# Optimization of Service Scheduling in Computing Force Network

Yongqiang Dong
School of Computer Science and Engineering, CNII Lab of MoE
Southeast University
Nanjing, China
dongyq@seu.edu.cn

Chenchen Guan
School of Computer Science and Engineering
Southeast University
Nanjing, China
gcc@seu.edu.cn

Yunli Chen
School of Computer Science and Engineering
Southeast University
Nanjing, China
220181605@seu.edu.cn

Kun Gao
School of Cyber Science and Engineering
Southeast University
Nanjing, China
gaokun@seu.edu.cn

Lu Lu*
China mobile Research Institute
Beijing, China
lulu@chinamobile.com

Yuexia Fu
China mobile Research Institute
Beijing, China
fuyuexia@chinamobile.com

2022 International Conference on Service Science (ICSS) | 978-1-6654-9861-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICSS55994.2022.00031

***Abstract*—To improve the users satisfaction requesting services in Computing Force Network(CFN), the issues and challenges of existing service scheduling mechanisms are examined firstly in this paper. Then in view of users' specific preference and expectation for services, a service optimization scheduling algorithm based on Bkd-tree is proposed, taking service response time, service scheduling cost, availability, and successability into consideration. The method covers a search algorithm and a reconstructing strategy. To improve the efficiency of selecting the service instances that satisfy user expectation, a pruning strategy is used to narrow search space. To support dynamic updates of the service instances, a reconstructing strategy is proposed. The experimental results show that the improved service scheduling algorithm can schedule service instances with the best overall performance while maintaining higher efficiency.**

***Keywords—CFN; Service Scheduling; User Preference; Bkd-tree***

## I. INTRODUCTION

Cloud computing, with its virtualization technology [1], allows virtual machines or containers with divergent resources to be provisioned in physical machines, thus providing on-demand IT services to users[2,3].In recent years, emerging businesses such as the Internet of Things, autonomous driving, and smart factory have been gradually developed, driving computing to move down from the cloud to the edge close to the data source. MEC (Multi-access Edge Computing) places tasks close to the data source for edge-side computing, shortening the data transmission path and reducing the communication latency [4,5]. Multi-node computing collaboration has become a key research topic in MEC. The development of cloud computing and edge computing has made the network full of different types and scales of computing force resources, and these resources are connected through the network, forming a trend of ubiquitous computing force and network integration[6]. The Computing Force Network (CFN) architecture[7] is proposed in this context, relying on network connections to flexibly allocate computing force resources in the network, thereby providing business guarantees.

The computing force of CFN can be regarded as computing resources and storage resources, or services that provide customized functions based on the computing force [8]. After service deployment, service instances are located at different locations in the network, and service scheduling is the process of provisioning based on the status of the service instances and incorporating the user preference and expectation for the service. All existing cloud platforms provide service scheduling capabilities. Kubernetes[9] is a relatively mature platform in cloud-native technology. It accomplishes the scheduling of services through Kube-proxy, and supports strategies such as Round-Robin, Weighted Round-Robin, Random and Least Connection. Istio [10] is a popular platform in service mesh, and in terms of architecture, it is divided into control plane and data plane. The Pilot in the control plane sends the configuration to Envoy in the data plane through the xDs protocols and executes the scheduling of services through Envoy, which also supports Round-Robin, Weighted Round-Robin, Random, and Least Connection. However, the service scheduling strategies in the existing platforms do not consider user-related information and cannot meet the requirements of integrating user preference and expectation for service instances into scheduling in CFN. Therefore, we focus on the service scheduling problem in CFN, aiming to select the service instance with best overall performance to respond to the user's request.

In a nutshell, we make the following contributions in this paper:

- The service scheduling problem in CFN is abstracted as a multi-criteria decision-making (MCDM) problem that integrates user preference and expectation, aiming to find service instances that meet user requirements and have better service performance.
- We propose a service optimization scheduling algorithm based on Bkd-tree, which utilizes pruning strategy to quickly find service instances that meet user expectation.

* the corresponding author

978-1-6654-9861-6/22/$31.00 ©2022 IEEE
DOI 10.1109/ICSS55994.2022.00031

- An improved Bkd-tree reconstructing strategy is proposed, which can support dynamic updates of service instances in CFN.
- We verify the algorithm by simulation experiments. The results reveal that our proposed algorithm can select better service instances and has higher efficiency.

The rest of this paper is organized as follows. II reviews related work on service scheduling. III introduces the notations, definitions, and assumptions involved in the service scheduling problem in CFN. IV discusses the design of the improved Bkd-tree algorithm in detail. V presents the conduction of experiments and results. The conclusion is drawn in VI.

## II. Related Work

In the cloud computing domain, the goal of the service selection problem is to select the best service among services that offer the same functionality but with different quality of service (QoS). The service scheduling in CFN is similar to the service selection in that the best overall service instance among the service instances which can provide the same functionality by considering user expectation and preference. In the process of scheduling a service or service instance, due to the multi-dimensional nature of the QoS attributes of services or service instances, the above two problems based on QoS attribute values can be considered as an MCDM problem. In the service selection problem, TOPSIS and Skyline methods are often used to solve.

### A. Service Selection Based on TOPSIS Method

The goal of the TOPSIS method is to find the closest alternatives to the positive ideal solution and the farthest to the negative ideal solution, and the attribute information is used to provide the cardinality ranking of the alternatives and complete the service selection.

Kumar et al.[11] use the best and worst algorithm to calculate the weight of QoS (Quality of Service), and obtains the final ranking of cloud services by combining the TOPSIS method. Fang et al.[12] use the interval number to characterize the dynamic change of QoS, use the TOPSIS method to solve the objective weight of the QoS index, and finally integrate the subjective and objective weights to calculate the closeness of the QoS attribute to the user demand value. Liu et al. [13] also use the interval number to represent the dynamic change of QoS. In [13], the Skyline method is used to filter some services, and the objective weight is calculated by the entropy weight method and combined with the user preference to synthesize the mixed weight. Finally, the TOPSIS method is used to order services.

### B. Service Selection Based on Skyline Approach

The service selection based on the Skyline method [14] is to find out the service instances that are not dominated by other service instances from the candidate service instances and sort them.

Alrifai et al.[15] use the Skyline method to solve the problem of service selection with user constraints in cloud computing. First, according to the definition of Skyline, it selects the dominant service, reduces the search space, and selects from the dominant service sets. But when the user's requirement involves multiple value constraints, this method will filter all alternative services and return an empty set. Benouaret et al.[16] improve the Skyline method that may return empty sets, and calculate the regret value for the user's constraints. When the traditional Skyline method filters all the alternative services, the algorithm selects the service according to the regret value to respond, avoiding the case of returning an empty set. Wang et al.[17] propose a Web service selection method based on the Skyline method. By constructing a kd-tree to determine whether a service is a skyline service, the efficiency of finding a dominant service is improved. When electing the skyline service, the voting method is used. Then each service will get a certain number of votes, and the service with more votes will be allocated more requests. Purohit et al.[18] use a clustering algorithm to cluster the QoS of the services. First, the distance between the user constraints and each class is compared, and then the class with closer distance is selected, which reduces the search space. Li et al.[19] consider the scenario of multiple user requests, and a concurrency service selection algorithm with fairness perception is proposed. The basic idea is to iteratively solve the single-objective sub-problem into a linear programming problem.

To sum up, the methods based on TOPSIS need to sort and select all service instances when facing different preferences, which incur certain time overhead. The Skyline-based method has high implementation complexity and cannot make good use of user preference information, and does not support dynamic changes of service instances.

## III. Problem Statement

In this section, we define and model the service scheduling problem in CFN, introduce related notations and concepts, and present the related assumptions.

Definition 1: (Service). Service can be seen as a specific function which located in a VM or a container. $QoS_{S_l}$ is the Quality of Service(QoS) of a service $S_l$, which represents the performance of the service. In this paper, we consider four QoS attributes: response time, cost, availability, and successability. Response time is the time taken to send a request and receive a response. Cost is the cost per service instance request of invocation. Availability is the ratio of time period when a service is available. Successability is the ratio of quantity of successful responses to all service requests.

Definition 2: (Service Instance Set). A service instance set is composed of all service instances, which provide the same function. The instances of a service usually locate in different areas.

Definition 3: (QoS-based Service Scheduling). A request from user is defined as a tuple of $U = (S_l, \omega, req)$. $S_l$ is the type of service requested by the user, $\omega$ is user preference

and *req* is user expectation. The user preference has the following constraint:

$$\sum_{t=1}^{k} \omega_t = 1, \omega_t > 0, 1 \le t \le k \tag{1}$$

The goal is to schedule the service instance that satisfies user expectation in four dimensions and has the best overall performance, which is defined as follows:

$$\max Score = \sum_{t=1}^{k} \omega_t \times tanh\left(\frac{Nor(QoS_{S_{l,j}}{}^t)}{Nor(req^t)}\right) \tag{2}$$

s.t.

$$\frac{Nor(QoS_{S_{l,j}}{}^t)}{Nor(req^t)} \ge 1, (1 \le t \le k) \tag{3}$$

Where $Nor(QoS_{S_{l,j}}{}^t)$ represents the normalized t-th QoS attribute of instance $j$ of service $S_l$. To avoid the situation that the ratio of $Nor(QoS_{S_{l,j}}{}^t)$ to $Nor(req^t)$ is too large, the hyperbolic tangent function is used to limit the ratio between 0 and 1.

In the four dimensions of QoS, response time and cost are usually smaller and more attractive, which are negative indicators. Availability and successability are larger and more attractive, which are positive indicators. Therefore, response time and cost are normalized by Eq.(4). And availability and successability are normalized by Eq.(5).

$$Nor(QoS_{S_{l,j}}{}^t) = \frac{max(QoS_{S_l}{}^t) - QoS_{S_{l,j}}{}^t}{max(QoS_{S_l}{}^t) - max(QoS_{S_l}{}^t)} \tag{4}$$

$$Nor(QoS_{S_{l,j}}{}^t) = \frac{QoS_{S_{l,j}}{}^t - min(QoS_{S_l}{}^t)}{max(QoS_{S_l}{}^t) - min(QoS_{S_l}{}^t)} \tag{5}$$

Where $max(QoS_{S_l}{}^t)$ is the maximum value of the service $S_l$ in dimension $t$ and $min(QoS_{S_l}{}^t)$ is the minimum value.

To solve this problem, the following reasonable assumptions are made:

- A service instance is usually an application running in a virtual machine or container. We ignore the startup time of the service instance.
- The end-to-end delay in the network needs to be obtained through telemetry.
- The user preference in each dimension of service quality can be obtained through user registration or other means.
- The user expectation of service quality can be obtained by signing a Service-Level Agreement (SLA).

In CFN, the number of service instances is massive. Meanwhile, the preference and expectation of different users are quite different. If the traversal method is used to schedule service instances, the time complexity is $O(n)$. Besides, the status of a service instance is constantly changing, so the update of status needs to be considered.

## IV. Service Scheduling Based On Bkd-tree

A method based on Bkd-tree(Bulk k-dimensional tree) is proposed to solve the above optimization problem with multi-dimensional attributes and constraints. As shown in Fig.1, firstly, the status data of service instances are normalized. Then the service instances that satisfy the Eq.(3) are filtered out through the Bkd-tree upon user's request. Finally, the comprehensive score of each service instance is calculated according to the user preference.

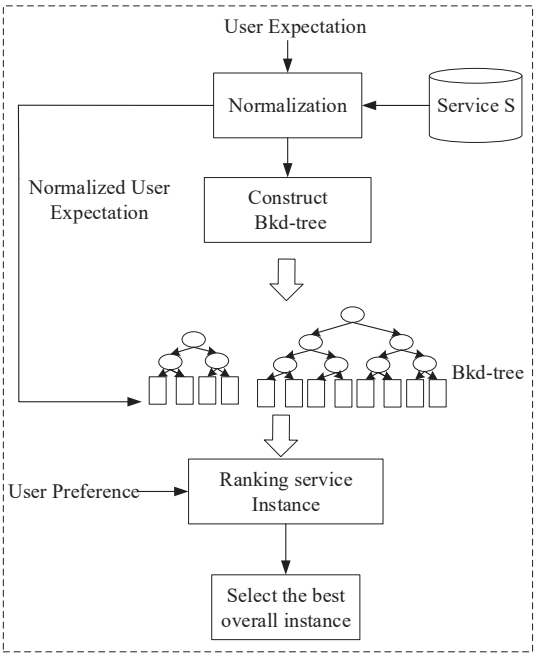


Figure 1. The service scheduling method based on Bkd-tree

### A. Kd-tree Based Service Status Maintenance and Search

A Bkd-tree can be regarded as a forest composed of multiple kd-trees. A kd-tree is a special kind of binary search tree. Each layer corresponds to a slice plane on a search dimension, which can divide the data of this dimension into two parts. A k-dimension binary search tree is finally constructed by cyclically using split plane on each dimension. A kd-tree with 2-dimension is shown in Fig.2.

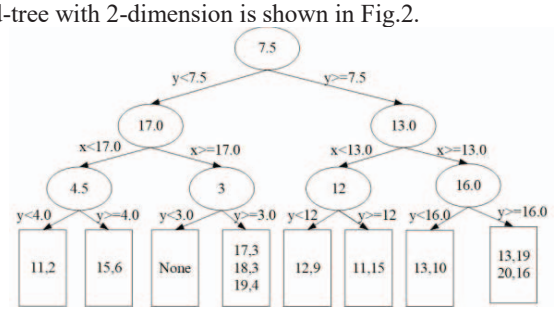


Figure 2. The service scheduling method based on Bkd-tree

When constructing the kd-tree, the variance is calculated for each QoS attribute (as individual dimension in kd-tree). Then, all service instances are sorted according to the dimension with the largest variance and the median is obtained. With the median as the cutting point, the service instances with attribute value less than the median are assigned to the left subtree, and the service instances larger than the median are assigned to the right subtree. The above process is repeated until all the service instance stops at a leaf node. In the end, the non-leaf nodes in the constructed kd-tree represent split conditions in distinct dimensions, and the service instances located at different leaf nodes according to the split conditions. Algorithm 1 is used to construct the kd-tree.

**Algorithm 1:** $kd-tree\_Construction$

**Input**: normalized status data of service instances $D=\{D_1, D_2, ..., D_k\}$

**output**: the *root* of constructed kd-tree

1: $split = \max(variance(D_i)), i = 1,2,...,k$;
2: $sort(D_{split})$;
3: $median = length(D_{split})/2$;
4: for $d : D_{split}$ :
    if $d \le D_{split}[median]$ :
        $QoS_{S_{l1}}.add(d)$;
    else:
        $QoS_{S_{l2}}.add(d)$;
5: $root = Node(D_{split}[median])$;
6: $root.left = kd-tree\_Construction(QoS_{S_{l1}})$;
7: $root.right = kd-tree\_Construction(QoS_{S_{l2}})$;

After building the kd-tree, the instances with better service status will be more concentrated on the right side of the kd-tree, but the rightmost service instance is not necessarily the best. For example, the service instance data <0.8, 0.9, 0.9, 0.9> is kept in the rightmost leaf node, and <0.78, 0.99, 0.99, 0.99> is assigned to the left subtree because its value of first dimension is smaller than the median. However, the overall performance of the latter is not inferior to that of the rightmost service instance.

The search space can be reduced by a pruning strategy when searching for service instances that meet the user expectation in the kd-tree. For example, when searching the data that satisfies $< x \ge 14, y \ge 10 >$, the left subtree with y=7.5 as the root and the left subtree with x=13.0 as the root can be pruned, due to the fact that they cannot satisfy the lookup requirement in the y and x dimensions, respectively. Only the right subtree with 13.0 as the root is searched. And the lookup size of the data is greatly reduced.

In our scenario, the pruning strategy can be performed according to the user expectation. Before searching, the user expectation need to be normalized by Eq.(4) and Eq.(5) as that in constructing the kd-tree. Then, pruning is performed by comparing the value of user expectation with the split conditions in the kd-tree, and the left subtree is pruned when the cutting point is smaller than the user expectation. Algorithm 2 is used to search in the kd-tree.

**Algorithm 2:** $kd-Search$

**Input**: *root* : the root of kd-tree,
*query* : the user expectation,
*rd* : the dimensions of user expectation,
*nA* : the dimension of current split condition,
*con* : split conditions saved in non-leaf nodes

**Output:** all service instances that satisfy the user expectation

1: if $root == Null$ :
2: return
3: if $root.isLeaf()$ :
4: $isSatisfied()$;
5: $con[nA] = root.value$;
6: if $root.value \ge query[nA]$ :
7: if $isSatisfied(con)$ :
8: $findAllLeaves(root, query)$;
9: else:
10: $kd-Search(root.right, query, (nA+1)\%rd, rd, con)$;
11: $kd-Search(root.left, query, (nA+1)\%rd, rd, con)$;
12: else:
13: $kd-Search(root.right, query, (nA+1)\%rd, rd, con)$;

### B. Improved Bkd-Tree Reconstructing Strategy

Although kd-tree can support the maintenance of multi-dimensional data and efficient lookup, it is only suitable for static data. In CFN, the state of service instances will change dynamically. To address this issue, an improved Bkd-tree is used. Compared to the kd-tree, the logarithmic method is used in Bkd-tree. This structure can be seen in Fig.3. A Bkd-tree consists of a small buffer region followed by exponentially growing static kd-trees. Inserts are performed by rebuilding a minimum number of trees to maintain fully constructed static trees.

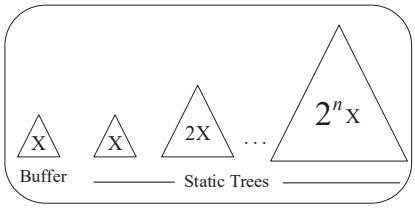


Figure 3. The logarithmic method used in Bkd-tree

When the status of a service instance changes, it is stored into the buffer temporarily. If the data amount in the buffer reaches $M$ , the first empty kd-tree is constructed with the updated data.

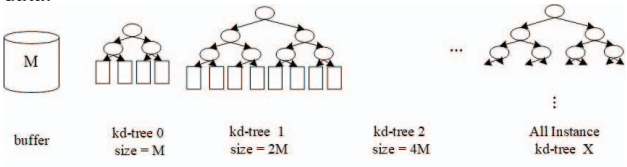


Figure 4. Improved Bkd-tree

In Fig.4, the first empty tree is kd-tree 2, whose data size is 4M. Consequently, the data in the buffer, kd-tree 0 and kd-tree 1 are emptied to form kd-tree 2. Therefore, there will be two trees in use, kd-tree X and kd-tree 2. In response to the user's request, the service instances that satisfy the user expectation will be found from kd-tree X and kd-tree 2.

In the Bkd-tree, the number of kd trees to maintain data will increase when data is continuously inserted. While in CFN, the state of service instances is updated regularly. The constant construction of new kd trees will bring great space overhead while keeping some obsolete data. Therefore, we propose a refactoring strategy. Specifically, the initially constructed kd-tree will maintain the state of all service instances, and updated service instances data are inserted into the buffer and the reconstructed kd-tree. When the size of updated service instance data exceeds half of the original kd-tree, the initial kd-tree is reconstructed for all service instances. After that, the buffer and the kd-trees where the updated data is temporarily stored are emptied. Algorithm 3 is used to update the Bkd-tree.

**Algorithm 3:** $Bkd-tree\_Update$

**Input:** new status data $newD$

**Output:** new Bkd-tree

1: $insert(newD)$ to buffer;
2: if $size(buffer) \geq M$ :
3: $FET = findFirstEmptyTree()$
4: $data = merge(buffer, KDTree-i), (0 \leq i < FET)$
5: $KDTree-T = kd-tree\_Construction(data)$
6: $erase(buffer, KDTree-i), (0 \leq i < FET)$

Assuming that after the update, the kd-tree X and kd-tree 2 are maintained. The service instances in the kd-tree X are the original data, while the kd-tree 2 maintains the newly updated data. There will be the following situations:

- The search results of kd-tree 2 are not found in kd-tree X. In this case, the search results of kd-tree 2 can be directly added to the candidate set.
- The search results of kd-tree X and kd-tree 2 have an intersection. In this case, we will remove the service instances from kd-tree X that are located in the intersection.
- A service instance is found in kd-tree X, but it is not included in the search results of kd-tree 2. In this case, there is no way to distinguish whether the service instances found in kd-tree X have not been updated, or they have been updated but do not satisfy user expectations. Here CBF (Counting Bloom Filter) is used to solve this issue. When the status of a service instance is updated, the service instance ID is inserted into the CBF. The service instance found in kd-tree X is examined further by CBF. If the CBF contains the service instance, it means that the latest service instance status does not satisfy the user expectation and should be removed. If the CBF does not contain the service instance, it means that the service instance status in the kd-tree X is the latest. This service instance still satisfies the user expectation and should be kept.

Searching all kd-trees and the buffer can quickly find all service instances that satisfy user expectation. In order to express the overall performance of a service instance, the service instance status data are weighted and computed by Eq.(2). Then, the service instance with the highest score will be selected.

## V. Performance Evaluation

### A. Design of Simulation

Since service-oriented CFN does not have a real and available environment currently, the simulated method is adopted to evaluate the proposed service scheduling algorithm. The tasks that follow the Poisson distribution are launched to simulate the requests from users, and the task arrivals is generated by Eq.(6).

$$P_n(t) = e^{-\lambda t} \frac{(\lambda t)^n}{n!} \tag{6}$$

Where $\lambda$ represents the number of task arrivals per unit time, and tasks are independent of each other.

QWS dataset is used to test the proposed service scheduling algorithm, which contains nine kinds of QoS parameters and 2507 QoS data. However, QWS dataset lacks cost metric of the service instance. In general, the cost of a service instance has a certain relationship with the load rate. The higher the load rate of the service instance, the higher the cost. When the load rate reaches a high level, the service instance may stop responding to requests, and has a infinite cost value. Therefore, the cost of service instance is calculated by using Eq.(7) in our simulation. When the load rate of the service instance is low, the cost of the service instance increases slowly, but as the load rate gradually increases, the cost of the service instance increases rapidly.

$$Cost = \frac{1}{1-\eta} \tag{7}$$

Based on Eq.(6), three sets of Poisson-distributed task flows are generated. The arrival rare of tasks is shown in Fig.5. Task flows in 30s are generated, and the average number of user requests per second are 10, 30, 50 in Fig.5(a), (b), and (c).

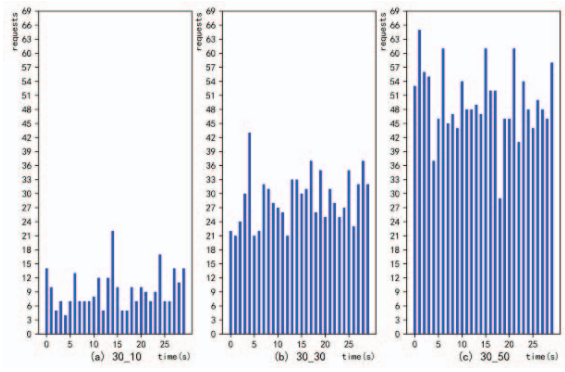


Figure 5. The arrival distribution of tasks

As there is no available dataset on user expectation of service quality, user expectation are generated based on the data distribution in the QWS dataset. The statistics of response time, availability and successability in the QWS dataset are shown in Fig.6.

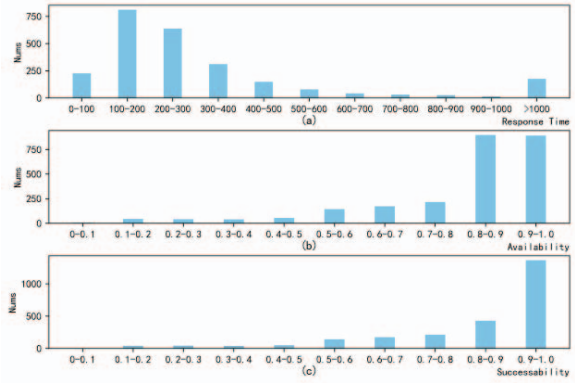


Figure 6. The statistics of the QWS dataset

In Fig.6, response time lies between 1ms and 300ms, and availability and successability are between 0.8 and 1. Therefore, the simulated user expectation takes values as shown in Table 1.

TABLE I. SIMULATED USER EXPECTATION

| name | range | unit |
|---|---|---|
| response time | <1, 300> | ms |
| cost | <1, 100> | |
| availability | <0.8, 1> | % |
| successability | <0.8, 1> | % |

## B. *Result analysis*

The algorithm based on Bkd-tree is compared with RR (Round Robin) algorithm and TOPSIS, and the user satisfaction with service scheduling and efficiency of algorithm are used as evaluation measures.

*1) User satisfaction:*

The user satisfaction with the RR, TOPSIS, and the Bkd-tree algorithm are displayed in Fig.7. RR algorithm is simple and efficient, but it does not consider the user expectation and preference, which makes the score of service scheduling rather low. The TOPSIS-based method sorts the service instance by calculating the closest to the positive ideal solution. However, it cannot guarantee that the selected service instance can satisfy user expectation in every dimension. Therefore, the user satisfaction obtained is slightly lower than the Bkd-tree algorithm. Service scheduling based on Bkd-tree performs better when user expectation and preference are considered.

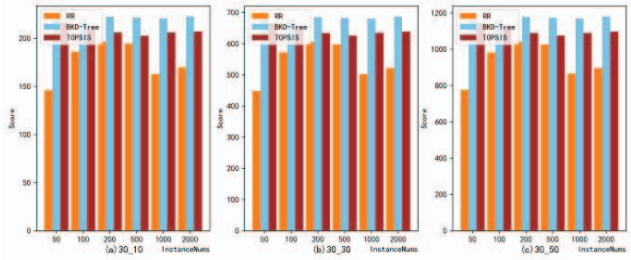


Figure 7. User satisfaction with service scheduling

*2) Efficiency of algorithm:*

The execution time of the TOPSIS algorithm and the Bkd-tree algorithm is compared in Table 2. When there is less service instances, the execution time of these two algorithms are not much different. As the number of service instances increases, the TOPSIS algorithm needs to re-order the data structure according to user preference when different requests arrive. So the execution time of TOPSIS increases linearly with the number of service instances. The algorithm based on Bkd-tree maintains data in multi-dimensional partition trees. When requests from different users arrive, a binary search can be performed on these trees with logarithmic time complexity. Therefore, the algorithm based on Bkd-tree has higher efficiency.

TABLE II. EXECUTION TIME OF TOPSIS AND THE IMPROVED BKD-TREE

| number of service instances | TOPSIS (s) | Improved Bkd-tree(s) |
|---|---|---|
| 50 | 0.5725 | 0.3819 |
| 100 | 1.0941 | 0.4158 |
| 200 | 2.1323 | 0.4827 |
| 500 | 5.3267 | 0.6742 |

The efficiency of searching for the service scheduling algorithm based on the kd-tree is shown in Fig.8. The time complexity of searching in a kd-tree is $O(n^{(k-1)/k})$. We record the average times of searching when requests come from 1500 users. The blue line represents a diagram of $y = x^{3/4}$, the theoretic time complexity of our algorithm. As shown in Fig. 8, the average times of searching in experiments is mostly distributed near the curve. This result indicates that the efficiency of our method is satisfactory.

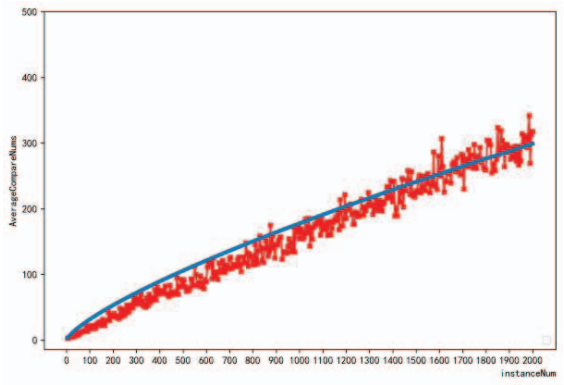


Figure 8. Average searching times of the Bkd-tree algorithm

## VI. CONCLUSION

In this paper, we propose an improved Bkd-tree algorithm to schedule service instances in CFN. It combines user preference and expectation for services. Compared with the existing service scheduling algorithms, our method can effectively select the service instance that satisfies the user expectation and cooperate with the dynamic update of service instances. The experimental results show that our proposed algorithm can schedule the most appropriate service instance, improve user satisfaction while maintaining higher efficiency.

## ACKNOWLEDGMENT

We would like to thank reviewers for their precious comments on this paper. This work was supported partly by Southeast University-China Mobile Research Institute Joint Innovation Foundation (under Grant No. CMYJY-202100163), and National Natural Science Foundation of China (under Grant No. 62072100).

## REFERENCES

[1] Kaur, Kuljeet, et al. A multi-objective optimization scheme for job scheduling in sustainable cloud data centers. IEEE Transactions on Cloud Computing (2019).

[2] H. Wang, H. Shen, Q. Liu, K. Zheng, and J. Xu. A reinforcement learning based system for minimizing cloud storage service cost. In Proc. of ICPP, 2020.

[3] H. Wang, Z. Liu, and H. Shen. Job scheduling for large-scale machine learning clusters. In Proc. of CoNEXT, 2020.

[4] Brecher, Christian, Melanie Buchsbaum, and Simon Storms. Control from the cloud: Edge computing, services and digital shadow for automation technologies. 2019 International Conference on Robotics and Automation (ICRA). IEEE, 2019.

[5] B. Liu, J. Mao, L. Xu, R. Hu and X. Chen. CFN-dyncast: Load Balancing the Edges via the Network. 2021 IEEE Wireless Communications and Networking Conference Workshops (WCNCW), 2021, 1-6.

[6] IRTF, Computing in the Network Research Group (COINRG). Directions for Computing in the Network. 2020, 3-13.

[7] China Mobile Research Institute. Computing Force Network White Paper. 2021.11

[8] China Mobile Research Institute. Computing Force Awareness Network Technology White Paper. 2021

[9] Chang, Chia-Chen, et al. A kubernetes-based monitoring platform for dynamic cloud resource provisioning. GLOBECOM 2017 - IEEE Global Communications Conference.

[10] Sheikh, Ozair, et al. Modernize digital applications with microservices management using the istio service mesh. 28th Annual International Conference on Computer Science and Software Engineering,2018.

[11] Kumar R R, Kumari B, Kumar C. CCS-OSSR: A framework based on Hybrid MCDM for Optimal Service Selection and Ranking of Cloud Computing Services. Cluster Computing, 2020(4).

[12] Fang Chen, Wang Jindong, Yv Zhiyong. Web Service Selection Method Based on Dynamic QoS. Computer Science. 2017, 44(05):245-250

[13] Liu Jinghua. Study on User Requirement-Based Dynamic QoS Service Selection Method. Computer Engineering and Applications. 2021, 57(17):106-115

[14] Borzsony S, Kossmann D, Stocker K. The Skyline operator[C]// International Conference on Data Engineering. IEEE, 2002.

[15] Alrifai M, Skoutas D, Risse T. Selecting skyline services for QoS-based web service composition. 19th international conference on World wide web, 2010: 11-20.

[16] Benouaret K, Elmi S, Tan K L. Relaxing the Sky: Handling Hard User Constraints in Skyline Service Selection. IEEE International Conference on Services Computing (SCC), 2021: 62-70.

[17] Wang Y , Song Y , Liang M. A skyline-based efficient web service selection method supporting frequent requests. 20th international conference on computer supported cooperative work in design (CSCWD), IEEE, 2016: 328-333.

[18] Purohit L, Kumar S. Clustering based approach for web service selection using skyline computations. IEEE international conference on web services (ICWS), 2019: 260-264.

[19] Li, Songyuan, et al. Fass: A fairness-aware approach for concurrent service selection with constraints. IEEE International Conference on Web Services (ICWS), 2019.