Journal of Systems Engineering and Electronics
Vol. 33, No. 4, August 2022, pp.792 – 804

Distributed point-to-point routing method for tasks in cloud control systems

WANG Guan[1,2], ZHAN Yufeng[1], XIA Yuanqing[1], and YAN Liping[1,*]

1. School of Automation, Beijing Institute of Technology, Beijing 100081, China;
2. School of Information Science and Engineering, Zaozhuang University, Zaozhuang 277100, China

Abstract: With the rapid development of cloud computing and control theory, a new paradigm of networked control systems called cloud control systems is proposed to meet the requirements of large-scale and complex applications. Currently, cloud control systems are mainly built by using a centralized architecture. The centralized system is overly dependent on the central control plane and has huge challenges in large-scale heterogeneous node systems. In this paper, we propose a decentralized approach to establish cloud control systems by proposing a distributed point-to-point task routing method. A considerable number of tasks in the system will not rely on the central plane and will be directly routed to the target devices through the point-to-point routing method, which improves the horizontal scalability of the cloud control system. The point-to-point routing method directly gives a unique address to every task, making inter-task communication more efficient in a complex heterogeneous and busy cloud control systems. Finally, we experimentally demonstrate that the distributed point-to-point task routing approach is compatible against the state-of-the-art central systems in large-scale task situations.

Keywords: cloud control system, task routing, cloud-edge-device cooperation.

DOI: 10.23919/JSEE.2022.000079

1. Introduction

Cloud control systems [1] are complex systems that unify cloud, edge and terminal devices. As cloud control systems continue to develop, system requirements for performance and reliability continue to increase. Meanwhile, many edge and terminal devices exist and are newly added to the system. For example, the cloud control system in intelligent transportation needs to complete the collection and calculation of massive traffic data in real-time and control the colossal road traffic equipment to complete complex vehicle control tasks. Cloud control systems are more different from cloud computing systems. Cloud computing systems are mainly cloud devices, complex computing, and the cloud control system in addition to the cloud also contains complex heterogeneous edge and terminal devices, cloud devices, edge devices, and terminal devices in the system to form a highly unified, collaborative whole, forming a closed-loop from computing to control. Edge and end devices have low computing power. Generally, they run small tasks (low amount of required system resources), and as the number of edge and end devices increases, the number of small tasks within the cloud control system increases. The number of devices and tasks within cloud control systems and the heterogeneity of devices and tasks are much higher than typical cloud computing systems.

Existing cloud control systems are built based on common container orchestration platforms and virtual machine management platforms [2−4]. The system is addressed only for devices (usually using IPv4 protocol addressing), and inter-task communication uses device addresses, making full use of the protocol stack provided by the operating system to ensure the reliability and stability of the system. However, there are three main problems: (i) address space is limited by the Internet protocol (IP), (ii) address is randomly assigned and lacks semantics [5], and (iii) tasks are migrated between devices, requiring central nodes to record task addresses [6,7]. Containerized orchestration platforms and virtual machine (VM) management platforms can usually be divided into control and execution planes. The control plane can be considered the central node of the system, which is responsible for creating, launching, scheduling, and managing container or VM tasks. The system's functional implementation relies heavily on the central node, which often becomes the system bottleneck [8]. Systems such as Kubernetes alleviate the central node pressure to a cer-

Manuscript received March 01, 2022.
*Corresponding author.
This work was supported by the National Key Research and Development Program of China (2018AAA0103203), the National Natural Science Foundation of China (62073036;61836001;62102022;62122014), and the Beijing Natural Science Foundation of China (42020741).

tain extent by setting up a multi-node control plane and by load balancing. However, for the case of multiple devices and multiple tasks in the cloud control system, the control plane is still a system bottleneck, which severely restricts the increase of the device size and the number of schedulable execution tasks in the cloud control system.

This paper eliminates the central structure and builds a ring-type system by unified addressing of tasks and distributed point-to-point routing to solve the above problems in current cloud control systems. The main contributions of this paper are as follows:

(i) We propose a unified addressing method for tasks and devices within cloud control systems. The unified addressing algorithm addresses semantics to tasks and devices, and the addressing space is controlled by the algorithm parameters, which are not restricted by the IP protocol. Tasks and devices within the cloud control system use the same addressing algorithm. The obtained addresses will be in the same addressing space, and a suitable distance algorithm can measure the distance between tasks and devices.

(ii) Quality of service for devices (DQoS) in a group is proposed. Devices with the same address form a group, and tasks are routed to the group need to select specific execution devices. The selection priority of devices within the group needs to be considered. The device with higher priority will be selected as the execution device. Based on the careful consideration of device performance and reliability, this paper proposes a method to calculate the quality of service of a device and uses the DQoS score of a device as the selection priority.

(iii) We propose a distributed point-to-point ring system structure, where there is no central control plane in the system. The task-to-device scheduling problem in the system is transformed into a task-to-device routing problem. A task-to-device routing method applicable to the cloud control system is proposed based on the unified addressing of tasks and devices.

The article is structured as follows: After this introduction, background and relative works are given in Section 2. The distributed point-to-point routing method for tasks in cloud control systems is introduced in Section 3. The experimental findings and insights on the main results are presented in Section 4. Related work is summarized in Section 5.

2. Background and relative works

In this section, we first introduce the current architecture commonly used in the cloud control system, that is, container orchestration platform-based centralized cloud control systems, illustrating the difficulties faced in large-scale distributed tasks for which a novel solution will be proposed in this paper. Then, the quality of service for devices and the difficulties of existing methods to measure both device performance and reliability are described. We will propose novel methods in this paper to enable more balanced routing of tasks within groups of devices.

2.1 Container orchestration platform-based cloud control systems

With the development of cloud control systems, the complexity of the system increases, and the scheduling and management of a large number of tasks within the system becomes a significant issue that hinders the development of cloud control systems. Container technology is a technology to virtualize applications in a lightweight manner and has been adopted on a large scale in cloud computing systems [9].

Usually, containers are processed isolated and resource-limited through the Namespace and Cgroup functionalities of the operating system kernel. Cgroup provides an interface to limit the number of resources (e.g., CPU, memory, read/write speed) per container. Hardware virtualization technology virtualizes the hardware device and operating system again on the host operating system. Containers are faster to boot and more resource-efficient for the host than virtualization. There are a large number of heterogeneous hardware and software and a large number of tasks within the cloud control system, and using containers to represent tasks is more efficient than using virtual machines to represent tasks in task initiation and task scheduling methods.

In the actual application of cloud control systems, there are often dependencies between tasks, and task containers do not appear singularly. However, multiple task containers are connected together, and they are deployed as a unified unit. Also, task containers do not run only on a single device but run task containers in bulk as a cluster, so task container orchestration includes task container scheduling, lifecycle management, elastic scaling, network management, storage management, and cluster management of devices running task containers.

Kubernetes is a more mature container orchestration framework, which not only enables container application deployment, container service orchestration, elastic scaling, container scheduling, and other functions [10], but also has features such as high availability and rolling upgrade [11]. A complete Kubernetes cluster generally contains two parts: the control plane (including master nodes) and the execution plane (including worker nodes) [3]. Improving system performance and reliability, the control plane can consist of multiple load-balanceable

device nodes.

When tasks migrate to a new device, the address of the task changes, and a central node is needed to record the change of address so that tasks can discover and communicate with each other [12,13]. When the number of tasks in the system increases, the pressure on the central recording node increases, limiting the expansion of the number of tasks in the system and further limiting the scale of the system.

Twine improves the performance of Kubernetes on very large clusters [4]. The Twine scheduler allows to split the scheduling of different subsets of entitlement under the same region, allowing for larger scale cluster management (scalability). The sharding mechanism increases the cluster size, but the cross-shard scheduling of tasks remains limited by a new model of central control plane bottleneck.

The central control plane constrains the horizontal scaling of centralized system clusters. Various solutions have been proposed by numerous researchers. Literature [14,15] proposed to improve the system performance by optimizing the central control plane synchronization algorithm. Literature [16] proposed the use of federation to turn multiple independent clusters into logically unified clusters. Federated clusters increase cluster size in part, but interconnecting multiple clusters efficiently poses a new problem. Literature [17] simulated the network environment in disadvantaged tactical networks and evaluated the performance of the Kubernetes system in a big federated cluster. The study revealed that the central control plane causes problems in a large number of task loads, deteriorating the system's efficiency, and proving difficult to improve.

These studies have driven the development of centralized cluster systems, but have not essentially solved the bottleneck caused by the central control plane. The elimination of the central control plane in the system can significantly solve the problem of system efficiency improvement in the case of a large number of tasks and a busy system, and can essentially eliminate the factors that limit the horizontal expansion of the system.

In this paper, we propose a task-distributed point-to-point routing method to eliminate the central control plane in the system and complete the task-to-device routing by proposing a task and device unified addressing method.

2.2 DQoS

For the dynamic calculation, the DQoS is usually approximated by the arithmetic mean of the historical DQoS of the device [18], and it is assumed that the DQoS remains unchanged during the selection and evaluation of the device. Although this approach is simple to implement, it does not differentiate the functions of device services, resulting in it not reflecting the DQoS dynamically and effectively. The changes in the device operating environment and the different task types and task granularity make the DQoS highly dynamic.

Literature [19] proposed to predict the stability of the service provided by the device based on long short-term memory (LSTM) neural network and to calculate DQoS in combination with the cost of the service spent. Literature [20] proposed a model for calculating DQoS in real-time based on a multi-signal mechanism of the immune system, which only gives the calculation of the service reputation and does not give the calculation of other important DQoS metrics. The work mentioned above promotes the study of dynamic computation of DQoS, but treats all known devices as a single set and then calculates the QoS of each device without making a clear distinction between services that provide similar or identical functions or not.

3. Distributed point-to-point routing method for tasks

3.1 Unified addressing method for tasks and devices

There are two main entities within cloud control systems, tasks and devices. The devices are the execution units and are the carriers of the tasks. One of the main functions of the cloud control system platform is task scheduling to the execution device. The task t can be formulated in a binary way: $t_{\text{dev}} = (b, P(b))$, where P is the set of attributes of the executable code b, $P(b) = \{p_1(b), p_2(b), \cdots, p_n(b)\}$. Uniform encoding of tasks and devices within the cloud control system and giving task and device address semantics is the basis for task-to-device routing.

In cloud control systems, many task attributes are continuous attributes, e.g., CPU usage limitation, memory usage limitation. Different continuous attributes take different ranges of values.

In cloud control systems, the tasks are performed by various heterogeneous devices containing some low computational power edge and end devices. The restricted Boltzmann machine (RBM) is a two-tier network with simple computation and implementation to accommodate different device computational capabilities.

The RBM is a variant of the Boltzmann machine that restricts the model to be a bipartite graph and can learn randomly generated neural networks with probability distributions from the input data set. RBMs have found applications in dimensionality reduction [21], classification [22], collaborative filtering [23], feature learning [24], and topic modeling [25]. RBMs can be trained by

using by supervised or unsupervised learning methods depending on the requirement. The model contains visible units corresponding to the input parameters and hidden units corresponding to the training results, and each edge in the graph must connect one visible unit and one hidden unit. In contrast, the "unrestricted" Boltzmann machine contains edges between hidden units, making it a recurrent neural network. This restriction allows for more efficient training algorithms compared to general Boltzmann machines, especially the gradient-based contrastive divergence algorithm [26].

Given the training set $S_{\text{train}} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_i, y_i)\}$, where i denotes the ith sample in the set consisting of the input vector $\boldsymbol{x}_i$ and the corresponding target class $\boldsymbol{y}_i \in \{1, 2, \cdots, n\}$. We use a specific RBM to model the joint distribution between a layer of N hidden variables $\boldsymbol{h} = (h_1, h_2, \cdots, h_n)$ (often referred as features) and the observed variables (x, y). It is a parametric model with parameters $\boldsymbol{\Theta} = (\boldsymbol{W}, \boldsymbol{U}, b, c, d)$ representing the following: $\boldsymbol{W}$ is weights matrix between $\boldsymbol{x}$ and $\boldsymbol{h}$; $\boldsymbol{U}$ is weights matrix between $\boldsymbol{e}_y$ and $\boldsymbol{h}$; b, c, d are respective biases of $\boldsymbol{x}, \boldsymbol{e}_y$ and $\boldsymbol{h}$, and $\boldsymbol{e}_y = (1_{i=j})$ for $i \in (1, 2, \cdots, C)$ is the 'one out of C' vector representation of $\boldsymbol{y}$.

After the RBM is trained, the computation can be completed with the conditional probability $p(y|x)$. $p(y|x)$ can be calculated from (1).

$$\begin{cases} f(j,y) = c_i + u_{iy} + \sum_{i=1}^{m} w_{ji} x_i \\ F(y,h) = -d_y - \sum_{j=1}^{n} \log_2\left(1 + \mathrm{e}^{f(j,y)}\right) \\ p(y|x) = \operatorname{argmin} F(y,x) \end{cases} \tag{1}$$

The framework of the RBM neural network computation used in this paper is shown in Fig. 1. It consists of an RBM visible layer and a hidden layer. First, the data of task and device attributes are preprocessed. Second, the sequence of each attribute of the task and device is used as the input to the network. Finally, the value of the hidden layer is output as the attribute encoding.

Algorithm 1 provides the pseudo-code of training encoder.

Algorithm 1 Pseudo-code of training the RBM encoder in a python-like style

```
# data: training examples
# w: weight matrix, random initialization
# v: visiable unit matrix
# h: hidden unit matrix
# l: leaning rate, constant
# σ(x): function of activation
# p: positive matrix
# n: negative matrix
for d in data do
    v:=d
# forward 1
for j from 1 to length of h do
    for i from 1 to length of v do
        a: = a+w[i,j]v[i]
    h[j]=σ(a)
# calculate the positive matrix
for i from 1 to length of v do
    for j from 1 to length of h do
        p[i,j]=v[i]h[j]
# backward, reconstruct the visible units
for i from 1 to length of v do
    for j from 1 to length of h do
        a: = a+w[i,j]h[j]
    v[i]=σ(a)
# forward 2, update the hidden units again
h: = σ(∑ w_ij v_i)
# calculate the negative matrix
for i from 1 to length of v do
    for j from 1 to length of h do
        n[i,j]=v[i]h[j]
# update the weight matrix
w[i,j]<−w[i,j]+l(p[i,j]−n[i,j])
```

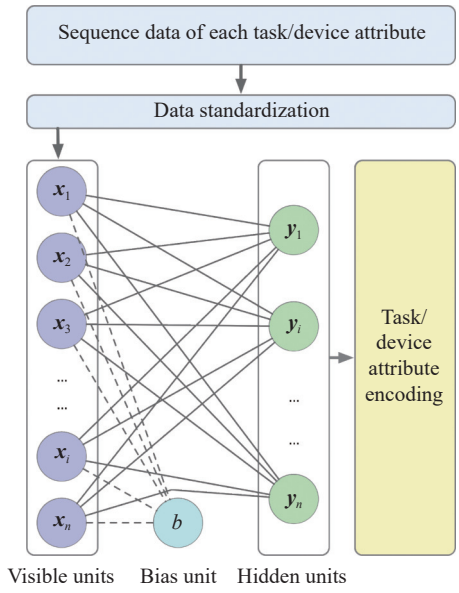


Fig. 1 RBM network encoding model

Set the state of the visible unit to the input sequence of attribute data. Update the state of the hidden unit using the following activation rule: for the jth hidden unit, calculate its activation energy $a_j = w_{ij} x_j$, and set a_j to 1 according to the probability $\sigma(a_j)$ derived from (2), and set a_j to 0 according to the probability $1 - \sigma(a_j)$. Then for each edge e_{ij} of the network, calculate $P(e_{ij}) = x_i x_j$,

i.e., for each pair of units, detect whether they are activated or not.

$$\sigma(x)=\frac{1}{1+\mathrm{e}^{-x}} \tag{2}$$

Similarly, reconstruct the visible units: for each visible unit, calculate its activation energy a_j and update its state (this reconstruction may not be consistent with the original preferences). Then, the hidden units are updated again, and $\hat{P}(e_{ij})=\hat{x}_i x_j$ is computed for each edge in the network. The weights of each edge e_{ij} in the network are updated using (3).

$$w'_{ij}=w_{ij}+L\left(P(e_{ij})-\hat{P}(e_{ij})\right) \tag{3}$$

where L is the learning rate.

$\hat{P}(e_{ij})$ is determined by taking the product of the ith and jth cells after reconstructing the visible cell once and then updating the hidden cell again. We can also take the product after more reconstructions, i.e., repeatedly updating the visible unit, then the hidden unit, then the visible unit again, which can more accurately describe the daydreams of the network. In addition to using $P(e_{ij})=x_i x_j$, where x_i and x_j are binary 0 or binary 1, one can also let x_i and/or x_j be assigned the activation probability value $\sigma(x)$. The same is true for $\hat{P}(e_{ij})$. We can penalize larger edge weights to obtain a sparser or more normalized model. When updating the edge weights, we use a momentum factor: the current step described above (i.e., $L(P(e_{ij})-\hat{P}(e_{ij}))$) and the weighted sum of the previous steps taken are added to each edge. Instead of using only one training example in each calendar time, we can use batches of examples in each calendar time and update the network's weights only after passing all the examples in the batch. This can speed up learning by using a fast matrix multiplication algorithm.

The output of the RBM network is the encoding of the attributes. An RBM network needs to be trained for each attribute and used for uniform encoding of task and device (or device group which will be described in Subsection 3.2) attributes.

The task and device address encoding algorithm is illustrated in Fig. 2. Task, containing some instance attributes, is the input part. All attributes of the task are encoded by Rb_i respectively. Combining all the features, the task unified code can be obtained. For a task or a device, the encoding of the attributes, also called attribute features, is obtained using the attribute encoding algorithm proposed above. Features are multiplied with the weight matrix ω to obtain the encoding. All attributes are weighted and summed, and the weight coefficients are equal to the weights of the attributes. The obtained sum vector characterizes this task, and the cosine angle between the vectors can be used to measure the distance between the corresponding tasks. In order to obtain an n-bit signature, the sum vector needs to be compressed. If a dimension of the sum vector is greater than 0, the corresponding bit of the final signature is binary 1, otherwise, it is binary 0. Finally, we obtain the uniform code (UC).

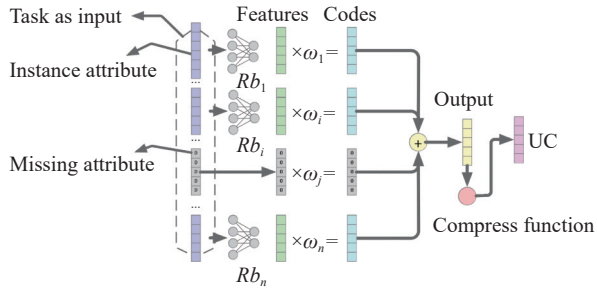


Fig. 2 Task/device encoding model

The function of a task (i.e., executable codes) is generally well defined before the system runs it. The attributes of the task also have a clear physical meaning. Therefore, a weight matrix ω can be determined based on the physical meaning of the task attributes.

The addresses of devices and tasks after uniform address encoding have the minimum value $\chi=\min(\mathrm{UC})=0$ and the maximum value $\psi=\max(\mathrm{UC})=2^n-1$. Let $\chi==\psi$, the addresses of devices and tasks are close together in the address space to form a ring pattern, as shown in Fig. 3.

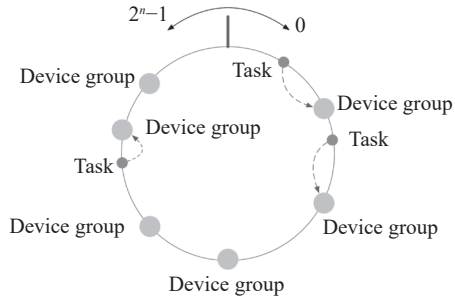


Fig. 3 Uniform address ring

The distance between addresses can be obtained using the cosine similarity in (4).

$$\begin{cases} D_{\mathrm{sim}}=\dfrac{A\cdot B}{\|A\|\|B\|} \\ D=1-\dfrac{\arccos(D_{\mathrm{sim}})}{\pi} \end{cases} \tag{4}$$

where A and B are the UC of a task or a device (group).

3.2 QoS for grouped device

There are various factors affecting DQoS, which can be roughly divided into two kinds of objective factors and subjective factors. Objective factors are factors related to the device's function, which are the essential factors determining the DQoS and are internal factors, such as

the operational efficiency of the device and the reliability of the device. Subjective factors are factors that are not related to the function of the equipment but can affect the rating of the device service quality and are external factors, such as the price of the device service (usually given by the device provider), and the service quality rating given by the device provider. We believe that the rating of device service should reflect both subjective and objective factors comprehensively but should be dominated by objective factors and supplemented by subjective factors. Because of the objectivity of the objective factors, it should be possible to measure them in real-time and objectively on any device. We choose the performance and reliability of the device as the primary objective factors to evaluate the DQoS.

The optimal target device that can meet the performance requirements is selected based on DQoS in the task-to-device routing process. The optimal target device satisfies the following two cases: (i) when there are more devices providing the same performance, the selection margin of the device is larger, and the DQoS score should be biased toward the device performance; (ii) when there are fewer devices providing the same performance, the selection margin of the target device is small, and the DQoS score should be biased toward the reliability of the device once the number of replaceable devices is small when the selected target device cannot provide the service properly. The relationship between performance and reliability is given in (5).

$$\mathrm{DQoS} = \rho S + (1-\rho) P \tag{5}$$

where $\rho \in [0,1]$ is a factor, S is the device stability, and P is the device performance. For ease of use, S and P are generally normalized at $[0,1]$, and then $\mathrm{DQoS} \in [0,1]$.

The device is considered as a single individual to measure DQoS, the DQoS obtained will not reflect the information of the number of services with the same function, and thus it is impossible to dynamically adjust the proportion of device performance and reliability in the DQoS calculation results according to the number of devices that meet the relevant requirements, and finally, it is difficult to obtain the optimal target device for device selection according to DQoS.

Firstly, calculate the addresses of all devices in the system according to the unified addressing method of devices proposed in Subsection 3.1, divide the devices with the same UC into a group. Then obtain the relative parameters of each device in the group. Finally, give the DQoS score of each device in the group to accurately and effectively reflect the actual QoS and select the best target device in the routing process. After grouping devices, the number of devices in a single group is reduced, which is conducive to improving the efficiency of DQoS calculation. In addition, after grouping the devices, the devices between different groups do not affect each other, which is conducive to improving the stability of the corresponding DQoS of a single device and facilitating the selection of the optimal target device in the task-to-device routing process. Therefore, we group devices with the same addressing into one group called a device group.

When evaluating devices within a group, the influence of the reliability of a single device within the group on its DQoS score should be jointly determined by the size of the device group and the reliability of all devices within the group. When the devices within the group are generally reliable, and the size of the group is large, the importance of the reliability of a single device will be reduced, and the influence of the reliability on its DQoS score will be reduced; conversely, the importance of the reliability of individual services will be more obvious. The performance of a device is also a relative concept, a measure of the relative performance among the devices in the group. It is not practical to examine the performance of a single device alone but to put the device in the device group and examine the device's performance in the group relative to other devices in the group. Because the first thing to consider when selecting a device is the availability of the device, i.e., reliability, it is meaningless to consider only the performance of the device in isolation from the reliability of the device, so the influence of the performance of a single device within the group on its DQoS score should also be combined with the reliability of the device.

Equation (6) is used to calculate the DQoS score of the ith device in a group.

$$\begin{cases} U = \dfrac{\sum_{j=1}^{n} S_j}{\sum_{j=1}^{n} C_j} \\ \overline{T} = \dfrac{t_i}{\sum_{j=1}^{n} t_j} \\ V = \left(1 - \dfrac{1}{n}\right) U \\ \mathrm{DQoS}_i = \overline{T} V + (1-V)\dfrac{S_i}{C_i} \end{cases} \tag{6}$$

where t_i is the total accumulated running time of the ith device in the group; n is the group size, $n \geqslant 1$; S_i is the number of successfully received execution tasks of the ith device in the group; C_i is the total number of received execution tasks of the ith device in the group.

In the application system based on service computing technology, the DQoS of each device in the group can be dynamically calculated according to (6) by collecting various parameters involved in (6) in real-time, such as the

execution time of the task on the device, and the number of device calls. After a task is routed to the device group, the priority of every device in the group will be determined according to the DQoS, and the device with high priority will execute the task first. The detailed process is described in Subsection 3.3.

3.3 Distributed point-to-point routing method

Any device in the system can generate (receive) a task and route the task to a device suitable for task execution. The task-to-device routing is done in two steps. Firstly, any task in the system route to a device group, and then the target device is selected within the device group based on DQoS.

According to the previous section, all tasks and devices within the system are distributed in an address space. The number of devices that can be accommodated in the system is related to the length of UC. For an n-dimensional UC addressing space, up to 2^n devices and/or tasks can be accommodated. Each device in the system keeps the UC addresses of other devices, which will result in an oversized device routing table and reduce the efficiency of lookup and discovery. We use a binary tree model to compress the routing table.

The routing table consists of multiple lists, each corresponding to one binary bit of the UC (e.g., with 128 bits of UC, the node's routing table would contain 128 lists). Each list contains multiple entries, containing some data necessary to locate other nodes. In this paper, the lists are referred to as D-Buckets. The length of a D-Bucket is generally a runtime constant, set by parameter δ.

Fig. 4 shows the process of task routing to a device group. The task generated by any device is used as the address code of the task by calculating the uniform code of the task according to the method described in Subsection 3.1. It is assumed that the ith D-Bucket is the closest among all D-Bucket in the current device routing table. To speed up the routing process, the maximum number of concurrent routing queries for a device is allowed by defining α as the query concurrency. Query messages are sent concurrently to each device in the ith D-Bucket, and then they send no more than β next-hop device UC lists back to the device from which the query was sent as a load of returned data. Each device has a pool to maintain the next-hop list, and β next-hop device UCs for each query response are put into the pool, i.e., a total of $\alpha\beta$ next-hops are put into the pool. If the remaining space in the pool is more extensive than $\alpha\beta$, then $\alpha\beta$ next-hops are put into the pool; otherwise, each next-hop device must be checked to see if it has already been queried. If so, the next-hop device will be discarded. When all next-hops in the pool are tracked, and no UC is found, the device tries to send a route lookup request for a re-selected device in the ith D-Bucket. Until the UC device group is found or there are no unlooked-up devices in the ith D-Bucket. If there are no unsearched devices in the ith D-Bucket and the device group with the same UC is found, the device group closest to the task UC during the route lookup is selected as the route result (target device group).

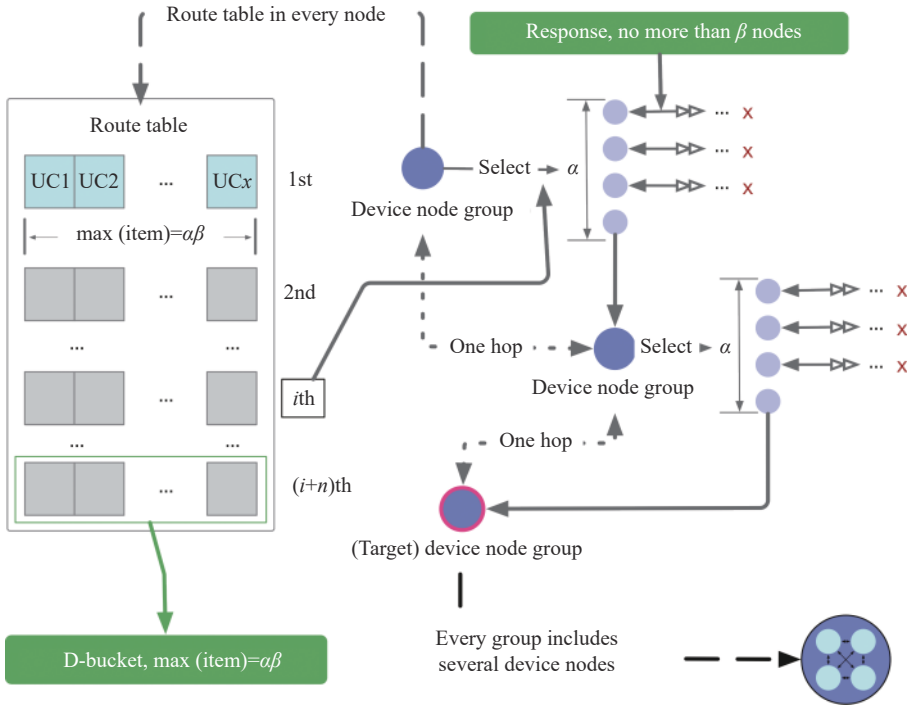


Fig. 4 Process of task route to device

The device selects α devices at the ith D-Bucket as the target for sending a route lookup request based on the round-trip time (RTT). When each device receives a route lookup response, the RTT between it and the responding device is calculated and stored in the D-Bucket for the next route lookup. Initially, since there is no RTT information between devices, the distance between UCs defined in (4) is used as the $\widehat{\mathrm{RTT}}$. Traditionally, additional communication is required to obtain RTT (e.g., RTT through active Internet control massage protocol), but the total number of messages increases. The communication frequency can be adjusted not to affect the available bandwidth, but the real-time performance of the system can be seriously affected. In this paper, we use network coordinate systems [27−30] to obtain RTT information between groups of devices.

When the routing query is sent back to the device from which the query is sent, a new entry is added during the iterative lookup if the ith D-Bucket of the device is not full. A device needs to be eliminated from the D-Bucket t if the ith D-Bucket is full. An existing entry ε_{old} is selected according to (7) and replaced with a new entry ε_{new}.

$$\begin{cases} T_{\text{rtt}}(\varepsilon_x, \varepsilon_{\text{new}}) < T_{\text{rtt}}(\varepsilon_x, \varepsilon_{\text{old}}) \\ \sigma^2\left(\dfrac{b_s^x}{\{\varepsilon_{\text{old}}\} \cup \{\varepsilon_{\text{new}}\}}\right) = \min\limits_{p_i \in b_s^x}\left\{\sigma^2\left(\dfrac{b_s^x}{\{\varepsilon_i\} \cup \{\varepsilon_{\text{new}}\}}\right)\right\} \end{cases} \tag{7}$$

where $T_{\text{rtt}}(\varepsilon_x, \varepsilon_y)$ is the RTT time from ε_x to ε_y, ε_x is the routing query issuing device, b_s^x is the sth D-Bucket of ε_x, and $\varepsilon_{\text{old}} \in b_s^x$. σ is the variance of the UC distance between neighboring devices. When the D-Bucket is full, i.e., the number of entries is δ, the replacement process of D-Bucket t in the device is executed. If ε_{new} satisfies the condition of (7), it replaces ε_{old}, always ensuring that the length of the D-Bucket does not exceed the parameter δ.

After the device issuing the task finds a device group suitable for task execution, it queries the DQoS of all devices in the group, selects the device with higher DQoS as the target device, and sends a start task request to the target device.

4. Experiments

4.1 Simulation setup

We use a self-managed cluster that contains sixty virtual machines with the exact specifications to complete the simulation experiment. Table 1 shows the relevant parameters of those virtual machines. These physical machines which locate virtual machines use single root I/O visualization (SR-IOV) to virtualize the network interface controller (NIC) (also known as a network interface card) for virtual machines to ensure that the network does not become a bottleneck for the experiment. Some of the large-scale inter-task communication efficiencies are compared using OverSim, which simulates the underlying and overlay communication. Many parameters can be defined in OverSim before the simulation, and the relevant parameters are shown in Table 2.

Table 1 Virtual machine configurations

Configuration	Value
Number of vCPU	8 cores @ AMD EPYC 7742 Processor
Type of vCPU usage	Host
Memory	16 GB @ DDR4 ECC 2666MHz
Network	10 GE @ Intel x710
Type of network interface usage	SR-IOV
Disk storage	50 GB @ Intel P4600 (shared in ZFS, RAID Z)

Table 2 OverSim parameters

Parameter	Value
Sim. Time/s	500
Churn model	Pareto churn
Lookup type	Iterative lookup
Application type	KBRTest

4.2 Comparison with the IP-based device address encoding method

In this section, to verify the performance advantage of our proposed method for communicating between a large number of tasks within cloud control systems, we compare the impact of the IP-based device addressing encoding (IP-DAE) and the use of the task addressing encoding (TAE) method proposed in this paper on the communication between a large number of tasks within cloud control systems.

The traditional approach is to give each device an individual address. This ensures that devices can communicate across the network. The communicable address of a device is generally an IP address. Tasks on a device share the device address, which is the device's IP address. The operating system of the device where the task is located generally identifies the local task address using a pair as (IP: Port). When a task is dispatched from one device to another by the scheduling system due to a failure or other reason of the device where the task is located, the task will get a new IP and port number pair as the new address through the operating system within the newly located device. Generally, the updated address is not the same as the original address.

To show that the direct task address encoding method proposed in this paper improves the efficiency of mutual discovery between tasks and the efficiency of interruption re-communication in a system where tasks are frequently scheduled, compared with the traditional approach of giving IP addresses to devices where tasks share IP addresses within devices, experiments in this section are designed and the results clarify the advantages of our proposed approach.

Communication between tasks latency generally consists of two parts, task lookup latency and data transfer latency. The communication data between tasks is 1 kB fixed data, and PING-PONG operation is performed between tasks. Since the received data latency recorded by the task may be affected by other factors, both ends of the communication only record the send latency as the end-to-end communication latency. During the experiment, no tasks are scheduled to ensure that the communication is not affected by task execution device switching.

The result is in Fig. 5. When there are few tasks (no more than 1 024) in the system, the device address encoding method can achieve higher communication efficiency with the help of the kernel's well-established protocol stack. However, as the task number increases, the task unified addressing encoding method has no kernel overhead and is more efficient in communication. In addition, in a massive tasks system, the IP-DAE method takes more time to lookup tasks, reducing communication efficiency.

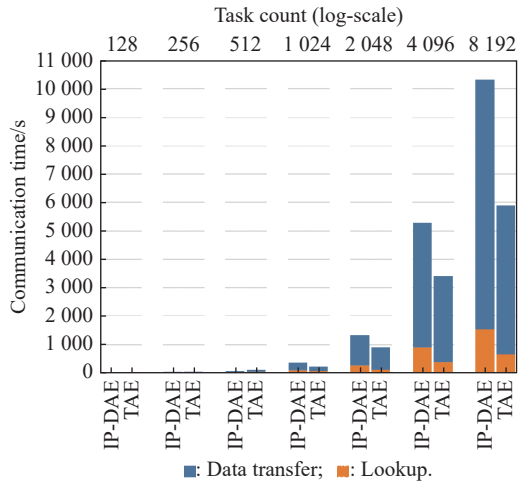


Fig. 5 Comparison with the IP-DAE

In summary, using a task address encoding algorithm to give each task an address, compared to giving a device address where all tasks share the same device address allows for more efficient inter-task communication within the cloud control system, especially with massive tasks. The size of the cloud control system can be further scaled up without the limitation of the device address space.

4.3 DQoS performance and reliability

4.3.1 Stability of DQoS calculation results for devices in a group

In this experiment, we define a device group and add a device to it. A simple addition task is used for simulation, and the time of each task performed is increased by a constant. The experimental result is shown in Fig. 6.

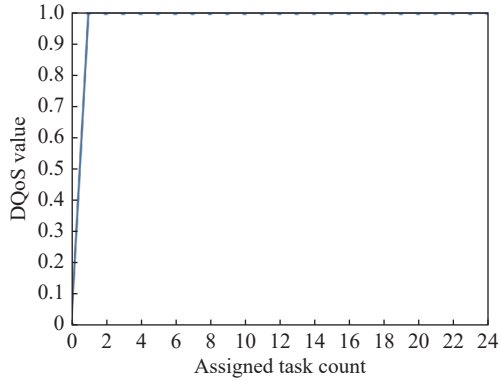


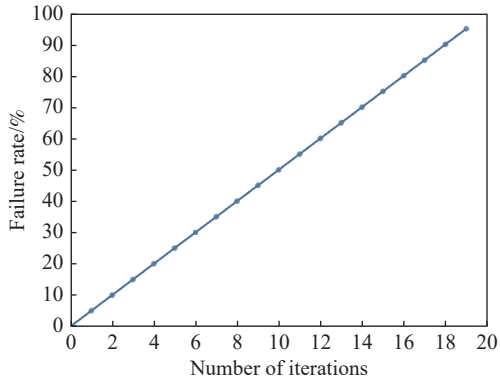
Fig. 6 Stability of DQoS

As the number of assigned tasks increases, the DQoS rises and stabilizes at 1.0. This experiment shows that the DQoS calculated using (6) is not affected by the device's performance when there is only one device in the device group. This result is in line with the actual situation when there is only one device in the group to be selected for the task. It also proves that the DQoS calculated using (6) is stable.

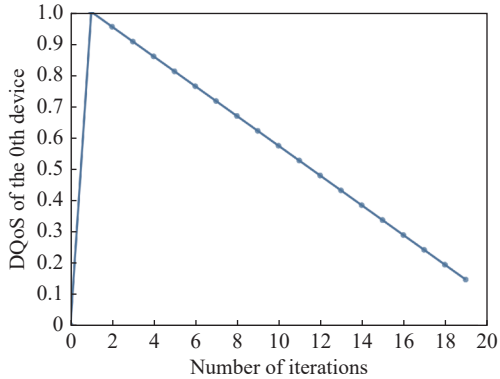
4.3.2 Impact of grouped device reliability and count on DQoS preferences

In this experiment, we define the device group and add 20 devices (virtual machines) with the same configuration, with serial numbers from 0 to 19. Twenty tasks are assigned to devices in the group, and the number of failed tasks created by devices increases by 5% each time, as shown in Fig. 7(a). At the same time, DQoS of the 0th device is recorded during each iteration, and the results are shown in Fig. 7 (b). The value of parameter $1-\rho$ of the 0th device is recorded in the DQoS process during each iteration, and the result is shown in Fig. 7(c). As Fig. 7(b) shows, the DQoS of the devices in the group calculated using (6) decreases as the percentage of the number of failed creation tasks of devices in the group increases. As the number of unreliable devices in the group increases, the reliability of the whole device group decreases in the external view. It can be seen that (6) pro-

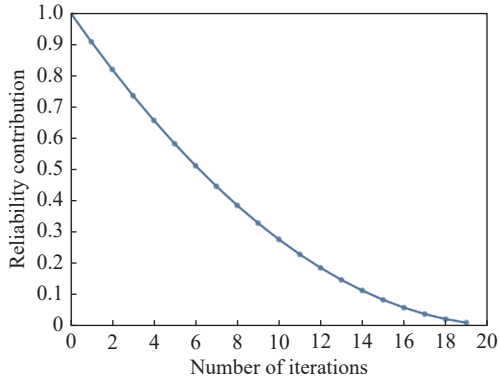
posed in this paper can correctly reflect that when the devices within the group are generally unreliable while the group size is large, the reliability of devices within the group becomes particularly important and increases the influence on the DQoS. In contrast, the influence of the performance index on the DQoS decreases, as shown in Fig. 7(c).



(a) The condition of some tasks fails in iteration



(b) DQoS of the 0th device in iteration



(c) Reliability contribution of the 0th device in iteration

Fig. 7 Impact of the grouped device reliability and count

We define a new device group. Firstly, a device (virtual machine) is added to it and successfully assigned a task once. Then, 29 devices with the same configuration are added to the continuation. Each newly added device is successfully assigned a task. The reliability contribution (parameter ρ) of all devices in the current group was measured after each addition and successful assignment of a task. The result is shown in Fig. 8.

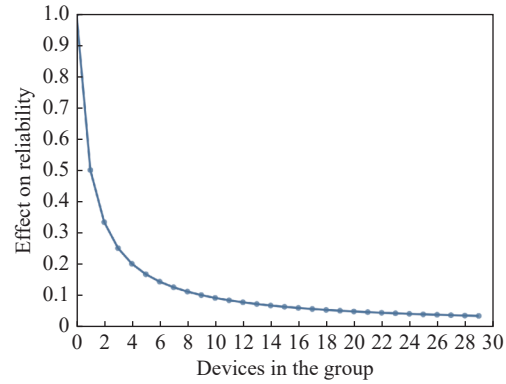


Fig. 8 Impact of the grouped device reliability when device count increases

As shown in Fig. 8, the reliability contribution decreases as the number of devices in the group increases. The DQoS within the group, calculated according to (6), reflects that the more devices with the same UC value (providing similar function), the larger the selection space for task routing to the devices, the smaller the impact on the device group from the reduced reliability of a small number of devices, and the smaller the DQoS change.

4.4 Comparison with Kubernetes

In this experiment, we use 60 virtual machines with the exact specifications (virtual machine specifications in Table 1) to build the Kubernetes cluster for testing, with Kubernetes cluster system version 1.19.2. Three of them are used to build the control plane (master nodes) and the remaining 57 to build the execution plane (worker nodes). In general, at least three servers are needed to build a highly available Et Cetera Daemon (ETCD) cluster. In this experiment, there is no need to consider the reliability of ETCD, and according to the operation principle of ETCD, reducing the number of ETCD service instances is beneficial to improve the performance of ETCD service instances [31]. To ensure test accuracy and reduce the impact of the ETCD component on the Kubernetes system [32,33], we use a high-performance server independent of the test cluster as the host for running ETCD instances. The ETCD server is interconnected with the test cluster using an 80GE network.

After the Kubernetes cluster is tested, 60 virtual machines are deleted, and then 60 virtual machines with the exact specifications (virtual machine specifications in Table 1) are recreated to test our proposed method. Sixty virtual machines are randomly divided into 20 groups,

with the number of devices in each group being random and not less than 1. After the test cluster is built, the cluster is given 10 min to start and initialize according to the principles of the distributed point-to-point task routing algorithm described in Subsection 3.3. The cluster is given 10 min to start and initialize according to the principle of distributed point-to-point task routing algorithm described in Subsection 3.3.

During the test, 10^5 test tasks are sent to the two clusters, and the time for the clusters to receive the tasks is recorded, as shown in Fig. 9. The time from the Kubernetes cluster receiving a task to making the corresponding one is the response time of the API-Server. The time from receiving a task to making a corresponding in the proposed system is from task reception to successful routing to the target device. The experimental results show that the execution efficiency of the cluster built by the distributed point-to-point task routing algorithm proposed in this paper is much higher than that of the Kubernetes cluster under the same hardware environment as the number of tasks in the system increases.

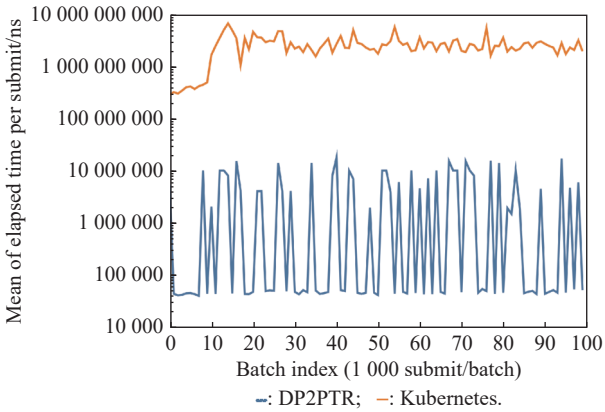


Fig. 9 Comparison with Kubernetes

5. Conclusions

In this paper, we first propose a unified addressing algorithm for tasks and devices, incorporating the properties of tasks and devices so that the binary codes obtained by the unified addressing algorithm have semantics. Through unified addressing, tasks and devices are distributed in one address space. The problem of scheduling tasks to devices for execution can be converted into the problem of routing tasks to the nearest devices. Second, considering many devices in the cloud control system, multiple devices may compute the same uniform addressing, and there is more than one device that can provide the same task execution capability. However, the task can only be eventually routed to one target execution device. We propose to group devices with the same uniform address into a group and determine the priority of the device when it is assigned a task based on the QoS of the device within the group, which is called DQoS. Then, based on the uniform address and DQoS, we propose a distributed point-to-point task routing algorithm. When a device generates or receives a task, it first uniformly codes the task, routes it to the device group according to the distributed point-to-point task routing algorithm, and selects the optimal execution device according to the DQoS of the device. Finally, through simulation experiments, it is verified that the unified coding algorithm improves the efficiency of the system platform, verifies the efficiency and stability of the DQoS algorithm within the system, and compares the distributed point-to-point task routing algorithm system platform with the Kubernetes platform to verify that the method proposed in this paper can effectively solve the system center bottleneck problem within complex cloud control systems.

As our future work, to improve the adaptability of the task address encoding algorithm, we will collect data related to the operation of the cloud control system and use a machine learning approach to obtain the weight matrix. Then, we will improve the distributed point-to-point task routing algorithm and increase the security of cloud control systems by introducing communication security protocols.

References

[1] XIA Y Q. Cloud control systems. IEEE/CAA Journal of Automatica Sinica, 2015, 2(2): 134–142.

[2] HINDMAN B, KONWINSKI A, ZAHARIA M, et al. Mesos: a platform for fine-grained resource sharing in the data center. Proc. of the USENIX Symposium on Networked Systems Design and Implementation, 2011. DOI: 10.1016/0375-6505(85)90011-2.

[3] BURNS B, GRANT B, OPPENHEIMER D, et al. Borg, Omega, and Kubernetes. Communications of the ACM, 2016, 59(5): 50–57.

[4] TANG C, YU K, VEERARAGHAVAN K, et al. Twine: a unified cluster management system for shared infrastructure. Proc. of the 14th USENIX Symposium on Operating Systems Design and Implementation, 2020: 787–803.

[5] BAIER J. Getting started with kubernetes. Birmingham: Packt Publishing, 2021.

[6] ZHU L L, HUANG K, HU Y F, et al. A self-adapting task scheduling algorithm for container cloud using learning automata. IEEE Access, 2021, 9: 81236–81252.

[7] CASQUERO O, ARMENTIA A, SARACHAGA I, et al. Distributed scheduling in kubernetes based on mas for fog-in-the-loop applications. Proc. of the 24th IEEE International Conference on Emerging Technologies and Factory Automation, 2019: 1213–1217.

[8] LAWRENCE J J, PRAKASH E, HEWAGE C. Kubernetes: essential for cloud transformation. Cardiff, Wales: Cardiff Metropolitan University, 2021.

[9] PAHL C, BROGI A, SOLDANI J, et al. Cloud container technologies: a state-of-the-art review. IEEE Trans. on Cloud Computing, 2019, 7(3): 677–692.
[10] BREWER E A. Kubernetes and the path to cloud native. Proc. of the 6th ACM Symposium on Cloud Computing, 2015. DOI: 10.1145/2806777.2809955.
[11] GOETHALS T, DETURCK F, VOLCKAERT B. Extending Kubernetes clusters to low-resource edge devices using virtual kubelets. IEEE Trans. on Cloud Computing, 2020. DOI: 10.1109/TCC.2020.3033807.
[12] HEIDARI A, NAVIMIPOUR N J. Service discovery mechanisms in cloud computing: a comprehensive and systematic literature review. Kybernetes, 2021, 51(3): 952–981.
[13] HOUMANI Z, BALOUEK-THOMERT D, CARON E, et al. Enhancing microservices architectures using data-driven service discovery and QoS guarantees. Proc. of the 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, 2020: 290–299.
[14] OLIVEIRA C, LUNG L C, NETTO H, et al. Evaluating raft in docker on kubernetes. Proc. of the International Conference on Systems Science, 2016: 123–130.
[15] NETTO H, PEREIRA OLIVEIRA C, RECH L O, et al. Incorporating the Raft consensus protocol in containers managed by Kubernetes: an evaluation. International Journal of Parallel, Emergent and Distributed Systems, 2020, 35(4): 433–453.
[16] NETTO H V, LUNG L C, CORREIA M, et al. State machine replication in containers managed by Kubernetes. Journal of Systems Architecture, 2017, 73: 53–59.
[17] FOGLI M, KUDLA T, MUSTERS B, et al. Performance evaluation of Kubernetes distributions (K8s, K3s, KubeEdge) in an adaptive and federated cloud infrastructure for disadvantaged tactical networks. Proc. of the International Conference on Military Communication and Information Systems, 2021: 1–7.
[18] TAHERIZADEH S, GROBELNIK M. Key influencing factors of the Kubernetes auto-scaler for computing-intensive microservice-native cloud-based applications. Advances in Engineering Software, 2020, 140: 102734.
[19] LIN C F, SHEU R K, CHANG Y S, et al. A relaxable service selection algorithm for qos-based web service composition. Information and Software Technology, 2011, 53(12): 1370–1381.
[20] GAO H H, HUANG W Q, DUAN Y C. The cloud-edge-based dynamic reconfiguration to service workflow for mobile ecommerce environments: a QoS prediction perspective. ACM Transactions on Internet Technology, 2021, 21(1): 1–23.
[21] HUANG J W, HU Z H. Multiple-signal prediction model for QoS of web services inspired by immune system. Journal of Guangxi University (Natural Science Edition), 2009, 34(4): 535–539. (in Chinese)
[22] HINTON G E, SALAKHUTDINOV R R. Reducing the dimensionality of data with neural networks. Science, 2006, 313(5786): 504–507.
[23] VARSAMOU M, ANTONAKOPOULOS T. Classification using discriminative restricted boltzmann machines on spark. Proc. of the International Conference on Software, Telecommunications and Computer Networks, 2019: 1–6.
[24] SALAKHUTDINOV R, MNIH A, HINTON G. Restricted boltzmann machines for collaborative filtering. Proc. of the 24th International Conference on Machine Learning, 2007: 791–798.
[25] COATES A, NG A, LEE H. An analysis of single-layer networks in unsupervised feature learning. Proc. of the 14th International Conference on Artificial Intelligence and Statistics, 2011: 215–223.
[26] HINTON G E, SALAKHUTDINOV R R. Replicated softmax: an undirected topic model. Advances in Neural Information Processing Systems, 2009, 22: 1607–1614.
[27] CARREIRA-PERPINAN M A, HINTON G. On contrastive divergence learning. Proc. of the International Workshop on Artificial Intelligence and Statistics, 2005: 33–40.
[28] FRANCIS P, JAMIN S, JIN C, et al. Idmaps: a global internet host distance estimation service. IEEE/ACM Trans. on Networking, 2001, 9(5): 525–540.
[29] COSTA M, CASTRO M, ROWSTRON R, et al. PIC: practical internet coordinates for distance estimation. Proc. of the 24th International Conference on Distributed Computing Systems, 2004: 178–187.
[30] SHARMA P, XU Z, BANERJEE S, et al. Estimating network proximity and latency. ACM SIGCOMM Computer Communication Review, 2006, 36(3): 39–50.
[31] DABEK F, COX R, KAASHOEK F, et al. Vivaldi: a decentralized network coordinate system. ACM SIGCOMM Computer Communication Review, 2004, 34(4): 15–26.
[32] LARSSON L, TARNEBERG W, KLEIN C, et al. Impact of etcd deployment on Kubernetes, Istio, and application performance. Software: Practice and Experience, 2020, 50(10): 1986–2007.
[33] JEFFERY A, HOWARD H, MORTIER R. Rearchitecting kubernetes for the edge. Proc. of the 4th International Workshop on Edge Systems, Analytics and Networking, 2021: 7–12.

Biographies

WANG Guan was born in 1986. He received his M.S. degree in computer science and technology from University of Ji'nan, China, in 2014. He is currently working toward his Ph.D. degree in the School of Automation, Beijing Institute of Technology, China. He was a lecturer in the School of Information Science and Engineering, University of Zaozhuang, China in 2019. His research interests include networking systems, cloud computing, gene expression data, and machine learning.
E-mail: netspecters@126.com

ZHAN Yufeng was born in 1989. He received his Ph.D. degree from Beijing Institute of Technology (BIT), Beijing, China, in 2018. He is currently an assistant professor in the School of Automation with BIT. Prior to joining BIT, he was a post-doctoral fellow in the Department of Computing with The Hong Kong Polytechnic University. His research interests include networking systems, game theory, and machine learning.
E-mail: yu-feng.zhan@bit.edu.cn

XIA Yuanqing was born in 1971. He received his Ph.D. degree in control theory and control engineering from Beijing University of Aeronautics and Astronautics, Beijing, China, in 2001. From January 2002 to November 2003, he was a postdoctoral research associate with the Institute of Systems Science, Academy of Mathematics and System Sciences, Chinese Academy of Sciences, Beijing, China. From November 2003 to February 2004, he was with National University of Singapore as a research fellow, where he worked on variable structure control. From February 2004 to February 2006, he was with University of Glamorgan, Pontypridd, U.K., as a research fellow. From February 2007 to June 2008, he was a guest professor with Innsbruck Medical University, Innsbruck, Austria. Since 2004, he has been with School of Automation, Beijing Institute of Technology, Beijing, first as an associate professor, then, since 2008, as a professor. His research interests include networked control systems, robust control and signal processing, and active disturbance rejection control.
E-mail: xia_yuanqing@bit.edu.cn

YAN Liping was born in 1979. She received her B.S. and M.S. degrees in mathematics from Henan University, Kaifeng, China, in 2000 and 2003, respectively, and Ph.D. degree in control science and engineering from Tsinghua University, Beijing, China, in 2007. From 2007 to 2009, she was a postdoctoral research associate with the Equipment Academy of Air Force, Beijing. Since 2009, she has been with the School of Automation, Beijing Institute of Technology, Beijing, first as an assistant professor, from 2011 to 2021 as an associate professor, and then, since 2021, as a full professor. From 2012 to 2013, supported by China Scholarship Council, she was a visiting scholar with the University of New Orleans, New Orleans, Louisiana, USA. From September 2018 to August 2019, she was a visiting scholar with the University of Windsor, Windsor, Ontario, Canada. Her research interests include multisensor data fusion, target tracking, fault detection, image registration, intelligent navigation, and integrated navigation.
E-mail: ylp@bit.edu.cn