QoS-Aware Task Scheduling in Cloud-Edge Environment

SHIDA LU1, RONGBIN GU1, HUI JIN2, LIANG WANG1, XIN LI2, (Member, IEEE), AND JING LI2
1Information and Communication Company, State Grid Shanghai Electric Power Company, Shanghai 200122, China
2College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China
Corresponding author: Xin Li (lics@nuaa.edu.cn)
This work was supported by the Information and Communication Company, State Grid Shanghai Electric Power Company, under Contract SGSHXT00JFS1900093.

ABSTRACT Since the limitations of the user equipment, they do not have enough computing power to process large amounts of data. At the same time, it’s unbearable for users to spend a long time uploading data to the remote cloud center. In order to solve these problems above, the concept of mobile edge computing (MEC) is proposed. The computing and storage resources are placed close to the end equipment, reducing the transmission delay. MEC can meet the high real-time requirements of the user equipment. In the real-time face recognition application scenario, a three-layer hierarchy MS-CE is proposed for the shortcoming of the traditional centralized cloud center. And the distributed MEC servers are utilized at the MEC layer to provide parallel computing capabilities. Aiming at the problem of how to perform task scheduling in a geographically distributed MEC server, we propose a task scheduling based queue algorithm (TSBQ), which considers the data transmission delay and server load, and carries out a reasonable task allocation policy. We evaluate the MS-CE and TSBQ through simulation experiments. We can find the MS-CE architecture are better than others and TSBQ is more effective than Corral and Greedy.

INDEX TERMS Task scheduling, mobile edge computing, cloud computing, QoS-aware.

I. INTRODUCTION
With the development and wide application of the internet of everything, the mobile devices and applications on the edge of the network are exploding, the volume of traffic and service data is also growing rapidly, and data-intensive applications are being extended to mobile terminals. Furthermore, the sensor-cloud system (SCS) integrates sensors, sensor networks, and cloud for managing sensors, collecting data, and decision-making. The integration provides a platform for monitoring and controlling the applications, which is flexible, low-cost, and the reconstructed. The platform includes emerging applications of the Internet of things (IoT), machine to machine, and cyber-physical systems. The demand for processing rate and Quality of Services (QoS) [1], [2] is also increasing. However, the limited energy and computing resources of User Equipment (UE) make it difficult to handle a large number of computing tasks in a short time. Furthermore, to save sensor resources, end-users are becoming more dependent on cloud processing and decision-making [3], [4].

Although cloud computing platforms can be used to store data and compute, centralized cloud computing has exposed its problems in the face of massive user device connections, explosive growth in data traffic and users’ increasing demand for better service quality. First, the real-time is not enough. In addition, the transmission bandwidth between the cloud and edge devices is insufficient.

In view of the shortcomings of traditional cloud computing and the demand of Internet of everything, the edge computing model emerges at the historic moment. Edge computing can reduce the delay of data transmission and better support the real-time performance of local business by virtue of its characteristics of being on the edge of the network and close to the mobile terminal equipment.

In May 2016, professor Shi Weisong’s team [5] presented a definition: edge computing refers a new kind of calculation model for computing at the network edge perform. The edge computing refers to any device or server that can perform computing in the path from the data source to the cloud center. In March 2018, alibaba that is a world famous brand of e-commerce launched “LinkEdge” [6], a product about the edge computing, which provides local services and forms...
a cloud edge computing system by combining with the cloud platform. In the same year, baidu also put forward “baidu Intelligent cloud Intelligent Edge” [7] (IoT Intelligent Edge), which extends the cloud computing capacity to the user end, and the Intelligent Edge cooperates with the cloud to form an integrated solution of the end cloud.

As a result of the attention from academia and industry, Edge Computing has made rapid progress in the Internet of everything service. Among them, one branch of Edge Computing is Mobile Edge Computing (MEC). The European Telecommunications Standards Institute (ETSI) published a report on mobile in 2014 White paper of edge computing [8], realizing the standardization of moving edge computing. MEC aims to reduce latency, ensure efficient network operation and service delivery, and improve the user experience. MEC is located in the wireless access network and close to mobile devices, enabling low latency, high bandwidth, and high quality service for users. Mobile edge computing emphasizes the establishment of edge servers between the cloud center and mobile edge devices. The computing tasks of mobile devices can be completed on the edge servers, and mobile terminal devices are considered to have no or only a small amount of computing power [9].

The main contributions are as follows:

- In the MEC system, the network architecture of cloud edge fusion is proposed. It makes use of the cloud’s powerful storage, query and computing capabilities to provide comprehensive query and computing for real-time operations. By taking advantage of the fact that MEC is close to UE, it can handle complex computing jobs submitted by UE, filter out irrelevant data for the cloud and make up for the shortage of UE resources.
- Based on the network architecture, a distributed task scheduling mechanism is proposed. Multiple MEC servers collaborate to handle complex computing jobs, work in parallel, and ultimately reduce the response time of jobs. We conduct simulation experiments to verify that this algorithm has better performance.

II. RELATED WORK

A. TASK SCHEDULING MECHANISM

Quincy [10] improves the throughput of the cluster to 40 percent by considering the data locality when processing concurrent jobs in the cluster. The purpose of the delay scheduling algorithm [11] is to make every job execute local task as much as possible, that is, the task and its input data are on the same machine. If the local task cannot be performed, the task will wait for a fixed time. If the required local data is not available within this time period, it can only start non local tasks. Delay scheduling can ensure the data locality and improve the number of processing tasks while achieving fairness. Corral [12] allocates different number of rack groups for each job to ensure that the input data or its copy must be placed in the rack group, and considers the data and task location to reduce the completion time and cross rack traffic of batch jobs.

III. TASK SCHEDULING MECHANISM BASED ON CLOUD EDGE FUSION NETWORK ARCHITECTURE

A. MOTIVATION

For user devices, the most suitable applications for uninstalling computing tasks are often those with an urgent need for computing resources. At the same time, the unloading of this task only requires a small amount of energy of UE for data transmission, so the task that needs to transfer a large amount of data is more suitable for processing in UE [13]. With the development of machine learning and artificial intelligence, the data acquired by the camera can be processed for face recognition, intelligent monitoring, image recognition and other scenes. On the other hand, with the popularity of users’ smart devices, the use of face recognition in applications has become common (for example, face payment, face login authentication, face beauty, etc.), and face recognition on UE has become a new trend.

The process of face recognition requires high real-time performance. If the process is too long, users will lose the convenience of using the application and the service quality will be reduced. However, face recognition requires the extraction of facial features, which is a complex computational process. The limited resources of UE cannot handle a large number of computation-intensive recognition tasks in a short time. Therefore, in order to solve the problem of limited UE resources, the traditional idea is to unload these recognition tasks to the cloud which is rich in computing resources. The cloud can store a large amount of face data in the data center, which can quickly carry out face recognition. However, the cloud center is often too far away from UE, so cloud-based face recognition applications may cause a long time link and delay between UE and the cloud, which cannot satisfy its real-time performance. In addition, the number of UE devices is huge, too many terminals are connected to the cloud server, the load in the cloud center is too heavy, and the processing speed and stability will decline.

In view of the above problems, this paper proposes a cloud edge fusion edge computing network architecture. Based on the traditional cloud computing architecture, combined with MEC technology, it provides services for new face recognition applications. MEC uses edge servers with computing and storage capabilities close to edge devices to provide cloud services for edge devices, so as to support real-time face recognition applications and reduce response delay. UE offloads computing tasks to a single edge server. Considering a large number of intensive computing tasks, the processing capacity of edge servers is still limited compared with the cloud server. Therefore, this paper proposes to use multiple geographical distribution of edge servers to complete the business together, taking into account the load and transmission delay of each server. And we study the geographic distributed computing task scheduling mechanism DTS-MEC in MEC system. It will schedule many computation-intensive tasks in the face recognition operation reasonably in each edge server.
B. A EDGE COMPUTING NETWORK ARCHITECTURE OF CLOUD EDGE FUSION MS-CE

Literature [14] describes the mobile edge cloud network architecture of face recognition applications. Face recognition applications can be divided into four modules: face detection (FD), image processing (IP), feature extraction (FE) and face recognition (FR). To reduce data transfer, FD is better run on UE than on MEC. Since FR requires frequent access to the database, a large amount of facial data can be placed in a database in the cloud, with FR at the center of the cloud. When a user uses a face recognition application, the face image is acquired through the camera on UE and the face is detected. The application receives the detected face data and generates a recognition job. The identification job is offloaded to the nearby edge server, which is then processed by the edge server and the cloud computing center (IPFE and FR). The edge computing network architecture of cloud edge fusion MS-CE proposed in this paper is shown in the Fig.1.

![FIGURE 1. A edge computing network architecture of cloud edge fusion.](image)

The architecture is divided into three layers [15]: UE layer, MEC layer and cloud computing layer. In the application of mobile face recognition, UE layer is composed of smart terminals with camera functions such as smart phones, tablets, laptops, etc. UE layer can access to nearby edge servers in MEC layer through wireless networks such as WiFi access points or base stations (BS). Cloud computing layer and MEC layer provide necessary services for UE accessing network.

MEC layer is mainly composed of base stations and edge servers, and the computing and storage capacity of these edge servers are much lower than that of cloud servers. In the face of a large number of computation-intensive tasks, the capacity of a single edge server is insufficient. When users use face recognition applications, UE offloads [16] the processed image information and recognition tasks to MEC, which processes the two modules of IPFE. As the tasks in IPFE are computation-intensive, a single edge server cannot reduce too much time when processing this process. Therefore, the MEC layer of MS-CE architecture consists of multiple edge servers.

The cloud computing layer is composed of the core cloud server cluster, which has a strong storage capacity and can be used to store a large amount of face data information and establish a face database. The feature information extracted from the MEC layer is uploaded to the central cloud, and the central cloud performs FR operation. The cloud center can not only provide a large amount of face comparison information by using the database, but also quickly carry out comparison and matching by virtue of its powerful computing power, and return the final result to the user.

C. TASK SCHEDULING MECHANISM TSBQ IN MS-CE

In face recognition, most tasks are computation-intensive. Although a single edge server has more computing power than UE, it is still weak in the face of DAG jobs. In order to further reduce the response time delay of jobs, this paper proposes a task schedule base queue (TSBQ) in MS-CE, which combines with multiple edge servers distributed in different geographical locations to carry out distributed parallel computing, so as to achieve the goal of low response delay of jobs in IPFE phase submitted by multiple users in MEC layer. In TSBQ, considering the load condition and data transmission delay of each server, the jobs in IPFE phase are divided into several sub-tasks, and the sub-tasks are reasonably assigned to each MEC server.

1) PROBLEM ANALYSIS

In the architecture shown in Fig.1, by using the technology of computing task offload, the services on UE are partially offloaded to edge layer, which can effectively reduce the overall response delay of services. However, the MEC layer in this architecture is composed of multiple geographically distributed edge servers. Unreasonable task scheduling strategy will cause too much data transmission delay. Therefore, a reasonable distributed task scheduling is needed to achieve the low latency target of UE submitted jobs in MEC layer.

Firstly, this paper proposes the following three hypotheses:

- Non-preemptive, that is, the running task cannot be interrupted before the end;
- Non-overload, that is, a edge server can only run one task at most at any time, and multiple tasks cannot run in a edge server at the same time;
- Each task can only be run on one server at any time.

The above three assumptions ensure that the task can only be executed on one edge server, and once it starts to execute, it occupies the resources related to face recognition application on the edge server and cannot be used by other face recognition related tasks.

To achieve the goal of low latency, we need to figure out how to schedule tasks across multiple edge servers. Because each sub-task in a DAG job requires intermediate data for its precursor task. Each precursor task may be executed on a different server, so we need to consider the delay required to transfer the intermediate data from the precursor task to
the server where the current task resides. Also, the load on
the server varies depending on the number of tasks assigned
to it. In order to achieve the goal of low latency of MEC layer
processing jobs, how to consider the load of each server and
allocate tasks to the server with low transmission delay and
light load as far as possible is a problem to be considered in
the proposed task scheduling mechanism.

2) PROBLEM FORMALIZATION
There are S edge servers in the MEC system. The connection
between edge servers can be regarded as an undirected graph
\( G_s = (V_s, E_s) \), where \( V_s \) is the collection of edge servers in
MEC layer and \( E_s \) is the communication link between edge
servers. In \( E_s \), each edge has a parameter \( B \), and \( B[k, l] \) is
the transmission bandwidth between server \( k \) and server \( l \).
In addition, the Directed Acyclic Graph (DAG) \( G_j = (V_j, E_j) \),
\( V_j \) is each sub-task node in the job, and \( E_j \) is the dependency
between each task. Weight \([m, n]\) is the size of the intermedi-
ate data between job \( m \) and job \( n \). Job \( m \) is the precursor of job
\( n \). The intermediate data is from job \( m \) to job \( n \). Jobs in DAG
must wait until their precursor tasks are complete before they
can be executed. Fig.3 is an example of a DAG job.

Based on the MS-CE architecture, when several users use
face recognition applications, UEs will submit the DAG jobs
and input data in the IPFE phase to the MEC layer after the
basic face detection. MEC layer will receive a job set \( J \) from
the user, and \( J \) has \( j \) jobs. In the batch scenario, jobs in \( J \)
are submitted at the same time, with the goal of minimizing
the completion time (the time \( J \) takes from submission to
complete all tasks). In online scenarios, the goal is to min-
imize the average response time (the response time of the job
is the difference [17] between the submission time and the
completion time). The structure of the TSBQ task scheduling
system is described in detail below.

3) THE STRUCTURE OF TASK SCHEDULING SYSTEM
Fig.2 shows the structure of the task scheduling system in
MS-CE. The system is a cluster architecture consisting of
a master and multiple slaves. The edge server that receives
the recognition job from UE acts as the master and performs
the task while scheduling the task. Other edge servers act as
slaves and are only responsible for the execution of the task.
In each edge server, a waiting queue \( TQ \) and a standby queue
\( SQ \) are maintained. \( TQ \) is the task queue that the edge server is
waiting to execute, and \( SQ \) is the task queue that is assigned to
the edge server but whose input data has not been transferred.
In addition, the master maintains information about the entire
job, such as the dependencies between the task nodes and
whether the task has been processed.

Next, the processing process of the task scheduling system
is introduced according to the steps in the Fig.2:
(1) UE submits the DAG job and uploads the input data to
the nearby base station. The edge server that receives the
job and data through the base station will conduct task
scheduling as the master, while other edge servers will
act as slave.
(2) The master assigns the slave server for the tasks that can
be performed in the current stage according to the TQ
queuing situation and data transmission delay in each
server.
(3) Then, each slave servers receive the task assigned by
the master, it first checks whether all the input data is
available locally. If there is no data or data is missing,
the task first enters \( SQ \) and waits for the arrival of data.
Else if there are all input data, the task will enter \( TQ \)
directly and wait for execution.
(4) After receiving all the input data of the \( SQ \) task in the
edge servers, the task pops up from \( SQ \), enters \( TQ \) and
wait for execution.
(5) After the team head task of the \( TQ \) of the edge server
is completed, it pops up from the queue and informs the
master that the task has been completed.
Each time the master server receives the completion infor-
mation for all the assigned tasks in the current phase, it con-
tinues to perform steps 3-5 until the entire job is completed.

4) TSBQ TASK SCHEDULING MECHANISM
Due to the nature of DAG jobs, each task needs to wait for
its precursor tasks to complete before execution can begin.
Fig.3 shows an example of a running DAG job.
The whole DAG is divided into several stages. The lower stage has a dependency on the upper stage, and the next stage task cannot be scheduled until all the tasks in the upper stage are completed. For example, stage 2 has a dependency on stage 1, and the tasks in stage 2 cannot be scheduled and run until the tasks in stage 1 are completed. The stage where the task in the current stage can run is available, and all stages in the lower level are unavailable. When all the tasks in the available stage are completed, the next stage becomes available, and the master assigns the slave servers to all the tasks in that stage. Tasks in the available stage can be divided into three categories: running, waiting and standby. Running task is the task running on the edge server; The waiting task is the task waiting for execution in the TQ queue, for which all input data have been received and all precursor tasks have been completed. Standby task is a task in the SQ queue waiting for the incoming data to arrive.

The master server collects information about TQ and SQ in each server before assigning MEC servers for tasks in the available stage. The task information waiting to be scheduled is stored in TQ, and the information of each task in SQ can be represented by a triple (jobId, taskId, ptime). The jobId is the job which the task belongs, the taskId indicates the sequence number of the task in the job, and the ptime is the processing time of the task (the processing time is known because the task characteristics in the DAG job in the real-time face application scenario are known). The edge servers execute tasks in the order in which they arrive at TQ. Task information waiting for the arrival of data is stored in SQ, and the information of each task in SQ can be represented by a quad (jobId, taskId, ptime, atime). The first three variables are the same as TQ, and the fourth, atime, is the time when all the intermediate data required for the task arrives at the server. After all the intermediate data of the task in SQ has arrived at the server to which it is assigned, the task pops up from SQ and goes into TQ to wait for scheduling. Since the location of the precursor task of the task node in the available stage is known, the time each task needs to be assigned to each edge server is known.

The Alg.1 shows the process of task scheduling in available stage when a job $J_j$ performs the task scheduling. First, get the task $t_i$ (line 1) in the currently available stage in job $J_j$, and the $t_i$'s $sid$ is the stage sequence number of the task(line 2). Next, start looping through the edge server set $S$ (line 3) to find the most suitable server for the task $t_i$. If the server location of the precursor task of the task node is not consistent with the server $S(s)$ of the current loop, calculate the transmission delay of intermediate data $\text{Weight}(t_p, t_i)$ from $t_p$'s $pos$ (the execution location of precursor task $t_p$) to $S(s)$ between $t_p$ and $t_i$, and find the maximum transmission delay $\text{transmiss}$ (line 7-11). Then, calculate how many tasks in the SQ queue in edge server $S(s)$ to be executed (lines 14-22). Finally, the server with the shortest wait delay for $t_i$ is selected (lines 23-30).

Algorithm 1 TSBQ Task Scheduling Mechanism

Input:
- $J_j$ denotes job $j$;
- $\text{stageId}$ denotes available stage sequence in job $j$;
- $\text{time}$ denotes current time;
- $S$ denotes server set;
- $\text{TQlist}$ denotes TQ set;
- $\text{SQlist}$ denotes SQ set;
- $C$ denotes number of CPU cycles required by task;
- $F$ denotes frequency of calculation of server;
- $\text{B}$ denotes bandwidth between servers.

Output:
- $P$ denotes allocation position of task.

1: while $t_i \in J_j.$Nodes do
2: if $t_i.$sid $==$ $\text{stageId}$ then
3: $t_x \leftarrow \infty$;
4: $\text{pos} \leftarrow S(1)$;
5: for $s = 1 \leftarrow S.length$ do
6: $\text{tmp} \leftarrow 0$;
7: $\text{transmiss} \leftarrow 0$;
8: while $t_p \in t_i.$pred do
9: if $t_p.$pos $\neq S(s)$ then
10: $\text{transmiss} \leftarrow \max(\text{transmiss}, (J_j, \text{Weight}(t_p, t_i)/B(S(s), t_j.$pos)));
11: end if
12: end while
13: tsq $\leftarrow 0$;
14: while $\text{SQlist}(s).\text{isEmpty} = false$ and $s_q \in \text{SQlist}(s)$ do
15: if ($\text{time} + \text{transmiss}) \leq s_q.$atime then
16: tsq $\leftarrow tsq + s_q.$ptime;
17: end if
18: end while
19: $\text{tmp} \leftarrow \text{tmp} + tsq + \text{transmiss};$
20: while $\text{TQlist}(s).\text{isEmpty} == false$ and $t_q \in \text{TQlist}(s)$ do
21: $\text{tmp} \leftarrow \text{tmp} + t_q.$ptime;
22: end while
23: if $t_x.$tmp then
24: $\text{pos} \leftarrow S(s);$,
25: $t_x \leftarrow \text{tmp;}
26: end if
27: end for
28: $t_i.$pos $\leftarrow \text{pos};$
29: $e_p.$add($t_i$, pos);
30: end if
31: end while
task into the \( TQ \) or \( SQ \) queue based on whether the machine has all the input data of the assigned task.

IV. EVALUATION

In order to verify that MS-CE can effectively reduce the response delay of jobs, this paper compares the delay of MS-CE, traditional cloud computing architecture and a single edge server in processing face recognition jobs. In addition, in order to verify the effectiveness of the task scheduling strategy in MS-CE in processing DAG jobs in the geographically distributed environment, the task scheduling strategies in MS-CE are compared with the traditional Greedy algorithm and Corral algorithm in batch and real-time scenarios respectively over the completion time and average delay of the real-time face recognition operations.

In the simulation, we adopt simulation platform MATLAB. In the real network environment, the data transmission speed between each edge server is different, so the bandwidth between servers are randomly generated during simulation. The response time of a job consists of the data transfer time between servers and between servers and devices and the queued time waiting for tasks to execute.

The design of relevant parameters in the simulation experiment is set referring to reference [18]. The data related to the face recognition job in the experiment is simulation design. The photos taken by the camera on the user’s device are 3M Bytes, the relevant data of face detected by the user’s device is 500K Bytes, and the feature data extracted after IPFE stage is 1K Bytes. The number of CPU cycles of all tasks in the recognition business (FD task in the face recognition business and sub-task in the DAG job in the IPFE phase) is 1300M. The face recognition (FR) phase is performed in the cloud center, where a large amount of face data is stored for matching and comparison, and the time required for the FR phase is set to 0.1s. Some simulation parameters are shown in table 1. The edge server group J in MS-CE is designed with \( J = 10 \) edge servers. During the experiment, the specific number of submitted jobs and the number of tasks in each job are set according to different situations.

| TABLE 1. Network parameter design. |
|-------------------------------------|
| **Computation speed (GHz)** | **Bandwidth between cloud (Mbps)** | **Bandwidth between MEC servers (Mbps)** |
| UE | MEC server | cloud |
| 2 | [10,20] | 5 |
| 100 | 20 | 200 |

A. PERFORMANCE ANALYSIS OF MS-CE NETWORK ARCHITECTURE

Next, based on the MS-CE network architecture model, this simulation experiment compares the ability to schedule face recognition tasks under the cloud-side collaborative architecture with the ability to schedule face recognition tasks under the traditional cloud computing architecture or under the edge architecture with a single edge server. In the traditional cloud computing architecture, UE will not detect the acquired face image, but directly upload it to the cloud computing platform for processing. In the edge network architecture of a single MEC server and MS-CE architecture, UE first detects the face, then upload the data to the MEC server, and at the same time submit the DAG job in the IPFE stage. After the processing of the DAG job is completed in the MEC layer, the extracted features are uploaded to the cloud computing platform for face recognition. The number of edge servers in the MEC layer of the MS-CE architecture is designed to be 10, and the number of sub-tasks in each DAG job is 10. In a single MEC server architecture, UE selects the s1 server as the object to unload the task. In this simulation, the time from the submission to the end of real-time face recognition business in three architectures is mainly compared, that is, the business completion time of the four processes of face recognition application (the time period from the acquisition to the image source to the completion of face recognition). For the cloud computing architecture, the completion time of the business is mainly determined by the time of image source data transmission and the time of face recognition application in the cloud computing platform. The completion time is composed of the processing time of source data by user equipment, the processing time of DAG task in the stage of IPFE at MEC layer, the processing time of face recognition at cloud center, the transmission time of intermediate data from UE to MEC layer and the time of characteristic data from MEC layer to cloud computing center. The simulation results are shown in Fig.4.

![FIGURE 4. Performance comparison of MS-CE network with cloud computing architecture and a single MEC server.](image)

As shown in Fig.4, when the number of batch jobs is less than 2, there is not much difference in the response time of the three to multiple business requests. This is because the cloud server has a strong computing capacity, which can quickly complete the business processing of face recognition applications. In addition, the amount of image source data that needs to be transmitted is also small, and the transmission delay...
is small. When the single edge server architecture handles a small number of DAG jobs in the IPFE phase, although its computing power is weak, it has little impact on the overall completion time of the job. Compared with MS-CE and cloud computing architecture, it does not have the delay of data transmission inside the MEC layer. But with the increase of the number of users’ requesting batch jobs, although a single MEC server won’t produce too much of the overhead of data transmission, the ability to handle a large number of DAG homework is insufficient. Even it brings the advantage of its location which was covered with the lack of their computing power, the completion of the business of time is longer than the traditional cloud computing architectures. UE uploading a large amount of sensing source data to the remote public cloud causes long transmission delay. Even though it has powerful computing resources and can quickly handle a large number of business requests, its overall business completion time is not low. Although the completion time of cloud computing architecture is lower than that of the single edge server, those have the problem of high transmission delay and high computing delay respectively, so the difference between them is not large. Compared with the MS-CE, because the MS-CE combines multiple edge servers for parallel computing, the computing power is stronger and the response time is faster than the single edge server. Although there is a small amount of data transfer delay in the MS-CE, by virtue of the task scheduling mechanism, other tasks can be started first in the process of data transmission to improve the CPU utilization and throughput of the entire network architecture. The MS-CE is not only close to UE geographically, which reduces the transmission delay of input data, but also further improves the utilization of CPU and reduces the completion time of business through TSBQ task scheduling mechanism.

In addition, as you can see from Fig.4, the overall business completion time of the cloud computing architecture and individual edge servers increases proportionally with the number of business requests. In contrast, the MS-CE has a relatively smooth time increase rate. This is because DAG jobs in the IPFE phase of MS-CE can be run in parallel in multiple MEC servers, thus reducing business completion time. The single edge server and the traditional cloud architecture can only perform one task per time period. The edge server of MS-CE is deployed on the edge of the network, which is close to UE and has a short data transmission time. Although the computing power of the single edge server in MS-CE is far less than that of the cloud server, the computing power of multiple edge servers is integrated computing power is about the same as cloud computing architecture. It can be seen that the architecture of the single edge server and cloud computing are limited by computing power in the case of batch DAG jobs, and the MS-CE solves this problem. When the number of DAG jobs was 100, the completion time of MS-CE network architecture was 70.8 percent and 76.3 percent lower than that of cloud computing architecture and edge network architecture of the single edge server, respectively. The MS-CE edge network architecture not only has excellent response delay performance, but also can effectively reduce the traffic of core network and avoid the occurrence of link congestion in actual networks.

Therefore, the MS-CE edge network architecture can be applied to real-time mobile face recognition scenarios, which can effectively reduce the response time of the overall business and provide better QoS.

B. PERFORMANCE ANALYSIS OF TSBQ TASK SCHEDULING ALGORITHM

This section simulates the TSBQ task scheduling algorithm, and compares it with the following algorithm in batch scenario and online scenario:

- The Greedy algorithm: DAG jobs execute in topological order, each time assigning the first idle edge server to the task. A task can be scheduled only if its previous tasks have been completed.
- The Corral algorithm: According to the task scheduling algorithm proposed in literature [12], jobs are assigned to appropriate edge server group based on all the known information of task characteristics, and the jobs complete the tasks in the edge server group. According to the relevant parameter Settings of the above simulation experiment, the feature information (such as processing time, etc.) of DAG job in face recognition scene is all known. In addition, when Corral processes DAG jobs, it divides the jobs into multiple stages as shown in Fig.3, and each stage can be regarded as a MapReduce job to calculate the delay of each stage.

When comparing the performance of TSBQ with different task scheduling algorithms, the delay performance of different task scheduling algorithms in the MEC layer for DAG jobs in the IPFE phase when UE is submitted to the MEC layer is mainly compared. In the batch scenario, the completion time of the whole job when processing batch DAG task is compared, from the time of submission (0 time) to the last task completion time in the batch task set J. In the real-time scenario, the average response time of multiple jobs is compared. The response time of a DAG job is the difference from the time that the job was submitted (any time) to the time that the job was completed.

1) BATCH SCENARIO

Under the batch scenario, the job is submitted at the same time. Fig.5(c) shows the simulation results of the three algorithms in 10 jobs, with each DAG job being 10. Both TSBQ and Corral completed their first job relatively early, while the Greedy algorithm completed its first job 10 seconds later. This is because the Greedy algorithm needs to be executed in the order of topological sort. Only one task can be running for each job in the MEC layer at the same time. Multiple jobs are running in parallel. Therefore, the CDF of the Greedy algorithm is basically a vertical line, with a short difference between the time to complete the first task and the time to complete the last task. Both Corral and TSBQ algorithm are hierarchical in job scheduling, and the internal tasks of the job
FIGURE 5. Completion time comparison of TSBQ, Greedy and Corral algorithms. (a) The impact of the number of jobs on the completion time. (b) The impact of the number of tasks on the completion time. (c) Batch jobs CDF.

are processed in parallel, so the first job can be completed as soon as possible.

Moreover, the overall completion time of the TSBQ is shorter than the Greedy algorithm and the Corral algorithm. It is due to the Corral algorithm and the Greedy algorithm are adopting the tactics of the First Come First Server (FCFS). All the input data need to wait until the task after arrival to run the job, and waiting for the data transfer during the period, edge server CPU idle condition, cause waste. The TSBQ algorithm considers that when transferring data, tasks with input data can be run first, breaking the arrival order, avoiding the occurrence of CPU no-load, reducing the waiting time for scheduling, thus improving the completion time of the overall job. In addition, although both the Corral and the Greedy algorithm let the CPU enter the no-load state, the Corral is the CPU of multiple edge server groups waiting for the data of all the tasks of the current stage to start execution, and the Greedy algorithm only means that the CPU of a single edge server enters the waiting state. Therefore, the Corral completed the first job faster than the Greedy algorithm, but the overall completion time of multiple jobs was longer than the Greedy algorithm. As can be seen from the figure, the Greedy algorithm began to overtake the Corral algorithm after completing about 43 percent of the jobs. The TSBQ algorithm proposed in this paper reduced the overall completion time of the Corral and the Greedy algorithm by 67.8 percent and 57.1 percent, respectively.

The Fig.5(a) and Fig.5(b) respectively show the simulation results of the three algorithms in the number of batch jobs (the number of tasks for each job is set to 10) and the number of jobs (the number of jobs is 10). When the number of jobs or tasks is small, the overall completion time of the three algorithms is basically similar, but with the increase of the number of jobs, the completion time of the three algorithms also increases gradually, and the gap between the three algorithms is getting bigger and bigger. Among them, since Corral is assigned to each stage as a server group, the servers in the MEC server group need to wait for the transfer of all the tasks before starting to process the tasks. As a result, an excessive wait delay can result in a longer overall completion time, with a larger overall response time than both the Greedy algorithm and the TSBQ. Although the waiting time of the Greedy algorithm is not as much as that of Corral, the total waiting time increases with the increase of the number, and the gap with the TSBQ gradually becomes larger. When the number of batch jobs is 100, the completion time of TSBQ algorithm is 69.8 percent and 49.8 percent lower than that of Corral and Greedy algorithm, respectively. When the number of tasks per job was 100, the completion time decreased by 61.4 percent and 43.7 percent, respectively, compared with the Corral and the Greedy algorithm.

Thus, in the batch scenario, the TSBQ task scheduling algorithm is faster than the Corral and the Greedy algorithm, with shorter overall completion time and better service.

2) ONLINE SCENARIO
In the online scenario, jobs are submitted randomly, and the submission time is distributed randomly within the interval of [0,30]. Set a total of 50 jobs to be submitted randomly, with 10 tasks per job. The Fig.6(c) shows the cumulative distribution function (CDF) of the job completion time of the three algorithms in the simulation environment. Because each job is submitted randomly, there may be only one job
that needs to be processed by the MEC layer for a period of time. The Greedy algorithm performs all of the tasks in the job on a single edge server without the delay of data transfer. However, the Corral and the TSBQ algorithms are layered so that multiple tasks are processed in parallel on different servers, which increases the delay of data transmission. Therefore, the response time of the Greedy and the TSBQ algorithms is basically the same, and even more because of the response delay performance of the TSBQ. In addition, compared with the TSBQ algorithm, multiple edge servers need to wait for data transmission, so the transmission delay is much higher than that of TSBQ and the response delay is longer. However, although multiple jobs are submitted at different times, there are always situations where multiple jobs need to be processed in parallel, so the TSBQ is better than the other two algorithms in terms of the overall job completion time.

TABLE 2. Average response time of the three algorithms.

|         | TSBQ     | Corral   | Greedy   |
|---------|----------|----------|----------|
| Time(s) | 14.24s   | 22.15s   | 16.42s   |

The Tab.2 shows the average response time of the three algorithms, and it can be found that the response time of the TSBQ is improved by 4.7 percent and 9.4 percent compared with that of the Corral and the Greedy algorithm. It can be seen that although there is a small gap between the three algorithms, the average response delay of the TSBQ is slightly lower than that of the Corral and the Greedy algorithm.

The Fig.6(a) and Fig.6(b) show the impact of the number of jobs (the number of tasks for each job is 10) and the number of jobs (50 jobs in total) on the average response time of the three algorithms in the online scenario. With the increase of the number of jobs, the average response time of all three algorithms increased. Among them, the Corral grew the fastest and was much higher than the Greedy algorithm and the TSBQ. The difference between the Greedy and the TSBQ algorithm is small, but as the number increases, the difference with the TSBQ gradually increases. As the number of tasks per job increased, the average time of those three algorithms gradually increased. Among those algorithms, the Corral grew the fastest and the TSBQ grew slowest. It can be seen that the delay performance of the TSBQ is better than the other two algorithms in the online scenario.

V. CONCLUSIONS

In order to verify that the MS-CE architecture can effectively reduce the response delay of jobs, this paper compares the delay of the MS-CE, the traditional cloud computing and the single edge server architecture in processing face recognition jobs. In addition, in order to verify the effectiveness of the task scheduling strategy at the MS-CE architecture in processing DAG jobs in the geographically distributed environment, the task scheduling strategies in MS-CE are compared with the traditional Greedy algorithm and Corral algorithm in batch and real-time scenarios respectively over the completion time and average delay of the real-time face recognition operations.

REFERENCES

[1] Q. Liu, Y. Peng, J. Wu, T. Wang, and G. Wang, “Secure multi-keyword fuzzy searches with enhanced service quality in cloud computing,” IEEE Trans. Netw. Service Manage., early access, Dec. 17, 2020, doi: 10.1109/TNSM.2020.3045467.
[2] Q. Liu, Y. Peng, S. Pei, J. Wu, T. Peng, and G. Wang, “Prime inner product encoding for effective wildcard-based multi-keyword fuzzy search,” *IEEE Trans. Services Comput.*, early access, Sep. 1, 2020, doi: 10.1109/TSC.2020.3020688.

[3] Q. Liu, Y. Tian, J. Wu, T. Peng, and G. Wang, “Enabling verifiable and dynamic ranked search over outsourced data,” *IEEE Trans. Services Comput.*, early access, Jun. 11, 2019, doi: 10.1109/TSC.2019.2922177.

[4] Q. Liu, P. Hou, G. Wang, T. Peng, and S. Zhang, “Intelligent route planning on large road networks with efficiency and privacy,” *J. Parallel Distrib. Comput.*, vol. 133, pp. 93–106, Nov. 2019.

[5] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[6] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, “Mobile edge computing: A key technology towards 5G,” *IEEE Access*, vol. 6, pp. 261–276.

[7] W. S. Shi, X. Z. Zhang, and Y. F. Wang, “Edge computing: Current situation and outlook,” *Comput. Res. Develop.*, vol. 56, no. 1, pp. 69–89, 2019.

[8] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, “Mobile edge computing—A key technology towards 5G,” ETSI White Paper, vol. 11, no. 11, pp. 1–16, 2015.

[9] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, “Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling,” in *Proc. 5th Eur. Conf. Comput. Syst.*, 2010, pp. 265–278.

[10] P. Mach and Z. Becvar, “Edge computing: A survey,” *IEEE Access*, vol. 5, no. 1, pp. 407–420, 2015.

[11] J. Liu, Y. Yao, J. Zhang, and B. Letaief, “Delay-optimal computation task scheduling for mobile-edge computing systems,” in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2016, pp. 1451–1455.

[12] P. Mach and Z. Becvar, “Mobile edge computing: A survey on architecture and computation offloading,” *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 3rd Quart., 2017.

[13] S. Wang, M. Zafer, and K. K. Leung, “Online placement of multi-component applications in edge computing environments,” *IEEE Access*, vol. 5, pp. 2514–2533, 2017.

[14] J. Liu, Y. Tian, J. Wu, T. Peng, and G. Wang, “Enabling verifiable and dynamic ranked search over outsourced data,” *IEEE Trans. Services Comput.*, early access, Jun. 11, 2019, doi: 10.1109/TSC.2019.2922177.

[15] Q. Liu, P. Hou, G. Wang, T. Peng, and S. Zhang, “Intelligent route planning on large road networks with efficiency and privacy,” *J. Parallel Distrib. Comput.*, vol. 133, pp. 93–106, Nov. 2019.

[16] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[17] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, “Mobile edge computing: A key technology towards 5G,” *IEEE Access*, vol. 6, pp. 261–276.

[18] W. S. Shi, X. Z. Zhang, and Y. F. Wang, “Edge computing: Current situation and outlook,” *Comput. Res. Develop.*, vol. 56, no. 1, pp. 69–89, 2019.

[19] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, “Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling,” in *Proc. 5th Eur. Conf. Comput. Syst.*, 2010, pp. 265–278.

[20] P. Mach and Z. Becvar, “Mobile edge computing: A survey on architecture and computation offloading,” *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 3rd Quart., 2017.

[21] S. Wang, M. Zafer, and K. K. Leung, “Online placement of multi-component applications in edge computing environments,” *IEEE Access*, vol. 5, pp. 2514–2533, 2017.

[22] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, “Mobile edge computing: A survey,” *IEEE Internet Things J.*, vol. 5, no. 1, pp. 450–465, Feb. 2018.

[23] M.-H. Chen, B. Liang, and M. Dong, “A semidefinite relaxation approach to mobile cloud offloading with computing access point,” in *Proc. IEEE 16th Int. Workshop Signal Process. Adv. Wireless Commun. (SPAWC)*, Jun. 2015, pp. 186–190.

[24] X. Zhang, Z. Qian, S. Zhang, Y. Li, X. Li, X. Wang, and S. Lu, “Towards reliable (and efficient) job executions in a practical geo-distributed data analytics system,” 2018, arXiv:1802.00245. [Online]. Available: http://arxiv.org/abs/1802.00245.