Senjab *et al. Journal of Cloud Computing* (2023) 12:87
https://doi.org/10.1186/s13677-023-00471-1

Journal of Cloud Computing:
Advances, Systems and Applications

REVIEW **Open Access**

A survey of Kubernetes scheduling algorithms

Khaldoun Senjab[1], Sohail Abbas[1*], Naveed Ahmed[1] and Atta ur Rehman Khan[2]

Abstract

As cloud services expand, the need to improve the performance of data center infrastructure becomes more important. High-performance computing, advanced networking solutions, and resource optimization strategies can help data centers maintain the speed and efficiency necessary to provide high-quality cloud services. Running containerized applications is one such optimization strategy, offering benefits such as improved portability, enhanced security, better resource utilization, faster deployment and scaling, and improved integration and interoperability. These benefits can help organizations improve their application deployment and management, enabling them to respond more quickly and effectively to dynamic business needs. Kubernetes is a container orchestration system designed to automate the deployment, scaling, and management of containerized applications. One of its key features is the ability to schedule the deployment and execution of containers across a cluster of nodes using a scheduling algorithm. This algorithm determines the best placement of containers on the available nodes in the cluster. In this paper, we provide a comprehensive review of various scheduling algorithms in the context of Kubernetes. We characterize and group them into four sub-categories: generic scheduling, multi-objective optimization-based scheduling, AI-focused scheduling, and autoscaling enabled scheduling, and identify gaps and issues that require further research.

Keywords Cloud services, Data center infrastructure, Resource optimization, Containerized applications, Kubernetes, Container orchestration, Scheduling algorithm

Introduction

Kubernetes is an open-source platform for automating the deployment, scaling, and management of containerized applications. It allows developers to focus on building and deploying their applications without worrying about the underlying infrastructure. Kubernetes uses a declarative approach to managing applications, where users specify desired application states, and the system maintains them. It also provides robust tools for monitoring and managing applications, including self-healing mechanisms for automatic failure detection and recovery. Overall, Kubernetes offers a powerful and flexible solution for managing containerized applications in production environments.

Kubernetes is well-suited for microservice-based web applications, where each component can be run in its own container. Containers are lightweight and can be easily created and destroyed, providing faster and more efficient resource utilization than virtual machines, as shown in Fig. 1. Kubernetes automates the deployment, scaling, and management of containers across a cluster of machines, making resource utilization more efficient and flexible. This simplifies the process of building and maintaining complex applications.

Microservice-based architecture involves dividing an application into small, independent modules called microservices, Fig. 2. Each microservice is responsible for a specific aspect of the application, and they communicate through a message bus. This architecture offers several

*Correspondence:
Sohail Abbas
sabbas@sharjah.ac.ae
[1] Department of Computer Science, College of Computing and Informatics, University of Sharjah, Sharjah, UAE
[2] College of Engineering and Information Technology, Ajman University, Ajman, UAE

© The Author(s) 2023, corrected publication 2023. **Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

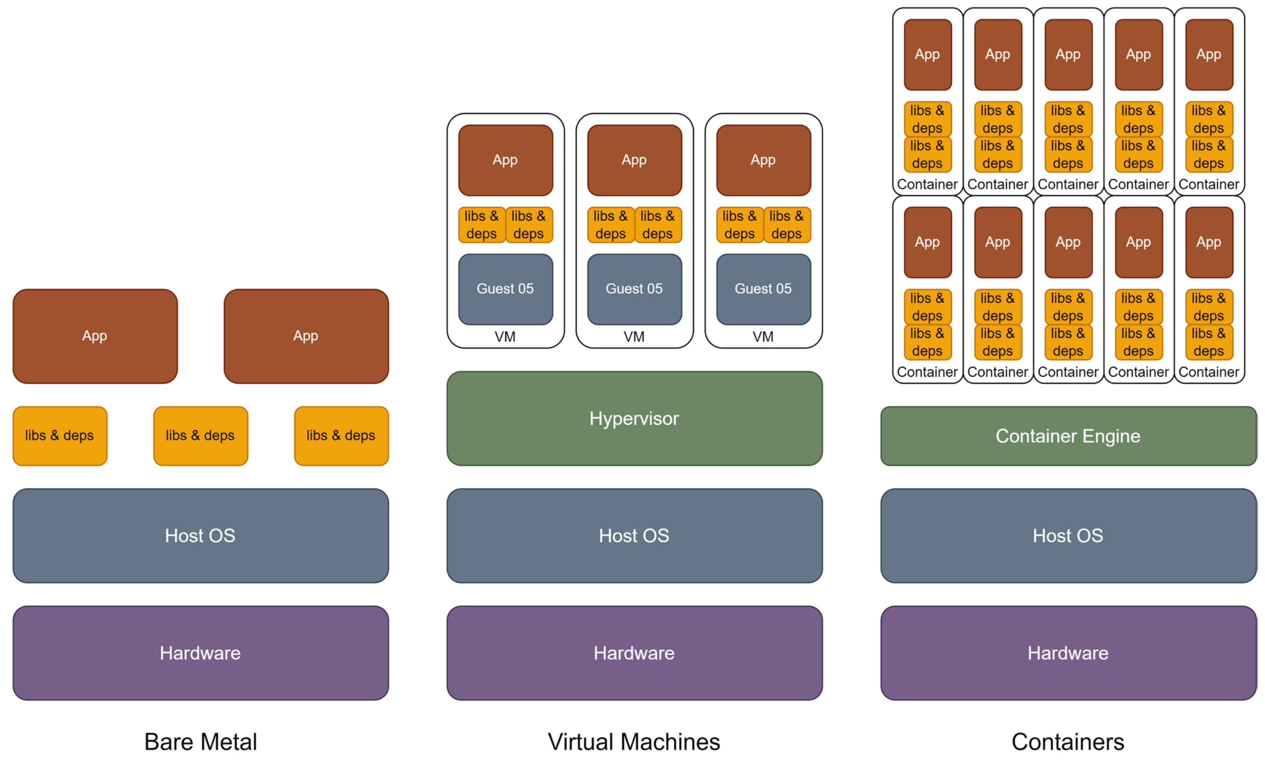

Fig. 1 Comparison between different types of applications deployments

benefits, such as the ability to automate deployment, scaling, and management. Because each microservice is independent and can be managed and updated separately, it is easier to make changes without affecting the entire system. Additionally, microservices can be written in different languages and can run on different servers, providing greater flexibility in the development process.

Kubernetes can quickly adapt to various types of demand intensities. For example, if a web application has few visitors at a given time, it can be scaled down to a few pods using minimal resources to reduce costs. However, if the application becomes extremely popular and receives a large number of visitors simultaneously, it can be scaled up to be serviced by a large number of pods, making it capable of handling almost any level of demand.

Kubernetes have been employed by many organizations in a diverse area of underlying applications and have gained the trust of being the best option for the management and deployment of containerized applications. In terms of recent applications, Kubernetes are proving to be an invaluable resource for IT infrastructure as they provide a sustainable path towards serverless computing that will result in easing up challenges in IT administration [1]. Serverless computing will provide end-to-end security enhancements but will also result in new infrastructure and security challenges as discussed in [1].

As the computing paradigm moves towards edge and fog computing, Kubernetes is proving to be a versatile solution that provides seamless network management between cloud and edge nodes [2–4]. Kubernetes face multiple challenges when deployed in an IoT environment. These challenges range from optimizing network traffic distribution [2], optimizing flow routing policies [3], and edge device's computational resources distribution [4].

As can be seen from the diverse range of applications, and challenges associated with Kubernetes, it is imperative to study proposed algorithms in the related area to identify the state-of-the-art and future research directions. Numerous studies have focused on the development of new algorithms for Kubernetes. The main motivation for this survey is to provide a comprehensive overview of the state-of-the-art in the field of Kubernetes scheduling algorithms. By reviewing the existing literature and identifying the key theories, methods, and findings from previous studies, we aim to provide a critical evaluation of the strengths and limitations of existing approaches. We also hope to identify gaps and open questions in the existing literature, and to offer suggestions for future research directions. Overall, our goal is

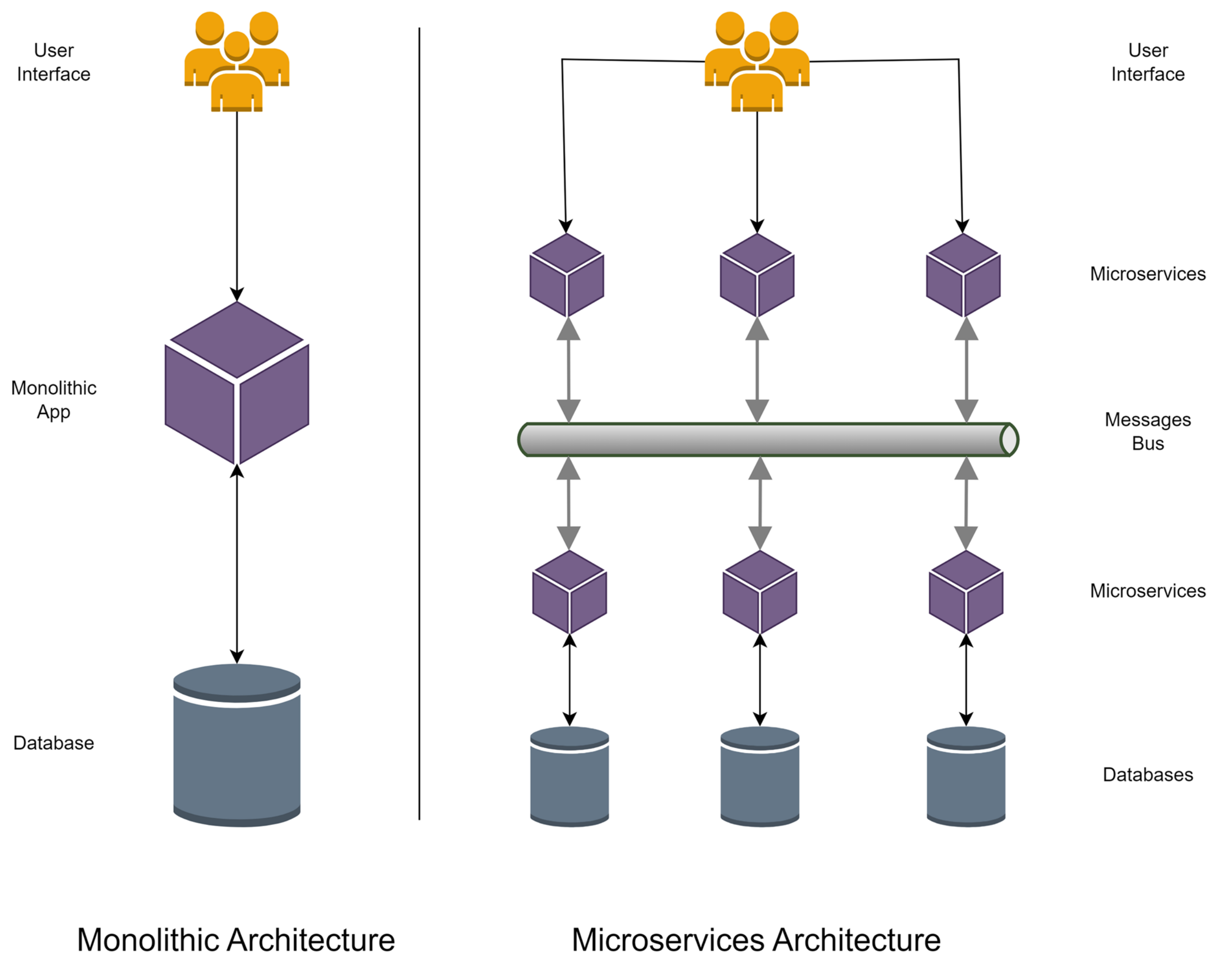

Fig. 2 Comparison between different applications architectures

to contribute to the advancement of knowledge in the field and to provide a useful resource for researchers and practitioners working with Kubernetes scheduling algorithms.

To the best of authors' knowledge, there are no related surveys found that specifically address the topic at hand. The surveys found are mostly targeted at the container orchestration in general (including Kubernetes), such as [5–8]. These surveys address Kubernetes breadthwise without targeting scheduling and diving deep into it and some even did not focus on Kubernetes. For example, some concentrated on scheduling in the cloud [9] and its associated concerns [10]. Others targeted big data applications in data center networks [11], or fog computing environments [12]. The authors have found two closely related and well-organized surveys [13] and [14] that targeted Kubernetes scheduling in depth. However, our work is different than these two surveys in terms of taxonomy, i.e., they targeted different aspects and objectives in scheduling whereas we categorized the literature into different four sub-categories: generic scheduling, multi-objective optimization-based scheduling, AI focused scheduling, and autoscaling enabled scheduling. Thereby focusing specifically on wide range of schemes related to multi-objective optimization and AI, in addition to the main scheduling with autoscaling. Our categorization, we believe, is more fine-grained and novel as compared to the existing surveys.

In this paper, the literature has been divided into four sub-categories: generic scheduling, multi-objective optimization-based scheduling, AI-focused scheduling, and autoscaling enabled scheduling. The literature pertaining to each sub-category is analyzed and summarized based on six parameters outlined in Literature review section.

Our main contributions are as follows:

- A comprehensive review of the literature on Kubernetes scheduling algorithms targeting four sub-categories: generic scheduling, multi-objective optimization-based scheduling, AI focused scheduling, and autoscaling enabled scheduling.
- A critical evaluation of the strengths and limitations of existing approaches.
- Identification of gaps and open questions in the existing literature.

The remainder of this paper is organized as follows: In Search methodology section, we describe the methodology used to conduct the survey. In Literature review section, we present the literature review along with results of our survey, including a critical evaluation of the strengths and limitations of existing approaches. A taxonomy of the identified research papers based on the literature review is presented as well. In Discussion, challenges & future suggestions section, we discuss the implications of our findings and suggest future research directions. Finally, in Conclusions section, we summarize the key contributions of the survey and provide our conclusions.

Search methodology

This section presents our search methodology for identifying relevant studies that are included in this review.

To identify relevant studies for our review, we conducted a comprehensive search of the literature using the following databases: IEEE, ACM, Elsevier, Springer, and Google Scholar. We used the following search terms: "Kubernetes," "scheduling algorithms," and "scheduling optimizing." We limited our search to studies published in the last 5 years and written in English.

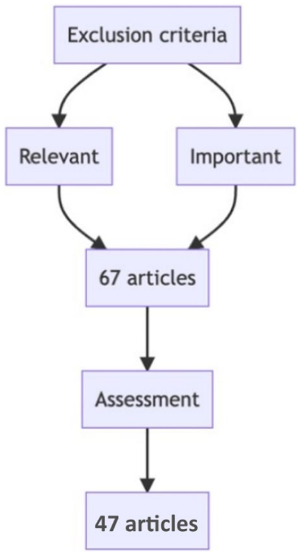

Fig. 4 Exclusion criteria

We initially identified a total of 124 studies from the database searches, see Fig. 3. We then reviewed the abstracts of these studies to identify those that were relevant to our review. We excluded studies that did not focus on Kubernetes scheduling algorithms, as well as those that were not original research or review articles. After this initial screening, we were left with 67 studies, see Fig. 4.

We then reviewed the texts of the remaining studies to determine their eligibility for inclusion in our review. We

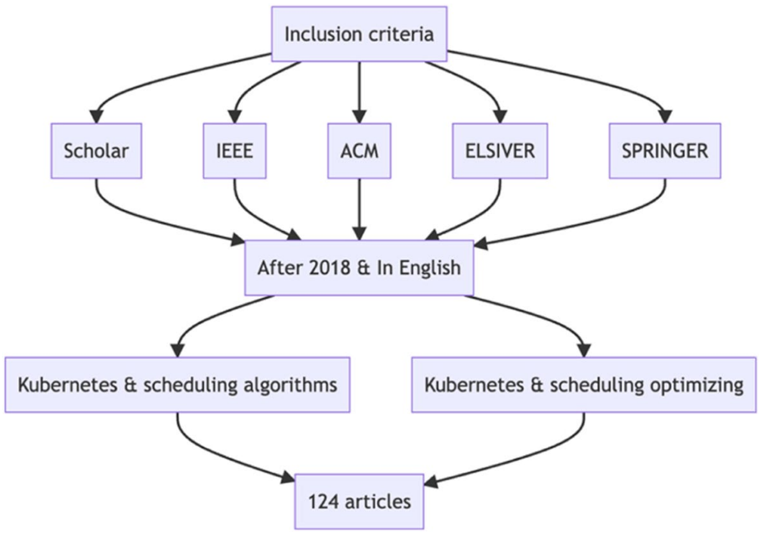

Fig. 3 Inclusion criteria

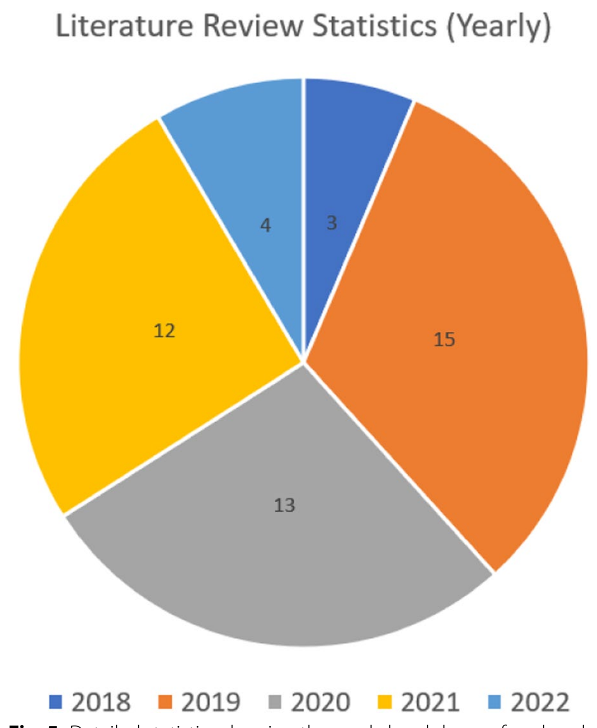

Fig. 5 Detailed statistics showing the yearly breakdown of analyzed studies

excluded studies that did not meet our inclusion criteria, which were: (1) focus on optimizing Kubernetes scheduling algorithms, (2) provide original research or a critical evaluation of existing approaches, and (3) be written in English and published in the last 5 years. After this final screening, we included 47 studies in our review, see Fig. 4. A yearly distribution of papers can be seen in Fig. 5.

We also searched the reference lists of the included studies to identify any additional relevant studies that were not captured in our database searches. We did not identify any additional studies through this process. Therefore, our review includes 47 studies on Kubernetes scheduling algorithms published in the last 5 years. These studies represent a diverse range of research methods, including surveys, experiments, and simulations.

Literature review

This section has been organized into four sub-categories, i.e., generic scheduling, multi-objective optimization-based scheduling, AI focused scheduling, and autoscaling enabled scheduling. A distribution of analyzed research papers in each category can be seen in Fig. 6. The literature in each sub-category is analyzed and then summarized based on six parameters given below:

- Objectives
- Methodology/Algorithms
- Experiments
- Findings
- Applications
- Limitations

Scheduling in Kubernetes

The field of Kubernetes scheduling algorithms has attracted significant attention from researchers and

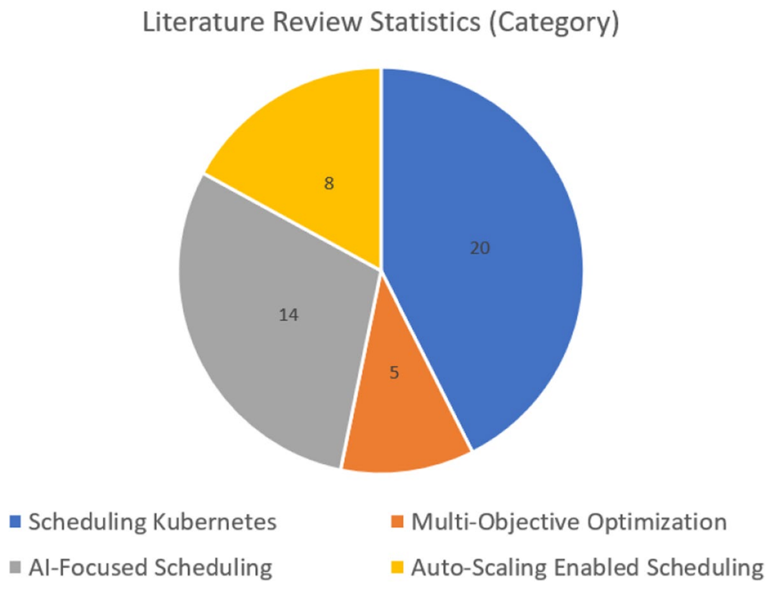

Fig. 6 Detailed statistics for each category in terms of analyzed studies

practitioners in recent years. A growing body of literature has explored the potential benefits and challenges of using different scheduling algorithms to optimize the performance of a Kubernetes cluster. In this section, we present a review of the key theories, methods, and findings from previous studies in this area.

One key theme in the literature is the need for efficient effective scheduling of workloads in a Kubernetes environment. Many studies have emphasized the limitations of traditional scheduling approaches, which often struggle to handle the complex and dynamic nature of workloads in a Kubernetes cluster. As a result, there has been increasing interest in the use of advanced scheduling algorithms to enable efficient, effective allocation of computing resources within the cluster.

Another key theme in the literature is the potential benefits of advanced scheduling algorithms for Kubernetes. Many studies have highlighted the potential for these algorithms to improve resource utilization, reduce latency, and enhance the overall performance of the cluster. Additionally, advanced scheduling algorithms have the potential to support the development of new applications and services within the Kubernetes environment, such as real-time analytics and machine learning and deep learning, see AI Focused Scheduling section.

Despite these potential benefits, the literature also identifies several challenges and limitations of Kubernetes scheduling algorithms. One key challenge is the need to address the evolving nature of workloads and applications within the cluster. Therefore, various authors focused on improving the autoscaling feature in Kubernetes scheduling to allow for automatic adjustment of the resources allocated to pods based on the current demand, more detailed discussion can be found in Autoscaling-enabled Scheduling section. Other challenges include the need to manage and coordinate multiple scheduling algorithms, and to ensure the stability and performance of the overall system.

Overall, the literature suggests that advanced scheduling algorithms offer a promising solution to the challenges posed by the complex and dynamic nature of workloads in a Kubernetes cluster. However, further research is needed to address the limitations and challenges of these algorithms, and to explore their potential applications and benefits.

In Santos et al. [15], for deployments in smart cities, the authors suggest a network-aware scheduling method for container-based apps. Their strategy is put into practice as an addition to Kubernetes' built-in default scheduling system, which is an open-source orchestrator for the automatic management and deployment of micro-services. By utilizing container-based smart city apps, the authors assess the suggested scheduling approach's performance and contrast it with that of Kubernetes' built-in default scheduling mechanism. Compared to the default technique, they discovered that the suggested solution reduces network latency by almost 80%.

In Chung et al. [16], the authors propose a new cluster scheduler called Stratus that is specialized for orchestrating batch job execution on virtual clusters in public Infrastructure-as-a-Service (IaaS) platforms. Stratus focuses on minimizing dollar costs by aggressively packing tasks onto machines based on runtime estimates, i.e., to save money, the allocated resources will be made either mostly full or empty so that they may then be released. Using the workload traces from TwoSigma and Google, the authors evaluate Stratus and establish that the proposed Stratus reduces cost by 17–44% compared to the benchmarks of virtual cluster scheduling.

In Le et al. [17], the authors propose a new scheduling algorithm called AlloX for optimizing job performance in shared clusters that use interchangeable resources such as CPUs, GPUs, and other accelerators. AlloX transforms the scheduling problem into a min-cost bipartite matching problem and provides dynamic fair allocation over time. The authors demonstrate theoretically and empirically that AlloX performs better than existing solutions in the presence of interchangeable resources, and they show that it can reduce the average job completion time significantly while providing fairness and preventing starvation.

In Zhong et al. [18], the authors propose a heterogeneous task allocation strategy for cost-efficient container orchestration in Kubernetes-based cloud computing infrastructures with elastic compute resources. The proposed strategy has three main features: support for heterogeneous job configurations, cluster size adjustment through autoscaling algorithms, and a rescheduling mechanism to shut down underutilized VM instances and reallocate relevant jobs without losing task progress. The authors evaluate their approach using the Australian National Cloud Infrastructure (Nectar) and show that it can reduce overall cost by 23–32% compared to the default Kubernetes framework.

In Thinakaran et al. [19], to create Kube-Knots, the authors combine their proposed GPU-aware resource orchestration layer, Knots, with the Kubernetes container orchestrator. Through dynamic container orchestration, Kube-Knots dynamically harvests unused computing cycles, enabling the co-location of batch and latency-critical applications and increasing overall resource utilization. The authors demonstrate that the proposed scheduling strategies increase average and 99th percentile cluster-wide GPU usage by up to 80% in the case of HPC workloads when used to plan datacenter-scale workloads using Kube-Knots on a ten-node GPU cluster. In addition, the suggested schedulers reduce energy consumption throughout the cluster by an average of 33% for three separate workloads and increase the average task completion times of deep learning workloads by up to 36% when compared to modern schedulers.

In Townend et al. [20], the authors propose a holistic scheduling system for Kubernetes that replaces the default scheduler and considers both software and hardware models to improve data center efficiency. The authors claim that by introducing hardware modeling into a software-based solution, an intelligent scheduler can make significant improvements in data center efficiency. In their initial deployment, the authors observed power consumption reductions of 10–20%.

In the work by Menouer [21], the author describes the KCSS, a brand-new Kubernetes container scheduling strategy. The purpose of KCSS is to increase performance in terms of makespan and power consumption by scheduling user-submitted containers as efficiently as possible. For each freshly submitted container, KCSS chooses the best node based on a number of factors linked to the cloud infrastructure and the user's requirements using a multi-criteria decision analysis technique. The author uses the Go programming language to create KCSS and shows how it works better than alternative container scheduling methods in a variety of situations.

In Song et al. [22], authors present a topology-based GPU scheduling framework for Kubernetes. The framework is based on the traditional Kubernetes GPU scheduling algorithm, but introduces the concept of a GPU cluster topology, which is restored in a GPU cluster resource access cost tree. This allows for more efficient scheduling of different GPU resource application scenarios. The proposed framework has been used in the production practice of Tencent and has reportedly improved the resource utilization of GPU clusters by about 10%.

In Ogbuachi et al. [23], the authors propose an improved design for Kubernetes scheduling that takes into account physical, operational, and network parameters in addition to software states in order to enable better orchestration and management of edge computing applications. They compare the proposed design to the default Kubernetes scheduler and show that it offers improved fault tolerance and dynamic orchestration capabilities.

In the work by Beltre et al. [24], utilizing fairness measures including dominant resource fairness, resource demand, and average waiting time, the authors outline a scheduling policy for Kubernetes clusters. KubeSphere, a policy-driven meta-scheduler created by the authors, enables tasks to be scheduled according to each user's overall resource requirements and current consumption. The proposed policy increased fairness in a multi-tenant cluster, according to experimental findings.

In Haja et al. [25], the authors propose a custom Kubernetes scheduler that takes into account delay constraints and edge reliability when making scheduling decisions. The authors argue that this type of scheduler is necessary for edge infrastructure, where applications are often delay-sensitive, and the infrastructure is prone to failures. The authors demonstrate their Kubernetes extension and release the solution as open source.

In Wojciechowski et al. [26], the authors propose a unique method for scheduling Kubernetes pods that makes advantage of dynamic network measurements gathered by Istio Service Mesh. According to the authors, this approach can fully automate saving up to 50% of inter-node bandwidth and up to 37% of application response time, which is crucial for the adoption of Kubernetes in 5G use cases.

In Cai et al. [27], the authors propose a feedback control method for elastic container provisioning in Kubernetes-based systems. The method uses a combination of a varying-processing-rate queuing model and a linear model to improve the accuracy of output errors. The authors compare their approach with several existing algorithms on a real Kubernetes cluster and find that it obtains the lowest percentage of service level agreement (SLA) violation and the second lowest cost.

In Ahmed et al. [28], the deployment of Docker containers in a heterogeneous cluster with CPU and GPU resources can be managed via the authors' dynamic scheduling framework for Kubernetes. The Kubernetes Pod timeline and previous data about the execution of the containers are taken into account by the platform, known as KubCG, to optimize the deployment of new containers. The time it took to complete jobs might be cut by up to 64% using KubCG, according to the studies the authors conducted to validate their algorithm.

In Ungureanu et al. [29], the authors propose a hybrid shared-state scheduling framework for Kubernetes that combines the advantages of centralized and distributed scheduling. The framework uses distributed scheduling agents to delegate most tasks, and a scheduling correction function to process unprioritized and unscheduled tasks. Based on the entire cluster state the scheduling decisions are made, which are then synchronized and updated by the master-state agent. The authors performed experiments to test the behavior of their proposed scheduler and found that it performed well in different scenarios, including failover and recovery. They also found that other centralized scheduling frameworks may not perform well in situations like collocation interference or priority preemption.

In Yang et al. [30], the authors present the design and implementation of KubeHICE, a performance-aware container orchestrator for heterogeneous-ISA architectures in cloud-edge platforms. KubeHICE extends Kubernetes with two functional approaches, AIM (Automatic Instruction Set Architecture Matching) and PAS (Performance-Aware Scheduling), to handle heterogeneous ISA and schedule containers according to the computing capabilities of cluster nodes. The authors performed experiments to evaluate KubeHICE and found that it added no additional overhead to container orchestration and was effective in performance estimation and resource scheduling. They also demonstrated the advantages of KubeHICE in several real-world scenarios, showing for example a 40% increase in CPU utilization when eliminating heterogeneity.

In Li et al. [31], the authors propose two dynamic scheduling algorithms, Balanced-Disk-IO-Priority (BDI) and Balanced-CPU-Disk-IO-Priority (BCDI), to address the issue of Kubernetes' scheduler not taking the disk I/O load of nodes into account. BDI is designed to improve the disk I/O balance between nodes, while BCDI is designed to solve the issue of load imbalance of CPU and disk I/O on a single node. The authors perform experiments to evaluate the algorithms and find that they are more effective than the Kubernetes default scheduling algorithms.

In Fan et al. [32], the authors propose an algorithm for optimizing the scheduling of pods in the Serverless framework on the Kubernetes platform. The authors argue that the default Kubernetes scheduler, which operates on a pod-by-pod basis, is not well-suited for the rapid deployment and running of pods in the Serverless framework. To address this issue, the authors propose an algorithm that uses simultaneous scheduling of pods to improve the efficiency of resource scheduling in the Serverless framework. Through preliminary testing, the authors found that their algorithm was able to greatly reduce the delay in pod startup while maintaining a balanced use of node resources.

In Bestari et al. [33], the authors propose a scheduler for distributed deep learning training in Kubeflow that combines features from existing works, including autoscaling and gang scheduling. The proposed scheduler includes modifications to increase the efficiency of the training process, and weights are used to determine the priority of jobs. The authors evaluate the proposed scheduler using a set of Tensorflow jobs and find that it improves training speed by over 26% compared to the default Kubernetes scheduler.

In Dua et al. [34], the authors present an alternative algorithm for load balancing in distributed computing environments. The algorithm uses task migration to balance the workload among processors of different capabilities and configurations. The authors define labels to classify tasks into different categories and configure clusters dedicated to specific types of tasks.

The above-mentioned schemes are summarized in Table 1.

Scheduling using multi-objective optimization

Multi-objective optimization scheduling takes into account multiple objectives or criteria when deciding how to allocate resources and schedule containers on nodes in the cluster. This approach is particularly useful in complex distributed systems where there are multiple competing objectives that need to be balanced to achieve the best overall performance. In a multi-objective optimization scheduling approach, the scheduler considers multiple objectives simultaneously, such as minimizing response time, maximizing resource utilization, and reducing energy consumption. The scheduler uses optimization algorithms to find the optimal solution that balances these objectives.

Multi-objective optimization scheduling can help improve the overall performance and efficiency of Kubernetes clusters by taking into account multiple objectives when allocating resources and scheduling containers. This approach can result in better resource utilization, improved application performance, reduced energy consumption, and lower costs.

Some examples of multi-objective optimization scheduling algorithms used in Kubernetes include genetic algorithms, Ant Colony Optimization, and particle swarm optimization. These algorithms can help optimize different objectives, such as response time, resource utilization, energy consumption, and other factors, to achieve the best overall performance and efficiency in the Kubernetes cluster.

In this section, multi-objective scheduling proposals are discussed.

In Kaur et al. [35], the authors propose a new controller for managing containers on edge-cloud nodes in Industrial Internet of Things (IIoT) systems. The controller, called Kubernetes-based energy and interference driven scheduler (KEIDS), is based on Google Kubernetes and is designed to minimize energy utilization and interference in IIoT systems. KEIDS uses integer linear programming to formulate the task scheduling problem as a multi-objective optimization problem, taking into account factors such as energy consumption, carbon emissions, and interference from other applications. The authors

Table 1 Literature Summary (Scheduling Kubernetes)

#	Objectives	Methodology/Algorithms	Experiments	Findings	Applications	Limitations
[15]	To develop a scheduling strategy for container-based apps in Smart City deployments that is network-aware.	As an addition to the default scheduling mechanism built into Kubernetes, a network-aware scheduling method is suggested and put into practice.	Evaluated utilizing container-based Smart City applications and validated on the Kubernetes platform.	Comparing the suggested method to the default scheduling mechanism, network latency is reduced by 80%.	Can be used in Fog Computing environments for delay-sensitive and data-intensive services.	Further testing and implementation may reveal limitations and future improvements.
[16]	Public cloud container scheduling with consideration for cost.	A cluster scheduler with a focus on organizing batch job execution on virtual clusters, which is termed as Stratus.	On the basis of cluster workload traces from Google and TwoSigma, simulation experiments were conducted.	Stratus reduces virtual cluster scheduling costs by 17–44% compared to state-of-the-art approaches.	Batch job execution on virtual clusters in the public cloud	Limited to the context of batch job execution on virtual clusters in the public cloud
[17]	To improve functionality and ensure user equality in a shared cluster with swappable hardware resources for deep learning frameworks.	min-cost bipartite matching	Large-scale simulations and evaluations on a small-scale CPU-GPU hybrid cluster.	AlloX can drastically shorten the average work completion time, eliminate starvation, and ensure fairness.	Scheduling jobs over interchangeable resources in a shared cluster	Interchangeable resources exceeding the threshold of two may cause many problems.
[18]	Containers' initial placement is optimized via task packing, enabling cluster size adjustment to meet changing workloads through autoscaling algorithms, and developing a rescheduling mechanism to shut down underutilized VM instances for cost savings while preserving task progress.	Heterogeneous job configurations Autoscaling algorithms Rescheduling mechanism	Validated using the Australian National Cloud Infrastructure (Nectar).	When compared to the standard Kubernetes framework, the suggested solution could lower overall costs for various cloud workload patterns by 23% to 32%.	Container orchestration with low costs on cloud computing infrastructures powered by Kubernetes.	VM types may also be taken into consideration.
[19]	To develop a GPU-aware resource orchestration layer for datacenters To improve resource utilization and reduce operational costs in datacenters. To improve Quality of Service (QoS) for user-facing queries	Presented Kube-Knots, a Kubernetes-integrated resource orchestration layer that is GPU-aware. Kube-Knots uses dynamic container orchestration to dynamically harvest available computation cycles. Two GPU-based scheduling methods (CBP and PP) are created to schedule workloads at datacenter scale using Kube-Knots.	Evaluated CBP and PP on a ten node GPU cluster Compared results with state-of-the-art schedulers	For HPC applications, CBP and PP increase GPU usage across the cluster by up to 80% on average. Deep learning workloads' average job completion increased by up to 36% and 33% cluster-wide energy reduction For latency-critical queries, PP ensures end-to-end QoS by lowering QoS breaches by up to 53%.	To improve resource utilization and reduce operational costs in GPU-based datacenters	–
[20]	To improve efficiency of data centers through holistic scheduling in Kubernetes To consider virtual and physical infrastructures and business processes in scheduling	Replaced the Kubernetes default scheduler with a proposed all-encompassing scheduling framework. Both software and hardware model considerations are made by the scheduler. System was deployed in a real data center.	Deployment in real data center	Reductions in power consumption of 10% to 20% were noted The effectiveness of the data center can be significantly increased by an intelligent scheduler.	To improve efficiency of data centers through software-based solutions	Further research is needed in this area

Table 1 (continued)

#	Objectives	Methodology/Algorithms	Experiments	Findings	Applications	Limitations
[21]	A new Kubernetes container scheduling strategy (KCSS) has been introduced. Boost the efficiency of many online-submitted containers' scheduling.	Using a variety of factors, choose the best node for each newly submitted container. To combine all criteria into a single rank, use the Technique for the Order of Prioritization by Similarity to Ideal Solution (TOPSIS) method.	Conducted experiments on different scenarios Used data from cloud infrastructure and user need	When compared to other container scheduling algorithms, KCSS enhances performance.	Can be used in industrial and academic fields for container-orchestration systems	Limited to the six key criteria used in the experiments Potential to expand the criteria and improve the performance further in future work.
[22]	Presented a Kubernetes GPU scheduling mechanism based on topology. Increase resource efficiency and load distribution in the GPU cluster.	The foundation of the system is the established Kubernetes GPU scheduling mechanism. In a resource access cost tree, the topology of the GPU cluster is restored. The resource access cost tree is used to schedule and adapt various GPU resource application scenarios.	Tencent has employed GaiaGPU in actual production.	Improved resource utilization by about 10% Improved performance on load balance	Used in production at Tencent	–
[23]	To develop a context-aware Kubernetes scheduler that takes into account physical, operational, and network parameters in order to improve service availability and performance in 5G edge computing	Real-time edge device data integration into the scheduler decision algorithm.	Comparison with the default Kubernetes scheduler	The suggested scheduler offers increased fault tolerance capabilities along with advanced orchestration and management.	5G edge computing	–
[24]	To develop a policy-driven meta-scheduler for Kubernetes clusters that enables efficient and fair resource allocation for multiple users	Dominant Resource Fairness (DRF) policy Additional fairness metrics based on task resource demand and average waiting time	–	The proposed meta-scheduler improves fairness in multi-tenant Kubernetes clusters	Kubernetes clusters	–
[25]	To modify Kubernetes to be better suited for edge infrastructure, with a focus on network latency and self-healing capabilities	Custom Kubernetes scheduler that considers applications' delay constraints and edge reliability	–	The modified Kubernetes is better suited for edge infrastructure	Edge computing	–
[26]	To improve Kubernetes scheduling for performance-sensitive containerized workflows, particularly in the context of 5G edge applications	NetMARKS is a cutting-edge method for scheduling Kubernetes pods that makes advantage of dynamic network metrics gathered with Istio Service Mesh.	Validated using different workloads and processing layouts	NetMARKS can save up to 50% of inter-node bandwidth while reducing application response times by up to 37%.	Kubernetes in 5G edge computing and machine-to-machine communication	–

Table 1 (continued)

#	Objectives	Methodology/Algorithms	Experiments	Findings	Applications	Limitations
[27]	Create a feedback control approach for Kubernetes-based systems' elastic container provisioning of Web systems.	Combining a linear model with a varying-processing-rate queuing model can increase the accuracy of output errors.	Evaluated on a real Kubernetes cluster	When compared to cutting-edge algorithms, the suggested approach achieves the lowest percentage of SLA violation and the second-lowest cost.	Elastic container provisioning in Kubernetes-based systems	–
[28]	Create a dynamic Kubernetes scheduler to help a heterogeneous cluster deploy Docker containers more effectively. Utilize past data on container execution to speed up task completion.	Developed the KubCG dynamic scheduling platform. Introduced a new scheduler that takes into account past data on container execution as well as the timetable for Kubernetes Pods.	Conducted different tests to validate the new algorithm	In experiments, KubCG was able to cut task completion times from 100 to 64% of the original time.	Used for the deployment of cloud-based services that require GPUs for tasks like deep learning and video processing.	Further testing and validation are needed to determine the effectiveness of the algorithm in a variety of scenarios.
[29]	Describe a new method for arranging workloads in a Kubernetes cluster.	Framework model for hybrid shared-state scheduling. On the basis of the cluster's overall state, scheduling decisions are determined.	Tested proposed scheduler behavior under different scenarios, including failover/recovery in a deployed Kubernetes cluster	The suggested scheduler operates in circumstances like priority preemption or collocation interference. The features of both centralized and distributed scheduling frameworks are included in the scheduler.	Used in Kubernetes cluster to optimize resource utilization	Further testing and implementation needed to fully evaluate the effectiveness of the proposed scheduler.
[30]	Develop and put into use KubeHICE, a container orchestrator for heterogeneous ISA architectures on cloud edge platforms. Assess the efficiency and performance of KubeHICE in handling heterogeneous-ISA clusters.	By using AIM and PAS, KubeHICE expands open source Kubernetes. AIM automatically locates a node that is appropriate for the ISAs that the containerized application supports. PAS schedules containers based on the computational capacity of cluster nodes.	KubeHICE was tested in several real-world scenarios.	KubeHICE is efficient in performance estimation and resource scheduling while adding no further overhead to container orchestration. When handling heterogeneity, KubeHICE can improve CPU utilization by up to 40%.	KubeHICE is beneficial for containerized applications in heterogeneous cloud-edge platforms	–
[31]	Make the Kubernetes scheduler more efficient by incorporating the disk I/O load.	To improve the disk I/O balance between nodes, a dynamic scheduling approach called Balanced-Disk-IO-Priority (BDI) was proposed. Also presented the Balanced-CPU-Disk-IO-Priority (BCDI) dynamic scheduling algorithm to address the problem of unbalanced CPU and disk I/O load on a single node.	According to experimental findings, the BDI and BCDI algorithms are superior to the default scheduling algorithms in Kubernetes.	The load imbalance of CPU and disk I/O on a single node is resolved by the BDI and BCDI algorithms, which also enhance the disk I/O balance between nodes.	Can be used to improve the performance of Kubernetes in managing containerized applications	Further research may be needed to optimize the BDI and BCDI algorithms and evaluate their performance in different scenarios.

Table 1 (continued)

#	Objectives	Methodology/Algorithms	Experiments	Findings	Applications	Limitations
[32]	Investigate how Serverless frameworks built on Kubernetes systems can schedule pods more efficiently in large-scale concurrent applications.	To further maximize the effectiveness of pod scheduling in Serverless cloud paradigms, a scheduling approach leveraging concurrent scheduling of the same pod is proposed.	Preliminary verification is performed to test the effectiveness of the proposed algorithm.	The suggested approach can significantly cut down on pod startup time while maintaining resource balance on each node.	The proposed algorithm is used to improve efficiency of pod scheduling in Serverless cloud paradigms.	The effectiveness is only verified via preliminary experiments. Also, the algorithm is only applicable to Serverless frameworks.
[33]	Present a resource rescheduling and Kubernetes scheduler extension that combines QoE metrics into SLOs.	Use the QoE metric proposed in the ITU P.1203 standard Evaluate architecture using video streaming services co-located with other services	Evaluate architecture using video streaming services co-located with other services	The average QoE is increased by 50%. The average QoE was raised by 135% as a result of resource rescheduling. Over-provisioning was completely removed by the suggested architecture.	Improving QoE for cloud environments	Limited to the specific QoE metric used. Further research may be needed to evaluate the effectiveness of the proposed architecture with other QoE metrics.
[34]	Enable the secure colocation of best-effort processes and latency-sensitive services in Kubernetes clusters to increase resource utilization. Flexibly divide resources among various workload categories. Improve hardware and software isolation capabilities for containers.	Based on Kubernetes extension mechanisms, Zeus was developed. Best-effort jobs are scheduled by Zeus based on actual server use. Through the coordination of hardware and software isolation elements, Zeus improves container isolation.	In a large-scale production setting, Zeus is assessed using latency-sensitive services and best-effort jobs.	Zeus can increase CPU usage from 15 to 60% on average without breaking SLO. Zeus can significantly increase how efficiently Kubernetes clusters use their resources.	Zeus can be used to improve the resource utilization of Kubernetes clusters	–

evaluate KEIDS using real-time data from Google compute clusters and find that it outperforms existing state-of-the-art schemes.

In Lin et al. [36], the authors propose a multi-objective optimization model for container-based microservice scheduling in cloud architectures. They present an ant colony algorithm for solving the scheduling problem, which takes into account factors such as computing and storage resource utilization, the number of microservice requests, and the failure rate of physical nodes. The authors evaluate the proposed algorithm using experiments and compare its performance to other related algorithms. They find that the proposed algorithm achieves better results in terms of cluster service reliability, cluster load balancing, and network transmission overhead.

In Wei-guo et al. [37], the authors propose an improved scheduling algorithm for Kubernetes by combining ant colony optimization and particle swarm optimization to better balance task assignments and reduce resource costs. The authors implemented the algorithm in Java and tested it using the CloudSim tool, showing that it outperformed the original scheduling algorithm.

In the work by Oleghe [38], the idea of container placement and migration in edge servers, as well as the scheduling models created for this purpose, are discussed by the author. The majority of scheduling models, according to the author, are based mostly on heuristic algorithms and use multi-objective optimization models or graph network models. The study also points out the lack of studies on container scheduling models that take dispersed edge computing activities into account and predicts that future studies in this field will concentrate on scheduling containers for mobile edge nodes.

In Carvalho et al. [39], The authors offer an addition to the Kubernetes scheduler that uses Quality of Experience (QoE) measurements to help cloud management Service Level Objectives (SLOs) be more accurate. In the context of video streaming services that are co-located with other services, the authors assess the suggested architecture using the QoE metric from the ITU P.1203 standard. According to the findings, resource rescheduling increases average QoE by 135% while the proposed scheduler increases it by 50% when compared to other schedulers.

The above-mentioned schemes are summarized in Table 2.

AI focused scheduling

Many large companies have recently started to provide AI based services. For this purpose, they have installed machine/deep learning clusters composed of tens to thousands of CPUs and GPUs for training their deep learning models in a distributed manner. Different machine learning frameworks are used such as MXNet [40], TensorFlow [41], and Petuum [42]. Training a deep learning model is usually very resource hungry and time consuming. In such a setting, efficient scheduling is crucial in order to fully utilize the expensive deep learning cluster and expedite the model training process. Different strategies have been used to schedule tasks in this arena, for examples, general purpose schedulers are customized to tackle distributed deep learning tasks, example include [43] and [44]; however, they statically allocate resources and do not adjust resource under different load conditions which lead to poor resource utilization. Others proposed dynamic allocation of resources after carefully analyzing the workloads, examples include [45] and [46].

In this section, deep learning focused schedulers are surveyed.

In Peng et al. [46], the authors propose a customized job scheduler for deep learning clusters called Optimus. The goal of Optimus is to minimize the time required for deep learning training jobs, which are resource-intensive and time-consuming. Optimus employs performance models to precisely estimate training speed as a function of resource allocation and online fitting to anticipate model convergence during training. These models inform how Optimus dynamically organizes tasks and distributes resources to reduce job completion time. The authors put Optimus into practice on a deep learning cluster and evaluate its efficiency in comparison to other cluster schedulers. They discover that Optimus beats conventional schedulers in terms of job completion time and makespan by roughly 139% and 63%, respectively.

In Mao et al. [47], the authors propose using modern machine learning techniques to develop highly efficient policies for scheduling data processing jobs on distributed compute clusters. They present their system, called Decima, which uses reinforcement learning (RL) and neural networks to learn workload-specific scheduling algorithms. Decima is designed to be scalable and able to handle complex job dependency graphs. The authors report that their prototype integration with Spark on a 25-node cluster improved average job completion time by at least 21% over existing hand-tuned scheduling heuristics, with up to $2\times$improvement during periods of high cluster load.

In Chaudhary et al. [48], a distributed fair share scheduler for GPU clusters used for deep learning training termed as Gandivafair is presented by the authors. This GPU cluster utilization system offers performance isolation between users and is created to strike a balance between the competing demands of justice and efficiency. In spite of cluster heterogeneity, Gandivafair is the first scheduler to fairly distribute GPU time among all active

Table 2 Literature summary (Multi-Objective Optimization)

#	Objectives	Methodology/Algorithms	Experiments	Findings	Applications	Limitations
[35]	Present a capable controller for managing containers on edge-cloud nodes in industrial IoT contexts while accounting for interference and energy use.	Integer linear programming based on multi-objective optimization.	Data obtained in real time from the Google compute cluster.	By reducing the energy consumption of edge-cloud nodes and scheduling applications optimally with the least amount of interference, the proposed Kubernetes-based energy and interference driven scheduler (KEIDS) improves performance for end users.	Container management and scheduling for Industrial IoT.	The limitations and future potential of KEIDS are not specified in the given text.
[36]	Build a multi-objective scheduling model for container-based microservices and to suggest an ant colony method to handle scheduling issues.	Ant colony algorithm	Real data from Alibaba cluster Trace V2018 having an application with 17 micro servers	In comparison to previous relevant algorithms, the suggested optimization method outperformed them in the optimization of cluster service dependability, cluster load balancing, and network transmission overhead.	Container-based microservice scheduling in cloud architectures	High time complexity plus real cloud container should be used.
[37]	To improve Kubernetes' resource scheduling scheme	The authors examine the source code of Kubernetes' scheduling module, extract its model, and create and carry out a simulation experiment using the model. The K8s scheduling model is then enhanced by combining the ant colony and particle swarm optimization algorithms.	The authors schedule resources for K8s using the Java programming language and the CloudSim tool.	The experimental results demonstrate that the suggested approach outperforms the original scheduling technique, resulting in a lower overall resource cost, a higher maximum node load, and more evenly distributed job assignment.	Kubernetes can deploy containerized applications on a wide scale in private, public, and hybrid cloud environments using the better resource scheduling scheme.	–
[38]	To describe in more detail the idea of container placement and migration in edge computing, as well as to examine the scheduling models created for this purpose.	The container placement problem can be abstracted using graph network models or multi-objective optimization models. Algorithms based on heuristics to address the scheduling issue.	–	Most existing container scheduling models are heuristic-based and consider only static edge computing tasks, with limited research on decentralized scheduling systems	Container-based edge computing	Future research in container scheduling should focus on decentralized systems and mobile edge nodes.
[39]	By assigning virtual network functions (VNFs) to appropriate places, virtual networks' resilience can be increased.	Optimization models and heuristic algorithms to solve VNF placement problems	–	Implementation of function scheduler plugins that can connect multiple optimization models with Kubernetes and allocate functions automatically	Allocating VNFs to nodes in Kubernetes	–

users. The authors demonstrate that Gandivafair delivers both fairness and efficiency under realistic multi-user workloads by evaluating it using a prototype implementation on a heterogeneous 200-GPU cluster.

In Fu et al. [49], the authors propose a new container placement scheme called ProCon for scheduling jobs in a Kubernetes cluster. ProCon uses an estimation of future resource usage to balance resource contentions across the cluster and reduce the completion time and makespan of jobs. The authors demonstrate through experiments that ProCon decreases completion time by up to 53.3% for a specific job and enhances general performance by 23.0%. In addition, ProCon shows a makespan improvement of up to 37.4% in comparison to Kubernetes' built-in default scheduler.

In Peng et al. [50], the authors propose DL2, a deep learning-based scheduler for deep learning clusters that aims to improve global training job expedition by dynamically resizing resources allocated to jobs. The authors implement DL2 on Kubernetes and evaluate its performance against a fairness scheduler and an expert heuristic scheduler. The results show that DL2 outperforms the other schedulers in terms of average job completion time.

In Mao et al. [51], the authors propose a new container scheduler called SpeCon optimized for short-lived deep learning applications. SpeCon is designed to improve resource utilization and job completion times in a Kubernetes cluster by analyzing the progress of deep learning training processes and speculatively migrating slow-growing models to release resources for faster-growing ones. The authors conduct experiments that demonstrate that SpeCon improves individual job completion times by up to 41.5%, improves system-wide performance by 14.8%, and reduces makespan by 24.7%.

In Huang et al. [52], for scheduling independent batch jobs across many federated cloud computing clusters, the authors suggest a deep reinforcement learning-based job scheduler dubbed RLSK. The authors put RLSK into use on Kubernetes and tested its performance through simulations, demonstrating that it can outperform conventional scheduling methods.

The work by Wang et al. [53] describes MLFS, a feature-based task scheduling system for machine learning clusters that can conduct both data- and model-parallel processes. To determine task priority for work queue ordering, MLFS uses a heuristic scheduling method. The data from this method is then used to train a deep reinforcement learning model for job scheduling. In comparison to existing work schedules, the proposed system is shown to reduce job completion time by up to 53%, makespan by up to 52%, and increase accuracy by up to 64%. The system is tested using real experiments and large-scale simulations based on real traces.

In Han et al. [54], the authors present KaiS, an edge-cloud Kubernetes scheduling framework based on learning. KaiS models system state data using graph neural networks and a coordinated multi-agent actor-critic method for decentralized request dispatch. Research indicates that when compared to baselines, KaiS can increase average system throughput rate by 14.3% and decrease scheduling cost by 34.7%.

In Casquero et al. [55], the Kubernetes orchestrator's scheduling task is distributed among processing nodes by the authors' proposed custom scheduler, which makes use of a Multi-Agent System (MAS). According to the authors, this method is quicker than the centralized scheduling strategy employed by the default Kubernetes scheduler.

In Yang et al. [56], the authors propose a method for optimizing Kubernetes' container scheduling algorithm by combining the grey system theory with the LSTM (Long Short-Term Memory) neural network prediction method. They perform experiments to evaluate their approach and find that it can reduce the resource fragmentation problem of working nodes in the cluster and increase the utilization of cluster resources.

In Zhang et al. [57], a highly scalable cluster scheduling system for Kubernetes, termed as Zeus, is proposed by the authors. The main feature of Zeus is that based on the actual server utilization it schedules the best-effort jobs. It has the ability to adaptively divide resources between workloads of two different classes. Zeus is meant to enable the safe colocation of best-effort processes and latency-sensitive services. The authors test Zeus in a real-world setting and discover that it can raise average CPU utilization from 15 to 60% without violating Service Level Objectives (SLOs).

In Liu et al. [58], the authors suggest a scheduling strategy for deep learning tasks on Kubernetes that takes into account the tasks' resource usage characteristics. To increase task execution efficiency and load balancing, the suggested paradigm, dubbed FBSM, has modules for a GPU sniffer and a balance-aware scheduler. The execution of deep learning tasks is sped up by the suggested system, known as KubFBS, according to the authors' evaluation, which also reveals improved load balancing capabilities for the cluster.

In Rahali et al. [59], the authors propose a solution for resource allocation in a Kubernetes infrastructure hosting network service. The proposed solution aims to avoid resource shortages and protect the most critical functions. The authors use a statistical approach to model and solve the problem, given the random nature of the treated information.

The above-mentioned schemes are summarized in Table 3.

Table 3 Literature summary (AI-Focused Scheduling)

#	Objectives	Methodology/Algorithms	Experiments	Findings	Applications	Limitations
[46]	To develop an effective deep learning cluster resource scheduler.	Predicts model convergence during training via online fitting and creates performance models to calculate training speed as a function of the resources allotted to each job. Deep learning tasks are placed, and resources are dynamically allocated in order to reduce the amount of time needed to complete each task.	Deployed on a deep learning cluster that runs 9 MXNet training jobs on 7 CPU servers and 6 GPU machines.	Optimus performs about 139% and 63% better than comparable cluster schedulers in terms of job completion time and makespan, respectively.	Can be used in production clusters with deep learning workloads.	Further testing and implementation may reveal limitations and future improvements.
[47]	Data processing job scheduling on distributed computing clusters.	Utilize neural networks and reinforcement learning to learn workload-specific scheduling algorithms.	Spark integration prototype on a 25 node cluster.	Comparing Decima to hand-tuned scheduling heuristics, the average job completion time is improved by at least 21%.	Scheduling data processing jobs on distributed compute clusters	Open research studies on resource management and computation optimization in edge computing
[48]	Develop a fair share scheduler for deep learning training on GPU clusters that strikes a balance between the competing demands of efficiency and fairness.	Gandivafair provides performance isolation between users and allocates cluster-wide GPU time fairly among active users. Gandivafair incentivizes users to use older GPUs with a novel resource trading mechanism that maximizes cluster efficiency without affecting fairness guarantees.	Realistic multi-user workloads were used to implement and assess the system in a heterogeneous 200-GPU cluster.	Gandivafair achieves both fairness and efficiency.	Can be used in GPU clusters for deep learning training.	Further testing and implementation may reveal limitations and future improvements.
[49]	Fully exploit and harness the power of big data, as well as to speed up processing times and enhance Kubernetes cluster performance in general.	Presented a container placement strategy based on progress (ProCon). ProCon takes into account both the current resource usage of the workforce and the projection of future resource demand. ProCon decreases completion time and makespan while balancing resource contentions among clusters.	Extensive experiments conducted to test ProCon	ProCon boosts overall performance by 23.0% and can cut completion times for certain jobs by up to 53.3%. It shows a makespan improvement of up to 37.4% over the Kubernetes scheduler by default.	To improve performance of Kubernetes clusters	–

Table 3 (continued)

#	Objectives	Methodology/Algorithms	Experiments	Findings	Applications	Limitations
[50]	Create a general-purpose, effective deep learning cluster scheduler, and to get the most out of expensive deep learning clusters.	Presented DL2, a scheduler for deep learning clusters that is driven by deep learning. The supervised learning and reinforcement learning approaches are combined in DL2. During the course of DL jobs' training, offline supervised learning is used to warm up the neural network and reinforcement learning is used to fine-tune it. Online resource allocation decisions for jobs are made by DL2 using neural networks.	Kubernetes was used to develop DL2, which allowed for dynamic resource scalability in DL jobs on MXNet. A thorough analysis was done to compare DL2 with the expert heuristic scheduler and the fairness scheduler (DRF) (Optimus).	In terms of average work completion time, DL2 performs better than DRF by 44.1% and Optimus by 17.5%.	To improve resource scheduling in deep learning clusters	–
[51]	To improve scheduling for deep learning applications in Kubernetes clusters To optimize resource management for deep learning workloads	SpeCon, a unique container scheduler that is tailored for fleeting deep learning applications, was proposed. The foundation of Scheduler is virtualized containers like Kubernetes and Docker. In order to free up resources for quickly expanding models, algorithms keep track of training progress and speculatively move slow-growing models.	Extensive experiments were performed to evaluate the proposed scheduler	SpeCon reduces the time it takes for each job to be completed by up to 41.5%. It also increases makespan by 24.7% and system performance by 14.8%.	To optimize scheduling for deep learning workloads on Kubernetes	–
[52]	RLSK is a deep reinforcement learning-based job scheduler that can adaptively distribute independent batch processes across many federated cloud computing clusters.	RLSK is based on reinforcement learning and is implemented on Kubernetes	Simulations are conducted to evaluate the performance of RLSK	RLSK outperforms traditional scheduling algorithms	Scheduling independent batch jobs in federated cloud computing clusters.	–
[53]	Create a machine learning cluster scheduling system and increase its efficiency and precision.	A heuristic scheduling approach that takes an ML job's spatial and temporal characteristics into account. When the system is overloaded, this system load control method removes tasks that produce little to no gain in accuracy and shifts tasks from overloaded servers to underloaded servers depending on task priority.	Large-scale simulations based on actual data and actual experiments	When compared to current ML job schedulers, MLFS decreases JCT by up to 53%, makespan by up to 52%, and improves accuracy by up to 64%.	Job scheduling for large-scale machine learning clusters	–

Table 3 (continued)

#	Objectives	Methodology/Algorithms	Experiments	Findings	Applications	Limitations
[54]	Presented a learning-based scheduling framework for edge-cloud systems focused on Kubernetes (KaiS) that raises the throughput rate of processing requests over the long term.	In order to provide decentralized request dispatch and dynamic dispatch spaces within the edge cluster, KaiS employs a coordinated multi-agent actor-critic method. In order to reduce the orchestration dimensionality by stepwise scheduling, it additionally employs graph neural networks to embed system state information and combines the results with multiple policy networks.	Experiments were conducted using real workload traces.	Regardless of request arrival patterns or system scales, KaiS was successful in learning the proper scheduling policies. In comparison to baselines, it increased system throughput rate by 14.3% and decreased scheduling cost by 34.7%.	KaiS can be used to improve the performance of Kubernetes-oriented edge-cloud systems.	–
[55]	Create a custom scheduler for the Kubernetes orchestrator that uses a model-based, multi-purpose Multi-Agent System (MAS) platform to divide the scheduling task among the processing nodes.	MAS platform Kubernetes orchestrator	–	The new scheduling approach is faster than the default scheduler of Kubernetes.	Fog-in-the-loop (FIL) applications	–
[56]	Develop a Kubernetes scheduling plan that blends the LSTM neural network prediction method with the grey system theory.	The method uses the grey system theory and LSTM neural network prediction to optimize the container scheduling algorithm.	The method was tested using experience results.	The algorithm was able to reduce resource fragmentation in the cluster and increase cluster resource utilization.	The method can be used to improve the performance of Kubernetes in managing containers in a cluster.	The limitations of the method are not mentioned in the text. Future work may focus on improving the performance of the algorithm and testing it in more complex scenarios.
[57]	Create a dynamic resource scheduler in Kubernetes for distributed deep learning training.	Combine the methods of DRAGON and OASIS to create a scheduler that supports weighted autoscaling and gang scheduling for its jobs.	Evaluation using a set of Tensorflow jobs	Scheduler increases training speed by over 26% compared to default Kubernetes scheduler	Used for distributed deep learning training in Kubeflow	–
[58]	Suggests a more effective scheduling method for deep learning platforms that can accommodate team cooperation. integrate a Docker-based deep learning platform with the improved Kubernetes scheduling algorithm.	Model users of the team as virtual clusters and routinely check the load on the clusters. Use a Docker-based deep learning platform with an improved Kubernetes scheduling algorithm.	The proposed scheduling algorithm is tested using a deep learning platform and multiple teams of users	The proposed algorithm ensures load balance and meets the needs of users	The proposed scheduling algorithm can be used in deep learning platforms to support multi-team collaboration	–

Table 3 (continued)

#	Objectives	Methodology/Algorithms	Experiments	Findings	Applications	Limitations
[59]	Offer a balanced, fine-grained scheduling mechanism for deep learning workloads.	Create a balanced, fine-grained scheduling model that takes into account the DL task's resource consumption characteristics. It is suggested to build a scheduling system named KubFBS using specialized GPU sniffer and balance-aware scheduler modules.	The proposed system is evaluated using real-world DL tasks and the cluster is a 16-node Kubernetes cluster	KubFBS expedites the completion of DL activities and enhances the cluster's capacity for load balancing.	KubFBS can be used to schedule deep learning tasks in a Kubernetes cluster	Further experiments and evaluations are needed to demonstrate the effectiveness of the proposed system in different scenarios. Future work could also focus on improving the scalability of the proposed system.

Autoscaling-enabled scheduling

Autoscaling is an important feature in Kubernetes scheduling because it allows for automatic adjustment of the resources allocated to pods based on the current demand. It allows efficient resource utilization, improved performance, cost savings, and high availability of the application. Auto rescaling and scheduling are related in that auto rescaling can be used to ensure that there are always enough resources available to handle the tasks that are scheduled. For example, if the scheduler assigns a new task to a worker node, but that node does not have enough resources to execute the task, the auto scaler can add more resources to that node or spin up a new node to handle the task. In this way, auto rescaling and scheduling work together to ensure that a distributed system is able to handle changing workloads and optimize resource utilization. Some of the schemes related to this category are surveyed below.

In Taherizadeh et al. [60], the authors propose a new dynamic multi-level (DM) autoscaling method for container-based cloud applications. The DM method uses both infrastructure- and application-level monitoring data to determine when to scale up or down, and its thresholds are dynamically adjusted based on workload conditions. The authors compare the performance of the DM method to seven existing autoscaling methods using synthetic and real-world workloads. They find that the DM method has better overall performance than the other methods, particularly in terms of response time and the number of instantiated containers. SWITCH system was used to implement the DM method for time-critical cloud applications.

In Rattihalli et al. [61], the authors propose a new resource management system called RUBAS that can dynamically adjust the allocation of containers running in a Kubernetes cluster. RUBAS incorporates container migration to improve upon the Kubernetes Vertical Pod Autoscaler (VPA) system non-disruptively. The authors evaluate RUBAS using multiple scientific benchmarks and compare its performance to Kubernetes VPA. They find that RUBAS improves CPU and memory utilization by 10% and reduces runtime by 15% with an overhead for each application ranging from 5–20%.

In Toka et al. [62], the authors present a Kubernetes scaling engine that uses machine learning forecast methods to make better autoscaling decisions for cloud-based applications. The engine's short-term evaluation loop allows it to adapt to changing request dynamics, and the authors introduce a compact management parameter for cloud tenants to easily set their desired level of resource over-provisioning vs. service level agreement (SLA) violations. The proposed engine is evaluated in simulations and with measurements on Web trace data, and the results show that it results in fewer lost requests and slightly more provisioned resources compared to the default Kubernetes baseline.

In Balla et al. [63], the authors propose an adaptive autoscaler called Libra, which automatically detects the optimal resource set for a single pod and manages the horizontal scaling process. Libra is also able to adapt the resource definition for the pod and adjust the horizontal scaling process if the load or underlying virtualized environment changes. The authors evaluate Libra in simulations and show that it can reduce the average CPU and memory utilization by up to 48% and 39%, respectively, compared to the default Kubernetes autoscaler.

In another work by Toka et al. [64], the authors propose a Kubernetes scaling engine that uses multiple AI-based forecast methods to make autoscaling decisions that are better suited to handle the variability of incoming requests. The authors also introduce a compact management parameter to help application providers easily set their desired resource over-provisioning and SLA violation trade-off. The proposed engine is evaluated in simulations and with measurements on web traces, showing improved fitting of provisioned resources to service demand.

In Wu et al., the authors propose a new active Kubernetes auto scaling device based on prediction of pod replicas. They demonstrate that their proposed autoscaler has a faster response speed compared to existing scaling strategies in Kubernetes.

In Wang et al. [65] the authors propose an improved automatic scaling scheme for Kubernetes that combines the advantages of different types of nodes in the scaling process. They found that their scheme improves the performance of the system under rapid load pressure and reduces instability within running clusters compared to the default auto scaler.

In Kang et al. [66], the authors propose a method for improving the reliability of virtual networks by using optimization models and heuristic algorithms to allocate virtual network functions (VNFs) to suitable locations. The authors also develop function scheduler plugins for the Kubernetes system, which allows for the automatic deployment and management of containerized applications. The proposed method is demonstrated to be effective in allocating functions and running service functions correctly. This work was published in the 2021 edition of the IEEE Conference on Decision and Control.

In Vu et al. [67], propose a hybrid autoscaling method for containerized applications that combines vertical and horizontal scaling capabilities to optimize resource utilization and ensure quality of service (QoS) requirements. The proposed method uses a predictive approach based on machine learning to forecast future demand and a

Table 4 Literature summary (Auto-Scaling Enabled Scheduling)

#	Objectives	Methodology/Algorithms	Experiments	Findings	Applications	Limitations
[60]	Create a multi-level dynamic autoscaling technique for containerized apps.	a dynamic multi-level (DM) autoscaling technique employing monitoring information from the infrastructure and applications.	Both simulated and actual workload settings.	The DM method outperforms other autoscaling techniques already in use.	DM method implementation for time-sensitive cloud applications in the SWITCH system.	Limited to the context of container-based cloud applications implemented in the SWITCH system.
[61]	Create a system that can flexibly change how many containers are operating in a Kubernetes cluster. To include container migration into the Kubernetes VPA system in a non-disruptive manner.	Resource Utilization Based Autoscaling System (RUBAS)	Multiple scientific benchmarks	RUBAS increases the cluster's CPU and memory consumption by 10% and decreases runtime by 15%, with a 5%–20% overhead for each application.	Dynamic allocation of containers running in a Kubernetes cluster	–
[62]	To make cloud-based applications' management easier and more effective To improve Kubernetes autoscaling decisions by adapting to actual variability of incoming requests	In a short-term assessment loop, various machine learning forecasting techniques compete with one another. Compact management parameter that application providers can use to find the ideal trade-off between resource over-provisioning and SLA violations.	Simulations and measurements on gathered Web traces were used to assess the scaling engine and management parameter.	In comparison to the default baseline, the multi-forecast scaling engine produces much fewer dropped requests with somewhat higher provided resources.	To improve Kubernetes autoscaling decisions in cloud-based applications	–
[63]	To develop an adaptive autoscaling algorithm for Kubernetes pods To automatically detect optimal resource set for pods and manage horizontal scaling process	Libra automatically determines the best resource combination for a single pod and updates resource description for the pod and the horizontal scaling procedure in the dynamic environment.	–	–	To improve scalability of Kubernetes pods	–
[64]	To develop an adaptive AI-based autoscaling system for Kubernetes that makes better scaling decisions and is easier for application providers to use	Various AI-based forecast methods Short-term evaluation loop	Simulations Collected web traces	The approach yields noticeably fewer dropped requests and slightly more supplied resources.	Kubernetes	–
[65]	Improve the efficiency of Kubernetes resource scaling.	Proposed Kubernetes autoscaler based on Pod replicas prediction.	Conducted experiments to verify the proposed autoscaler	Autoscaler had faster response speed	Can be used to improve resource scaling in Kubernetes	Further research and experimentation is to fully evaluate the proposed autoscaler.
[66]	Present a better automatic scaling plan for Kubernetes based on a variety of node types with pre-loaded images.	The suggested method incorporates the benefits of various node types in the scaling process.	The proposed scheme is tested and compared with the default auto scaler	The suggested approach decreases instability within the active clusters and enhances system performance under high load pressure.	The proposed scheme improved the performance and stability of the system under load pressure	Further testing and evaluation is needed to determine the full range of applications and limitations of the proposed scheme.
[67]	Address the diversity of 5G use cases with maximum flexibility and cost effectiveness. Improve network functions availability and resilience	Modular design for network functions. Statistical approach for modeling and resolution of resource allocation problem.	Used Kubernetes infrastructure hosting different network services	The suggested technique protects crucial operations while preventing resource limitation in cluster nodes.	Can be applied to Kubernetes infrastructure hosting network services to improve availability and resilience	Limited information provided on experimental setup and specific results obtained

burst identification module to make scaling decisions. The authors evaluate the proposed method and find that it improves response time and resource utilization compared to existing methods that only use a single scaling mode.

The above-mentioned schemes are summarized in Table 4.

Discussion, challenges & future suggestions

In Literature review section, a comprehensive review has been presented covering four sub-categories in the area of Kubernetes scheduling. It is crucial to provide a brief discussion on the categorized literature review that is presented in this section.

In the area of multi-objective optimization-based scheduling in Kubernetes, several research studies have been conducted to optimize various objectives such as minimizing the energy consumption and cost while maximizing resource utilization and meeting application performance requirements. These studies employ different optimization techniques such as genetic algorithms, particle swarm optimization, and ant colony optimization. Some studies also incorporate machine learning-based approaches to predict workload patterns and make scheduling decisions. There are still several challenges that need to be addressed. Firstly, the multi-objective nature of the problem poses a significant challenge in finding optimal solutions that balance conflicting objectives. Second, the dynamic nature of the cloud environment requires real-time adaptation of scheduling decisions to changing conditions. Overall, the research in multi-objective optimization-based scheduling in Kubernetes shows great potential in achieving efficient and effective resource management. Still, further work is needed to address the challenges and validate the effectiveness of these approaches in real-world scenarios.

On the other hand, AI-based scheduling in Kubernetes has been a popular area of research in recent years. Many studies have proposed different approaches to optimize scheduling decisions using machine learning and other AI techniques. One of the key accomplishments in this area is the development of scheduling algorithms that can handle complex workloads in a dynamic environment. These algorithms can consider various factors, such as resource availability, task dependencies, and application requirements, to make optimal scheduling decisions. Some studies have proposed reinforcement learning-based scheduling algorithms, which can adapt to changing workload patterns and learn from experience to improve scheduling decisions. Other studies have proposed deep learning-based approaches, which can capture complex patterns in the workload data and make accurate predictions. Overall, these studies have demonstrated that AI-based scheduling can improve the efficiency and performance of Kubernetes clusters. However, there are still some challenges that need to be addressed in this area. One of the main challenges is the lack of real-world datasets for training and evaluation of AI-based scheduling algorithms. Most studies use synthetic or simulated datasets, which may not reflect the complexities of real-world workloads. Another challenge is the trade-off between accuracy and computational complexity. Future research in this area could focus on developing more efficient and scalable AI-based scheduling algorithms that can handle large-scale, real-world workloads. This could involve exploring new machine learning and optimization techniques that can improve scheduling accuracy while reducing computational complexity.

Lastly, autoscaling enabled scheduling is an emerging research area that aims to optimize resource utilization and improve application performance by combining autoscaling and scheduling techniques. Several research studies have been published in this area in recent years. The analysis of these studies reveals that autoscaling enabled scheduling can lead to significant improvements in resource utilization and application performance. The studies have shown they can help reduce resource wastage, minimize the risk of under-provisioning, and improve application response times. However, despite these promising results, there are still some challenges that need to be addressed in this area. One of the main challenges is the complexity of designing effective autoscaling enabled scheduling algorithms. Developing algorithms that can adapt to dynamic workload changes and optimize resource utilization while maintaining application performance is a non-trivial task. Furthermore, there is a need for more research on the practical implementation of autoscaling enabled scheduling in real-world scenarios. Most of the existing studies have been conducted in controlled experimental settings, and there is a need to evaluate the effectiveness of auto scaling enabled scheduling in real-world applications. There are still several challenges that need to be addressed, including algorithm design, standardization, and practical implementation. Future research in this area should focus on addressing these challenges and developing more effective and practical auto scaling enabled scheduling techniques.

The research papers use diverse algorithms to enhance Kubernetes scheduling. These algorithms are tested on various platforms and environments, such as Spark, MXNet, Kubernetes, Google and TwoSigma's GPU cluster, workloads, Google compute, CPU-GPU, the National Cloud Infrastructure, benchmarks, ProCon, DL2, DRF, Optimus, CBP, PP, scaling, data centers, schedulers, CloudSim and Java, scenarios, cloud infrastructure, user need, RLSK, real trace, GaiaGPU and Tencent, real workload traces, simulations and web traces, Kubernetes, a new algorithm, Kubernetes failover and recovery, KubeHICE, real-world scenarios, BDI, BCDI, Kubernetes, a proposed algorithm, autoscalers, default auto scalers, video streaming, Tensorflow, Zeus, and latency-sensitive services. Some papers did not specify the details of the algorithms they used or the platforms and environments they tested on.

As can be seen in the previous sections, the survey extensively analyzes the current literature, and composes a taxonomy to not only effectively analyze the current state-of-the-art but also identify the challenges and future directions. Based on the analysis, the following areas have been identified as potential future research in the field:

- As Kubernetes becomes more popular, there will be a growing need for advanced computation optimization techniques. In the future, Kubernetes may benefit from the development of more sophisticated algorithms for workload scheduling and resource allocation, potentially using AI or machine learning. Additionally, integrating Kubernetes with emerging technologies like serverless computing could lead to even more efficient resource usage by enabling dynamic scaling without pre-provisioned infrastructure. Ultimately, the future of computation optimization in Kubernetes is likely to involve a combination of cutting-edge algorithms, innovative technologies, and ongoing advancements in cloud computing.
- Testing and implementation to reveal limitations or current learning algorithms for scheduling and potential improvements on large scale clusters. One important focus is on improving the tooling and automation around testing and deployment, including the development of new testing frameworks and the integration of existing tools into the Kubernetes ecosystem. Another key area is the ongoing refinement of Kubernetes' implementation and development process, with a focus on streamlining workflows, improving documentation, and fostering greater collaboration within the open-source community. Additionally, there is a growing emphasis on developing more comprehensive testing and validation strategies for Kubernetes clusters, including the use of advanced techniques like chaos engineering to simulate real-world failure scenarios. Overall, the future of testing and implementation in Kubernetes is likely to involve ongoing innovation, collaboration, and an ongoing commitment to driving the platform forward.
  A number of methods are employing learning algorithms for resource balancing inside and outside the cluster. Even though the methods given encouraging results, new learning algorithms can be found to improve the scheduler, especially on large scale clusters.
- Limitations and potential improvements in specific contexts, e.g., Green Computing. Minimizing the carbon footprint of a cluster is an ongoing challenge. Advanced schedulers are needed to be proposed in order to reduce the energy consumption and carbon footprint of clusters in IIoT setups. There is a huge opportunity for improving the existing methods and proposing new methods in this area.
- Future research in Kubernetes resource management. Kubernetes resource management mostly relies on optimization modelling framework and heuristic-based algorithms. The potential for improving and proposing new resource management algorithms is a very promising area of research. Future research in Kubernetes resource management may focus on addressing the challenges of managing complex, dynamic workloads across distributed, heterogeneous environments. This may involve developing more sophisticated algorithms and techniques for workload placement, resource allocation, and load balancing, as well as exploring new approaches to containerization and virtualization. Additionally, there may be opportunities to leverage emerging technologies like edge computing and 5G networks to enable more efficient and scalable resource management in Kubernetes.
- Most of the work done in the area of Kubernetes scheduling has been evaluated on small clusters. However, this might not always be tempting. One future research direction in Kubernetes scheduling is to use larger cluster sizes for algorithm evaluation. While Kubernetes has been shown to be effective in managing clusters of up to several thousand nodes, there is a need to evaluate its performance in even larger cluster sizes. This includes evaluating the scalability of the Kubernetes scheduler, identifying potential bottlenecks, and proposing solutions to address them. Additionally, there is a need to evaluate the impact of larger cluster sizes on application performance and resource utilization. This research could lead to the development of more efficient scheduling algorithms and better management strategies for large-scale Kubernetes deployments.
- Scheduling should not only be considered from the static infrastructure point of view, but rather advanced context-aware scheduling algorithms may be proposed that could focus on developing new approaches to resource allocation and scheduling that take into account a broader range of contextual

factors, such as user preferences, application dependencies, and environmental conditions. This may involve exploring new machine learning techniques and optimization algorithms that can dynamically adapt to changing conditions and prioritize resources based on real-time feedback and analysis. Other potential areas of research may include developing new models and frameworks for managing resources in Kubernetes clusters, improving container orchestration and load balancing, and enhancing monitoring and analytics capabilities to enable more effective use of context-aware scheduling algorithms.

As can be seen from the diversity of future directions, the potential for new research in Kubernetes is ripe with challenges of myriad levels of difficulty and effort. It provides future researchers with exciting opportunities to pursue and problems to tackle. We hope that this survey will facilitate future researchers in selecting a suitable challenge and solve new problems to expand the state-of-the-art in the area of Kubernetes.

Conclusions

In conclusion, the survey on Kubernetes scheduling provides a comprehensive overview of the current state of the field. It covers the objectives, methodologies, algorithms, experiments, and results of various research efforts in this area. The survey highlights the importance of scheduling in Kubernetes and the need for efficient and effective scheduling algorithms. The results of the experiments show that there is still room for improvement in this area, and future work should focus on developing new algorithms and improving existing ones. Overall, the survey provides valuable insight into the current state of Kubernetes scheduling and points to promising directions for future research.

Acknowledgements
None.

Authors' contributions
Research was supervised by Sohail Abbas and Naveed Ahmed. Data collection, material preparation, and analysis were performed by Khaldoun. All authors read and approved the final manuscript. Conceptualization and revisions done by Sohail Abbas, Naveed Ahmed and Atta ur Rehman.

Funding
The author(s) received no financial support for the research and publication of this article.

Availability of data and materials
The corresponding author may provide the supporting data on request.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Authors provide consent for publication.

Competing interests
The authors declare no competing interests.

Received: 27 January 2023 Accepted: 1 June 2023
Published online: 13 June 2023

References

1. Mondal SK, Pan R, Kabir HMD, Tian T, Dai HN (2022) Kubernetes in IT administration and serverless computing: an empirical study and research challenges. J Supercomput 78(2):2937–2987
2. Phuc LH, Phan LA, Kim T (2022) Traffic-Aware horizontal pod autoscaler in kubernetes-based edge computing infrastructure. IEEE Access 10:18966–18977
3. Zhang M, Cao J, Yang L, Zhang L, Sahni Y, Jiang S (2022) ENTS: An Edge-native Task Scheduling System for Collaborative Edge Computing. IEEE/ACM 7th Symposium on Edge Computing, SEC. pp 149–161
4. Kim SH, Kim T (2023) Local scheduling in kubeedge-based edge computing environment. Sensors 23(3):1522
5. E. Casalicchio (2019) "Container orchestration: A survey," Syst Model, 221–235.
6. Pahl C, Brogi A, Soldani J, Jamshidi P (2017) Cloud container technologies: a state-of-the-art review. IEEE Transact Cloud Comput 7(3):677–692
7. Rodriguez MA, Buyya R (2019) Container-based cluster orchestration systems: A taxonomy and future directions. Software Pract Experience 49(5):698–719
8. Truyen E, Van Landuyt D, Preuveneers D, Lagaisse B, Joosen W (2019) A comprehensive feature comparison study of open-source container orchestration frameworks. Appl Sciences (Switzerland) 9(5):931
9. Arunarani AR, Manjula D, Sugumaran V (2019) Task scheduling techniques in cloud computing: a literature survey. Futur Gener Comput Syst 91:407–415
10. Vijindra and S. Shenai, (2012) Survey on scheduling issues in cloud computing. Procedia Eng 38:2881–2888
11. Wang K, Zhou Q, Guo S, Luo J (2018) Cluster frameworks for efficient scheduling and resource allocation in data center networks: a survey. IEEE Commun Surveys Tutor 20(4):3560–3580
12. Hosseinioun P, Kheirabadi M, Kamel Tabbakh SR, Ghaemi R (2022) A task scheduling approaches in fog computing: a survey". Transact Emerg TelecommunTechnol 33(3):e3792
13. Rejiba Z, Chamanara J (2022) Custom scheduling in Kubernetes: a survey on common problems and solution approaches. ACM Comput Surv 55(7):1–37
14. Carrión C (2022) Kubernetes scheduling: taxonomy, ongoing issues and challenges. ACM Comput Surv 55(7):1–37
15. Santos J, Wauters T, Volckaert B, De Turck F (2019) Towards network-Aware resource provisioning in kubernetes for fog computing applications. Proceedings of the IEEE Conference on Network Softwarization: Unleashing the Power of Network Softwarization. pp 351–359
16. Chung A, Park JW, Ganger GR (2018) Stratus: Cost-aware container scheduling in the public cloud. Proceedings of the ACM Symposium on Cloud Computing. pp 121–134
17. Le TN, Sun X, Chowdhury M, Liu Z (2020) AlloX: Compute allocation in hybrid clusters. Proceedings of the 15th European Conference on Computer Systems, EuroSys
18. Zhong Z, Buyya R (2020) A Cost-Efficient Container Orchestration Strategy in Kubernetes-Based Cloud Computing Infrastructures with Heterogeneous Resources. ACM Trans Internet Technol 20(2):1–24
19. Thinakaran P, Gunasekaran JR, Sharma B, Kandemir MT, Das CR (2019) Kube-Knots: Resource Harvesting through Dynamic Container Orchestration in GPU-based Datacenters. Proceedings - IEEE International Conference on Cluster Computing, ICCC
20. Townend P et al (2019) Invited paper: Improving data center efficiency through holistic scheduling in kubernetes. Proceedings - 13th IEEE International Conference on Service-Oriented System Engineering, 10th

International Workshop on Joint Cloud Computing, and IEEE International Workshop on Cloud Computing in Robotic Systems, CCRS. pp 156–166
21. Menouer T (2021) KCSS: Kubernetes container scheduling strategy. J Supercomput 77(5):4267–4293
22. Song S, Deng L, Gong J, Luo H (2019) Gaia scheduler: A kubernetes-based scheduler framework. 16th IEEE International Symposium on Parallel and Distributed Processing with Applications, 17th IEEE International Conference on Ubiquitous Computing and Communications, 8th IEEE International Conference on Big Data and Cloud Computing. pp 252–259
23. Ogbuachi MC, Gore C, Reale A, Suskovics P, Kovacs B (2019) Context-aware K8S scheduler for real time distributed 5G edge computing applications. 27th International Conference on Software, Telecommunications and Computer Networks, SoftCOM
24. Beltre A, Saha P, Govindaraju M (2019) KubeSphere: An approach to multi-tenant fair scheduling for kubernetes clusters. 3rd IEEE International Conference on Cloud and Fog Computing Technologies and Applications, Cloud Summit. pp 14–20
25. Haja D, Szalay M, Sonkoly B, Pongracz G, Toka L (2019) Sharpening Kubernetes for the Edge. ACM SIGCOMM Conference Posters and Demos, Part of SIGCOMM. pp 136–137
26. Wojciechowski L et al (2021) NetMARKS: Network metrics-AwaRe kubernetes scheduler powered by service mesh. Proceedings - IEEE INFOCOM
27. Cai Z, Buyya R (2022) Inverse Queuing Model-Based Feedback Control for Elastic Container Provisioning of Web Systems in Kubernetes. IEEE Trans Comput 71(2):337–348
28. El Haj Ahmed G, Gil-Castiñeira F, Costa-Montenegro E (2021) KubCG: A dynamic Kubernetes scheduler for heterogeneous clusters. Software Pract Experience 51(2):213–234
29. Ungureanu OM, Vlădeanu C, Kooij R (2019) Kubernetes cluster optimization using hybrid shared-state scheduling framework. ACM International Conference Proceeding Series
30. Yang S, Ren Y, Zhang J, Guan J, Li B (2021) KubeHICE: Performance-aware Container Orchestration on Heterogeneous-ISA Architectures in Cloud-Edge Platforms. 19th IEEE International Symposium on Parallel and Distributed Processing with Applications, 11th IEEE International Conference on Big Data and Cloud Computing, 14th IEEE International Conference on Social Computing and Networking and 11th IEEE Internation. pp 81–91
31. Li D, Wei Y, Zeng B (2020) A Dynamic I/O Sensing Scheduling Scheme in Kubernetes. ACM International Conference Proceeding Series. pp 14–19
32. Fan D, He D (2020) A Scheduler for Serverless Framework base on Kubernetes. ACM International Conference Proceeding Series. pp 229–232
33. Bestari MF, Kistijantoro AI, Sasmita AB (2020) Dynamic Resource Scheduler for Distributed Deep Learning Training in Kubernetes. 7th International Conference on Advanced Informatics: Concepts, Theory and Applications, ICAICTA
34. Dua A, Randive S, Agarwal A, Kumar N (2020) Efficient Load balancing to serve Heterogeneous Requests in Clustered Systems using Kubernetes. IEEE 17th Annual Consumer Communications and Networking Conference, CCNC
35. Kaur K, Garg S, Kaddoum G, Ahmed SH, Atiquzzaman M (2020) KEIDS: Kubernetes-Based Energy and Interference Driven Scheduler for Industrial IoT in Edge-Cloud Ecosystem. IEEE Internet Things J 7(5):4228–4237
36. Lin M, Xi J, Bai W, Wu J (2019) Ant colony algorithm for multi-objective optimization of container-based microservice scheduling in cloud. IEEE Access 7:83088–83100
37. Wei-guo Z, Xi-lin M, Jin-zhong Z (2018) Research on kubernetes' resource scheduling scheme. ACM International Conference Proceeding Series
38. Oleghe O (2021) Container placement and migration in edge computing: concept and scheduling models. IEEE Access 9:68028–68043
39. Carvalho M, MacEdo DF (2021) QoE-Aware Container Scheduler for Co-located Cloud Environments," Faculdades Catolicas
40. Chen T, Li M, Li Y, Lin M, Wang N, Wang M, Xiao T, Xu B, Zhang C, Zhang Z (2015) Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. arXiv preprint arXiv:1512.01274
41. Abadi M et al (2016) Tensorflow: a system for large-scale machine learning. Osdi 2016(16):265–283
42. Xing EP et al (2015) Petuum: A new platform for distributed machine learning on big data. Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp 1335–1344
43. Verma A, Pedrosa L, Korupolu M, Oppenheimer D, Tune E, Wilkes J (2015) Large-scale cluster management at Google with Borg. 10th European Conference on Computer Systems, EuroSys. pp 1–15
44. Vavilapalli VK et al (2013) Apache hadoop YARN: Yet another resource negotiator. 4th Annual Symposium on Cloud Computing, SoCC. pp 1–16
45. Bao Y, Peng Y, Wu C, Li Z (2018) Online Job Scheduling in Distributed Machine Learning Clusters. Proceedings - IEEE INFOCOM. pp 495–503
46. Peng Y, Bao Y, Chen Y, Wu C, Guo C (2018) Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters. Proceedings of the 13th EuroSys Conference, EuroSys
47. Mao H, Schwarzkopf M, Venkatakrishnan SB, Meng Z, Alizadeh M (2019) Learning scheduling algorithms for data processing clusters. SIGCOMM Conference of the ACM Special Interest Group on Data Communication. pp 270–288
48. Chaudhary S, Ramjee R, Sivathanu M, Kwatra N, Viswanatha S (2020) Balancing efficiency and fairness in heterogeneous GPU clusters for deep learning. Proceedings of the 15th European Conference on Computer Systems, EuroSys
49. Fu Y et al (2019) Progress-based Container Scheduling for Short-lived Applications in a Kubernetes Cluster. IEEE International Conference on Big Data, Big Data. pp 278–287
50. Peng Y, Bao Y, Chen Y, Wu C, Meng C, Lin W (2021) DL2: A Deep Learning-Driven Scheduler for Deep Learning Clusters. IEEE Trans Parallel Distrib Syst 32(8):1947–1960
51. Mao Y, Fu Y, Zheng W, Cheng L, Liu Q, Tao D (2022) Speculative Container Scheduling for Deep Learning Applications in a Kubernetes Cluster. IEEE Syst J 16(3):3770–3781
52. Huang J, Xiao C, Wu W (2020) RLSK: A Job Scheduler for Federated Kubernetes Clusters based on Reinforcement Learning. IEEE International Conference on Cloud Engineering, IC2E. pp 116–123
53. Wang H, Liu Z, Shen H (2020) Job scheduling for large-scale machine learning clusters. Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies. pp 108–120
54. Han Y, Shen S, Wang X, Wang S, Leung VCM (2021) Tailored learning-based scheduling for kubernetes-oriented edge-cloud system. Proceedings - IEEE INFOCOM
55. Casquero O, Armentia A, Sarachaga I, Pérez F, Orive D, Marcos M (2019) Distributed scheduling in Kubernetes based on MAS for Fog-in-the-loop applications. IEEE International Conference on Emerging Technologies and Factory Automation, ETFA. pp 1213–1217
56. Yang Y, Chen L (2019) Design of Kubernetes Scheduling Strategy Based on LSTM and Grey Model. Proceedings of IEEE 14th International Conference on Intelligent Systems and Knowledge Engineering, ISKE. pp 701–707
57. Zhang X, Li L, Wang Y, Chen E, Shou L (2021) Zeus: Improving Resource Efficiency via Workload Colocation for Massive Kubernetes Clusters. IEEE Access 9:105192–105204
58. Liu Z, Chen C, Li J, Cheng Y, Kou Y, Zhang D (2022) KubFBS: A fine-grained and balance-aware scheduling system for deep learning tasks based on kubernetes. Concurrency Computat Pract Exper 34(11):e6836. https://doi.org/10.1002/cpe.6836
59. Rahali M, Phan CT, Rubino G (2021) KRS: Kubernetes Resource Scheduler for resilient NFV networks. IEEE Global Communications Conference
60. Taherizadeh S, Stankovski V (2019) Dynamic multi-level auto-scaling rules for containerized applications. Computer J 62(2):174–197
61. Rattihalli G, Govindaraju M, Lu H, Tiwari D (2019) Exploring potential for non-disruptive vertical auto scaling and resource estimation in kubernetes. IEEE International Conference on Cloud Computing, CLOUD. pp 33–40
62. Toka L, Dobreff G, Fodor B, Sonkoly B (2021) Machine Learning-Based Scaling Management for Kubernetes Edge Clusters. IEEE Trans Netw Serv Manage 18(1):958–972
63. Balla D, Simon C, Maliosz M (2020) Adaptive scaling of Kubernetes pods. IEEE/IFIP Network Operations and Management Symposium 2020: Management in the Age of Softwarization and Artificial Intelligence, NOMS

64. Toka L, Dobreff G, Fodor B, Sonkoly B (2020) Adaptive AI-based auto-scaling for Kubernetes. IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGRID. pp 599–608
65. Wang M, Zhang D, Wu B (2020) A Cluster Autoscaler Based on Multiple Node Types in Kubernetes. IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference, ITNEC. pp 575–579
66. Kang R, Zhu M, He F, Sato T, Oki E (2021) Design of Scheduler Plugins for Reliable Function Allocation in Kubernetes. 17th International Conference on the Design of Reliable Communication Networks, DRCN
67. Vu DD, Tran MN, Kim Y (2022) Predictive hybrid autoscaling for containerized applications. IEEE Access 10:109768–109778

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Submit your manuscript to a SpringerOpen journal and benefit from:

► Convenient online submission
► Rigorous peer review
► Open access: articles freely available online
► High visibility within the field
► Retaining the copyright to your article

Submit your next manuscript at ► springeropen.com