servers. Load balancing logic is built into the client; it maintains a list of servers and uses an algorithm to determine which server a given request should be sent to. Software load balancers are another name for client-side load balancers [38].

1.6 Auto-Scaling in Microservice

Auto-scaling is a cloud computing technology provided by Amazon Web Services (AWS) that lets customers deploy or terminate virtual instances based on predefined criteria, health status checks, and scheduling [14]. As defined by [14], It automates the provisioning of system capacity to applications. "In cloud IaaS and PaaS solutions, auto-scaling is a popular feature.

In cloud computing systems, the auto-scaling technique enables on-demand resource availability based on specific workloads. The auto-scaling service provides capacity management strategies to be configured and used to dynamically decide whether to acquire or release resource instances for a certain application [25]. Academics and cloud technology vendors have defined the concept with differing degrees of clarity in a variety of scenarios. In academic words, auto-scaling is a feature of cloud computing infrastructures that enables the dynamic provisioning of virtualized resources [38, 39]. The auto-scaling capacity of services is critical in the cloud computing environment to optimize cloud operating costs and Quality of Service [40]. The addition of Auto Scaling groups to network architecture improves the performance and fault tolerance of the application. A load balancer can be used in architecture to distribute traffic among the instances in auto-scaling groups [25].

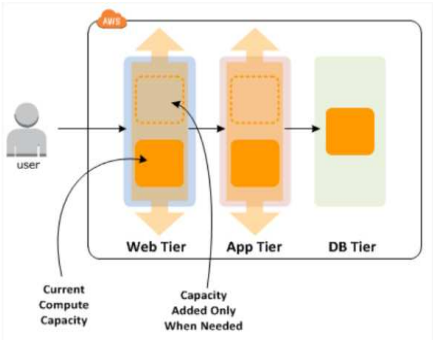

Figure 6. Auto-scaling in Amazon Web Service Microservice [41]

To achieve scalability in microservices, auto-scaling is usually considered the key strategy. When it comes to resource provisioning, scalability, and elasticity, auto-scaling is frequently mentioned.

Under dynamic workloads, resource provisioning allows a system to scale out and in resources [42]. Improved scalability is the result of efficient resource provisioning. When demand increases, scalability allows a system to keep performance by adding or removing hardware resources, which is usually performed by the system administrator [38].

2. Related work

A cloud-based container microservices application is made up of a collection of small, independent services that run in their processes and communicate with one another using a lightweight approach. Over the last few years, a significant amount of research [43, 20, 17, 45] has been focused on the deployment and management of microservice containers.

Several significant companies, including banks, financial institutions, and worldwide retail outlets, are utilizing the MS architecture to incrementally, flexibly, and cost-effectively create their services. Containers have recently gained popularity as a means of deploying MSs across geographically dispersed clouds. The use of microservices design can provide numerous benefits, including loose coupling, rapid disaster recovery, and greater resilience in the case of a network failure. Only a limited amount of the service is affected. The entire system will not be jeopardized as a result of a single failure [45]. Microservices are easy to use, but they have their own set of problems. As the size of the application grows, so does the number of API requests, necessitating the use of a load balancer to handle API calls across the architecture. Load balancing, which distributes the dynamic workload over numerous nodes to ensure that no one resource is overwhelmed or underutilized, is one of the most challenging tasks in Cloud computing. It is a major problem that impacts the server's performance and resource consumption. It tries to optimize resources across computer clusters using network links to increase throughput while minimizing response time. The authors proposed a static and dynamic load balancing technique for automatic resource usage in [16]. This algorithm effectively balances the load by minimizing the response time of running applications in the VM and allocating the load on the servers. In their paper, [48] present a self-organizing optimization strategy for system load balancing based on VM communications records while minimizing response time during VM migration. In addition, it improves QoS by minimizing VM downtime, [32] proposes a chain-oriented load balancing algorithm (COLBA) based entirely on message queues that model the load balancing problems as a non-cooperative game and leverages Nash bargaining to coordinate microservice allocation across chains to reduce response time.

To increase the performance of existing centralized container orchestration systems, a simple swarm-like decentralized load balancing system for microservices running inside OpenVZ containers was presented [31]. In contrast, [24] proposed an improved particle algorithm for resource load balancing optimization in the cloud environment, which takes into account the characteristics of complex networks to develop a corresponding resource-task allocation model for archiving preferable performance cloud-computing environments to improve containers' microservice QoS.

Researchers, on the other hand, have used a different type of auto-scaling algorithms in a cloud context to scale out/down resources based on the current load of Physical Machines (PMs). Without human intervention, auto-scaling is a mechanism for automatically adjusting resources provisioned to applications based on real-time demands. It enables application providers to reduce their cloud resource expenses while still achieving their customers' QoS standards. Designing and implementing an auto-scalar, on the other hand, is a difficult task. Many studies have been conducted on this topic, and many auto-scalars with various properties have been proposed.

A Virtual Hadoop framework was developed to scale out the processing resources necessary for the applications to appease the stated real-time requirements [47]. For heterogeneous computing systems, the authors upgraded their resource inference and allocation approaches. To create an automated system for managing the entire application via scale-out / in with IaaS, a new auto-scaling microservice framework based on predicted workload, as well as an artificial neural network, recurrent neural network, and resource scaling optimization algorithm, were proposed [48].

Meanwhile, the authors of [49, 50] used microservices-based applications to reduce application deployment costs in DCs and enable auto-scaling of cloud applications as their workload varies.

Interestingly, auto-scaling and load balancing of a system may minimize response time, and downtime, and distribute workloads uniformly across servers to avoid service failure, protecting against data loss [50, 17] which is the major challenging task in cloud-based container microservices.

To this extent, it becomes evident that none of the researchers address the issues of auto-scaling and load balancing for cloud-based container microservices at the same time. Therefore, our research has been set to explore the combination of load balancing and auto-scaling in cloud-based container microservices. Integrating load balancing and auto-scaling techniques in cloud-based container microservices at the same time is critical to improving the performance of the container microservices cloud-based system. The related works for the proposed research are summarized and presented in the table below:

Table 1. Related Work Summary

Objective Constraints	Authors	Problems Addressed	Approach's	QoS Metrics	Tools/ Libraries	Limitations
Load Balancing	Ashwin et al., 2015	Designed a Particle Swarm Optimization-based technique to minimize response time and efficiently distribute load across available VMs.	Modified PSO Algorithm algorithm to assign all jobs uniformly Calculate and compare the response time it takes to serve incoming jobs. Used in Virtual Machines.	Response time, Load distribution	Cloudsim	The load balancing algorithm is only main to allocate/distribute the load in the cloud environment, so does not helps to stabilize the swarm algorithm
	Niu et al., 2018.	Focus on balancing strategies to reduce response time and prevent significant networking costs and compete for shared microservices requirements across chains.	Proposed a chain-oriented load balancing algorithm (COLBA) that balances load based on microservices for message queues. Enable hybrid microservice communication by mixing HTTP with a message queue. Use a convex optimization strategy with rounding in a constant gap between the optimum and the target.	Response time Iteration Times	Trace-driven simulations	Their work proposes balancing load across microservices while taking request heterogeneity and inter-chain competition into account but fails to scale in/out the load across microservices.
Auto-Scaling	Guan et al., 2017	Designed a unique application Oriented Docker Container (AODC)-based resource	Develop a scalable algorithm with a diverse set of dynamic applications and	Total energy cost Acceptance ratio Response	Docker Container	The designed framework is only main to minimize the application

	allocation system to reduce application deployment costs in DCs and enable automated scaling as cloud application workloads change.	vast physical resources. Based on application needs and available resources, determines the number and capacity of containers. For VM placement (Best Fit-VM), use a greedy best-fit algorithm, while for VM placement (Best Fit-VM), use a greedy worst fit method (Worst Fit-VM)	time		deployment cost-constrained while satisfying QoS requirements but does not improve the performance of the system for better QoS to the users.
Prachitmutita et al., 2018,	Strive to maintain the service within the Service Level Agreement, the cost of the servers charged by the cloud provider, and automate the scale-out / in.	Use auto-scaling microservices on infrastructure as a service under a Service-Level Agreement. A cost-effective architecture was used to implement the automated system of scaling servers and services with proactive and reactive strategies. Create the system via scale-out / in with Infrastructure as a Service using Artificial Neural Networks, Recurrent Neural Networks, and Resource Scaling Optimization Algorithm (IaaS).	Response Time Workload Distribution Root Mean Square Error	Artificial Neural Network. Recurrent Neural Network.	Their scaling algorithm avoids unnecessary scaling actions but does not improve resource planning and feedback.
Srirama et al., 2020.	Focus on reducing the number of physical machines (PMs) in the cloud data centre by designing a heuristic-based auto-scaling policy, as well as reducing the number of computing resources wasted.	Use an auto-scaling policy to suggest a new container-aware application scheduling method for processing microservices. The technique uses the best-fit lightweight containers to deploy desired apps.	Processing Time Processing Cost. Deployment time. Resource's utilization	Docker Swarm, Google Cluster Tracelog	Their method optimizes computing resources by reducing overall PM resource waste and balancing workloads among them, but it fails to adapt to

					workload changes in container microservices.
Perez et al., 2018	Decoupling complex and monolithic systems into smaller stateless services in order to reduce overall processing time while maintaining QoS.	Use serverless computing in scientific scenarios. Uses a serverless execution approach with massively parallel event-driven file processing.	Throughput Execution Time	Docker Container AWS Lambda	Their method runs a generic application on specific runtime environments defined by Docker Images stored in Docker Hub, however, it fails to increase the performance and resource usage of a running Docker container application.

3. Results and Discussion

The majority of the pertinent issues regarding load balancing and auto-scaling for Cloud-based Container Microservices are to avoid the challenges of server overload, traffic spikes, services, or application failure and to scale up/out the available number of instance servers based on the incoming loads. This is because so many businesses have built their applications using microservices, which break down monolithic software into a collection of small services, each of which runs in its container. The challenges encountered during the microservices communication process included server overload, server failure, and traffic spikes, making it difficult to provide better QoS to users. Until now, the vast majority of cloud-based container microservice researchers have proposed solutions based on one or two constraints (Load balancing and Auto-scaling). Most related research works in cloud-based container microservices that deal with load balancing and auto-scaling fall short of being effective for better QoS to users [16, 17, 46, 48]. Most current cloud-based container microservices address the problems of workload distribution, response time, and scalability using either load balancing or auto-scaling. It was also discovered that the load balancing and auto-scaling objective constraints interact with one another. Auto-scaling performance has a direct impact on or influences load balancing performance. The cloud-based system's key features are load balancing and auto-scaling, which are in charge of adjusting available resources to meet user QoS demands. Load balancing by itself reduces response time and distributes workloads evenly across servers. However, there is still a need for the services infrastructure to be automatically scaled up/down during low/high service activity. This issue could be auto-scaled based on service availability, allowing instances to be dynamically increased or decreased based on loads.

4. Conclusion

This paper looked at the most critical issues related to load balancing and auto-scaling for Cloud-based Container microservices in great detail. Microservice architecture, cloud-based microservices, container microservices, load balancing, and auto-scaling problems in microservices are just a few of the topics addressed. The important issues related to load balancing and auto-scaling in cloud-based container microservices were addressed in order to enhance performance and QoS for users. Our future work will include the implementation of load balancing and auto-scaling hybrid features

in real-time cloud-based container microservices and analyzing docker container features in a microservice cloud-based environment to optimize workload delivery, response time, and scalability.

References

[1] S. Li, "Understanding quality attributes in microservice architecture," Proc. - 2017 24th Asia-Pacific Softw. Eng. Conf. Work. APSECW 2017, vol. 2018-Janua, pp. 9–10, 2018, doi: 10.1109/APSECW.2017.33.

[2] S. Zhuo, "A HIGH AVAILABILITY AND HIGH RELIABILITY MICROSERVICES ARCHITECTURE By," no. August, 2020.

[3] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Migrating to Cloud-Native architectures using microservices: An experience report," Commun. Comput. Inf. Sci., vol. 567, no. July, pp. 201–215, 2016, doi: 10.1007/978-3-319-33313-7_15.

[4] N. Viennot, M. Lécuyer, J. Bell, R. Geambasu, and J. Nieh, "Synapse: A microservices architecture for heterogeneous-database web applications," Proc. 10th Eur. Conf. Comput. Syst. EuroSys 2015, 2015, doi: 10.1145/2741948.2741975.

[5] H. Zhao, H. Lim, M. Hanif, and C. Lee, "Predictive Container Auto-Scaling for Cloud-Native Applications," ICTC 2019 - 10th Int. Conf. ICT Converg. ICT Converg. Lead. Auton. Futur., pp. 1280–1282, 2019, doi: 10.1109/ICTC46691.2019.8939932.

[6] M. Villamizar, O. Garcés, H. Castro, M. Verano, L. Salamanca, and S. Gil, "Evaluating the Monolithic and the Microservice Architecture Pattern to Deploy Web Applications in the Cloud Evaluando el Patrón de Arquitectura Monolítica y de Micro Servicios Para Desplegar Aplicaciones en la Nube," 10th Comput. Colomb. Conf., pp. 583–590, 2015.

[7] T. Erl, J. Fontenla, M. Caeiro, and M. Llamas, "Web Services and Contemporary SOA," Serv. Architecture Concepts, Technol., Des., pp. 25–81, 2005.

[8] D. Bhamare, M. Samaka, A. Erbad, R. Jain, and L. Gupta, "Exploring microservices for enhancing internet QoS," Trans. Emerg. Telecommun. Technol., vol. 29, no. 11, 2018, doi: 10.1002/ett.3445.

[9] A. Sundberg, "A study on load balancing within microservices architecture," 2019.

[10] J. A. Valdivia, X. Limon, and K. Cortes-Verdin, "Quality attributes in patterns related to microservice architecture: a Systematic Literature Review," pp. 181–190, 2020, doi: 10.1109/conisoft.2019.00034.

[11] Z. Ding, S. Wang, and M. Pan, "QoS-Constrained Service Selection for Networked Microservices," IEEE Access, vol. 8, pp. 39285–39299, 2020, doi: 10.1109/ACCESS.2020.2974188.

[12] Y. Gan et al., "An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems," 2019.

[13] M. Villamizar et al., "Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures," Serv. Oriented Comput. Appl., vol. 11, no. 2, pp. 233–247, 2017, doi: 10.1007/s11761-017-0208-y.

[14] A. Hanieh, L. Yan, and H.-L. Abdelwahab, "Analyzing Auto-scaling Issues in Cloud Environments," Proc. 24th Annu. Int. Conf. Comput. Sci. Softw. Eng. IBM Corp., no. March 2015, pp. 75–89, 2014.

[15] Y. Gan and C. Delimitrou, "The Architectural Implications of Cloud Microservices," vol. 17, no. 2, pp. 155–158, 2018.

[16] A. Dave, B. Patel, G. Bhatt, and Y. Vora, "Load balancing in cloud computing using particle swarm optimization on Xen Server," 2017 Nirma Univ. Int. Conf. Eng. NUiCONE 2017, vol. 2018-Janua, pp. 1–6, 2018, doi: 10.1109/NUICONE.2017.8325618.

[17] S. N. Srirama, M. Adhikari, and S. Paul, "Application deployment using containers with auto-scaling for microservices in cloud environment," J. Netw. Comput. Appl., vol. 160, no. August 2019, 2020, doi: 10.1016/j.jnca.2020.102629.

[18] V. Singh and S. K. Peddoju, "Container-based microservice architecture for cloud applications," Proceeding - IEEE Int. Conf. Comput. Commun. Autom. ICCCA 2017, vol. 2017-Janua, pp. 847–852, 2017, doi: 10.1109/CCAA.2017.8229914.

[19] E. Casalicchio and V. Perciballi, "Auto-Scaling of Containers: The Impact of Relative and Absolute Metrics," Proc. - 2017 IEEE 2nd Int. Work. Found. Appl. Self* Syst. FAS*W 2017, pp. 207–214, 2017, doi: 10.1109/FAS-W.2017.149.

[20] X. Wan, X. Guan, T. Wang, G. Bai, and B. Y. Choi, "Application deployment using Microservice and Docker containers: Framework and optimization," J. Netw. Comput. Appl., vol. 119, no. December 2017, pp. 97–109, 2018, doi: 10.1016/j.jnca.2018.07.003.

[21] M. V. L. N. Venugopal, "Containerized Microservices architecture," Int. J. Eng. Comput. Sci., vol. 6, no. 11, 2017, doi: 10.18535/ijecs/v6i11.20.

[22] NetApp, "What are Microservices?" 2020. [Online]. Available: https://www.netapp.com/knowledge-center/what-are-microservices/.

[23] S. Sharma, Mastering microservices with Java 9: build domain-driven microservice-based applications with Spring, Spring Cloud, and Angular. 2017.

[24] K. Pan and J. Chen, "Load balancing in cloud computing environment based on an improved particle swarm optimization," Proc. IEEE Int. Conf. Softw. Eng. Serv. Sci. ICSESS, vol. 2015-Novem, pp. 595–598, 2015, doi: 10.1109/ICSESS.2015.7339128.

[25] A. R and J. Agarkhed, "Evaluation of Auto Scaling and Load Balancing Features in Cloud," Int. J. Comput. Appl., vol. 117, no. 6, pp. 30–33, 2015, doi: 10.5120/20561-2949.

[26] A. M. Alakeel, "A Guide to Dynamic Load Balancing in Distributed Computer Systems," vol. 10, no. 6, pp. 153–160, 2010.

[27] P. B. Agavane, "Improve Load Balancing Performance and Efficiency Using Equally Spread Current Execution Algorithm working with response time clustering in Microservices."

[28] M. Yakut, "Load Balancing In Microservices," 2020. [Online]. Available: https://mesutyakut.medium.com/load-balancing-in-microservices-474ad84b847d#:~:text=Load balancing is the process, Round Robin%2C Least Connections etc.

[29] M. D. Cojocaru, A. Oprescu, and A. Uta, "Attributes assessing the quality of microservices automatically decomposed from monolithic applications," Proc. - 2019 18th Int. Symp. Parallel Distrib. Comput. ISPDC 2019, no. June, pp. 84–93, 2019, doi: 10.1109/ISPDC.2019.00021.

[30] N. Erl, "Microservice and Containerization Patterns," 2019. [Online]. Available: https://patterns.arcitura.com/microservice-patterns/design_patterns/workload_distribution.

[31] M. Rusek, D. Rzegorz, and A. Orłowski, "A decentralized system for load balancing of containerized microservices in the cloud," Int. Conf. Syst. Sci., vol. 539, no. November, pp. 142–152, 2016, doi: 10.1007/978-3-319-48944-5.

[32] Y. Niu, F. Liu, and Z. Li, "Load Balancing Across Microservices," Proc. - IEEE INFOCOM, vol. 2018-April, pp. 198–206, 2018, doi: 10.1109/INFOCOM.2018.8486300.

[33] T. Le Duc, R. G. Leiva, P. Casari, and P. O. Östberg, "Machine learning methods for reliable resource provisioning in edge-cloud computing: A survey," ACM Comput. Surv., vol. 52, no. 5, 2019, doi: 10.1145/3341145.

[34] C. Yi, X. Zhang, and W. Cao, "Dynamic weight based load balancing for microservice cluster," ACM Int. Conf. Proceeding Ser., 2018, doi: 10.1145/3207677.3277955.

[35] C. Heggem, "Container Load Balancing," 2019.

[36] J. Tpoint, "Client-Side Load Balancing with Ribbon," 2018. [Online]. Available: https://www.javatpoint.com/client-side-load-balancing-with-ribbon.

[37] D. Kumar, "Load balancing Spring Boot Microservices using Netflix's Ribbon," 2020. [Online]. Available: https://www.studytonight.com/post/load-balancing-spring-boot-microservices-using-netflixs-ribbon.

[38] N. Ma, A. Maghari, N. Sarkissian, and W. Clark Lambert, "Parakeratosis: What it is and what it is not," Skinmed, vol. 8, no. 6, pp. 361–362, 2010.

[39] N. M. Calcavecchia, B. A. Caprarescu, E. Di Nitto, D. J. Dubois, and D. Petcu, "DEPAS: A decentralized probabilistic algorithm for auto-scaling," Computing, vol. 94, no. 8–10, pp. 701–730, 2012, doi: 10.1007/s00607-012-0198-8.

[40] H. Zhao, H. Lim, M. Hanif, and C. Lee, "Predictive Container Auto-Scaling for Cloud-Native Applications," ICTC 2019 - 10th Int. Conf. ICT Converg. ICT Converg. Lead. Auton. Futur., pp. 1280–1282, 2019, doi: 10.1109/ICTC46691.2019.8939932.

[41] A. W. Service, "Amazon EC2 Auto Scaling benefits," 2021. [Online]. Available: https://docs.aws.amazon.com/autoscaling/ec2/userguide/auto-scaling-benefits.html.

[42] W. Iqbal, M. N. Dailey, D. Carrera, and P. Janecek, "Adaptive resource provisioning for read intensive multi-tier applications in the cloud," Futur. Gener. Comput. Syst., vol. 27, no. 6, pp. 871–879, 2011, doi: 10.1016/j.future.2010.10.016.

[43] M. Lin, J. Xi, and W. Bai, "Ant Colony Algorithm for Multi-Objective Optimization of Container-Based Microservice Scheduling in Cloud," IEEE Access, vol. 7, pp. 83088–83100, 2019, doi: 10.1109/ACCESS.2019.2924414.

[44] M. Vijayalakshmi, D. Yakobu, D. Veeraiah, and N. G. Rao, "Automatic Healing of Services in Cloud Computing Environment," no. 978, pp. 740–745, 2016.

[45] M. Bravetti, S. Giallorenzo, J. Mauro, I. Talevi, and G. Zavattaro, Optimal and automated deployment for microservices, vol. 11424 LNCS. Springer International Publishing, 2019.

[46] S. Aslanzadeh and Z. Chaczko, "Load balancing optimization in cloud computing: Applying Endocrine-particale swarm optimization," IEEE Int. Conf. Electro Inf. Technol., vol. 2015-June, pp. 165–169, 2015, doi: 10.1109/EIT.2015.7293424.

[47] Y.-W. Chen, S.-H. Hung, C.-H. Tu, and C. W. Yeh, "Virtual Hadoop," pp. 201–206, 2016, doi: 10.1145/2987386.2987408.

[48] I. Prachitmutita, W. Aittinonmongkol, N. Pojjanasuksakul, and M. Supattatham, "Auto - scaling microservices on IaaS under SLA with cost - effective Framework," pp. 583–588, 2018.

[49] X. Guan, X. Wan, B. Choi, S. Song, and J. Zhu, "Application Oriented Dynamic Resource Allocation for Data Centers Using Docker Containers," vol. 1, no. c, pp. 1–4, 2016, doi: 10.1109/LCOMM.2016.2644658.

[50] B. Stevant, J. L. Pazat, and A. Blanc, "Optimizing the Performance of a Microservice-Based Application Deployed on User-Provided Devices," Proc. - 17th Int. Symp. Parallel Distrib. Comput. ISPDC 2018, pp. 133–140, 2018, doi: 10.1109/ISPDC2018.2018.00027.

[51] V. Podolskiy, A. Jindal, and M. Gerndt, "Multilayered Autoscaling Performance Evaluation: Can Virtual Machines and Containers Co-Scale?" Int. J. Appl. Math. Comput. Sci., vol. 29, no. 2, pp. 227–244, 2019, doi: 10.2478/amcs-2019-0017.