An Efficient Scheduling Strategy for Containers Based on Kubernetes

Xurong Zhang[1], Xiaofeng Wang[1,2(✉)], Yuan Liu[1], and Zhaohong Deng[1]

[1] School of Artificial Intelligence and Computer Science, Jiangnan University, Wuxi, China
wangxf@jiangnan.edu.cn
[2] Peng Cheng Laboratory, Shenzhen, China

Abstract. Container clouds are an important supporting technology for collaborative edge computing, and Kubernetes has become the de facto standard for container orchestration. To solve the problem that the scheduling mechanism of Kubernetes has a single scheduling resource index and is unable to adapt the refined resource scheduling requirements in collaborative edge computing, this paper proposes an efficient multicriteria container online scheduling strategy based on Kubernetes, named E-KCSS. To improve the resource utilization of the cluster, the proposed E-KCSS strategy takes into account the global view of edge nodes and containers. An adaptive weight mechanism based on real-time utilization is proposed to solve the problem that preset Kubernetes weighting coefficients do not meet the individual resource requirements of applications. The experimental results show that compared with the scheduling mechanism of Kubernetes, the deployment efficiency of E-KCSS is improved by 35.22%, the upper limit of container application deployment is increased by 29.82%, and the cluster resource imbalance is reduced by 6.87%, which can make the multi-dimensional resource utilization of the cluster more balanced.

Keywords: Collaborative edge computing · Kubernetes · Container online scheduling strategy · Adaptive weight mechanism · Resource utilization

1 Introduction

With the large growth of intelligent terminal data traffic, edge computing can significantly improve computing performance by enabling cloud computing services to be deployed at the edge of the network to alleviate the great pressure of data [1]. Edge computing can decompose large-scale services into smaller and easier to manage parts, which enables real-time processing and analysis of data at the source of data generation. It is considered an effective distributed computing architecture [2].

However, with the emerging applications of computation-intensive and delay-sensitive [3] requirements, the limited computing power of edge nodes has been greatly challenged. Collaborative edge computing uses different edge nodes to collaborate by sharing computing resources and data, which provides a low time delay and highly reliable computing services [4]. In the collaborative edge computing scenario, computing

H. Gao et al. (Eds.): CollaborateCom 2022, LNICST 460, pp. 326–342, 2022.
https://doi.org/10.1007/978-3-031-24383-7_18

tasks are scheduled to edge nodes to fully use the advantages of edge nodes in computing and storage and reduce the network burden of cloud data centres.

In recent years, relevant research has further explored the fusion of containers and collaborative edge computing [5–8]. Most studies sink the container to the edge node and use its features of second-level deployment, easy portability and elastic scaling [7] to perform task computing. The fusion of containers and collaborative edge computing considers the collaboration among edge nodes to achieve mobile container scheduling. The lightweight nature of containers enables the cluster to quickly deploy application instances when needed, which provides technical support for scheduling the computing power of edge nodes by flexibly adjusting the number of service applications [8].

Before the public release of Kubernetes, it was difficult to jointly develop complex collaborative computing systems due to hardware differences between computing infrastructures [8]. However, since Kubernetes provides a declarative interface for upwards services, it can use software virtualization to abstract resources and provide unified objects to the outside world [9], which perfectly fits with the collaborative edge computing scenario.

The Kubernetes resource scheduler plays a decisive role in cluster and resource utilization and is the core of the container cloud. However, it faces the problems of a single scheduling mechanism and fixed resource scheduling weight and it cannot ensure that the computing power of edge nodes will be available on demand. Therefore, it is necessary to explore a new container scheduling mechanism for collaborative edge computing according to the container characteristics, system state and optimization objectives.

To overcome the shortcomings of Kubernetes scheduling in collaborative edge computing scenarios, an efficient scheduling strategy for containers based on Kubernetes, named E-KCSS, is presented. The novelty of E-KCSS lies in the introduction of a multicriteria scheduling strategy and an adaptive weight mechanism for resources. This strategy is considered at the node and container levels to achieve the load balancing of the cluster.

In summary, our main contributions are as follows:

1. A multicriteria container scheduling strategy for Kubernetes for collaborative edge computing is proposed, which selects nodes with a good compromise among multiple criteria for each container.
2. An adaptive weight algorithm is proposed, which can automatically model and solve the container resource weight set according to the dynamically obtained multidimensional resource utilization of edge nodes.
3. Based on the global node load balancing of the cluster as the objective function, the scheduling of a set of containers submitted online is optimized considering the multi-dimensional resource idleness and resource imbalance of the nodes.

The remainder of this article is organized as follows. In Sect. 2, some related works are introduced. Section 3 describes the proposed E-KCSS architecture and key technologies of E-KCSS in detail. Section 4 presents the exhaustive experiments and analysis to validate the E-KCSS method. Finally, Sect. 5 summarizes the full text and describes future research.

2 Related Work

The resource scheduling module in the container cloud platform plays a decisive role in the cluster performance [9]. There have been considerable studies on scheduling strategies for container technology, which mainly focus on collaborative computing, improving system resource utilization, achieving system load balancing, and predicting models based on machine learning.

Reference [4] proposes a dynamic offloading strategy based on a probabilistic evolutionary game-theoretic model. The mechanism comprehensively considers computing power and task requirements, establishes a resource consumption model and a time computing model, and uses a greedy algorithm to determine the optimal scheduling node of the container in the mobile edge computing scenario. Reference [10] proposes a cloud container-based collaboration framework that performs dynamic resource provisioning according to different workloads in collaborative computing scenarios, which outperforms virtual machine-based systems in terms of completion time and throughput.

The authors of [11] perform network prioritization of containers by introducing a quality of service mechanism to provide priority delivery services for containers. It can effectively determine the scheduling and allocation of bandwidth-sensitive containers and reduce operating costs. In [12], the authors analyse the correlation between absolute and relative CPU utilization under different workload scenarios and select the most appropriate performance metric for automatic scaling control while ensuring service quality constraints. This method can reduce the energy consumption of the cloud infrastructure platform and maximize the CPU utilization. Reference [13] proposes an I/O scheduling strategy at the cluster and node levels for the contention of I/O shared resources by containers. The disk usage is collected in real time at the cluster level, and the container priority is set at the node level, to achieve node-level throttling and cluster-level load balancing. References [11–13] are optimized based on single-resource scheduling, which plays an optimization role when the container focuses on such resources.

Reference [14] investigates and introduces a scheduling mechanism based on the resource utilization rate for index weight self-learning, which calculates the resource weight set of containers through dynamic perception of the real-time resource utilization rate, and improves the resource utilization in the container cloud environment. Reference [15] proposes a container scheduling method based on the multi-dimensional resource idle rate to achieve OpenShift cluster load balancing. It mathematically models the factors that affect scheduling, automatically solves the dynamic weights of multi-dimensional resources required by the container, and ensures the efficient use of node resources. References [14, 15] aim to improve the utilization of system resources, and they simulate the container cloud environment on the ContainerCloudSim [16] simulator to verify the scheduling strategy.

With the development of artificial intelligence, some predictive algorithms based on machine learning have emerged in the field of container scheduling. The authors of [17] consider the threshold distance of container replicas, use a genetic algorithm to address the problems of container scheduling and automatic scalability of containers, optimize the uniform distribution of microservices, enhance system resource allocation, and reduce network overhead. Reference [18] proposes a microservice container scheduling strategy based on a multi-objective ant colony optimization algorithm from

the four dimensions of node computing resource utilization, storage resource utilization, number of microservice requests and failure rate and improves the cluster service reliability and resource utilization. Reference [19] proposes a forecasting model based on time series, which collects the historical resource usage of nodes, and predicts future resource usage through the forecasting model. However, scheduling strategies based on heuristic algorithms and machine learning predictive algorithms must learn a large number of data sets in advance, which will increase the difficulty of practical application. Based on these studies, there is less research on container scheduling in container clouds. The traditional scheduling method cannot fully consider the characteristics of containers, which often leads to low resource utilization and load imbalance in container clouds. More research should consider both.

3 E-KCSS Scheduling Strategy

In this section, we propose a new Kubernetes multicriteria container scheduling strategy, E-KCSS, to optimize the container scheduling performance in edge collaboration scenarios. This strategy considers the differences in multi-dimensional resource utilization on edge nodes and individual requirements of containers and combines the adaptive weight mechanism to perform load balancing of cluster resources.

3.1 E-KCSS Architecture

In Kubernetes, the node is the carrier to execute tasks, and the pod is the smallest unit of scheduling. The scheduler plays an important role in linking the preceding and the following. It is responsible for receiving requests to create pods, and it notifies the service process to manage the life cycle of the pod.

As shown in Fig. 1, the E-KCSS architecture is divided into two independent control loops: the Informer Path and Scheduling Path. The Informer Path starts a series of informers, monitors the changes to the multi-dimensional resources in the database, and adds the pods to be scheduled to the scheduling priority queue. The main logic of the Scheduling Path is to constantly pull pods out of the scheduling priority queue. First, the Predicates strategy is called to obtain a list of all nodes that meet pod requirements. Then, taking the responsible balance of the cluster as the objective function, the nodes are prioritized through the Priorities strategy, and the optimal node is selected. In addition, the scheduling part of E-KCSS is responsible for updating the scheduler cache and caching as much cluster information as possible to fundamentally improve the execution efficiency of the Predicates strategy and Priorities strategy.

The Scheduling mechanism is the key point of the Scheduling Path. It mainly introduces multicriteria indicators, combines the adaptive weight mechanism for resources, and takes the load balance of the cluster nodes as the objective function to select the optimal node for a pod. First, Sect. 3.2 introduces the Kubernetes multicriteria metrics. Then, Sect. 3.3 introduces the adaptive weight mechanism. Finally, Sect. 3.4 takes the global node load balancing of the cluster as the objective function to optimize the container scheduling performance.

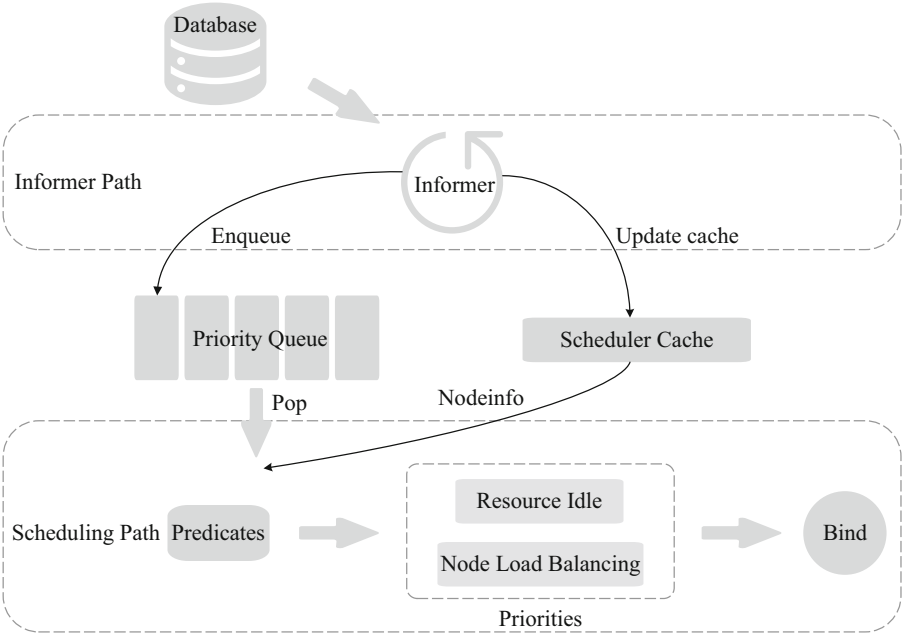

Fig. 1. E-KCSS architecture

3.2 Multicriteria Indicators

The scheduling algorithm of Kubernetes only considers the CPU and memory, and does not consider the needs of pod applications for other resources. Reference [14] adds two evaluation indicators, bandwidth and disk utilization, to the scheduler to avoid deploying containers to nodes that cannot adapt to edge computing scenarios. However, this strategy deploys many pods to a certain node. Therefore, E-KCSS additionally considers the number of pods deployed. Pods are added to the E-KCSS consideration to more evenly deploy containers to different nodes of the cluster.

Because edge nodes are limited by the deployment environment, network edge storage, computing, bandwidth and other resources are very scarce. According to the characteristics of the container, E-KCSS introduces multicriteria indicators, which additionally take the bandwidth, disk utilization, and number of deployed pods as the basis for scoring, to avoid deploying pod applications to nodes in which bandwidth, disk, and pod resources are saturated. Another aspect of E-KCSS considering bandwidth is to pull container images from container repositories faster. The transmission speed of the image directly affects the normal operation of the pod application and startup speed of the business. E-KCSS considers the disk factor to more reasonably schedule disk-intensive containers.

E-KCSS integrates a monitoring component to monitor resource changes on each node, collects the usage and total amount of resources in each dimension on each node, and calculates the idle utilization of various resources on the node. Assume that the number of filtered Kubernetes working nodes to satisfy the pod's requirement is K; this

is expressed as $M = (m_1, m_2, \cdots, m_k)$. The nodes in the cluster consider resources in N dimensions. In E-KCSS, there are five types of resources: CPU, memory, bandwidth, disk utilization, and number of pods, so $N = 5$. The total resources of the nodes in the cluster are expressed as $R = \left(r_1^n, r_2^n, \cdots, r_k^n\right), \forall n \in N$, where r_k^n is the total number of n th-dimension resources owned by node k. The resource usage of the node is expressed as $U = \left(u_1^n, u_2^n, \cdots, u_k^n\right), \forall n \in N$, where u_k^n is the usage of the n th-dimension resource owned by node k. Therefore, the idle rate of the n th-dimension resource on node k $Free_k^n$ is expressed as

$$Free_k^n = \left(1 - \frac{u_k^n}{r_k^n}\right), \forall n \in N \tag{1}$$

3.3 Adaptive Weight Mechanism

When Kubernetes deploys pod applications, because the importance of pods to multidimensional resource requirements often varies, a scheduling policy using a fixed weight coefficient cannot satisfy the individual requirements of pod applications. Reference [15] adopts the fuzzy analytic hierarchy process to automatically model and solve the multi-dimensional resource weight parameters of container applications. Although the utilization of resources and cluster performance improve, this is more related to weight prediction at the subjective level. Based on the above research, E-KCSS obtains the dynamic weight coefficient triggered by dynamic events. E-KCSS automatically generates multi-dimensional resource weights according to the specific resource requirements of pod applications and combines the weights with the idle resource rates of the cluster nodes to calculate the scores of candidate nodes. This idea considers the actual situation of edge nodes and actual needs of pod applications, and their combination is applied to E-KCSS.

E-KCSS calculates the dynamic weight of a pod according to the proportion of the resources of the pod to the CPU, memory, bandwidth, disk, and pod number in the corresponding total resources of the cluster. The total amounts of various resources on the node are collected through the monitoring module, and the total resource R_n on each node is calculated.

$$R_n = \left(r_1^n, r_2^n, \cdots, r_k^n\right), \forall n \in N \tag{2}$$

where R_n is the n th-dimensional resource set of each node in the Kubernetes cluster and r_k^n is the total amount of n th-dimensional resources owned by node k. Therefore, the total resource S_n of the n th dimension of the cluster is expressed as

$$S_n = \sum\nolimits_{i=1}^{k} r_i^n \tag{3}$$

The mathematical model is built according to the resource requirements of the pod, and the pod application container records the set of resource requirements as $P = (p_1, p_2, \cdots, p_n)$. Therefore, the proportion w_n of the resource demand of the n th-dimension of the pod with respect to the total global resources can be obtained by

combining formula (3) to obtain

$$w_n = \frac{P_n}{S_n} = \frac{P_n}{\sum_{i=1}^{k} r_i^n}, \forall n \in N \tag{4}$$

Therefore, the set of resources required by the pod application in the overall resource weight of the cluster is set to $Weight = (w_1, w_2, \cdots, w_n)$. Thus, deploying the pod application consumes w_1 CPU resources, w_2 memory resources, w_3 bandwidth resources, w_4 disk resources, and w_5 pod quantity resources in the cluster.

Finally, the weight set is normalized. Define the weight coefficients $\beta_i, \forall i \in [1, 5]$, which represent the weights of the CPU, memory, bandwidth, disk, and pod number after normalization, respectively; $\beta_1 + \beta_2 + \beta_3 + \beta_4 + \beta_5 = 1$, where β_i is expressed as

$$\beta_i = \frac{w_i}{\sum_{j=1}^{N} w_j}, \forall i \in N \tag{5}$$

The adaptive weight algorithm is shown in Algorithm 1. The input of Algorithm 1 is all candidate node information (nodeInfo), pod application information (PodInfo) and dimension resource name sets (resourceSet). The output is the weight vector (weightVector) of the pod, which is used to save the dynamic weight coefficient.

Algorithm 1. Adaptive Weight Algorithm

Input: *nodeInfo*, *PodInfo*, *resourceSet*;
Output: *weightVector*;
1. *weightVector*, *PodVector*, *nodeVector*, *Sum* ← {};
2. **for** *Pod* **in** *PodInfo* **do**:
3. *PodVector* ← Get the resource demand the of pod from *PodInfo*;
4. **end for**
5. **for** *node* **in** *nodeInfo* **do**:
6. *nodeVector* ← Get the total resources of the node from *nodeInfo*;
7. *Sum* += *nodeVector*;
8. **end for**
9. *weight* ← According to *Sum* and *PodVector*, calculate the proportion of resources required by pod to the overall global resources, and judge the tendency of the pod;
10. *weightVector* ← Normalize *weight*;
11. **return** *weightVector*

3.4 Load Balancing Strategy

The load balancing strategy of E-KCSS is jointly determined by the node resource idle module and multi-dimensional resource balancing module. The node resource idle module implies that a higher idle resource rate of the candidate node corresponds to a higher score; the multi-dimensional resource balance module implies that a more balanced resource utilization of the candidate node corresponds to a higher score. The final score of the candidate node is determined by the weighted sum of the two scores.

E-KCSS calculates the n th-dimension resource idleness on node K as $Free_k^n$ based on formula (1). According to formula (5), the pod resource weight set $W = (\beta_1, \beta_2, \cdots, \beta_n)$ is obtained. Therefore, the idle resource score S_k of the pod deployed on node k is expressed as

$$S_k = \sum_{i=1}^{n} Free_k^n \times \beta_i \tag{6}$$

Therefore, scoring set S of the idle resource degree of the cluster nodes is

$$S = \left(S_1, S_2, \cdots, S_j\right), \forall j \in K \tag{7}$$

E-KCSS obtains the multi-dimensional resource utilization set on the candidate node k according to formula (1), which is denoted as $U_k = \left(u_k^1, u_k^2, \cdots, u_k^n\right), \forall n \in N$. Therefore, the average resource utilization U_{avg} on candidate node k is expressed as

$$U_k^{avg} = \sum_{i=1}^{N} u_k^i \tag{8}$$

In probability theory, variance is often used to measure the degree of deviation between a random variable and the mean; it describes the distribution range of variable values. A larger variance corresponds to a larger data fluctuation, more unbalanced resources of the cluster, and a lower node score. Therefore, the multi-dimensional resource balance score B_k on node k is expressed as

$$B_k = \sqrt{\frac{1}{n}\sum_{i=1}^{N}\left(u_k^i - u_k^{avg}\right)^2} \tag{9}$$

To obtain B_k, after amplification, score set B of the multi-dimensional resource balance degree of the cluster nodes is expressed as

$$B = \left(B_1, B_2, \cdots, B_j\right), \forall j < K \tag{10}$$

Therefore, according to formula (7) and formula (10), the comprehensive score of candidate node k is expressed as $f_k = S_k + B_k$. Therefore, the comprehensive score set F of cluster candidate nodes is expressed as

$$F = \left(f_1, f_2, \cdots, f_j\right), \forall j < K \tag{11}$$

4 Experimental Evaluation

In this section, we verify the analysis of E-KCSS through the following experiments: 1) comparison of cluster resource imbalance, 2) comparison of deployment efficiency, and 3) comparison of resource utilization.

Table 1. Kubernetes node information.

Node number	CPU/core	Memory/GB	Disk/GB	Bandwidth/Mbps
Master1	4	8	200	1000
Node1	8	8	1000	1000
Node2	8	12	1000	1000
Node3	16	24	1000	1000
Node4	16	32	1700	1000
Node5	12	64	1100	1000

4.1 Experimental Environment

To verify the validity and feasibility of the proposed E-KCSS, a Kubernetes cluster was built, including one master node and five worker nodes. The total resource information of each node is shown in Table 1.

Simultaneously, T pod applications for scheduling are constructed, and their resource requirements simulate four different resource-intensive applications in terms of the CPU, memory, bandwidth and disk in the cloud computing platform. The resource specifications are shown in Table 2.

Table 2. Pod application resource specifications.

Number	CPU/MHz	Memory/MB	Disk/GB	Bandwidth/Mbps	Pod
1	400	230	8	20	1
2	100	700	2	10	1
3	150	100	20	20	1
4	75	175	7	10	1
…	…	…	…	…	…
T	200	100	5	80	1

The normalized multi-dimensional resource weights of the T pods in Table 2 are solved according to the adaptive weight mechanism, and the corresponding multi-dimensional weight parameters are shown in Table 3.

Table 3. Pod multi-dimensional resource weight parameters.

Number	CPU/MHz	Memory/MB	Disk/GB	Bandwidth/Mbps	Pod
1	0.517	0.128	0.124	0.154	0.077
2	0.181	0.552	0.045	0.111	0.111
3	0.245	0.070	0.392	0.196	0.097
4	0.209	0.209	0.206	0.209	0.167
…	…	…	…	…	…
T	0.238	0.051	0.071	0.569	0.071

4.2 Performance Indicators

There are three main performance indicators.
1) Cluster resource imbalance

For nodes, the standard deviation of multi-dimensional resource utilization can reflect the resource balance of nodes. We calculate the average utilization rate U_k^{avg} of each resource dimension for node k according to formula (1), which is expressed as

$$U_k^{avg} = \frac{1}{N}\sum_{i=1}^{N}(1 - Free_k^i) \tag{12}$$

Therefore, the standard deviation of the node k resource utilization SD_k is expressed as

$$SD_k = \sqrt{\sum_{i=1}^{N}(1 - Free_k^i - U_k^{avg})^2} \tag{13}$$

Define the cluster resource imbalance degree $IBD = \frac{1}{K}\sum_{i=1}^{K} SD_i$. A smaller value corresponds to less cluster resource imbalance, more balanced overall resource utilization, and a higher load balance.

2) Deployment efficiency

The pod deployment time is the time from when the control node issues the deployment command until the pod becomes available. A shorter pod deployment time corresponds to a higher deployment efficiency.
3) Resource utilization of each dimension

The resource utilization of each dimension on the cluster node measures whether the resources of the node are overloaded. A smaller probability of node resource tilt corresponds to better scheduling performance.

4.3 Comparison of Cluster Resource Imbalance

In the experiment, the Kubernetes scheduler and scheduler based on the E-KCSS mechanism were used to maximally deploy pod applications. To fully use cluster resources as

much as possible, the application with pod number 4 in Table 2 was selected for scheduling. The experiment deployed as many pod applications as possible in order, where each round lasted 60 s, and recorded the CPU, memory, bandwidth, and disk utilization of each node and the proportion of the number of deployed pods to the total number of pods on the node. The imbalance degree of cluster resources was calculated according to formula (12) and formula (13), and the experimental results are shown in Fig. 2.

After E-KCSS scheduling, the cluster resource imbalance degree is better than that of the Kubernetes scheduling strategy on the whole, with an average reduction of 6.87%. The Kubernetes scheduler node failed in the 173rd round; the pod application could not be deployed on the node, and 285 pods were deployed in total. The E-KCSS scheduler counted 253 rounds of information and deployed a total of 370 pods, which is 29.82% greater than the number scheduled by the Kubernetes scheduler. E-KCSS maximized the use of multi-dimensional cluster resources, which greatly improved the utilization of resources. The reason is that E-KCSS considers the weights of the four resources and the number of node pod deployments at the pod and node levels, which effectively reduces the possibility that one node in a cluster is exhausted and other resources are too often unused. E-KCSS also takes the load balance of the cluster nodes as the objective function, which combines the idle rate of multi-dimensional resources and the resource imbalance between nodes. As a result, the imbalance of cluster resources is reduced to satisfy the deployment requirements of more pod applications.

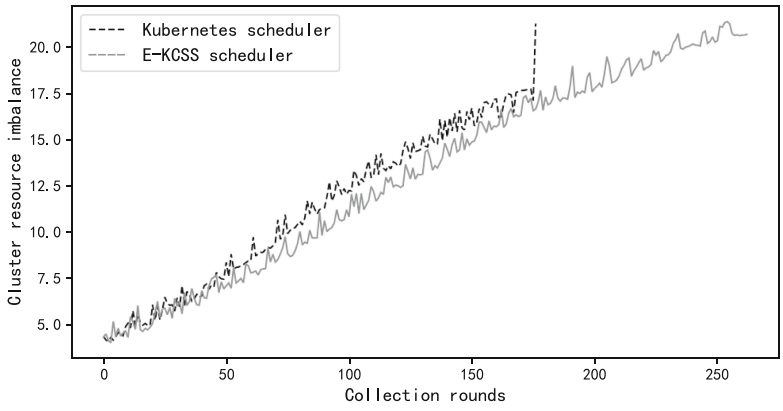

Fig. 2. Comparison of cluster resource imbalance

4.4 Comparison of Deployment Efficiency

In the experiment, the Kubernetes scheduler and E-KCSS scheduler were used to deploy pod applications, and the total time required to deploy 20 pods, deployment time of each pod, and network load rate before deploying each pod were compared. The method to determine the total time consumption of pod deployment in the experiment took the deployment request time of the first pod as the start time; the remaining pods were

deployed at intervals of 10 s, and the time at which all pods became available was considered the completion time.

The experimental results are shown in Fig. 3. Figure 3(a) shows the total time taken from the creation command delivery to the pod successful creation for each pod. Figure 3(b) shows the network load rate before the node creates the pod. From the experimental result calculations, the Kubernetes scheduler takes 521 s to deploy 20 pods and 157.7 s to deploy one pod on average. E-KCSS takes 345 s to deploy 20 pods, and it takes 102.15 s to deploy one pod on average, a decrease of 0.35 times in deployment time. Figure 3(b) shows that the network load rate of the node selected by E-KCSS before creating the pod is lower than that of the node selected by the Kubernetes scheduler. In contrast to the Kubernetes scheduling policy, the improved E-KCSS takes the network load rate of the nodes as a reference factor for scheduling during the scheduling process, and preferentially selects nodes with a low network load rate for scheduling. Therefore, the efficiency of the container image distribution will be improved and reduce the speed of pod deployment. In summary, E-KCSS determines the network bandwidth utilization of nodes; it has higher deployment efficiency than the Kubernetes scheduler and can rapidly deploy containers.

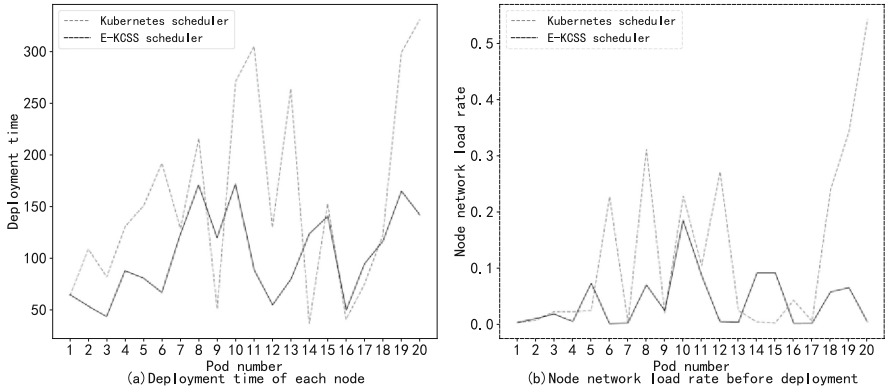

Fig. 3. Comparison of deployment efficiency

4.5 Comparison of Resource Utilization

In the experiment, the Kubernetes scheduler and scheduler based on E-KCSS were used to deploy CPU-intensive pod applications, memory-intensive pod applications and disk-intensive pod applications, corresponding to the applications with pod numbers 1, 2 and 3, respectively, in Table 2. The experiment deployed 100 pod applications in order, and compared the CPU utilization, memory utilization, disk utilization and pod distribution of each work node after deployment. The experimental results of deploying CPU-intensive pod applications are shown in Fig. 4. The experimental results of deploying memory-intensive pod applications are shown in Fig. 5. The experimental results of deploying disk-intensive pod applications are shown in Fig. 6.

As seen from Fig. 4(d), compared with the Kubernetes scheduler, the pod distribution of E-KCSS after deploying CPU-intensive applications is 12, 13, 26, 29, and 20, and many pods are deployed on node4. Table 1 shows that node4 has the most CPU resources, so E-KCSS solves the multi-dimensional resource weight parameters of the specific pod application according to the resource demand characteristics of pod applications and node resource utilization, enlarges the proportion of the CPU resource weight, and prioritizes the deployment of nodes with more CPU resources. Simultaneously, as shown in Fig. 4(a), the Kubernetes scheduler deploys a large number of pods on node5, which makes the CPU utilization of node5 reaching 96.40%, while the E-KCSS more evenly deploys pods on all cluster nodes. The CPU utilization of node5 decreases by 27.50%, and the CPU utilization of the other nodes increases by 6.15% on average to avoid resource utilization overload.

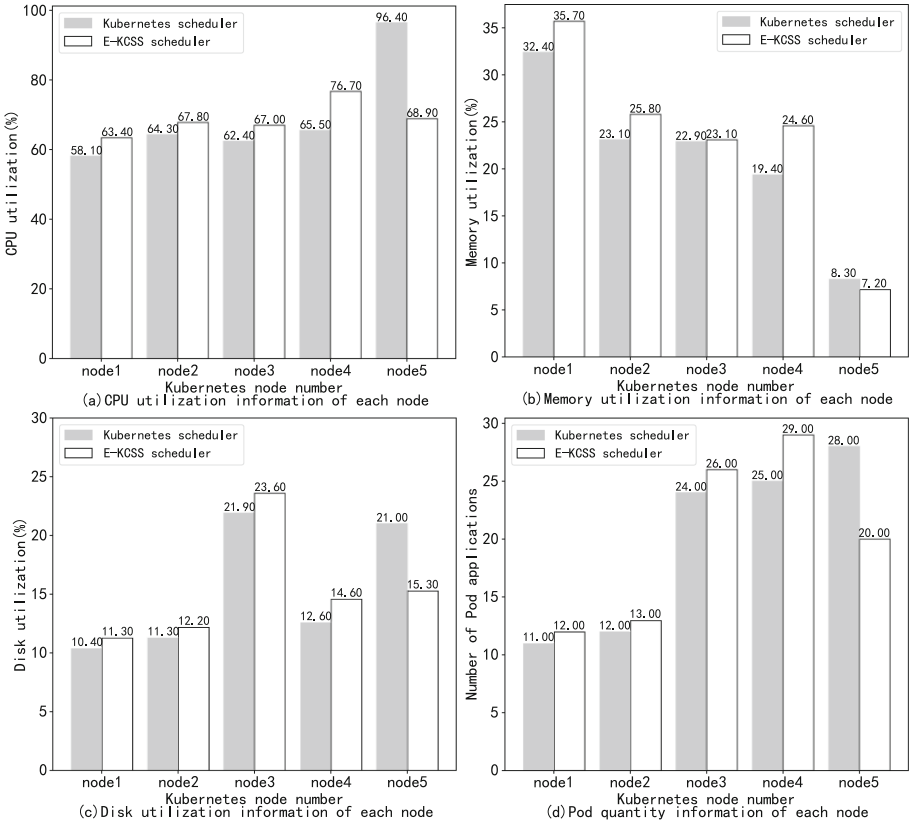

Fig. 4. CPU intensive pod application resource utilization

As Fig. 5 shows, compared with the Kubernetes scheduler, E-KCSS is more balanced in deploying memory-intensive applications. As shown in Fig. 5(d), the pod distribution of E-KCSS after deploying memory-intensive applications is 7, 11, 21, 22, and 39. Because node1 and node2 have relatively tight memory resources, while node5 has

the most memory resources, combined with the adaptive weight mechanism, E-KCSS deploys many pods on node5 to alleviate the memory pressure on the other working nodes. As shown in Fig. 5(b), the memory utilization of node5 increased by 15.50%, and the memory utilization of the other nodes decreased by 8.53%.

As Fig. 6 shows, the deployment of disk-intensive pod applications using E-KCSS-based schedulers is more balanced than that using Kubernetes schedulers. According to Table 1, node4 has the most disk resources, followed by node5. Simultaneously, as shown in Fig. 6(c), the Kubernetes scheduler deploys a large number of pods on node3, which makes the Disk utilization of node3 reaching 61.40%, while the E-KCSS more evenly deploys pods on all cluster nodes. According to Fig. 6(d), the pod distribution of E-KCSS after deploying disk-intensive applications is 13, 15, 19, 30, and 23. Many disk-intensive pod applications are deployed on node4 and node5. In summary, E-KCSS makes the disk utilization of cluster nodes more balanced.

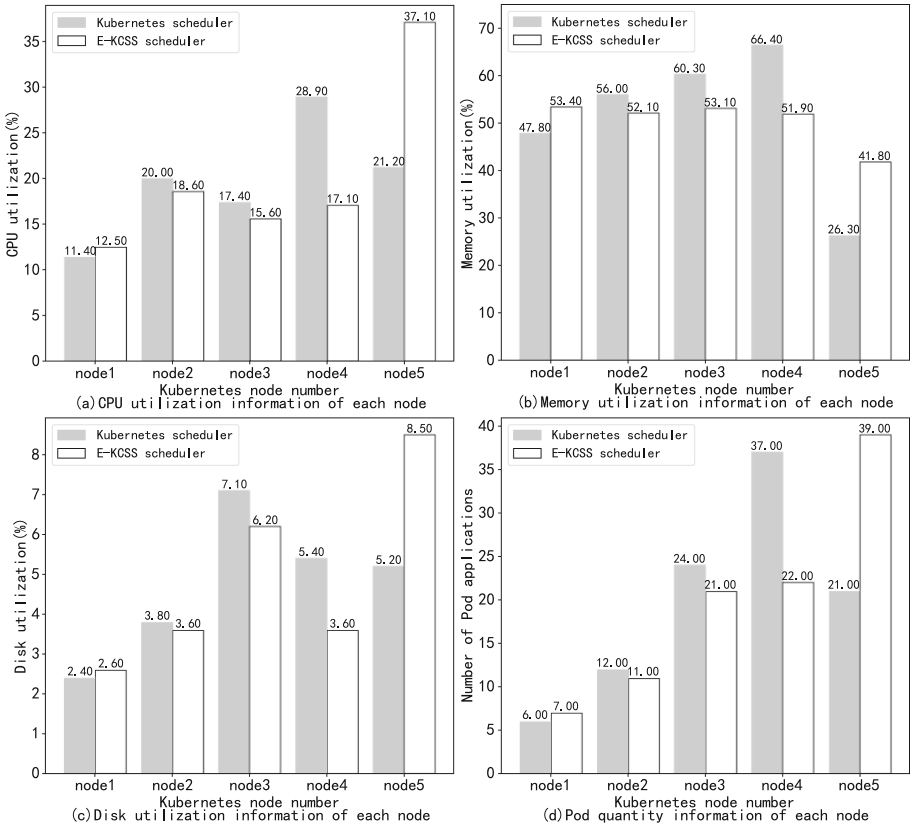

Fig. 5. Memory intensive pod application resource utilization

The Kubernetes scheduler considers only the resource utilization of nodes, while E-KCSS considers the pod application requirements and multi-dimensional resource utilization of nodes. E-KCSS automatically adjusts the multi-dimensional resource weight

and gives higher weight to the high-performance resource nodes to deploy more pods to these nodes. In addition, E-KCSS also uses the load balancing strategy to take into account the resource requirements of the pod and the actual surplus of the node. Using this strategy, the pod can be deployed on cluster nodes more evenly.

As the number of Pods increases, the Kubernetes scheduler does not consider factors such as disk resources and the number of pod deployments. Once a work node in the cluster is overloaded with resources, it will cause the waste of other resources, and the scheduler will be unable to schedule pod applications to the node. However, E-KCSS uses the adaptive weight algorithm to obtain the dynamic weight coefficient triggered by the dynamic event, and combines the resource idle rate of node to obtain the optimal solution. To sum up, compared with the Kubernetes scheduler, E-KCSS makes the node resource utilization more balanced.

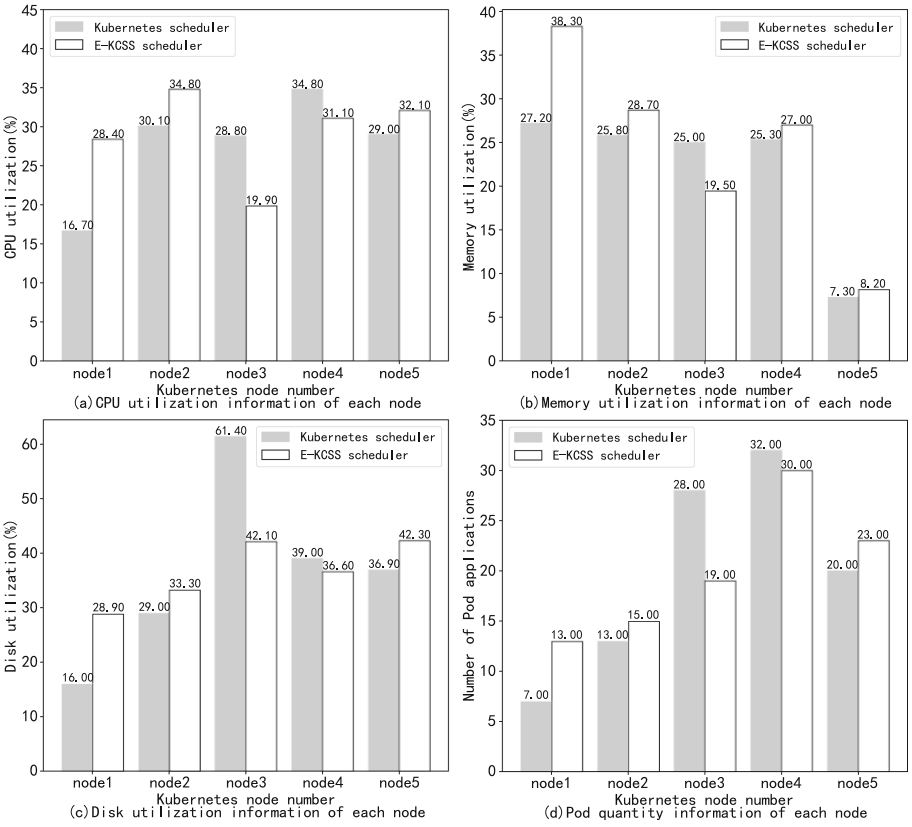

Fig. 6. Disk intensive pod application resource utilization

5 Conclusion

Container clouds are an important supporting technology for collaborative edge computing. The resources of edge nodes are relatively scarce, and the difference between edge applications and traditional applications brings new requirements for the scheduling mechanism of container clouds. Based on research on the Kubernetes scheduling policy, this paper proposes a Kubernetes container scheduling policy for collaborative edge computing to address the shortcomings of its single-criteria scheduling mechanism. This strategy comprehensively considers the CPU, memory, bandwidth, disk, and number of pods, automatically calculates multi-dimensional resource weights according to the application's resource requirements, and schedules pod applications based on the node resource load balance. Finally, experiments are designed to verify the performance of the E-KCSS scheduling method proposed in this paper and scheduling method of Kubernetes. Compared with the Kubernetes scheduling strategy, the deployment efficiency of E-KCSS increases by 35.22%, the upper limit of deployment increases by 29.82%, and the cluster resource imbalance decreases by 6.87%. The experiments show that the new scheduling method can effectively improve multi-dimensional resource utilization and cluster load balancing. In the future, more factors, such as node affinity and anti-affinity, internal load interference and data localization of pod applications, will be considered to make the cluster resource scheduling more balanced and efficient and satisfy the real-time needs of multiple tenants.

Acknowledgements. This research was funded by the National Natural Science Foundation of China (grant nos. 62172191 and 61972182), the National Key R&D Program of China (grant no. 2016YFB0800803), and the Peng Cheng Laboratory Project (grant no. PCL2021A02).

References s

1. Ren, J., Yu, J., He, Y.: Collaborative cloud and edge computing for latency minimization. IEEE Trans. Veh. Technol. **68**(5), 5031–5044 (2019)
2. Chiang, M., Zhang, T.: Fog and IoT: an overview of research opportunities. IEEE Internet Things J. **3**(6), 854–864 (2016)
3. Yang, L., Cao, J., Cheng, H.: Multi-user computation partitioning for latency sensitive mobile cloud applications. IEEE Trans. Comput. **64**(8), 2253–2266 (2014)
4. Lei, Y., Zheng, W., Ma, Y., Xia, Y., Xia, Q.: A novel probabilistic-performance-aware and evolutionary game-theoretic approach to task offloading in the hybrid cloud-edge environment. In: Gao, H., Wang, X., Iqbal, M., Yin, Y., Yin, J., Ning, G. (eds.) Collaborative Computing: Networking, Applications and Worksharing. LNICSSITE, vol. 349, pp. 255–270. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-67537-0_16
5. Xiao, X., Li, Y., Xia, Y., Ma, Y., Jiang, C., Zhong, X.: Location-aware edge service migration for mobile user reallocation in crowded scenes. In: Gao, H., Wang, X., Iqbal, M., Yin, Y., Yin, J., Ning, G. (eds.) Collaborative Computing: Networking, Applications and Worksharing. LNICSSITE, vol. 349, pp. 441–457. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-67537-0_27
6. Gao, H., Huang, W., Zou, Q., Yang, X.: A dynamic planning framework for QOS-based mobile service composition under cloud-edge hybrid environments. In: Wang, X., Gao, H., Iqbal,

M., Min, G. (eds.) Collaborative Computing: Networking, Applications and Worksharing. LNICSSITE, vol. 292, pp. 58–70. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30146-0_5
7. Zhang, J., Li, Y., Zhou, L., Ren, Z., Wan, J., Wang, Y.: Priority-Based optimization of I/O isolation for hybrid deployed services. In: Wang, X., Gao, H., Iqbal, M., Min, G. (eds.) Collaborative Computing: Networking, Applications and Worksharing. LNICSSITE, vol. 292, pp. 28–44. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30146-0_3
8. Xu, Y., Chen, L.: An adaptive mechanism for dynamically collaborative computing power and task scheduling in edge environment. IEEE Internet Things J. **1**(1), 232–245 (2021)
9. Li, J.: Design and implementation of machine learning cloud platform based on Kubernetes. Master thesis, Nanjing University of Posts and Telecommunications (2021)
10. Suresh, S., Manjunatha, R.: CCCORE: cloud container for collaborative research. Int. J. Elect. Comput. Eng. **8**(3), 1659–1670 (2018)
11. Dusia, A., Yang, Y., Taufer, M.: Network quality of service in docker containers. In: 2015 IEEE International Conference on Cluster Computing, pp. 527–528. IEEE (2015)
12. Casalicchio, E.: A study on performance measures for auto-scaling CPU-intensive containerized applications. Clust. Comput. **22**(3), 995–1006 (2019). https://doi.org/10.1007/s10586-018-02890-1
13. McDaniel, S., Herbein, S., Taufer, M.: A two-tiered approach to I/O quality of service in docker containers. In: 2015 IEEE International Conference on Cluster Computing, pp. 490–491. IEEE (2015)
14. Kong, D., Yao, X.: Kubernetes resource scheduling strategy for 5G edge computing. Comput. Eng. **47**(2), 32–38 (2021)
15. Gong, K., Wu, Y., Chen, K.: Container cloud multi-dimensional resource utilization balanced scheduling. App. Res. Comput. **37**(4), 1102–1106 (2018)
16. Piraghaj, S., Dastjerdi, A., Calheiros, R.: ContainerCloudSim: an environment for modeling and simulation of containers in cloud data centers. Softw. Pract. Exp. **47**(4), 505–521 (2017)
17. Guerrero, C., Lera, l., Juiz, C.: Genetic algorithm for multi-objective optimization of container allocation in cloud architecture. J. Gird Comput. **16**(1), 113–135 (2018)
18. Lin, M., Xi, J., Bai, W.: Ant colony algorithm for multi-objective optimization of container-based microservice scheduling in cloud. IEEE Access. **7**, 83088–83100 (2019)
19. Yang, M., Rao, R., Xin, Z.: CRUPA: a container resource utilization prediction for auto-scale based on time series analysis. In: 2016 International Conference on Progress in Informatics and Computing, pp. 468–472. IEEE (2016)