International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Container Scheduling: A Taxonomy, Open Issues and Future Directions for Scheduling of Containerized Microservices in Cloud Environments

Anil Prajapati[1] *, Dr.Manish M Patel[2]

Submitted: 03/05/2024 **Revised**: 16/06/2024 **Accepted**: 23/06/2024

Abstract: Containerization offers lightweight virtualization and modern applications are adopting containers because of their scalability, portability, and flexible deployment, particularly with microservices. In contrast to monolithic architecture, microservice design is a new paradigm that delivers granular and loosely coupled services. With containerization, it is feasible to create a scalable application architecture made up of several microservices. For their production environment, companies like Netflix, Google, Microsoft, and others have been using cloud environments based on containers. We have presented an extensive review of containerization, and microservice architecture in this paper. The study also discusses container orchestration, classification of container scheduling techniques, optimization objectives for scheduling of container-based microservices, and a comparison of different container orchestration platforms. Container scheduling strategies are widely developed using heuristic and meta-heuristic techniques. Designing a resource efficient scheduling technique for containerized microservice is a key challenge due to various factors such as dynamic workload and diverse resource requirements. Machine learning has great potential and machine learning-based techniques have been employed for the optimized scheduling of containerized microservices in recent years. It is possible to implement an intelligent container scheduling approach to forecast performance. Machine learning-based multi-objective container scheduling solutions can be proposed to obtain effective resource usage of cloud environment. We mention areas that still need investigation in the field of container scheduling for containerized microservices.

Keywords: *Cloud Computing, Containerization, Container scheduling, Microservice, Machine learning*

1. Introduction

These days, cloud computing is becoming more and more popular. Many apps are moving from private infrastructures to cloud-based environments to take advantage of its features, which include scalability, flexibility, agility, and cost-effectiveness. When physical resources of the cloud, such as memory, CPU, storage, and network resources, are allocated to different cloud-based applications and computational resources are used effectively, the cost of application deployment and operation is decreased. One of the main concerns for cloud service providers is how to deploy applications on the cloud at a reasonable cost. Resource management, scheduling, and allocation have all been extensively researched for the deployment of a variety of dynamic applications to address these problems. To streamline processes and minimize costs associated with developing and deploying cloud-based services, we have investigated the multiple architectural paradigms [1].

Resource virtualization—or sharing of resources amongst applications—is made possible by virtualization technologies. Applications on the same system can therefore operate in many execution environments. To provide the necessary resource isolation, hypervisor-based virtualization incurs expense by performing various duplicate functionalities to abstract the system resources. CPU overhead, memory overhead, network overhead, and Disk overhead are the overheads that occur in virtual machines [2].

Containers are quickly gaining popularity and replacing virtual machines in recent years due to their several promising characteristics, like shared host operating system, quick launch time, scalability, portability, and quick deployment. With containers, applications can boost productivity and portability by wrapping all required dependencies as code, runtime, code, system libraries, and system tools, into a single box. This creates a run-time environment that is platform-independent [3]. Software applications that offer virtualization at the operating system level are called containers. Container-based computing platforms can be installed, terminated, replicated, recovered from, and relocated in milliseconds owing to their architecture [4].

The paradigm of cloud computing is being revolutionized by containerization [3, 4, 5], and in response to the increasing demand for these services, numerous cloud service providers have begun to offer services based on containers. A few instances are Amazon Elastic Container Service, Google Container Engine, and Azure Container Services. Modern applications must communicate in real-

[1]*Research Scholar (209999913024), Gujarat Technological University, Gujarat, India*
ORCID ID : 0009-0002-6314-4987
[2] *Professor, Department of Information Technology, Sankalchand Patel College of Engineering, Visnagar, Gujarat, India*
** Corresponding Author Email: anil.apit18@gmail.com, it43manish@gmail.com*

time, get regular updates, and deal with fluctuating traffic patterns.

A new paradigm for developing applications for cloud computing is microservices. It separates an application into a collection of fine-grained, loosely connected services. It is a deployable and independently scalable application component. The traditional "monolithic" approach for developing applications, where every application is a stand-alone, independent component, is contrasted with the microservice-based method. For example, in client-server applications, the server is a single unit that executes logic, gets and/or changes data, and responds to HTTP requests. Therefore, the challenge with these monolithic architectures is that every time the logic of the application changes, an updated version of the whole code base must be deployed. Because each microservice only handles a single subtask or service, there is less overhead in communication and less computing power needed.

Numerous major corporations, such as Netflix, Amazon, IBM, Uber, Alibaba, and others, have moved their apps to this development framework because of microservices [6]. These days, cloud-based applications are frequently developed using the microservice architecture, which is growing in popularity in application design and development [7]. Microservices can be deployed and encapsulated in a cloud environment using containers, a lightweight virtualization technology. Certain scheduling techniques for containers have been suggested as a result of the advancements in container technology and the growing popularity of microservice architecture. Nonetheless, there are still a few significant issues with scheduling of containerized microservice in cloud environments that need to be resolved.

The growing popularity of microservice architecture and container architecture for cloud computing offers the chance to enhance the elasticity and scalability of application development. Compared to virtual machines, containers offer more portability, faster deployment, and reduced utilization of resources. As a result, it's a great tool for scheduling, encapsulating, and deploying microservices. At present, Google Kubernetes, Apache Mesos, and Docker Swarm are the most popular container cluster management tools [8, 9, 10].

To the diverse workloads and resources available on the cloud, container scheduling has become essential for the cost-effective operation of microservices on the cloud platform. Researchers have proposed various container scheduling approaches to accomplish different performance objectives, including response time, energy consumption, resource utilization, and availability, load balancing, and cost. There was some investigation into the practical deployment of microservice-based applications using containers. In a cloud system based on containers, we investigated container scheduling for microservice deployment. One researcher has developed approaches to figuring out how much resources to allocate to each microservice and how to scale effectively when workload varies.

In addition to meeting the cloud cluster's load requirements, an efficient container resource allocation strategy guarantees the cluster's reliability and efficiency. Additional study is necessary to investigate the performance of microservice applications, cloud cluster reliability, and network transmission overhead between microservices, all of which can be enhanced [7].

The remainder of the paper has been arranged in this manner. A brief history of containerization, container architecture, and microservices architecture is given in the second section. The third section presents a comparison of several container orchestration technologies, classifies container scheduling strategies, and explains the optimization objectives used to schedule microservices based on containers. The scheduling of containerized microservices is discussed in the fourth section along with the relevant container scheduling work for microservices. In the fifth section, the significance of machine learning is discussed, along with an in-depth investigation of the techniques based on machine learning that are employed in container scheduling. Additionally, it shows how well machine learning techniques perform in different optimization objectives and the scheduling container technology. The sixth section presents research and the future. The future and research directions in the field of containerized microservices scheduling are presented in the sixth section. The seventh section brings survey to its conclusion.

2. Background

Significant background information on container architecture, microservices architecture, and container engines is presented in this section. We have also showcased the associated research that compares the performance of virtual machines and containers made with Docker.

2.1 Containerization

High service downtime occurs while an application is being upgraded on a virtual machine. On the other hand, due to their faster startup times and improved performance, containers are promising lightweight virtualization technology for cloud-based services, particularly when it comes to microservices, edge computing, smart cars, and the Internet of Things [10]. The architectural approaches for virtual machines and container technologies are distinct, as seen in Figure 1. Considering the container architecture, a container engine represents the top layer of the stack's resource management system. It is situated directly on top of the host operating system, whereas the comparable layer

in the virtual machine architecture is called the hypervisor. The fact that a guest operating system is not needed for running containers is a fascinating characteristic of the container-based design. Compared to virtual machines, it offers portability, efficiency, and less overhead because the hypervisor layer is removed [11].

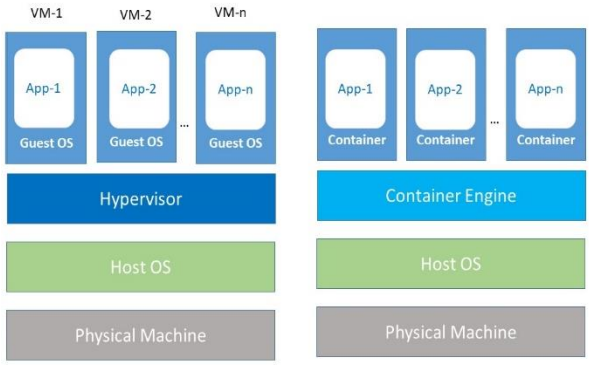

Fig 1. Hosting Approach: Virtual Machines and Containers Furthermore, the authors in [11][12] found that in terms of total throughput, services deployed using containers perform far better than virtual machines. Virtual machine deployment of real-time applications was also found to have a significant performance overhead, as shown by the CPU utilization and memory consumption. Moreover, Docker containers are allegedly launched on top of Amazon EC2 virtual machines by Amazon Cloud, which goes against the standard procedure for application deployment shown in Figure 1. The container deployment strategy used by Amazon Cloud Platform is depicted in Figure 2.

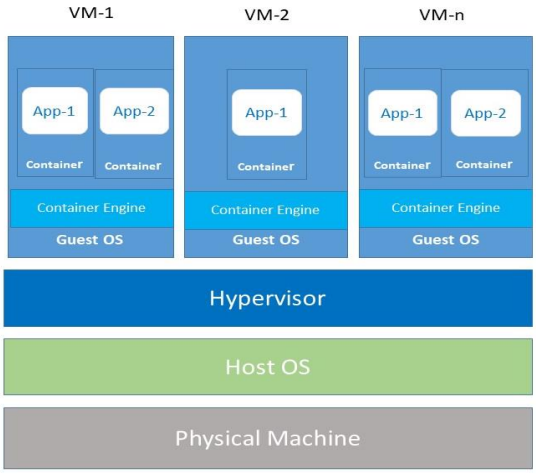

Fig 2. Container hosting approach in Public Cloud

Developers use container technologies extensively for deploying various microservices. Despite containers' enormous popularity in cloud computing, no comprehensive study is present that addresses scheduling approaches for containers from the perspective of containerized microservices [12,13]. When it comes to application deployment, developers who build and deploy application containers must make the most of the infrastructure's performance. The demand for various types of container scheduling algorithms for the deployment of containerized microservices is driven by this resistance to cloud providers.

2.2. Container Engine Architecture

Software developers build and distribute container images, which are files that include the data needed to run a containerized application. Containerization tools are used by developers to create read-only, non-modifiable images in containers. On top of any infrastructure, users can create containers with the use of containerization tools. The most widely used container solution available right now is called Docker, and here it is used to illustrate the concept of containerization.

Docker is an open-source platform that makes it simpler to build, deploy, and run services that use containers. Users can deploy applications more quickly by separating the applications from the infrastructure through the use of Docker containers. Docker Swarm, Docker Compose, Docker Images, Docker Daemon, and Docker Engine are major components of Docker. We may manage this infrastructure in the same manner that we would an application. The Docker container engine's architecture is depicted in Figure 3.

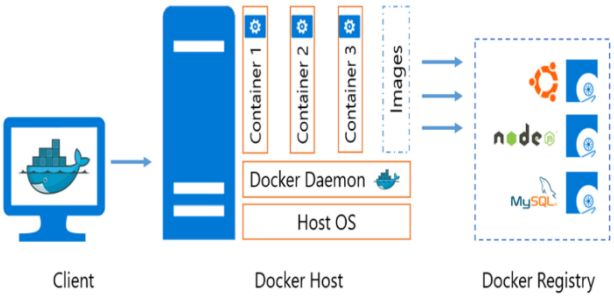

Fig 3. Architecture of Container Engine [14]

The component of Docker that creates and manages Docker containers is the Docker engine. The container is an instance of a Docker image that is currently running. A Docker image is a read-only template which contains the instructions on how to build a Docker container. The Docker daemon serves as a process that is used to manage and control the containers. The primary service that Docker users use to interact with Docker containers is the Docker client. Every Docker image is stored in a Docker registry. The Docker CLI (Command Line Interface) enables us to start, stop, or remove containers [14].

2.3. Microservice Architecture

A new paradigm for developing cloud applications is microservices. An application is divided into several fine-grained, loosely coupled services using microservice architecture. A microservice is a deployable, independently scaled application component. The traditional "monolithic" approach to application development, in which each

application is a single autonomous unit, is compared with the microservice-based approach. Many organizations, including Uber, Netflix, and Amazon, are switching from traditional monolithic architecture to microservice architecture because of their high scalability requirements and massive user counts. A comparison of the microservice and monolithic architectures is shown in Figure 4. Monolithic applications are scaled by replicating them on multiple servers, while microservices are scaled by distributing across cloud servers and replicating them as and when required.

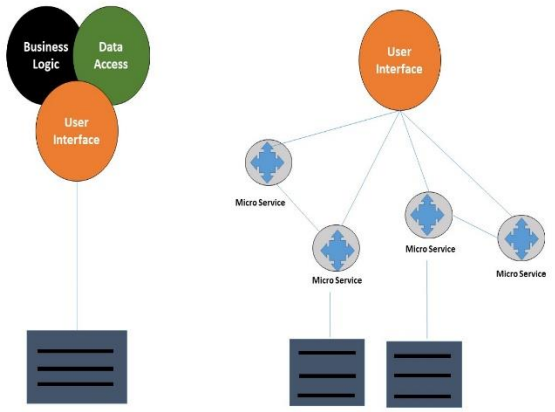

Fig 4. Monolithic and Microservice architecture

Compared to monolithic architectural approaches, microservices are independent components of an application that are small and easy to work with. The flexibility to adapt and be implemented on their own is referred to as the services' autonomy. Microservices can independently create, deploy, test, and operate; they are typically run by separate developer teams and structured around business logic. Moreover, microservices may utilize many technologies and be developed using different programming languages. Microservices are deployable and scalable using container-based virtualization [16]. The deployment of a microservice instance in a cloud container is depicted in Figure 5.

A newly created microservice should be able to register itself by saving its runtime configurations and updating the deployment information so that it can communicate with other microservices in the same application. Microservices will have their development framework and each will be developed independently. To create microservice binaries, the deployment server retrieves the source code from the repository, compiles, and tests the microservice code. The deployment server generates a container image with these binaries, which is then kept in the repository of containers. These container images are deployed to the deployment server after they are created. The developed microservice would be deployed in one or more containers.

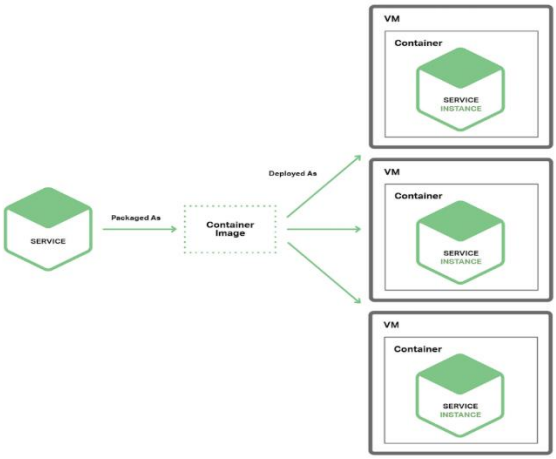

Fig 5. Deployment of Microservice in container [16]

A newly created microservice should be able to register itself by saving its runtime configurations and updating the deployment information so that it can communicate with other microservices in the same application. Microservices will have their development framework and each will be developed independently. To create microservice binaries, the deployment server retrieves the source code from the repository, compiles, and tests the microservice code. The deployment server generates a container image with these binaries, which is then kept in the repository of containers. These container images are deployed to the deployment server after they are created. The developed microservice would be deployed in one or more containers.

3. Container Scheduling for Microservices

Emerging requirements for container scheduling are imposed by the container-based infrastructure to guarantee the performance of deployed microservice-based containerized applications. The underlying physical machine or cloud cluster must provide the resources that a container requests, which are frequently a combination of several resources like CPU, memory, network, etc. Since there are frequent updates and data transfers, containers inside a distributed application frequently have a strong affinity for one another. Thus, affinity needs to be considered when planning container schedules. During the execution of the application, the scheduling algorithms are routinely called upon, especially when scaling out or recovering from failure, which typically involves crucial time constraints. As a result, the scheduling overhead ought to be minimal [17,4].

3.1 Container Scheduling Classification

Numerous research papers on container scheduling have been reviewed by us. It was discovered that the four kinds of container scheduling approaches under investigation each had different performance and quality characteristics. The majority of the suggested approaches fall into Mathematical modeling-based techniques, Heuristics based techniques, Meta-heuristics-based technique, and Machine learning-

based techniques.

Mathematical modeling offers the optimization of a linear function using a set of linear constraints, the technique is called integer linear programming. From the perspective of container scheduling, it does not offer optimum solutions promptly. Many researchers have proposed linear programming models for container deployment considering various optimization objectives to obtain optimum container scheduling and showed efficient results as well. It has been observed that mathematical modeling is suitable to the problem in small size.

After a detailed survey it is found that compared to mathematical modeling, the heuristic techniques are preferable in container scheduling. Most of the approaches investigated in recent years use some kind of heuristic to find the optimum container scheduling. In the majority of situations, heuristic algorithms are simple to use and offer effective scheduling in a small amount of time. Most of these approaches combine current container scheduling techniques and apply them in Kubernetes and Docker Swarm. [3] Summarized the heuristic techniques employed in container scheduling. A researcher has showed a scheme for efficient resource utilization. Container scheduling framework for the Docker environment also proposed best-fit scheduling that offers dynamic monitoring and showed a 23% energy-efficient solution as compared to Docker Swarm. A multi-objective approach that combines all three (Spread,Binpack,Random) of Docker Swarm to provide efficient container scheduling is proposed with considerations of memory utilization, network delay, CPU utilization, and interaction between cluster nodes and containers. The approach showed efficient performance. After the survey, it is observed that heuristic techniques generally offer quick optimization, furthermore, they can be integrated with other optimization approaches for enhanced container scheduling.

Meta-heuristic approaches are promising in getting the optimum solution in a variety of sectors nowadays. Ideally, these approaches are adaptive to the environment and they can be based on Genetic algorithms, Ant Colony Optimization, and Particle Swarm Optimization. [3] Summarizes meta-heuristic approaches used in container scheduling. Kaewsaki(2017) presented ACO(Ant colony optimization) based approach presented to improve resource usage and load balancing. The strategy was experimented on Docker Swarm and showed 15% enhanced performance. From the perspective of the microservice, Lin(2019) presented a multi-objective ACO scheduling technique that is used to optimize network transmission overhead, failure rate, and resource utilization of containerized microservices. Results showed enhanced resource usage and reliability. Genetic algorithms and Particle Swarm Optimization-based container scheduling techniques have also been proposed in recent years. Li(2018) proposed a PSO-based technique to improve load balancing and resource usage. Results showed enhanced performance by 20% compared to the spread technique of Docker Swarm.

After an extended literature survey, it is observed that researchers have proposed hybrid scheduling techniques for container scheduling in which ACO and PSO are combined to offer efficient scheduling mechanisms. Furthermore, it is found that meta-heuristic approaches are more suitable for multi-objective optimization. The field of machine learning has been an emerging field of study that has shown tremendous potential in numerous applications across various fields and it holds great potential for container scheduling. Particularly when it comes to scheduling the containerized microservice, machine learning approaches are thoroughly investigated. We have investigated the machine learning algorithms employed for the scheduling of the containerized microservices and discussed in Section-5 of the paper.

3.2. Performance Objectives for Container Scheduling

According to particular user requirements, the scheduling decisions typically need to accomplish several performance objectives. When deploying applications in containers, striking a balance between competing optimization objectives is still a major challenge. The objectives mentioned below are thought to be the most typical ones for scheduling cloud containers [3,4].

Energy efficiency: The huge amount of electricity used by cloud environments has become a major concern in the area of cloud computing due to the constantly expanding scale of cloud data centers. When deploying containers on nodes that are working, the amount of power used is referred to as energy consumption. The objective seeks to reduce the cluster's overall power consumption.

Cost: The total cost of running an application is determined by the price of several services, including computing, storage, and communication. The amount of time needed to run an application on the cloud cluster's available cores is referred to as the computation cost. The more time an application runs on a processor, the more costly it becomes.

Availability: It specifies how long a user has access to an application. Availability is a key component of cloud computing, where users should always be able to access a service or application. The objective is to ensure that the schedule that is generated offers some kind of fault tolerance for the different services that the application provides, requiring redundancy in the number of containers that are deployed in the cloud.

Resource Utilization: It is the effectiveness with which a worker node uses its memory, core, and network bandwidth. To achieve this goal, a cluster node's energy efficiency and

cost-effectiveness increase with its resource utilization. For energy and cost efficiency, infrastructure-level resource utilization measures like CPU and memory usage are typically regarded as vital application performance indicators.

Load Balancing: The task of dividing up the workload among available resources equally so that no one node becomes overloaded is known as load balancing. Throughput, cost, and response time are all impacted by this objective. This measure is

critical, particularly for container applications that take advantage of microservice architectures.

Scalability: The scalability of containers to maintain the required service even in the face of growing system demand for the application or service is another important parameter.

Latency: The duration needed for an application to run from start to finish is known as latency. Reducing latency is always the objective of a good scheduler. Cost and throughput are greatly impacted by this objective.

Security: An ability to protect data and services from malicious attacks or software bugs by encryption and access control methods. Orchestration tools must be able to offer essential security.

Network Bandwidth: The number of bits transferred in one second on the network is referred as network bandwidth. It would assist in assessing network delays and handling congestion that occurred during the communication of containers.

Carbon Emission: It is the volume of Carbon Dioxide that is emitted into the atmosphere as a result of using electric power that is specifically created when fossil fuels are burned.

SLA assurance: The majority of containerized applications are set up with precise performance parameters, including throughput, launch time, completion time, and response times. The majority of these constraints are outlined in SLA contracts, and breaking them may result in a penalty.

3.3. Container Orchestration

The containers are widely employed by organizations to deploy modern applications such as IoT and Big data in cloud data centres. As a result, container orchestration has emerged. Container orchestration facilitates defining, selecting, deploying, monitoring, and dynamically control the configuration of containerized services in the cloud. The container orchestrator performs the scheduling of containers, selecting the optimal node and ensuring the container is running in the desired state. Container orchestration system enables organizations to streamline application development by deploying the same application without having to rebuild it in a cloud computing environment.

The deployment of containerized microservices, cost, and performance are all significantly impacted by the intricate container scheduling problem. Applications are not just moved between servers and turned on and off with container orchestration. When a multi-container application is deployed, users can specify how to manage the cloud's containers through the use of container orchestration. Container orchestration describes the way multiple containers are managed as a single unit in addition to how they are initially deployed [18]. The section discusses taxonomy of container orchestration and analyses the orchestration systems in the context of the scheduling of the containerized applications. [19] Analyses the existing container orchestration systems concerning container technology used, the application model it follows, placement constraints, and resource granularity. Orchestration systems for the system objectives such as scalability, availability, throughput, and resource utilization are also investigated. The result of the investigation is summarized in Table 1.

Container orchestration tools available today are as follows.

Docker Swarm

The built-in scheduling and clustering mechanism for Docker containers is called Docker Swarm which orchestrates the application or service running in a Docker container. Applications are treated as services; they might be decomposed in microservices and may be deployed in one or more containers. The Swarm manager is used for controlling the entire life cycle of applications containerized in Docker containers. It supports three scheduling techniques by default: First, there is the Spread approach, which chooses the newly deployed containers for operation on the least loaded hosts; second, there is the BinPack strategy, which chooses the most loaded host with sufficient resources to run the containers. In addition to this, the third is the Random method, which chooses the node at random.

Kubernetes

An open-source container orchestrator called Kubernetes was first created by Google that facilitate the automated deployment and scaling of containerized applications. The pod is a group of containers and the basic building block of the Kubernetes orchestration framework. Kubernetes scheduler provides schedules for each pod based on available resources, filtered by user-specified requirements, and ranked according to application affinities that are individually defined. The Kubernetes cluster contains Master and Worker nodes, where the master node is the fundamental component and the worker nodes perform services to run pods.

Mesos

Two steps are available for scheduling with Apache Mesos. It allocates the physical node's resources to each application in order

during the first stage, which is referred to as the framework. After accepting the offer, the framework can use its built-in framework scheduler to plan its tasks on the obtained resources. Mesos does not offer service discovery. Hence, Kubernetes or Swarm is integrated to address this lack. Orchestration frameworks Aurora and Marathon depend on Mesos to manage the cluster resources of the container cluster.

Aurora is developed by Twitter, a scheduler that runs on top of Mesos and enables long-running services to be deployed on the container.

Table 1 . Taxonomy of Container orchestration tools

Orchestrator	Swarm	Kubernetes	Apache Mesos	Aurora	Marathon	YARN	Omega	Fuxi
Organization	Docker	Google	UC Berkeley	Twitter	Mesosphere	Apache	Google	Alibaba
Open Source	✓	✓	✓	✓	✓	✓	-	-
Technology of container	Docker	Docker	Mesos containers, Dokcer	Apache Mesos, Docker	Mesos containers, Docker	Linux cgroups-based, Docker	N/S	Linux cgroups-based
Pre-emption	-	-	-	✓	-	-	✓	-
Rescheduling	-	-	-	✓	-	-	✓	-
Scheduling constraints	Label and affinity-based	Label and affinity-based	N/A	Value and limit-based	Value and limit-based	Value and limit-based	N/S	Value based
Scalability	✓	✓	✓	✓	✓	✓	✓	✓
High Availability	✓	-	-	-	-	-	✓	✓
High Utilization	✓	-	-	-	-	-	✓	✓
High Throughput	✓	-	-	-	-	-	✓	✓
Resource granularity	Fine-grained	Fine-grained	Fine-grained	Fine-grained	Fine-grained	coarse-grained	Fine-grained	Fine-grained
Application workload	Long running tasks	All	All	Long running tasks	Long running tasks	Batch tasks	All	Batch tasks

Marathon is a framework for Mesos that is developed for the orchestration of long-running services. It offers fault tolerance and high availability; it ensures that deployed applications will continue running even in the situation of node failures.

YARN is designed for orchestration of Hadoop tasks, it also supports frameworks like Giraph, Spark, and Storm. Each application framework running on top of YARN coordinates its execution flows and optimizations as it sees fit.

Omega is Google's next-generation cluster management

system

that offers a platform that enables specialized and customized schedulers to be developed, providing users with great flexibility.

Fuxi is a resource management and scheduling system that supports Alibaba's proprietary data platform. It is the resource management module on their Aspara system, which is responsible for managing the physical resources of Linux clusters within a datacenter [20].

4. Container Scheduling for Microservices

In addition to fulfilling user service needs, efficient container scheduling minimizes running overhead and guarantees the performance of the application and the cloud environment underneath it [9,20]. We have reviewed the open issues within the scheduling of containerized microservices and presented herein this paper, different scheduling methods employed for the containerized microservices are investigated and discussed in this section.

4.1. Issues in Scheduling of Microservices

The number of independently running microservices must fulfil the necessary function, handling thousands of requests for access and ensuring high availability, scalability, and tolerance over network failures [21]. The services that comprise the application must communicate with one another constantly to use the microservices architecture. In order to complete the task, each service must simultaneously use the interface for interaction between its services. The communication between microservices is extremely difficult because an enormous number of microservices instances run on a large number of containers, and the placement of microservices instances is also constantly changing High complexity and dynamicity pose many challenges for handling microservices.For managing microservices networked together such as 2-tier or 3-tier web applications and Internet of Things (IoT) based applications, we have yet to discover a conventional large-scale, optimized scheduling platform. Following a thorough survey, we discovered the following issues with microservices scheduling [22].

Configuration and Management

Typically, a cloud application integrates several interdependent microservices, such as a web server, database server, and load balancer, to provide a variety of functionality. These microservices also have interference and dataflow dependencies. Dealing with different microservice configurations and cloud data center resources operated by diverse performance needs presents some issues.

Application Composition

The workloads associated with various microservices are interdependent, meaning that modifications to one microservice dataflow and execution will have an impact on others. All things considered, the application composition must address the entire life cycle, including deploy, patch, monitor, reconfigure, and shutdown, all of which are influenced by each of the microservices' objectives for performance.

Monitoring of microservice

A clear and updated understanding of objectives for performance across microservices and data center resources is necessary for optimal application performance. Measures of performance include, for instance, throughput and latency for storage resources, utilization and throughput for CPU resources, and query response time for microservices based on NoSQL and SQL databases. Thus, there is still work that needs to be done to define and construct performance objectives across microservices in a coherent manner that provides a comprehensive perspective of data and control flows.

Elastic Scheduling and runtime Adaptation

Microservice scheduling is challenging due to several runtime uncertainties. Microservice workload behavior, including request arrival rate, type, and processing time, as well as Input/output system behavior and the number of users connecting to different microservice types and mixtures, is difficult to estimate. Developing workload models specifically for microservices presents a significant challenge: accurately determining and fitting statistical functions for observed distributions, including those pertaining to request arrival patterns, CPU and memory utilization patterns, I/O system behaviors, request processing time distributions, and network usage patterns. Manually solving the issue wouldn't be possible.

Microservice Interference

If several microservices are deployed in a single container, there may be resource conflict and interference because various microservices may have similar resource requirements. To minimize interference and make the best use of available resources, it's essential to understand how microservices running in tandem fluctuate in terms of performance.

Service Discovery

During the running time of microservices, discovering the proper microservice needs the orchestration which needs to be aligned with the essential quality of the service. An essential quality parameter is the latency of the discovered and triggered microservice.

Performance isolation

In a containerized system, a single container can handle several heterogeneous microservices that offer different application-specific functionalities. Unexpected conflict and interference are possible in this scenario. Certain microservices have more storage requirements than others, some have more computational requirements than others (such as transactional query processing by database servers), and some have more communication requirements than storage requirements.

Communication and integration

These issues have arisen because of the distributed architecture of the microservices. It is challenging to ensure that the communication system is dependable and that the protocol to be used for communication and integration can manage challenging processes, even when microservices communicate with a more lightweight protocol. Reliability and durability are the two most crucial requirements for both problems; if these are not satisfied, the system's capacity to function properly and reliably will be impacted, perhaps leading to cascading failures.

4.2 Approaches for scheduling of containerised microservices

The deployment of containerized microservices poses some challenges, most of which are related to planning and deployment configuration in the container. The heterogeneous requirements of microservices demand an optimal container scheduling mechanism to meet these requirements. The microservices need to be monitored and automatically adjusted resources according to varying loads for the predictability and availability of the application. These requirements bring a lot of challenges for the scheduling. The chain of services must be arranged and managed, and the available resources must be scheduled for efficient utilization. The system's dependability and availability are directly impacted by ineffective scheduling. Additionally, container scheduling for microservices frequently needs to be reviewed for effective analysis and, further improvement. In this section, we have presented the studies and work related to the scheduling of microservices.

[MIAO LIN, 9] presented a model to enhance the microservice-based container resource allocation approach. This model includes reducing overhead of network transmission between microservices, load, and enhancing the cluster services' reliability. The researcher used the average number of microservice request failures, the maximum value of the resource utilization rate, and the network transmission overhead between microservices to achieve this. This was implemented using an Ant colony optimization (ACO_MCMS) and it was presented to address the issue of containerized microservices in the cloud platform. It is compared with the current container scheduling policies for an average number of microservice request failures and computing resource utilization. The result shows enhanced performance.

For microservices in clouds, [Sheng Wang, 23] suggested and developed a task scheduling algorithm of microservices as a cost optimization. Based on the workload statistics, the proposed strategy selects the container and adapts to the streaming load based on the statistics of the workload. The strategy was tested in simulation-based environment and results show reduced the cost in comparison with existing scheduling mechanisms.

A microservices scheduling method called LWFF presented by [H M Fard, 24] in which the scheduling problem is characterized as an advanced version of the knapsack problem and solved using a multi-objective optimization technique, Findings indicate that, in comparison to the state-of-the-art, the suggested mechanism is extremely scalable and concurrently boosts CPU and memory usage, which improves throughput. The suggested container scheduling technique outperforms the Spread and Binpack scheduling strategies in terms of throughput.

A novel Microservice-based container framework proposed by [XUEHUA ZHAO, 25] for running mobility and delay-sensitive applications at the lowest possible cost. The findings demonstrate that the suggested approaches can decrease costs, improve utilization of resources, and minimize service delay.

Using the TOPSIS algorithm, [Tarek, 26] presented a novel scheduling approach that combines the principles of Bin-Packing and the Spread techniques. The suggested approach aimed to identify, from a group of nodes that comprise a cloud infrastructure, the node with the best balance between three factors: (i) the total number of containers on the node; (ii) the total number of CPUs available; and (iii) the total amount of RAM accessible. The proposed scheduling mechanism is experimented with in Docker Swarm and compared with Spread, Binpack, and Random strategies. The result shows improved performance under various scenarios.

The first Ant Colony Optimization technique for container scheduling was introduced by [Kaewkasi, 3] to improve the utilization of resources through appropriate load balancing. Docker Swarm was used to integrate and test the proposed strategy. The outcomes demonstrated a 15% improvement in performance as compared to the Swarm Greedy technique. Nevertheless, the strategy only took into account a small number of optimization objectives, like load balancing and utilization of resources.

[Mehmet,27] focuses on the challenges of scheduling and auto-scaling of containerized microservices. The authors claimed that existing scheduling mechanisms underperform with streaming workloads and in architecture consisting of virtual machines and containers. Thus, to overcome these challenges, researchers proposed an elastic scheduling mechanism that handles task scheduling and auto-scaling, which is based on a variable-sized bin packing problem (VSBPP). The proposed mechanism improves the success ratio and cost.

C.T Joseph (2021) proposed a microservice rescheduling framework to address performance degradation and response time challenges. The author says, to define the quality of any microservice, response time is an important measure. The author claimed that the effect of configuration parameters of containerized microservices has not been handled well by the researchers.

5. Machine Learning in scheduling of microservice based containers

Automation of the container orchestration process for complex and heterogeneous workloads under cloud computing systems is very uncertain today. In the studies, we found that machine learning has already been employed in virtual machine orchestration. Current container scheduling mechanisms are typically designed with heuristic scheduling policies that do not take into account the variety of workload situations and QoS needs of an application. The majority of these techniques are used for small-scale systems that are configured offline based on specific workload scenarios. Such scheduling techniques cannot handle highly dynamic workloads where applications need to be scaled at runtime according to specific behavior patterns. When the system scales up, heuristic techniques may perform significantly worse. When resource provisioning, dependency structures between components of containerized systems are not taken into consideration. Within an application, several microservice units contain internal relationships and depend on one another. Microservice architecture suffers as a consequence of increased resource requirements and communication expenses.

The majority of the attention of current container orchestration approaches is on evaluating infrastructure-level metrics; application-level metrics and particular QoS (Quality of Service) needs are not given enough weight. The increasing complexity of managing applications on cloud platforms has compelled cloud service providers to use machine-learning approaches to optimize their container

Table 2. Machine learning-based models used for scheduling containerized microservices

Problem in scheduling of containerized applications	Machine Learning based solution	Advantage/ Limitations
How can the interdependencies between micro services be analyzed?	BO, GP, CNN, and LSTM	Increased accuracy of predictions/ Ignorance of updates regarding dependencies
How can task dependencies in containerized applications be analyzed?	SVM	Increased Accuracy/ The time overhead and computational expenses are explained implicitly
How can resource scheduling for containers be made energy-efficient?	MDP, Q-learning, and SARSA	Cost saving, Minimizing task execution time/ Limited scalability
When migrating micro services to cloud containers, how can the communication delay be minimized?	DNN, Q-learning	Minimizing task execution time/Limited scalability
How can workloads be characterized by forecasting and modeling the behavior of requests and the pattern of resource consumption?	ARIMA, Bi-LSTM, LSTM, K-means++,GRU, TSNNR	Optimized Resource utilization/ Scheduling delays

orchestration strategies [28]. Therefore, machine learning techniques could be used to model and forecast metrics at the infrastructure or application level, including workload characteristics, system resource consumption, and application performance. However, machine learning algorithms, which offer better accuracy and less time overhead in large-scale systems, could generate resource management decisions directly in place of

heuristic techniques. We investigated the research work

[3] Reported the work related to machine learning-based container scheduling. Resource optimization is very challenging in diverse workload scenarios. To boot resource usage, Nanda(2018) proposed a reinforcement learning-based approach for container consolidation. The proposed technique showed enhanced results compared SJF and random scheduling techniques. Lv(2019) presented a random forest regression model for the prediction of the need of containers. The model accurately predicted the future resource needs. Liu(2020) proposed a model to

Table 3. Container scheduling techniques and optimization objectives have been used by the researchers

Objectives						ML approach used	Container Technology used	Limitations
Energy	**Availability**	**Utilization**	**Load Balancing**	**Cost**	**Network**			
		✓	✓			ACO	Docker	Require parameter tuning
	✓	✓	✓		✓	First multi-objective GA	Kubernetes	No energy reduction
✓	✓	✓	✓			GA	Docker	Slow
		✓	✓			Random Forest	Kubernetes	Only predict the resources and lack of testing in real time
✓		✓	✓			K-means	Docker	Few optimization objectives are used
✓						NN based model	Docker	Performance is not considered
✓		✓				Linear Regression	Cloudsim	Cost is not taken in consideration

where

machine learning algorithms are utilized in container scheduling.

In past years, machine learning has gained immense popularity and is widely employed in many research areas. According to their optimization objectives, we found that the most often used machine learning models in the container scheduling area are Regression, Classification and Decision-making. The overall performance of the application and resource efficiency are directly impacted by the quality of scheduling selections. Following our survey, it was found that a few researchers used machine learning-based models in the last few years, particularly for scheduling of containerized applications.

predict the physical machine-level resource usage are considered to improve energy efficiency and Service Level Agreement of datacenters. The proposed model was tested in ContainerCloudSim and showed reduced energy consumption.

[28] Shows an evaluation of machine learning-based container scheduling technologies. In 2016, K-nearest neighbor was employed to estimate the resource consumption of containerized applications. However, the application model only considered the time series pattern of infrastructure-level resource metrics. In Shah(2017) long short-term memory (LSTM) model was applied to microservices dependency analysis, based on neural networks and fitted for classification, processing, and forecasting. The approach assessed the time series pattern of resource metrics as

well as the internal relationships among microservices. Tang(2018) presented a model for performance analysis of important resource metrics which was carried out using the bidirectional LSTM (Bi-LSTM) model. This model was employed to forecast application throughput and workload arrival rates. Comparing their training module to ARIMA and LSTM models, it has shown a notable gain in accuracy in terms of time series prediction. Podolskiy(2019) used Lasso regression (LASSO) to forecast the service level indicators (SLI) for the predictions of application response time and throughput. The ANN model was trained using the reinforcement learning technique, which provided the best scheduling mechanism with the least amount of performance interference.

In 2020, many Reinforcements Learning based approaches were suggested. Qiu (2020), employed a model for microservice dependency analysis and identification of the essential components most likely to encounter resource shortages and performance degradations. This model was implemented using SVM. Zhang (2021), proposed a model combination of Convolutional neural networks (CNNs) and boosted trees (BT) to analyze the dependability and performance of microservices, with promising results.

Table 2 displays findings following our thorough analysis of the machine learning-based container scheduling model. It demonstrates how different machine learning-based scheduling techniques have been designed for containerized microservices. A summary of meta-heuristics and machine learning-based container scheduling techniques summarized with the optimization objectives have been used by the researcher and shown in Table 3. Analysis has shown that approaches based on machine learning and meta-heuristics are better suited for multi-objective optimization.

To summarize, the majority of research studies use reinforcement learning models for scheduling decision-making to improve utilization of resources and reduce task completion time.

6. Discussion and Future Research Directions

Applications like the Internet of Things (IoT), microservices, smart infrastructure, and containers are growing at a rapid pace and are widely used in cloud computing environments. Containerization is a lightweight virtualization solution that facilitates microservice application encapsulation, scheduling, and deployment. For this reason, a lot of cloud service providers have integrated container technologies into their infrastructure to automate applications. Container scheduling is proposed as a key research challenge to address the automation of deployment, scheduling, auto-scaling, and networking of containerized applications. Applications that are highly diverse and dynamic, significantly increase the complexity of container scheduling.

Conventional scheduling and optimization techniques are insufficient for scheduling containerized applications, according to an analysis of different container scheduling methods. All of the important objectives for performance cannot be achieved by a single algorithm. Therefore, there are still several issues that need to be resolved as important areas for future research in the field of scheduling the deployment of containerized applications in cloud environments. With the rise of the fog computing paradigm, processing and storage are being pushed closer to the user for real-time applications resulting in reduced energy consumption and quicker response time. The essential technology used to provide such services are containerization, which allows for dynamic workloads and applications. To use this new field of microservice applications, more resource-aware and energy-efficient container scheduling strategies are also needed.

In the last few years, machine learning-based techniques have also been used for containerized application scheduling improvement. Nonetheless, we have shown how different strategies compare in terms of performance and optimization objectives. Only machine learning would be useful for dynamic and varied workload-specific scheduling strategies. It would also be able to predict future consumption of resources and workload, which would enable intelligent scheduling decisions. Microservices have become widely used in many domains, and most cloud-native applications these days might have a large number of microservices.

The number of user requests, their interdependence, interference, and scalability all affect how an application behaves, which increases overhead and lowers performance. Therefore, an intelligent and efficient container scheduling that takes into consideration every microservice architecture issue would be needed. Multi-dimensional metrics for performance would be predicted using machine learning methods. The quality of scheduling and resource provisioning decisions made in response to shifting user demands in complex systems with diverse resource utilization and dynamic workloads may be further enhanced by these insights.

7. Conclusion

In this paper, we showed that the most appropriate method for deploying containerized microservices in a cloud context is containerization, which offers lightweight virtualization. The growing popularity of cloud-based containers highlights how crucial microservices management and orchestration are to the entire microservices architecture. Based on the optimization strategy used for containerized application scheduling, we have presented the classification of scheduling approaches and discussed several container

orchestration tools.

Docker and Kubernetes would provide the best performance in container orchestration and management from the perspective of application developers. Our research indicates that estimating the behaviour of microservice workload concerning processing time, request arrival rate, type, and number of users connecting to various microservice types is challenging. For predicting performance objectives, the microservice-based workload model can be employed. In our studies, we found that Machine Learning has a great potential for implementing intelligent scheduling mechanisms. Our survey results show that machine learning techniques are the most effective for the scheduling of containerized microservices by making the most use of the underlying cloud resources. Based on these findings, we have suggested potential further studies in this paper.

Furthermore, to offer efficient container scheduling, multi-objective container scheduling techniques can be employed to manage resources efficiently. Hybrid machine learning-based approaches can also be employed. Our survey would help researchers identify the key characteristics of Machine learning-based approaches and choose the most suitable method for efficient container scheduling for the microservices.

Conflicts of interest

The authors declare no conflicts of interest.

References

[1] XiliWana , XinjieGuana,∗ , TianjingWanga , GuangweiBaia , Baek-Yong Choi, "Application deployment using Microservice and Docker containers: Framework and optimization", Journal of Network and Computer Applications 119 (2018) 97–109

[2] M. SRIRAGHAVENDRA1 & PRATEEK JAIN2," VIRTUAL MACHINE VS CONTAINER: AN APPLICATION PERFOMANCE" ,International Journal of General Engineering and Technology (IJGET) ,ISSN(P): 2278-9928; ISSN(E): 2278-9936 Vol. 6, Issue4, Jun – Jul 2017; 29-40

[3] Imtiaz Ahmad, Mohammad Gh. AlFailakawi , AsayelAlMutawa, LatifaAlsalman," Container scheduling techniques: A Survey and assessment", Journal of King Saud University – Computer and Information Sciences

[4] Y. Hu, H. Zhou, C.d. Laat et al.," Concurrent container scheduling on heterogeneous clusters with multi-resource constraints" ,Future Generation Computer Systems 102 (2020) 562–573

[5] MIAO LIN 1 , JIANQING XI1 , WEIHUA BAI 2 , AND JIAYIN WU ,"Ant Colony Algorithm for Multi-Objective Optimization of Container-Based Microservice Scheduling in Cloud"IEEE Access VOLUME 7, 2019

[6] Qian Qu, Ronghua Xu, SeyedYahyaNikouei, Yu Chen "An Experimental Study on Microservices based Edge Computing Platforms", IEEE Xplore 2017

[7] VindeepSingh,Sateesh K Peddoju,"Container-based Microservice Architecture for Cloud Applications", IEEE International Conference on Computing, Communication and Automation (ICCCA2017), ISBN: 978-1-5090-6471-7/17/$31.00 ©2017 IEEE

[8] OMOGBAI OLEGHE, "Container Placement and Migration in Edge Computing: Concept and Scheduling Models",IEEE Access 2021

[9] Guisheng Fan1,2 , Liang Chen1 , Huiqun Yu1 , and Wei Qi1,"Multi-Objective Optimization of Container-Based Microservice Scheduling in Edge Computin", Computer Science and Information Systems 2020

[10] Ouafa Bentaleb1,2,3 · Adam S. Z. Belloum3 · Abderrazak Sebaa4,5 Aouaouche El-Maouhab,"Containerization technologies: taxonomies, applications and challenges", The Journal of Supercomputing 2021

[11] Salah, M. Jamal Zemerly, Chan YeobYeun, Mahmoud Al-Qutayri, Yousof Al-Hammadi," Performance Comparison between Container-based and VM-based Services", 978-1-5090-3672-1/17/$31.00 ©2017 IEEE

[12] Sheng Wang, Zhijun Ding, Senior Member, IEEE, ChangjunJiang,"Elastic Scheduling for Microservice Applications in Clouds", IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS:1045-92(19)-2020

[13] JunmiBo An, Donggang Cao, Xiangqun Chen," Comparing Container-based Microservices and Workspace as a Service:Which One to Choose?", 2018 IEEE Symposium on Service-Oriented System Engineering", 0-7695-6394-5/18/$31.00 ©2018 IEEE

[14] Docker Containers and VMs.https://www.weave.works/blog/a-practical-guide-to-choosing-between-docker-containers-and-vms.Accessed January 2022

[15] Microservice vs Monolithic. https://www.suse.com/c/rancher_blog/microservices-vs-monolithic-architectures/. Accessed June 2023

[16] Chris Richardson of Eventuate,Building Microservices, https://www.nginx.com/blog/building-microservices-using-an-api-gateway/. Accessed: December 2023

[17] Tarek Menouer,PatriceDarmon ,"New Scheduling Strategy based on Multi-CriteriaDecision Algorithm" 2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2377-5750/19/$31.00 ©2019 IEEE

[18] Rajkumar Buyya1, Maria A. Rodriguez1, Adel NadjaranToosi, Jaeman Park," Cost-Efficient Orchestration of Containers in Clouds: A Vision, Architectural Elements, and Future Directions", Cloud Computing and Distributed Systems (CLOUDS) Lab

[19] Claus Pahl∗ and Antonio Brogi‡ and Jacopo Soldani‡ and PooyanJamshidi† "Cloud Container Technologies: a State-of-the-Art Review", IEEE Transactions on Cloud computing 2017

[20] Maria A. Rodriguez RajkumarBuyya," Container-based cluster orchestration systems: A taxonomy and future directions", © 2018 John Wiley & Sons, Ltd

[21] Abdul Saboor 1,* ,MohdFadzil Hassan 2, Containerized Microservices Orchestration and Provisioning in Cloud Computing: A Conceptual Framework and Future Perspectives", Appl. Sci. 2022, 12, 5793

[22] Maria Fazio,AntonioCelest,RajivRanjan,LydiaChen,Chang Liu," Open Issues in Scheduling Microservices in the Cloud" ,IEEE cloud computing , 2325-6095/16/$33.00 © 2016 IEEE September/October 2016

[23] Kaewkasi , KornrathakChuenmuneewong," Improvement of Container Scheduling for Docker using Ant Colony Optimization" ,IEEE cloud computing , 978-1-4673-9077/4/17/ © 2017 IEEE October 2017

[24] Hamid MohammadiFard ,"Dynamic Multi-objective Scheduling of Microservices in the Cloud" 2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)

[25] XUEHUA ZHAO," Microservice Based Computational Offloading Framework and Cost Efficient Task Scheduling Algorithm in Heterogeneous Fog Cloud Network",IEEE Access VOLUME 8, 2020

[26] Tarek Menouer Patrice Darmon," New Scheduling Strategy based on Multi-Criteria Decision Algorithm" , 2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing ,2377-5750/19/$31.00 ©2019 IEEE DOI 10.1109/PDP.2019.00022

[27] Mehmet Söylemez1 ,BedirTekinerdogan 2,* and AyçaKolukısaTarhan 1," Challenges and Solution Directions of Microservice Architectures: A Systematic Literature Review", Appl. Sci. 2022, 12, 5507. https://doi.org/10.3390/app12115507

[28] Z. Zhong, "Machine Learning-based Orchestration of Containers: A Taxonomy and Future Directions" ACM Computing Surveys, Vol. 54, No. 10s, Article 217. Publication date: September 2022

[29] NandanJha,Saurabh Garg, Prem Prakash Jayaraman,RajkumarBuyya,RajivRanjanCelest, " A Holistic Evaluation of Docker Containers for Interfering Microservices" , 2018 IEEE International Conference on Services Computing,2474-2473/18/$31.00 ©2018 IEEEDOI 10.1109/SCC.2018.00012

[30] MingchangWei,Yang Yu,," A Container Scheduling Strategy Based on Machine Learning in Microservice Architecture", 2019 IEEE International Conference on Services Computing (SCC), 2474-2473/19/$31.00 ©2019 IEEEDOI 10.1109/SCC.2019.00023

[31] Ye Wu ,HaopengChen,"ABP Scheduler :Speeding up service spread in Docker swarm" , 2017 IEEE International journal on Parallel and distributed Processing with Applications", 0-7695-6329-5/17 2017 IEEE

[32] Junming Ma, Bo An, Donggang Cao, Xiangqun Chen," Comparing Container-based Microservices and Workspace as a Service:Which One to Choose?", 2018 IEEE Symposium on Service-Oriented System Engineering", 0-7695-6394-5/18/$31.00 ©2018 IEEE

[33] Christophe C´ erin, Tarek Menouer, WalidSaad and Wiem Ben AbdallahChris," A New Docker Swarm Scheduling Strategy", 2017 IEEE 7th International Symposium on Cloud and Service Computing, 978-0-7695-6328-2/17 $31.00 © 2017 IEEE

[34] Ayman Noor∗†, DevkiNandanJha∗, Karan Mitra‡, Prem Prakash Jayaraman§, Arthur Souza¶, Rajiv Ranjan∗andSchahramDustdar, "A framework for monitoring microservice-oriented cloud applications in heterogeneous virtualization environments", 2019 IEEE 12th International Conference on Cloud Computing (CLOUD), 2159-6190/19/$31.00 ©2019 IEEE

[35] Michel Gokan Khan, Member, IEEE, Javid Taheri, Senior Member, IEEE,"PerfSim: A Performance Simulator for Cloud Native Microservice Chains:, arXiv:2103.08983v2 [2017]

[36] Guo ,"A Container Scheduling Strategy Based on Neighborhood Division in Micro Service" 978-1-5386-3416-5/18/$31.00 ©2018 IEEE

[37] Raj Gunasekaran,"Characterizing Bottlenecks in

Scheduling Microservices on Serverless Platforms", 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)

[38] Rohan Mahapatra,Exploring Efficient ML-based Scheduler for Microservices in Heterogeneous Clusters

[39] 1st Guozhi Li," Microservices: architecture, container, and challenges", 2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)

[40] Qilong Li, "Multi-Algorithm Collaboration Scheduling Strategy for Docker Container", 2017 International Conference on Computer Systems, Electronics and Control (ICCSEC)

[41] Christina Terese Joseph K. Chandrasekaran," Straddling the crevasse: A review of microservice software architecture foundations and recent advancements", DOI: 10.1002/spe.2729

[42] Yanjun Shi *, YijiaGuo, LinglingLv and Keshuai Zhang," An Efficient Resource Scheduling Strategy for V2X Microservice Deployment in Edge Servers", Future Internet 2020, 12, 172; doi:10.3390/fi12100172

[43] 1Pratham Jangra, 2Anuttam Anand, 3Dr. Amit Kumar Tyagi, "Survey on Container Systems and Their Efficient Orchestration Algorithms", International Journal of Creative Research Thoughts (IJCRT)

[44] Ebook for .Net microservice for containerized applications, Microsoft book

[45] Yiren Li1,2, Tieke Li1 , Pei Shen2 , Liang Hao2," Sim-DRS: a similarity-based dynamic resource scheduling algorithm for microservice-based web systems", PeerJComput. Sci. 7:e824 DOI 10.7717/peerj-cs.824

[46] Vishal Rao1 , Vishnu Singh1 "Scheduling Microservice Containers on Large Core Machines through Placement and Coalescing"