Distributed Query Processing in the Edge Assisted IoT Data Monitoring System

Zhipeng Cai, *Senior Member, IEEE,* Tuo Shi

***Abstract*—The massive amount of data generated by IoT devices places enormous pressure on sensory data query processing. Due to the limitations of computation and data transmission capabilities in traditional wireless sensor networks, the current query processing methods are no longer effective. Furthermore, processing vast amount of sensory data also overloads the cloud. To address these problems, we investigate query processing in an Edge Assisted IoT Data Monitoring System (EDMS). Multi-access Edge Computing (MEC) is an emerging topic in IoTs. Unlike wireless sensor networks, the edge servers in an EDMS can deploy the computation and storage resources to nearby IoT devices and offer data processing services. Therefore, queries towards massive sensory data can be processed in an EDMS in a distributed manner and the edge servers can handle the sensory data in a distributed manner, reducing the workload of the cloud. In this paper, we define a query processing problem in an EDMS which aims to deriving a distributed query plan with the minimum query response latency. We prove that this problem is NP-Hard and propose a corresponding approximation algorithm. The performance of the proposed algorithm is bounded. Furthermore, we evaluate the performance of the proposed algorithm through extensive simulations.**

***Index Terms*—Query Processing, Data Monitoring, MEC**

I. Motivation

The Internet of Things (IoT) acts as a bridge connecting the cyber world and the physical world. An IoT system allows users to collect data from the physical world via sensors, to monitor system performance, and to run queries towards the collected data [1]–[3]. Many useful techniques, such as DeepDirect for tie direction learning [4], are proposed for data monitoring and analysis in distributed environments. Multi-access Edge Computing (MEC) [5] is an emerging IoT architecture which is composed of edge servers.Unlike conventional IoT systems, MEC systems can obtain extra computation and storage resources from the physical world.

In this work, we address the query processing problem in modern IoT systems. Query processing is one of the most important topics in traditional IoT systems such as Wireless Sensor Networks (WSNs), and there are many existing works in this regard [6]–[12]. In WSNs, there are two major strategies for query processing which are centralized strategy and distributed strategy. In the centralized strategy, all sensory data are transmitted to the cloud, and the cloud is in charge of query processing. On the contrary, in a distributed strategy, queries are processed by cooperative sensor nodes. However, query processing is more challenging in modern IoT systems, and the above two query processing strategies are not feasible in modern IoT systems due to the following two challenges.

Challenge 1. Large volume of the sensory data [13]. According to a report by Cisco [14], there will be 31 billion connected IoT devices by 2020 and 75 billion devices by 2025. These IoT devices will generate an enormous volume of sensory data and query processing on this volume of data will place too much computation and transmission pressure on the underlying systems. The data transmission will be slow and the computation workload will overload the sinks (or cloud). Thus, the centralized strategy used in WSNs cannot be adopted.

Challenge 2. Complex queries. In WSNs, existing works have tried to process queries in a distributed manner, such as the top-k query, range query and curve query. However, in the recent IoT systems, queries are becoming more complex than what they are used to be in traditional WSNs. To answer a query in a modern IoT system, the data might have to be preprocessed by a series of services including image processing, speech recognition, data integration, and a node may have to deploy an AI model to process these data processing services. For example, if a user wants to select the license number of the fast vehicle, then the speed data of vehicles should be sorted to obtain the fast vehicle, the image data of vehicles should be recognized to obtain the license number, and the results of above two steps should be joined to obtain the final result. However, since the sensor nodes in WSNs have limited computation, storage and energy resources, the above-mentioned data processing services cannot be carried out at each individual sensor node, thus WSNs cannot process complex queries in a distributed manner. Therefore, the distributed strategy used in WSNs cannot be adopted.

To process complex queries in a modern IoT system and address the above mentioned challenges, we consider the Edge Assisted IoT Data Monitoring System (EDMS). The EDMS can be deployed in a city to monitor the traffic, the security, the change of the environment and so on. Moreover, the EDMS can also be deployed in a big factory to monitor the industrial parameters. An overview of the EDMS is shown in Fig.1. In an EDMS, sensory data are collected from different data sources and are stored in a distributed way at edge servers, and these edge servers are connected to a remote cloud. The cloud translates a query into a series of data processing services, such as image processing, information integration, top-k query processing and so on. These data processing services have been cached at the edge servers. When a query arrives at the cloud, it will generate a query plan and assign the tasks to some edge servers. Since edge

Z. Cai, is with the Department of Computer Science, Georgia State University, 25 Park Place, Atlanta, GA 30303, USA. Email: zcai@gsu.edu. T. Shi is with the Department of Computer Science and Technology, Harbin Institute of Technology, 92 West Dazhi Street, Nan Gang District, Harbin, China. Email: shituo@hit.edu.cn.

servers have more computation and storage resources, the large volume of sensory data (Challenge 1) and the complex queries (Challenge 2) can be processed in a distributed manner by them. The distributed query processing in EDMS can make the best use of the resources at edge servers and decrease the computation and transmission costs.

Unfortunately, query processing in an EDMS is still challenging. An EDMS is a heterogeneous system where different edge servers have different computation and communication capabilities. Furthermore, the data processing services in each query are interdependent, and the output of one service can be the input of another service. Therefore, an improper query plan may lead to unnecessary response latency. In this paper, we investigate how to generate a query plan to optimize query response delay in an EDMS. To the best of our knowledge, this is the first work investigating query processing in an EDMS. The contributions of the paper are summarized as follows.

(1) We propose a model for distributed query processing in an EDMS, and define the *Query Processing Latency Minimization* (MIN-QPL) problem. We prove that the MIN-QPL problem is NP-Hard.

(2) We show that in some specific cases, the optimal solution of the MIN-QPL problem can be obtained in polynomial time. Furthermore, we provide the upper bound and lower bound of the MIN-QPL problem.

(3) We propose an approximation algorithm to solve the MIN-QPL problem. We also prove that the approximation algorithm has an acceptable approximation ratio.

(4) We evaluate the performance of the proposed algorithm by carrying out extensive simulations. The simulation results show that the proposed algorithm is effective and efficient.

The rest of the paper is organized as follows. Section II provides a summary of related works. Section III defines the MIN-QPL problem. Section IV investigates two special cases of the MIN-QPL problem. An approximation algorithm for the MIN-QPL problem is proposed in Section V. The upper bound and lower bound of the MIN-QPL problem are provided in Section VI, and the approximation ratio of the proposed algorithm is analyzed. Section VII shows the simulation results and Section VIII concludes the paper.

II. Related Works

1) Multi-Access Edge Computing: Multi-Access Edge Computing (MEC) [15] aims at deploying computation and storage resources closer to users (or mobile devices) and releasing the workload pressure of the cloud. Mobile devices can send their sensory data to edge servers for further analysis. The works in [16], [17] investigate the offloading problem with a single mobile device and a single MEC server. The works in [18]–[25] consider the task offloading problem in an MEC system with multiple mobile devices and a single MEC server. Other works have considered the service placement and the requests routing problems. In the service placement problem [26], [27], the authors aim to investigate how to place/migrate services in edge services to reduce the respond latency. The request routing problem [28] aims to investigate how to send users' requests to feasible edge servers to improve the performance. However, none of these works addresses query processing at edge servers.

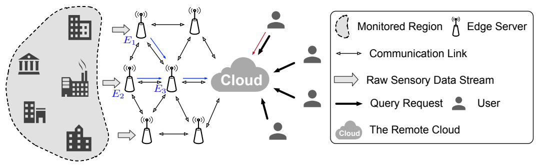

Fig. 1: An example of the EDMS.

2) Directed Acyclic Graph: The DAG (Directed Acyclic Graph) is always used to represent the relationships between services [18], [29], [30]. In these papers, the authors use a directed edge to represent the dependency between two tasks (or services). Each direct edge (f_i, f_j) means that f_j can process if and only if f_i has been processed. However, the authors in these papers have not fully considered the data dependency between services. In this paper, we propose the Data Based DAG (DDAG for short) by considering the data dependency between services.

3) Query Processing in WSNs: Query processing is a key method to help users to access the data [31]–[34] and it is also a classical issue in WSNs [35]. The authors in [36]–[39] investigated how to collect enough sensory data in the energy harvesting networks and battery-free sensor networks. The works in [6] and [7] propose some secure range query processing algorithms for WSNs. Two spatial-correlation based approximate aggregation algorithms are proposed in [8] and [9] with an adaptive clustering method for WSNs to support aggregation query processing. k-NN query processing is investigated in [10] based on an itinerary-based method. The work in [11] investigates curve query processing in WSNs. A sampling-based method is proposed in [12]. However, as mentioned in Section I, these methods cannot be adopted to solve the query processing problem in modern IoT systems.

III. Problem Definition

A. System Overview

An Edge assisted IoT Data Monitoring System (EDMS) is illustrated in Fig.1. An EDMS involves three major components, a network composed of servers $\mathcal{E}^+ = \{E_0, E_1, E_2, ..., E_{|\mathcal{E}|}\}$ including a remote cloud E_0 and a set of edge servers $\mathcal{E} = \{E_1, E_2, ..., E_{|\mathcal{E}|}\}$, a set of users $\mathcal{U}$, and a monitored region $\mathcal{R}$. The edge servers in $\mathcal{E}$ are deployed close to the monitored region and collect sensory data. Each edge server is equipped with a set of data processing services to process the sensory data. The users also can submit query requests to the cloud to retrieve the physical world information. The cloud will handle the submitted query requests and return results to users. As illustrated in Fig. 1, a user has submitted a query request (the red arrow) to the cloud. Then the cloud will generate a corresponding query processing plan and will send the plan to edge servers. After receiving the processing plan, the edge servers (E_1 and E_2) with the user's interested data will transmit the data to edge servers with the feasible data processing service (E_3) to be processed. After processing the data, the final query results will be transmitted to the cloud.

Unlike traditional DBMS, the sensory data in an EDMS are stored in a distributed manner at edge servers and the stored sensory data are raw data collected from different data sources, such as cameras, microphones, sensors and so on. These data need to go through a few preprocessing steps, such as sorting, data integration, image processing, speech recognition, *etc.* to enable query processing. Thus, if we process queries in the cloud, collecting all the sensory data will be time-consuming and processing complex queries will be resource-consuming. In an EDMS, we can picture a data query as a series of data processing tasks, and because the data processing services have been placed at edge servers, we can process queries in a distributed way to reduce query response latency and relieve the workload of the cloud.

B. Network Model

Sensory Data. Assume that we have D types of data sources in the monitored region, such as image data from surveillance cameras, temperature data from temperature sensors, speed data from speed detectors and so on. The set of sensory data generated in the monitored region is denoted by $\mathcal{S} = \{S^{(i)} | 1 \leq i \leq D\}$ where $S^{(i)}$ is the sensory data collected from the type i ($1 \leq i \leq D$) data source. We assume that each sensory data set $S^{(i)}$ is distributedly stored at the edge servers. Let $S(E_j) = \bigcup_{1 \leq i \leq D} S^{(i)}(E_j)$ be the set of sensory data kept at E_j where $S^{(i)}(E_j)$ is the set of type i data. For every two edge servers $E_k, E_j \in \mathcal{E}$, we have $S^{(i)}(E_j) \cap S^{(i)}(E_k) = \emptyset$ and for all the edge servers, we have $\bigcup_{E_j \in \mathcal{E}} S^{(i)}(E_j) = S^{(i)}$.

Communication. The edge servers and the remote cloud can communicate with each other through the backbone network. For each link (i, j) where $E_i, E_j \in \mathcal{E}^+$, let $l_{i,j}$ be the communication delay of transmitting one unit of data. Without loss of generality, the communication between an edge server and the cloud is much slower than the communication between two edge servers, *i.e.*, $\min\{l_{0,i} | E_i \in \mathcal{E}\} \gg \max\{l_{i,j} | E_i, E_j \in \mathcal{E}\}$. Suppose E_i transmits a set of data O to E_j. The transmission delay is $L_{i,j}(O) = |O| l_{i,j}$.

Computation. Edge servers and the remote cloud can provide different data processing services, such as image processing, data integration and speech recognition, to process the sensory data. Let $\mathcal{F}$ be the universal set of services. For each $E_i \in \mathcal{E} \cup \{E_0\}$, let C_i be the computing resource of E_i and $F_i = \{f_1, f_2, ..., f_{|F_i|}\} \subseteq \mathcal{F}$ be the set of services provided by E_i. The cloud E_0 provides all the services in $\mathcal{F}$, *i.e.*, $F_0 = \mathcal{F}$. We assume that each $E_i \in \mathcal{E}^+$ can only carry out one service at a time and the service cannot be interrupted. For service $f \in F_i$, let I and O be the input and output data, *a.k.a* $f(I) = O$, and $C(f(I))$ be the computing resources needed by f with input I. Thus, the processing latency $f(I)$ is

$$L_i(f(I)) = \frac{C(f(I))}{C_i}. \tag{1}$$

We notice that the computing resources needed by $f(I)$ and the size of the output O are related to the size of I. Therefore, Formula (1) can be rewritten as

$$L_i(f(I)) = \frac{c_f(|I|)}{C_i}, \tag{2}$$

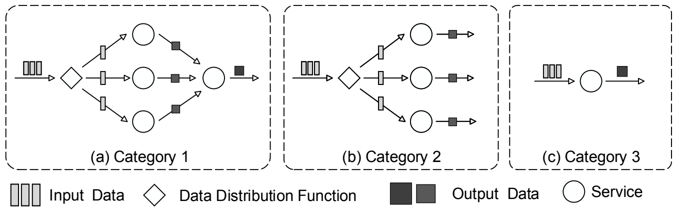

Fig. 2: Different categories of services.

and the relationship between $|I|$ and $|O|$ is

$$|O| = g_f(|I|), \tag{3}$$

where $c_f(\cdot)$ and $g_f(\cdot)$ are the functions related to service f. We assume that $c_f(\cdot)$ and $g_f(\cdot)$ are both convex functions.

We have noticed that each data processing service in $\mathcal{F}$ could be distributedly processed, partially distributedly processed or cannot be distributedly processed. Thus, the data processing services can be classified into the following three categories.

Category 1: $f(I) = f(f(I_1) \cup f(I_2) \cup ...)$**.** If a service $f \in \mathcal{F}$ belongs to Category 1, f can be partially distributedly processed. As illustrated in Fig.2 (a), service $f(I)$ can be decomposed into several sub-tasks $f(I_1), f(I_2), ..., f(I_k)$, where $I = I_1 \cup I_2 \cup ... \cup I_k$, and the outputs of the sub-tasks need to be aggregated by f to obtain the final output. Data processing services, such as top-k query and sorting, belong to this category.

Category 2: $f(I) = \bigcup_{1 \leq i \leq k} f(I_k)$**.** If service $f \in \mathcal{F}$ belongs to this category, f can be fully distributedly processed. As shown in Fig.2(b), the combination of the outputs of the sub-tasks is the final output of service f. The face recognition service [40] belongs to this category.

Category 3: $f(I) \notin \{f(\{f(I_1), f(I_2), ...\}), \bigcup_{1 \leq i \leq k} f(I_k)\}$**.** When service f belongs to this category (Fig.2(c)), f cannot be distributedly processed. For example, an event detection service [41] cannot be processed distributedly because all the sensory data should be input to the service function to obtain the entire information about the physical world.

C. Query Model

Given a query $Q(S_Q)$, where $S_Q \subseteq \mathcal{S}$ is a collection of different types of sensory data, the cloud can easily return a logical query processing strategy $P_Q(\cdot)$ based on the existing DBMS methods. The correct result of $Q(S_Q)$ is equal to $P_Q(S_Q)$. The query Q can also be represented by $P_Q(\cdot)$. To distributedly process queries, $P_Q(\cdot)$ can be divided into a series of data processing services $F_Q \subseteq \mathcal{F}$ and the dependency between these services can be represented by an Equivalent Data based Directed Acyclic Graph (E-DDAG). The DDAG and E-DDAG are defined as follows.

Definition 1 (**Data based Directed Acyclic Graph** (DDAG))**.** *Given a collection of different types of sensory data $S = \{S_1, S_2, ..., S_k\}$ and a set of different services $F = \{f_1, f_2, ..., f_m\}$, a* ***Data based Directed Acyclic Graph*** *$G(V, E)$ satisfies that*

(1) $V = F \cup S$ and E is a set of directed edges;

(2) $G(V, E)$ is a Directed Acyclic Graph;

(3) S is the leaf nodes of $G(V, E)$.

Different from traditional DAG, an edge (f_i, f_j) (or (S_i, f_j)) in a DDAG not only represents that f_j should be executed after f_i, but also means the output of f_i (or S_i) is the input of f_j.

Definition 2 (**Equivalent DDAG** (E-DDAG))**.** *Given a query $Q(S_Q)$, the logical query processing strategy $P_Q(\cdot)$ and a set of data processing services F_Q, an Equivalent DDAG of $P_Q(\cdot)$ $G_Q(V, E)$ satisfies $V = S_Q \cup F_Q$ and $f_r(A_{f_r}) = P_Q(S_Q)$ where $A_f = \bigcup_{S_i \in K_f} S_i \cup \bigcup_{f_i \in K_f} f_i(A_{f_i})$, $f_r \in V$ is the root of $G_Q(V, E)$, A_f is the input of f, and $K_f = \{\xi | (\xi, f) \in E\}$ is the set of children of f.*

An E-DDAG of $P_Q(\cdot)$ illustrates the dependency between the services in F_Q and Fig.4 gives an example E-DDAG. In Fig.4, a user submits the query "*the license plate number of the top-10 fastest vehicles*". In this case, $S = \{S_I, S_S\}$ is the set of vehicle image data S_I and the vehicle speed data S_S. $\mathit{\Pi}_Q(S) = f_{TK}(f_{JN}(S_S, f_{IR}(S_I)))$ where f_{TK}, f_{JN}, and f_{IR} are the top-k service, join service and image recognition service. $P_Q(S)$ can be represented by the E-DDAG shown in Fig.3. This E-DDAG has two leaf nodes, "image data" and "speed data", representing that the answer to this query can be found in these data sets. Based on the E-DDAG, the "image data" will be sent to the "image recognition" service to recognize the license plate number of the vehicles. The outputs of the "image recognition" are combined (joined) with the speed data, and the "top-k query" service will be carried out towards the output of the "join" service.

D. Query Processing in EDMS

Based on the E-DDAG $G_Q(S_Q \cup F_Q, E)$ of a query Q, the EDMS will map $G_Q(S_Q \cup F_Q, E)$ to the network composed of servers and process query Q distributedly. The distributed query processing in the EDMS has the following two steps.

Step 1. Data Supply. The EDMS will select a set of edge servers which can provide enough sensory data to cover S_Q.

Step 2. Data Processing. The servers in $\mathcal{E}^+$ will process the sensory data distributedly, obtain the query answer and return the answer to the cloud.

These two steps can be described by a collection of servers' actions. Let $\pi_i^t(f(S))$ denote the action that at time t, service $f \in F_i$ in $E_i \in \mathcal{E} \cup \{E_0\}$ will process the data in S. Specifically, we include a special service $f_{E_j}(\cdot)$, and $\pi_i^t(f_{E_j}(S))$ represents that at time t, E_i will transmit S to E_j and we define $f_{E_j}(S) = S$. We will use these actions to explain the workflow of EDMS as follows.

1. Data Supply. Let $\mathcal{E}_S \subseteq \mathcal{E}$ be the set of edge servers that will provide sensory data in the first step and $S'(E_i) \subseteq S(E_i)$ be the set of sensory data sets provided by $E_i \in \mathcal{E}_S$. $S'(E_i)$ and $\mathcal{E}_S$ satisfy $\bigcup_{E_i \in \mathcal{E}} S'(E_i) = S_Q$. Based on the dependency in E-DDAG, these sensory data will be transmitted to the servers with the corresponding services.

2. Data Processing. Suppose at time t, there are a set of servers $\mathcal{E}'$, and each $E_i \in \mathcal{E}'$ has a set of data S_i that need to be processed by service $f \in \mathcal{F}$. The data sets in $S(\mathcal{E}') = \{S_i | E_i \in \mathcal{E}'\}$ will be processed based on the following conditions.

Case 1: f **belongs to Category 1.** f can be partially distributedly processed. Let $\mathcal{E}_f = \{E_i | f \in F_i \wedge E_i \in \mathcal{E}^+\}$ be the servers to carry out service f. The data sets in $S(\mathcal{E}')$ can be processed based on the following steps.

(a) Partition each $S_i \in S(\mathcal{E}')$ into $|\mathcal{E}_f| \geq k \geq 1$ subsets $S_i = \bigcup_{1 \leq l \leq k} S_{i_l}$.

(b) Select k servers $E_{e_1}, E_{e_2}, ..., E_{e_k}$ from $\mathcal{E}_f$. Each server $E_i \in \mathcal{E}'$ transmits S_{i_j} to E_{e_j}. To transmit $S_{i_1}, ..., S_{i_k}$ from E_i to $E_{e_1}, ..., E_{e_k}$, we have the following actions $\{\pi_i^{t_{i_1}}(f_{E_{e_1}}(S_{i_1})),\ \pi_i^{t_{i_2}}(f_{E_{e_2}}(S_{i_2})), ...,\ \pi_i^{t_{i_k}}(f_{E_{e_k}}(S_{i_k}))\}$ where $t = t_{i_1} < t_{i_2} < ... < t_{i_k}$ and for each $1 \leq j \leq k$, $t_{i_{j-1}} + L_{i,e_{j-1}}(S_{i_{j-1}}) < t_{i_j}$.

(c) Let $t_{i_{j'}}$ be the time when S_{i_j} arrived at E_{e_j}. E_{e_j} receives all the data sets at time $t_j = \max\{t_{i'_j} | E_i \in \mathcal{E}'\}$. Each $E_{e_j} \in \{E_{e_j} | 1 \leq j \leq k\}$ processes $\bigcup_{E_i \in \mathcal{E}'} S_{i_j}$ and the action of E_{e_j} is $\pi_{e_j}^{t_{j'}}(f(\bigcup_{E_i \in \mathcal{E}'} S_{i_j}))$. Obviously, $t_{j'} > t_j$.

(d) Select a server $E_{e_{k+1}}$ from $\mathcal{E}_f$. Each $E_{e_j} \in \{E_{e_j} | 1 \leq j \leq k\}$ will transmit the output of f to $E_{e_{k+1}}$. The action of E_{e_j} is $\pi_{e_j}^{t_{j''}}(f(\bigcup_{E_i \in \mathcal{E}'} S_{i_j})))$ where $t_{j''} \geq t_{j'} + L_{e_j}(f(\bigcup_{E_i \in \mathcal{E}'} S_{i_j}))$. The relationship between $t_{j'}$ and $t_{j''}$ implies that E_{e_j} can transmit the output of f if and only if f has been finished.

(e) After receiving the outputs of $f(\bigcup_{E_i \in \mathcal{E}'} S_{i_1}))$, $f(\bigcup_{E_i \in \mathcal{E}'} S_{i_2}), ..., f(\bigcup_{E_i \in \mathcal{E}'} S_{i_k})$, $E_{e_{k+1}}$ will aggregate them together. Let $t' = \max\{t_{j''} + L_{e_j, e_{k+1}} | 1 \leq j \leq k\}$ be the time when $E_{e_{k+1}}$ has received all the data sets. The action of $E_{e_{k+1}}$ is $\pi_{e_{k+1}}^{t_{k+1}}(f(\bigcup_{1 \leq j \leq k, E_i \in \mathcal{E}'} f(S_{i_j})))$ where $t_{k+1} \geq t'$. At $t_{k+1} + L_{e_{k+1}}(f(\bigcup_{1 \leq j \leq k, E_i \in \mathcal{E}'} f(S_{i_j})))$, f is finished.

Based on the above five steps, data S in E_i can be processed by service f. These five steps can be divided into two phases: 1)**partition** ((a)-(c)) and 2)**aggregation** ((d)-(e)). In the first phase, S is partitioned into subsets and these subsets are assigned to other servers to be distributedly processed. The second phase selects a server to aggregate the outputs. The above five steps start at time t and end at time $t_{k+1} + L_{e_{k+1}}(f(\bigcup_{1 \leq j \leq k} f(S_j)))$.

Case 2: f **belongs to Category 2.** f can be fully distributedly processed. As shown in Fig.2, a Category 2 service is a special case of a Category 1 service without the "aggregation" phase. Therefore, we only need to adopt the first three steps in Case 1. Obviously, the processing of f starts at t and ends at $\max\{t_{j'} + L_{e_j}(f(\bigcup_{E_i \in \mathcal{E}'} S_{i_j})) | 1 \leq j \leq k\}$.

Case 3: f **belongs to Category 3.** f cannot be distributedly processed. The following steps are adopted to process f. Firstly, the servers in $\mathcal{E}'$ select a server $E_j \in \mathcal{E}_f$ and each $E_i \in \mathcal{E}'$ transmits S_i to E_j. For E_i, the action is $\pi_i^t(f_{E_j}(S_i))$. Secondly, E_j will process $f(\bigcup_{E_i \in \mathcal{E}'} S_i)$ at time $t_j > \max\{t + L_{i,j}(S_i) | E_i \in \mathcal{E}'\}$. The action is $\pi_j^{t_j}(f(\bigcup_{E_i \in \mathcal{E}'} S_i))$. In this case, the processing starts at t and ends at the time $t' \geq t_j + L_j(f(\bigcup_{E_i \in \mathcal{E}'} S_i))$ when E_j has been finished f.

Based on the above three cases, we can process f on the servers. Let $\mathit{\Gamma}_f(S(\mathcal{E}'))$ be the set of actions that process $f(S(\mathcal{E}'))$ on the servers. $\mathit{\Gamma}_f(S(\mathcal{E}'))$ can also be considered as a *component* with a set of inputs $S(\mathcal{E}')$ and output $f(\bigcup_{E_i \in \mathcal{E}'} S_i)$

(or $\{f(\bigcup_{E_i \in \mathcal{E}'} S_{i_1}), f(\bigcup_{E_i \in \mathcal{E}'} S_{i_2}), ..., f(\bigcup_{E_i \in \mathcal{E}'} S_{i_k})\}$ if f belongs to Category 2). The formal definition of the component is defined in Definition 3.

Definition 3 (**Component**). *A* ***Component*** $\mathit{\Gamma}_f(S(\mathcal{E}'))$ *is a set of actions that can process* $f(S(\mathcal{E}'))$ *on the servers.* $\mathit{\Gamma}_f(S(\mathcal{E}'))$ *has the following properties.*

(1) The input and output of $\mathit{\Gamma}_f(S(\mathcal{E}'))$ *are* $S(\mathcal{E}')$ *and* $f(\bigcup_{E_i \in \mathcal{E}'} S_i)$ *(or* $\{f(\bigcup_{E_i \in \mathcal{E}'} S_{i_1}), f(\bigcup_{E_i \in \mathcal{E}'} S_{i_2}), ..., f(\bigcup_{E_i \in \mathcal{E}'} S_{i_k})\}$ *if* f *belongs to Category 2);*

(2) The start time and end time of $\mathit{\Gamma}_f(S)$ *are* $t_s(\mathit{\Gamma}_f(S(\mathcal{E}'))) = \min\{\ t \mid \pi_i^t(\cdot) \in \mathit{\Gamma}_f(S(\mathcal{E}'))\}$ *and* $t_e(\mathit{\Gamma}_f(S(\mathcal{E}'))) = \max\{t|\pi_i^t(\cdot) \in \mathit{\Gamma}_f(S(\mathcal{E}'))\}$.

Specifically, each data supply edge server $E_i \in \mathcal{E}_S$ can be considered as a component $\mathit{\Gamma}_s(S'(E_i)) = \emptyset$ with the same input and output $S'(E_i)$, and the same start and end time. We can also connect two different components together to complete more complex data processing services. When connecting two components $\mathit{\Gamma}_f(\cdot)$ and $\mathit{\Gamma}_{f'}(\cdot)$ together, the output of $\mathit{\Gamma}_f(\cdot)$ is the input of $\mathit{\Gamma}_{f'}(\cdot)$, and the start time of $\mathit{\Gamma}_{f'}(\cdot)$ and the end time of $\mathit{\Gamma}_f(\cdot)$ are adjusted to satisfy $t_s(\mathit{\Gamma}_{f'}(\cdot)) \geq t_e(\mathit{\Gamma}_f(\cdot))$. Therefore, given an E-DDAG $G_Q(V, E)$ which illustrates the relationships between services, we can connect different components together to map the E-DDAG to the servers' network and process the query in a distributed way. Let $\mathit{\Gamma}_f^*(S)$ be the output of $\mathit{\Gamma}_f(S)$, then the query processing plan in an EDMS can be defined as follows.

Definition 4 (**Query Processing Plan (QPE) in an EDMS**). *Given an E-DDAG* $G_Q(V = F_Q \cup S_Q, E)$*, a* ***Query Processing Plan*** $\mathit{\Pi}(Q)$ *can be constructed through the following steps.*

(1) For each $(S, f) \in E$*, replace* S *by the components in* $\{\mathit{\Gamma}_s(S'(E_{i_1})), \mathit{\Gamma}_s(S'(E_{i_2})), ..., \mathit{\Gamma}_s(S'(E_{i_k}))\}$ *where* $\{E_{i_1}, ..., E_{i_k}\} \subseteq \mathcal{E}_S$ *and* $\bigcup_{1 \leq l \leq k} S'(E_{i_l}) = S$*. Connect the components in* $\{\mathit{\Gamma}_s(S'(E_{i_1})), ..., \mathit{\Gamma}_s(S'(E_{i_k}))\}$ *with component* $\mathit{\Gamma}_f(\bigcup_{1 \leq l \leq k} \mathit{\Gamma}_s^*(S'(E_{i_l})))$.

(2) For each $(f, f') \in E$*, connect components* $\mathit{\Gamma}_f(\cdot)$ *and* $\mathit{\Gamma}_{f'}(\cdot)$ *together.*

(3) Construct a set of actions $\mathit{\Gamma}_c(\mathit{\Gamma}_{f_r}^*(\cdot))$ *to transmit the output of* $\mathit{\Gamma}_{f_r}(\cdot)$ *to the cloud* E_0*, where* f_r *is the root of* $G_Q(V, E)$.

After the above steps, $\mathit{\Pi}(Q) = \bigcup_{f \in F_Q} \mathit{\Gamma}_f(\cdot) \cup \mathit{\Gamma}_c(\mathit{\Gamma}_{f_r}^*(\cdot))$.

The query plan $\mathit{\Pi}(Q)$ can obtain the answer of Q. The first and second steps in Definition 4 imply that the interested sensory data set of query $Q(S_Q)$ has been processed and the query has been answered. The third step aims to transmit the query answer to the cloud. Given a feasible query plan $\mathit{\Pi}(Q)$, the latency of $\mathit{\Pi}(Q)$ is equal to

$$L(\mathit{\Pi}(Q)) = t_e(\mathit{\Pi}(Q)) - t_s(\mathit{\Pi}(Q)) \tag{4}$$

where $t_s(\mathit{\Pi}(Q)) = \min\{t_s(\mathit{\Gamma}_f(\cdot))|\mathit{\Gamma}_f(\cdot) \subseteq \mathit{\Pi}(Q)\}$ and $t_e(\mathit{\Pi}(Q)) = \max\{t_e(\mathit{\Gamma}_f(\cdot))|\mathit{\Gamma}_f(\cdot) \subseteq \mathit{\Pi}(Q)\}$ are the start time and the end time of $\mathit{\Pi}(Q)$, respectively.

An example is shown in Fig.4. In the EDMS based vehicle monitoring system, the image data and speed data of vehicles are stored in edge servers. The image data $S_I = S_{I1} \cup S_{I2}$ is stored at E_4 and E_5, and the speed data is only stored at E_5. Based on the E-DDAG shown in Fig.3, we have a QPE $\mathit{\Pi}(Q) = \{\pi_4^{t_1}(f_{E_1}(S_{I1})), \pi_5^{t_1}(f_{E_2}(S_{I2} \cup S_S)), \pi_1^{t_2}(f_{IR}(S_{I1})), \pi_2^{t_2}(f_{IR}(S_{I2})), \pi_1^{t_3}(f_{E_2}(\bar{S}_{I1})), \pi_2^{t_4}(f_{JN}(\bar{S}_{I1} \cup \bar{S}_{I2} \cup S_S)), \pi_2^{t_5}(f_{E_3}(S_3)), \pi_3^{t_6}(f_{TK}(S_3)), \pi_3^{t_7}(f_{E_0}(S_4))\}$, where $\bar{S}_{I1} = f_{IR}(S_{I1})$, $\bar{S}_{I2} = f_{IR}(S_{I2})$, $S_3 = f_{JN}(\bar{S}_{I1} \cup \bar{S}_{I2} \cup S_S)$, and $S_4 = f_{TK}(S_3)$. $\mathit{\Pi}(Q)$ implies that the image data S_{I1}, S_{I2} will be processed by the image recognition service in E_1 and E_2 respectively. Then E_1 will send the output $\bar{S}_{I1}$ to E_2. After that, the image recognition results $\bar{S}_{I1}, \bar{S}_{I2}$ and the speed data S_S will be joined together in E_2. The join results S_3 will be sent to E_3 to be processed by the "top-k query" service. Finally, the query result S_4 will be transmitted to the cloud. In $\mathit{\Pi}(Q)$, $\pi_4^{t_1}(f_{E_1}(S_{I1}))$, $\pi_1^{t_2}(f_{IR}(S_{I1}))$, $\pi_1^{t_3}(f_{E_2}(\bar{S}_{I1}))$ and $\pi_5^{t_1}(f_{E_2}(S_{I2} \cup S_S))$, $\pi_2^{t_2}(f_{IR}(S_{I2})$ constitute the components $\mathit{\Gamma}_{f_{IR}}(S_I)$.

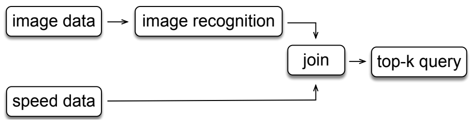

Fig. 3: An example of the E-DDAG.

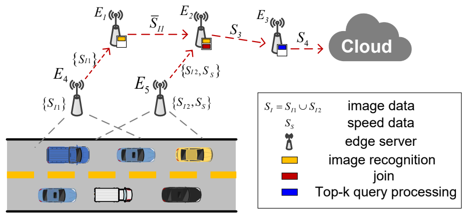

Fig. 4: Query processing in an EDMS. A user wants to figure out the license plate number of the top-10 fastest vehicles.

$$\begin{tikzcd}
S_1 \arrow[r] & f_1 \arrow[dr] & \\
S_2 \arrow[r] & f_2 \arrow[r] & f_{k+1} \\
\cdots\cdots & & \\
S_k \arrow[r] & f_k \arrow[uur] &
\end{tikzcd}$$

Fig. 5: The E-DDAG in the sub-MIN-QPL problem.

E. Problem Definition

We notice that given an E-DDAG $G_Q(V, E)$, there are many feasible distributed query processing plans. For a $G_Q(V, E)$, different QPEs will result in different latency. In order to reduce the latency of distributed query processing, we define the *Query Processing Latency Minimization* (MIN-QPL) problem.

MIN-QPL Problem

Input:

(1) A cloud E_0 and a set of edge servers $\mathcal{E}$;

(2) A query $Q(S_Q)$ and the E-DDAG $G_Q(V = S_Q \cup F_Q, E)$ generated by the cloud;

(3) The sensory data $S(E_i)$ stored at each edge server E_i;

(4) The computing capability C_i of each $E_i \in \mathcal{E} \cup \{E_0\}$;

(5) The set of services of each edge server E_i and F_i;

(6) The communication latency $l_{i,j}$ of transmitting one unit of data between $E_i, E_j \in \mathcal{E}^+$.

Outputs: A QPE $\mathit{\Pi}(Q)$ with the minimum latency.

Theorem 1. *The MIN-QPL problem is NP-Hard.*

Proof. Consider the following special case of the MIN-QPL problem.

sub-MIN-QPL Problem:

Input:

(1) A cloud E_0 and a set of edge servers $\mathcal{E}$;

(2) A query $Q(S_Q)$ and the E-DDAG $G_Q(V = S_Q \cup F_Q, E)$ (as illustrated in Fig.5) generated by the cloud, where $S_Q = \{S_1, S_2, ..., S_k\}$ is composed of Category 3 services $F_Q = \{f_1, f_2, ..., f_{k+1}\}$, $E = \{(S_1, f_1), (S_2, f_2), ..., (S_k, f_k), (f_1, f_{k+1}), (f_2, f_{k+1}), ..., (f_k, f_{k+1})\}$ and $k > |\mathcal{E}|$;

(3) The sensory data $S(E_i)$ stored at each edge server E_i, where $S(E_1) = S_Q$ and $\bigcup_{1<i<|\mathcal{E}|} S(E_i) \cap S_Q = \emptyset$;

(4) The computing capability C_i of each $E_i \in \mathcal{E}^+$, and $C_0 = C_1 = ... = C_{|\mathcal{E}^+|} = 1$;

(5) The set of services of each edge server E_i, F_i, and $F_i = F_Q - \{f_{k+1}\}$;

(6) The communication latency $l_{i,j} = 0$ of transmitting one unit of data between $E_i, E_j \in \mathcal{E}^+$.

Outputs: A QPE $\Pi(Q)$ with the minimum latency.

Based on the E-DDAG (Input (2)) and the service deployment (Input (5)) of the sub-MIN-QPL problem, the query processing must follow the following steps. First, the edge server E_1 will transmit the sensory data to the servers in $\mathcal{E}^+$ to process data services $\{f_1, f_2, ..., f_k\}$. Second, after processing data services $\{f_1, f_2, ..., f_k\}$, the outputs will be transmitted to the cloud to process data service f_{k+1}.

Since the transmission does not cause any delay, the latency of a feasible $\Pi(Q)$ is equal to $L_k + L_0(f_{k+1}(\bigcup_{1\le i\le k} f_i(S_i)))$, where L_k is the latency of conducting all the services in $\{f_1, f_2, ..., f_k\}$ and $L_0(f_{k+1}(\bigcup_{1\le i\le k} f_i(S_i)))$ is the total time of conducting $f_{k+1}(\cdot)$ at the cloud. Obviously, $L_0(f_{k+1}(\bigcup_{1\le i\le k} f_i(S_i)))$ is a constant. Therefore, the optimal $\Pi(Q)$ must minimize L_k, and in order to optimize the solution of MIN-QPL, we only need to consider how to finish the services in $\{f_1, f_2, ..., f_k\}$.

We reduce the Minimum Makespan Scheduling problem (MMS) to the sub-MIN-QPL problem to prove its NP hardness. Given an instance of the MMS problem $[J, M]$, where $J = \{J_1, J_2, ..., J_k\}$ is a set of k jobs each of which takes t_i time to be processed and $M = \{M_1, M_2, ..., M_m\}$ is a set of m identical machines. The reduction has the following steps.

Step 1. Construct a data processing service f_i and a type of data set S_i for each $J_i \in J$.

Step 2. Construct another service f_{k+1}. Then $F_Q = \{f_i | J_i \in J\} \cup \{f_{k+1}\}$ and $S_Q = \{S_i | J_i \in J\}$.

Step 3. Construct an E-DDAG $G_Q(S_Q \cup F_Q, E)$, where $E = \{(S_1, f_1), (S_2, f_2), ..., (S_k, f_k), (f_1, f_{k+1}), (f_2, f_{k+1}), ..., (f_k, f_{k+1})\}$.

Step 4. Construct m identical servers $\mathcal{E}^+ = \{E_0, E_1, ..., E_{m-1}\}$. Let $C_0 = 1$ and $F_0 = F_Q$. For each $i \neq 0$, let $C_i = 1$, $F_i = F_Q - \{f_{k+1}\}$.

Step 5. Let $S(E_1) = S_Q$ and $\bigcup_{1<i<|\mathcal{E}|} S(E_i) \cap S_Q = \emptyset$.

Step 6. Let $l_{i,j} = 0$ for each $E_i, E_j \in \mathcal{E}^+$.

Let A_i be the set of jobs assigned to machine M_i and $A^o = \{A_i^o | M_i \in M\}$ be the optimal solution of the MMS problem. We now derive a corresponding solution of the sub-MIN-QPL problem. For each $J_i \in A_j^o$, we assign data set S_i to server E_j as the input of service f_i, and $\Gamma_{f_i}(S_i) = \{\pi_1^{t_0}(f_{E_j}(S_i)), \pi_j^{t_0}(f_i(S_i))\}$. The services processed at server E_j is $\{f_i | J_i \in A_j^o\}$, and at time $T_j = \sum_{J_i \in A_j^o} t_i$, all the services in E_j has been finished. Then all the services in $\{f_1, f_2, ..., f_k\}$ have been finished at time $T_{max} = \max\{T_j | 1 \le j \le m\}$. Since A^o is the optimal solution of the jobs in J, the corresponding solution of the sub-MIN-QPL problem is also optimal. Similarly, given an instance of the sub-MIN-QPL problem, we can also obtain a corresponding optimal solution for the MMS problem.

Based on above analysis, the sub-MIN-QPL problem is NP-Hard, and the MIN-QPL problem is also NP-Hard. □

Symbol	Description
E_0	The remote cloud.
$\mathcal{E}$	The set of edge servers.
$S(E_i)$	The sensory data stored in E_i.
C_i	The computation capability of E_i.
F_i	The set of services in E_i.
$l_{i,j}$	The latency of transmitting data between E_i, E_j.
f_j	A service in the universal set of service $\mathcal{F}$.
$C(f_j(I))$	The computation resources needed by f_j with input I.
$L_i(f_j(I))$	The processing latency of $f_j(I)$.
$Q(S_Q)$	The query on data sets S_Q.
$G_Q(V, E)$	The E-DDAG of query $Q(S_Q)$.
$\Pi(Q)$	The QPE of query Q.

TABLE I: Symbol Table

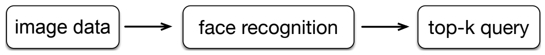

Fig. 6: An example of Case 2. User wants to find out the top-10 oldest people that have passed through the security camera.

IV. Some Special Cases of the MIN-QPL Problem.

We first consider the following two special cases of the MIN-QPL problem.

Case 1. $\bigcup_{E_i \in \mathcal{E}} F_i \cap F_Q = \emptyset$, *i.e.*, the query needs to be processed in the cloud.

Case 2. The E-DDAG $G_Q(V = S_Q \cup F_Q, E)$ of query Q is linear, and each service $f \in F_Q$ belongs to Category 2. That means, for each non-leaf node $f \in V$, f has only one child node and at most one parent.

Obviously, Case 1 is the case when query Q is processed centrally in the cloud. Case 2 means the data sets in S_Q is processed linearly, and it also describes a common scenario. An example of Case 2 is shown in Fig. 6. In such example, the user wants to find out the top-10 oldest people that have passed through the security camera. First, the image data collected by the camera will be processed by the "face recognition" service to recognized the age of each person. Then the output of the "face recognition" service will be sent to the "top-k query" service to obtain the final result. Although the MIN-QPL problem is NP-Hard, we will show that we can obtain the optimal solution of these two special cases.

A. Special Case 1

Special Case 1 of the MIN-QPL problem (C1-MIN-QPL) is defined as follows.

C1-MIN-QPL Problem

Input:

(1) A cloud E_0 and a set of edge servers $\mathcal{E}$;

(2) A query $Q(S_Q)$ and the E-DDAG generated by the cloud $G_Q(V = S_Q \cup F_Q, E)$;

(3) The sensory data $S(E_i)$ stored at each edge server E_i;

(4) The computation capability C_i of each $E_i \in \mathcal{E} \cup \{E_0\}$;

(5) The set of services F_i of each edge server E_i, and for each $E_i \in \mathcal{E}$, $F_i \cap F_Q = \emptyset$,

(6) The communication latency of transmitting one unit of data between $E_i, E_j \in \mathcal{E}^+$.

Outputs: A QPE $\mathit{\Pi}(Q)$ with the minimum latency.

According to the fifth input, the query needs to be answered by the cloud. Thus, the query processing plan (QPE) $\mathit{\Pi}(Q)$ can be divided into two sets $\mathit{\Pi}^{(1)}(Q)$ and $\mathit{\Pi}^{(2)}(Q)$. The actions in $\mathit{\Pi}^{(1)}(Q)$ aim to transmit the sensory data to the cloud, and based on the actions in $\mathit{\Pi}^{(2)}(Q)$, the cloud will process the sensory data based on the E-DDAG. It should be noted that the cloud can only carry out one service at a time. Therefore, the actions in $\mathit{\Pi}^{(1)}(Q)$ can be carried out in a parallel way and the actions in $\mathit{\Pi}^{(2)}(Q)$ can only be carried out sequentially. Obviously, the actions in $\mathit{\Pi}^{(2)}(Q)$ are affected by the actions in $\mathit{\Pi}^{(1)}(Q)$. In the following sections, we will discuss three problems:

(1) Given $\mathit{\Pi}^{(1)}(Q)$, how to construct $\mathit{\Pi}^{(2)}(Q)$?

(2) Given $\mathit{\Pi}^{(2)}(Q)$, how to construct $\mathit{\Pi}^{(1)}(Q)$?

(3) How to combine $\mathit{\Pi}^{(1)}(Q)$ and $\mathit{\Pi}^{(2)}(Q)$ together and obtain an optimal $\mathit{\Pi}(Q)$?

1) Construction of $\mathit{\Pi}^{(2)}(Q)$*:* Let $t(S)$ be the time when the data set $S \in S_Q$ has arrived in the cloud. $t(S)$ is determined by $\mathit{\Pi}^{(1)}(Q)$. In this section, we assume that $\mathit{\Pi}^{(1)}(Q)$ is given and the value of $t(S)$ is known. We will discuss how to construct $\mathit{\Pi}^{(2)}(Q)$ based on $\{t(S)|S \in S_Q\}$.

The cloud will process the services sequentially. Let $t_s(f)$ and $t_e(f)$ be the start time and end time of a service f. Based on $G_Q(V, E)$, $t_s(f)$ and $t_e(f)$ must satisfy the following conditions.

(a) *Data Dependency.* If $(S, f) \in E$, then $t_s(f) \geq t(S)$.

(b) *Service Dependency.* If $(f, f') \in E$, then $t_e(f) \leq t_s(f)$.

(c) *Interrupt Free.* $t_e(f) - t_s(f) = L_0(f(A_f))$, where A_f is the input data set of f.

Based on the above conditions, we propose Algorithm **??** to construct $\mathit{\Pi}^{(2)}(Q)$. Algorithm **??** initializes $\mathit{\Pi}^{(2)}(Q) = \emptyset$ and then it executes the following three steps.

Algorithm 1: $\mathit{\Pi}^{(2)}(Q)$ Construction Algorithm

```
Input: $G_Q(V = S_Q \cup F_Q, E)$; $\{t(S)|S \in S_Q\}$.
Output: $\mathit{\Pi}^{(2)}(Q)$
1  $\mathit{\Pi}^{(2)}(Q) = \emptyset$;
2  Let $G'_Q(F_Q, E_{F_Q})$ be the graph deduced from $F_Q$ and construct a
     topological order of $G'_Q$, $T(G'_Q) = \{f_{o_1}, ..., f_{o_{|F_Q|}}\}$;
3  foreach $1 \leq i \leq |F_Q|$ do
4      Let $N(f_{o_i}) = \{S|(S, f_{o_i}) \in E\}$;
5      if $N(f_{o_i}) \neq \emptyset$ then
6          $t_{max} = \max\{t(S)|S \in N(f_{o_i})\}$;
7          $t_s(f_{o_i}) = \max\{t_{max}, t_e(f_{o_{i-1}})\}$;
8          $t_e(f_{o_i}) = t_s(f_{o_i}) + L_0(f_{o_i}(A_{f_{o_i}}))$;
9      else
10         $t_s(f_{o_i}) = t_e(f_{o_{i-1}})$;
11         $t_e(f_{o_i}) = t_s(f_{o_i}) + L_0(f_{o_i}(A_{f_{o_i}}))$;
12     $\mathit{\Pi}^{(2)}(Q) = \mathit{\Pi}^{(2)}(Q) \cup \{\pi_0^{t_s(f_{o_i})}(f_{o_i}(A_{f_{o_i}}))\}$;
13 return $\mathit{\Pi}^{(2)}(Q)$;
```

Step 1. (Line 2) Construct a topological order of G'_Q which is a subgraph of G_Q and is deduced from F_Q. Since the cloud carries out services sequentially, the execution order of services must obey the topological order of E-DDAG.

Step 2 and Step 3 are executed iteratively.

Step 2. (Line 3-Line 11) Determine the start time and end time of each service based on the topological order and conditions (a)-(c).

Step 3. (Line 12) Derive the actions of each service based on the start time and end time.

Obviously, the time complexity of Algorithm **??** is $O(|V| + |E|)$ which is the time complexity of the topological order construction algorithm.

2) Construction of $\mathit{\Pi}^{(1)}(Q)$*:* We aim to construct $\mathit{\Pi}^{(1)}(Q)$ based on the $\mathit{\Pi}^{(2)}(Q)$ obtained in Algorithm **??**. After the iterations in Algorithm **??**, the total latency of $\mathit{\Pi}(Q)$ is

$$L(\mathit{\Pi}(Q)) = t_e(f_{o_{|F_Q|}}) = \max\{\Sigma(S)|S \in S_Q\} \qquad (5)$$

where $\Sigma(S) = t(S) + \sum_{f \in F_Q} x(S, f)L_0(f(A_f))$ and $x(S, f) \in \{0, 1\}$ is determined by $\mathit{\Pi}^{(2)}(Q)$. Based on the computation model in Section III-B, $L_0(f(A_f))$ can be estimated for each f and it can be considered as a constant. Without loss of generality, we sort the data sets in S_Q based on the value of $\sum_{f \in F_Q} x(S, f)L_0(f(A_f))$, and $S_Q = \{S_1, S_2, ..., S_{|S_Q|}\}$ where

$$\sum_{f \in F_Q} x(S_i, f)L_0(f(A_f)) > \sum_{f \in F_Q} x(S_{i+1}, f)L_0(f(A_f)).$$

Then we have the following theorem.

Theorem 2. *Let* $S_Q = \{S_1, S_2, ..., S_{|S_Q|}\}$ *where*

$$\sum_{f \in F_Q} x(S_i, f)L_0(f(A_f)) > \sum_{f \in F_Q} x(S_{i+1}, f)L_0(f(A_f)).$$

Given an optimal $\mathit{\Pi}^{(1)}(Q)$*, based on* $\mathit{\Pi}^{(1)}(Q)$*, the data sets in* S_Q *arrive at the cloud at time* $\{t^o(S_j)|S_j \in S_Q\}$*, then we can always construct another optimal* $\bar{\mathit{\Pi}}^{(1)}(Q)$*, and based on* $\bar{\mathit{\Pi}}^{(1)}(Q)$*, the data sets arrive at the cloud at time* $t(S_1) \leq t(S_2) \leq ... \leq t(S_{|S_Q|})$*.*

Proof. Let $T = \max\{\Sigma(S_j)|S_j \in S_Q\}$. For each $S_j \in S_Q$, we can change $t^o(S_j)$ to $\bar{t}^o(S_j) = T - \sum_{f \in F_Q} x(S_j, f)L_0(f(A_f))$. Obviously, $\bar{t}^o(S_j) \geq t^o(S_j)$. Therefore, we can easily construct another feasible solution $\bar{\mathit{\Pi}}^{(1)}(Q)$, based on which the arrival time of each data set equals to $\bar{t}^o(S_j)$, and $\max\{\bar{t}^o(S_j)|S_j \in S_Q\} = T$. It is easy to see that for each two $\bar{t}^o(S_j)$ and $\bar{t}^o(S_i)$, if $j < i$, then $\bar{t}^o(S_j) \leq \bar{t}^o(S_i)$. □

According to Formula (5), latency $L(\mathit{\Pi}(Q))$ is related to $t(S)$s. Based on Theorem 2, we can assume that the arrival time of the data sets in S_Q satisfies $t(S_1) \leq t(S_2) \leq ... \leq t(S_{|S_Q|})$. Given the data sets stored at edge server E_i, $\{S_1(E_i), S_2(E_i), ..., S_{|S_Q|}(E_i)\}$, ($S_j(E_i) \subseteq S_j$) E_i will transmit the data sets based on the order $S_1(E_i), S_2(E_i), ..., S_{|S_Q|}(E_i)$. For each $E_i \in \mathcal{E}$, the actions are $\pi_i^{t_1}(f_{E_0}(S_1(E_i)))$, $\pi_i^{t_2}(f_{E_0}(S_2(E_i))), ..., \pi_i^{t_{|S_Q|}}(f_{E_0}(S_{|S_Q|}(E_i)))$, where for

$$\begin{tikzcd}\boxed{\Gamma_s(\cdot)} \arrow[r] & \boxed{\Gamma_{f_1}(\cdot)} \arrow[r] & \boxed{\Gamma_{f_2}(\cdot)} \arrow[r] & \cdots \arrow[r] & \boxed{\Gamma_{f_{|F_Q|}}(\cdot)} \arrow[r] & \boxed{\Gamma_c(\cdot)}\end{tikzcd}$$

Fig. 7: The QPE $\varPi(Q)$ can be represented by a series of linear connected components.

each $1 \leq k \leq |S_Q|$, $t_k - t_{k-1} = l_{i,0}|S_k(E_i)|$. Thus, we can construct $\varPi^{(1)}(Q)$. Based on Theorem 2, $\varPi^{(1)}(Q)$ is optimal.

3) Construction of $\varPi(Q)$: After Algorithm **??** generates $\varPi^{(1)}(Q)$, a feasible QPE $\varPi(Q)$ can then be obtained. However, we notice that the outputs of Algorithm **??** are also related to the topological order constructed in Step 1. Different topological orders may result in different final QPEs. Considering that the number of topological orders of a graph is limited, we can use the brute force search method to obtain an optimal $\varPi(Q)$ for C1-MIN-QPL. The details are omitted here.

B. Special Case 2

In this special case, the E-DDAG $G_Q(V, E)$ of query Q is linear. We define this special case as follows.

C2-MIN-QPL Problem

Input:

(1) A cloud E_0 and a set of edge servers $\mathcal{E}$;

(2) A query $Q(S_Q)$ and the corresponding E-DDAG $G_Q(V = S_Q \cup F_Q, E)$ generated by the cloud, where G_Q is linear, *i.e.*, $|S_Q| = 1$ and $|E| = |V| - 1$, and each $f \in F_Q$ belongs to Category 2;

(3) The sensory data $S(E_i)$ stored at each edge server E_i;

(4) The computation capability C_i of each $E_i \in \mathcal{E} \cup \{E_0\}$;

(5) the set of services F_i of each edge server E_i;

(6) the communication latency of transmitting one unit of data between $E_i, E_j \in \mathcal{E}^+$.

Outputs: A QPE $\varPi(Q)$ with the minimum latency.

Without loss of generality, we assume $F_Q = \{f_1, ..., f_{|F_Q|}\}$ and $E = \{(S, f_1), (f_1, f_2), (f_2, f_3), ..., (f_{|F_Q|-1}, f_{|F_Q|})\}$. Since G_Q is linear as illustrated in Fig.7, $\varPi(Q)$ can be represented by a series of linear connected components, where $\varGamma_s(\cdot)$ is the component that provides a set of sensory data S and $\varGamma_c(\cdot)$ is the component that transmits the answer of the query to the cloud. Based on the definition in Section III-D, $\varGamma_s(\cdot) = \emptyset$. Therefore, in order to construct $\varPi(Q)$, we can construct $\{\varGamma_{f_1}(\cdot), \varGamma_{f_2}(\cdot), ..., \varGamma_{f_{|F_Q|}}(\cdot), \varGamma_c(\cdot)\}$ by iterations. In the following, we first investigate how to construct a component $\varGamma_{f_i}(\cdot)$, then we consider how to minimize the total latency of $\varPi(Q)$.

1) Component Construction: Suppose that service f_{i-1} has been finished at time $\bar{t}_{i-1}$, and based on the property of Category 2 service, the output of f_{i-1} is distributedly stored at the servers in $\mathcal{E}_{i-1} \subseteq \mathcal{E}^+$. Let S_j ($|S_j| = \alpha_j$) be the distributed output stored at E_j. (*If $i = 1$, then $\mathcal{E}_{i-1}$ is the set of servers that store the data set $S \in S_Q$ and $S_j = S$.*) These data need to be processed by service f_i. The servers in $\mathcal{E}_{i-1}$ will transmit data to the servers in $\mathcal{E}_i$, where $\mathcal{E}_i = \{E_j | f_i \in F_j \wedge E_j \in \mathcal{E}^+\}$ is the set of the servers that have carried out service f_i. Let $S_{j,k} \subseteq S_j$ ($|S_{j,k}| = \alpha_{j,k}$) be the set of data transmitted from $E_j \in \mathcal{E}_{i-1}$ to $E_k \in \mathcal{E}_i$. We have

$$\sum\nolimits_{E_k \in \mathcal{E}_i} \alpha_{j,k} = \alpha_j.$$

For each server $E_k \in \mathcal{E}_i$ and $E_j \in \mathcal{E}_{i-1}$, they will follow the following two steps to construct $\varGamma_{f_i}(\bigcup_{E_j \in \mathcal{E}_{i-1}} S_j)$.

Step 1. E_j transmits data sets $S_{j,k}$ to each server $E_k \in \mathcal{E}_i$, and we have action $\pi_j^{t_{i-1}}(f_{E_k}(S_{j,k}))$.

Step 2. E_k starts carrying out f_i at time

$$t_{k,i} = \bar{t}_{i-1} + \max\{l_{j,k}\alpha_{j,k} | E_j \in \mathcal{E}_{i-1}\}$$

which is the time when E_k has finished receiving data. E_k finishes carrying out f_i at time

$$\bar{t}_{k,i} = t_{k,i} + L_k(f_i(\bigcup_{E_j \in \mathcal{E}_{i-1}} S_{j,k})).$$

According to Formula (3), the size of the output of $f_i(\bigcup_{E_j \in \mathcal{E}_{i-1}} S_{j,k})$ is

$$|f_i(\bigcup_{E_j \in \mathcal{E}_{i-1}} S_{j,k})| = g_{f_i}(|\bigcup_{E_j \in \mathcal{E}_{i-1}} S_{j,k}|) = g_{f_i}(\sum_{E_j \in \mathcal{E}_{i-1}} \alpha_{j,k}).$$

In this step, we have action $\pi_j^{t_{k,i}}(f_i(\bigcup_{E_j \in \mathcal{E}_{i-1}} S_{j,k}))$.

Step 3. At time

$$\bar{t}_i = \max\{\bar{t}_{k,i} | E_k \in \mathcal{E}_i\},$$

service f_i has been finished. The output of f_i will be transmitted to the next component $\varGamma_{i+1}(\cdot)$.

Based on the above three steps, we have

$$\begin{aligned}\varGamma_{f_i}(\bigcup\nolimits_{E_j \in \mathcal{E}_{i-1}} S_j) =& \{\pi_j^{t_{i-1}}(f_{E_k}(S_{j,k})) | E_j \in \mathcal{E}_{i-1}\} \\ & \cup \{\pi_k^{t_{k,i}}(f_i(\bigcup\nolimits_{E_j \in \mathcal{E}_{i-1}} S_{j,k})) | E_k \in \mathcal{E}_i\}.\end{aligned}$$

Obviously, the component construction can be carried out iteratively and we can finally obtain the components in $\{\varGamma_{f_1}(\cdot), \varGamma_{f_2}(\cdot), ..., \varGamma_{f_{|F_Q|}}(\cdot)\}$. We only need to adopt the first step of the component construction method to construct $\varGamma_c(\cdot)$.

2) Minimizing total latency of $\varPi(Q)$: The above section provides a scheme to construct components. However, minimizing the total latency is still a problem. Based on the above section, the latency is mainly related to the amount of data transmitted between servers, denoted as $\alpha_{i,j}$. We have the following program.

$$\begin{aligned}
min \quad & \bar{t}_{|F_Q|} + \max\{l_{j,0}\alpha_{j,0} | E_j \in \mathcal{E}_{|F_Q-1|}\} \qquad (6)\\
\text{s.t.} \quad & f_i \in F_Q : \\
& C_{i_1} : \sum\nolimits_{E_k \in \mathcal{E}_i} \alpha_{j,k} = \alpha_j \qquad \forall E_j \in \mathcal{E}^+ \\
& C_{i_2} : t_{k,i} = \bar{t}_{i-1} + \max\{l_{j,k}\alpha_{j,k} | E_j \in \mathcal{E}_{i-1}\} \\
& \qquad\qquad \forall E_j \in \mathcal{E}^+, E_k \in \mathcal{E}_i \\
& C_{i_3} : \bar{t}_{k,i} = t_{k,i} + L_k(f_i(\bigcup\nolimits_{E_j \in \mathcal{E}_{i-1}} S_{j,k})) \\
& \qquad\qquad \forall E_j \in \mathcal{E}^+, E_k \in \mathcal{E}_i \\
& C_{i_4} : \bar{t}_i = \max\{\bar{t}_{k,i} | E_k \in \mathcal{E}_i\} \\
& C_{i_5} : \alpha_{j+1} = g_{f_i}(\sum\nolimits_{E_j \in \mathcal{E}_{i-1}} \alpha_{j,k}) \\
& C_{i_6} : \alpha_1 = |S|, \mathcal{E}_i = \{E_j | f_i \in F_j \wedge E_j \in \mathcal{E}^+\}
\end{aligned}$$

In Program (6), there are six conditions for each $f_i \in F_Q$. According to Formulas (2) and (3), we have

$$L_k(f_i(\bigcup_{E_j \in \mathcal{E}_{i-1}} S_{j,k})) = \frac{c_{f_i}(|\bigcup_{E_j \in \mathcal{E}_{i-1}} S_{j,k}|)}{C_k} = \frac{c_{f_i}(\sum_{E_j \in \mathcal{E}_{i-1}} \alpha_{j,k})}{C_k},$$

and $\alpha_{j+1} = g_{f_i}(\sum_{E_j \in \mathcal{E}_{i-1}} \alpha_{j,k})$. Since $c_{f_i}(\cdot)$ and $g_{f_i}(\cdot)$ are convex functions, Program (6) is a Convex Optimization Problem which has an optimal solution. Based on the solution of Program (6), we can obtain an optimal $\Pi(Q)$ which has the minimum latency.

V. Approximation Algorithm for the general MIN-QPL Problem

As illustrated in Algorithm **??**, we propose the *Latency Minimization Algorithm* (LMA) to approximately solve the MIN-QPL problem.

The MIN-QPL problem can be considered as an assignment problem which aims at assigning data transmission tasks to communication links and data processing tasks to edge servers to minimize the overall makespan. Action $\pi_i^t(f(S))$ can be considered as assigning task $f(S)$ to E_i at time t, and action $\pi_j^t(f_{E_i}(S))$ can be considered as assigning task S to link (j,i) between E_j and E_i at time t. Therefore, given a query processing plan $\Pi(Q)$, we can partition $\Pi(Q)$ into

$$\Pi(Q) = \bigcup_{E_i \in \mathcal{E}^+} J_i \cup \bigcup_{E_i, E_j \in \mathcal{E}^+} J_{i,j},$$

where J_i and $J_{i,j}$ are regarded as *task sets*, and

$$J_i = \{\pi_i^t(f(S)) | \pi_i^t(f(S)) \in \Pi(Q)\}$$

and

$$\begin{aligned} J_{i,j} =& \{\pi_i^t(f_{E_j}(S)) | \pi_i^t(f_{E_j}(S)) \in \Pi(Q)\} \cup \\ & \{\pi_j^t(f_{E_i}(S)) | \pi_j^t(f_{E_i}(S)) \in \Pi(Q)\}. \end{aligned}$$

The LMA algorithm shown in Algorithm **??** aims at constructing a query plan $\Pi(Q)$ by constructing J_i and $J_{i,j}$ for each server and link, respectively. Obviously, if all the services have been assigned, the query has been answered.

Initially (Line 1 - Line 5 in Alg. ??), given an E-DDAG $G_Q(F_Q \cup S_Q, E)$, we add another service f_{E_0} into F_Q and add another edge (f_r, f_{E_0}) into E, where f_{E_0} is the service that transmits the final answer of Q to the cloud E_0 and f_r is the root of G_Q. Obviously, f_{E_0} belongs to Category 2. For each server E_i and each link (i,j), we initialize $J_i = \emptyset$ and $J_{i,j} = \emptyset$, respectively. Furthermore, $\mathcal{F}_c$ is used to record the services that have been assigned to the servers and $\mathcal{F}_c^+$ is used to record the services that can be processed based on the current $\mathcal{F}_c$. Then the LMA algorithm executes the iterations until $\mathcal{F}_c = F_Q$ (Line 6 - Line 36). Each iteration has the following two phases.

Phase I. Service Preselection (Line 7-Line 14). The algorithm preselects a list of services $\mathcal{F}_c^+$ that can be assigned to the servers based on the current $\mathcal{F}_c$ and the services in $\mathcal{F}_c^+$ are weighted.

Phase II. Service Assignment (Line 15 - Line 36). The algorithm selects a service from $\mathcal{F}_c^+$ and assigns the service and the corresponding data sets to servers and links based on the service type.

The details of these two phases are described in the following sections.

Algorithm 2: Latency Minimization Algorithm

```
Input: $G_Q(V = S_Q \cup F_Q, E)$, $\mathcal{E}^+$.
Output: $\Pi(Q)$
1  $F_Q = F_Q \cup \{f_{E_0}\}$, $E = E \cup \{(f_r, f_{E_0})\}$;
2  $\mathcal{F}_c^+ = \emptyset$;
3  $\mathcal{F}_c = \emptyset$;
4  $\forall E_i \in \mathcal{E}^+, J_i = \emptyset$;
5  $\forall E_i, E_j \in \mathcal{E}^+, J_{i,j} = \emptyset$;
6  while $\mathcal{F}_c \neq F_Q$ do
7      $\mathcal{F}_c^+ = \emptyset$ ;
8      For each $f \in F_Q$, let $K_f$ be the set of children of $f$;
9      For each $f \in F_Q$, let $P_f$ be the set of ancestors of $f$;
10     foreach $f \in F_Q$ do
11         if $K_f \subseteq \mathcal{F}_t$ then
12             $\mathcal{F}_c^+ = \mathcal{F}_c^+ \cup \{f\}$;
13             $w_f = |P_f|$;
14     Sort $\mathcal{F}_c^+$ in descending order by the weight of each $f$;
15     $f^* = \arg\max\{w_f | f \in \mathcal{F}_c^+\}$;
16     if $f^*$ belongs to Category 3 then
17         foreach $E_i \in \mathcal{E}(f^*)$ do
18             $t = \max\{\tau_{j,i} + l_{j,i}|S_j(f^*)| | E_j \in \mathcal{E}^+ \wedge S_j(f^*) \neq \emptyset\}$;
19             $t_i(f^*) = \max\{t, \tau_i\}$;
20             $\bar{t}_i(f^*) = t_i(f^*) + L_i(f^*(\bigcup_{E_j \in \mathcal{E}^+} S_j(f^*)))$;
21             $w_i = \max\{L_c, \bar{t}_i(f^*)\}$;
22         $E_m = \arg\min\{w_i | E_i \in \{E_i | f^* \in F_i\}\}$;
23         Assign $f^*$ to $E_m$ and update task sets;
24         $\mathcal{F}_c = \mathcal{F}_c \cup \{f^*\}$;
25     if $f^*$ belongs to Category 2 then
26         Solve Programming (7);
27         Update task sets based on the solution of (7);
28         $\mathcal{F}_c = \mathcal{F}_c \cup \{f^*\}$;
29     if $f^*$ belongs to Category 1 then
30         for $E_i \in \mathcal{E}(f^*)$ do
31             Solve Programming (8);
32             Let $A_i$ be the assignment strategy;
33             $w_i = T_i$;
34         $E_m = \arg\min\{w_i | E_i \in \mathcal{E}(f^*)\}$;
35         Update task sets based on $A_m$;
36         $\mathcal{F}_c = \mathcal{F}_c \cup \{f^*\}$;
37 $\Pi(Q) = \bigcup_{E_i \in \mathcal{E}^+} J_i \cup \bigcup_{E_i, E_j \in \mathcal{E}^+} J_{i,j}$;
38 return $\Pi(Q)$;
```

Phase I. Service Preselection.

This phase has three steps.

Step 1 (Line 7-Line 9). Initialize $\mathcal{F}_c^+ = \emptyset$ and let K_f and P_f be the sets of children and ancestors of f respectively.

Step 2 (Line 10-Line 13). For each $f \in F_Q$, if $K_f \subseteq \mathcal{F}_c$, it means all f's children have been finished and f can be processed at the current time. Then we add f into $\mathcal{F}_c^+$ and set the weight of f as $|P_f|$.

Step 3 (Line 14). Sort $\mathcal{F}_c^+$ by w_f in descending order. If f has more ancestors and larger weight, then executing f may "unlock" more other services.

Phase II. Service Assignment

Given the task sets of servers $\{J_i | E_i \in \mathcal{E}^+\}$ and links $\{J_{i,j} | E_i, E_j \in \mathcal{E}^+\}$ in the current iteration, a server E_i will finish its current tasks at time

$$\tau_i = \max\{t | \pi_i^t(\cdot) \in J_i\}$$

and a link (i,j) will finish its transmission tasks at

$$\tau_{j,i} = \max\{t|\pi_i^t(\cdot), \pi_j^t(\cdot) \in J_{i,j}\}.$$

Then the current total latency L_c is the time when every server and link have finished their tasks and

$$L_c = \max\{\tau_S, \tau_L\},$$

where $\tau_S = \max\{\tau_i|E_i \in \mathcal{E}^+\}$ and $\tau_L = \max\{\tau_{i,j}|E_i, E_j \in \mathcal{E}^+\}$. The service assignment phase adopts the greedy strategy and aims at carrying out a service that can minimize the latency. Let $S_i = \bigcup_{f\in F_Q} S_i(f)$ be the set of data kept by E_i in the current iteration where $S_i(f)$ is the data set that needs to be processed by service f. The service assignment phase has the following steps.

Step 1 (Line 15). Select the service with the maximum weight f^* from $\mathcal{F}_c^+$.

Step 2 (Line 16 - Line 36). In order to assign f^* to the servers in $\{E_i|f^* \in F_i\}$ and assign the data sets in $\{S_j(f^*)|E_j \in \mathcal{E}^+\}$ to links, we need to consider the service category, and this step is carried out based on the following cases.

Case 1. f^* belongs to Category 3 (Line 16-Line 24). We need to select a server in $\mathcal{E}(f^*) = \{E_i|E_i \in \mathcal{E}^+ \wedge f^* \in F_i\}$ to carry out f^* and minimize the latency, where $\mathcal{E}(f^*)$ is the set of the servers that can carry out service f^*. For each server $E_i \in \mathcal{E}(f^*)$, its weight w_i is determined as follows.

(1) Suppose $E_i \in \mathcal{E}(f^*)$ is selected to carry out f^*.
(2) Each server $E_j \in \{E_j|E_j \in \mathcal{E}^+ \wedge S_j(f^*) \neq \emptyset\}$ will transmit $S_j(f^*)$ to E_i, and E_j has received all the data sets at time

$$t = \max\{\tau_{j,i} + l_{j,i}|S_j(f^*)|\ |E_j \in \mathcal{E}^+ \wedge S_j(f^*) \neq \emptyset\}.$$

(3) E_i starts carrying out $f(f^*)$ at time $t_i(f^*) = \max\{t, \tau_i\}$, and E_i finishes f^* at time

$$\bar{t}_i(f^*) = t_i(f^*) + L_i(f^*(\bigcup\nolimits_{E_j\in\mathcal{E}^+} S_j(f^*))).$$

(4) Let $w_i = \max\{L_c, \bar{t}_i(f^*)\}$.

After weighing each server, we assign f^* to the server with the minimum weight. Then update the corresponding task sets and let $\mathcal{F}_c = \mathcal{F}_c \cup \{f^*\}$.

Case 2. f^* belongs to Category 2 (Line 25-Line 28). Each server $E_i \in \mathcal{E}^+$ with $S_i(f^*) \neq \emptyset$ will assign $S_i(f^*)$ to the servers in $\mathcal{E}(f^*)$. Let $S_i(f^*) = \bigcup_{E_j\in\mathcal{E}(f^*)} S_{i,j}(f^*)$ where $S_{i,j}(f^*)$ is the set of data transmitted from E_i to $E_j \in \mathcal{E}(f^*)$. Obviously, in order to save communication and computation resources, for every two $S_{i,j}(f^*)$ and $S_{i,k}(f^*)$, we have $S_{i,j}(f^*) \cap S_{i,k}(f^*) = \emptyset$. Therefore, we have $|S_i(f^*)| = \sum_{E_j\in\mathcal{E}(f^*)} |S_{i,j}(f^*)|$. Let $|S_i(f^*)| = \alpha_i$ and $|S_{i,j}(f^*)| = \alpha_{i,j}$. Then this case can be formulated as the following program.

$$\begin{aligned} &min \quad T && (7)\\ &\text{s.t.} \quad T = \max\{\max\{\bar{t}_j(f^*)|E_j \in \mathcal{E}(f^*)\},\ L_c\}\\ &\forall E_j \in \mathcal{E}(f^*):\\ &\quad t_j^r = \max\{\tau_{i,j} + l_{i,j}\alpha_{i,j}|E_i \in \mathcal{E}^+ \wedge S_i(f^*) \neq \emptyset\}\\ &\quad t_j(f^*) = \max\{t_j^r, \tau_j\}\\ &\quad \bar{t}_j(f^*) = t_j(f^*) + L_j(f^*(\bigcup\nolimits_{E_i\in\mathcal{E}^+} S_{i,j}(f^*))) \end{aligned}$$

In Program (7), t_j^r is the time when server E_j has received all the data sets, and $t_j(f^*)$ is the time when E_j starts carrying out service f^*. Then at time $\bar{t}_j(f^*)$, E_j finishes f^*. Therefore, $T = \max\{\max\{\bar{t}_j(f^*)|E_j \in \mathcal{E}(f^*)\},\ L_c\}$ is the updated latency. Program (7) is a convex optimization problem, and by solving Program (7), we can obtain an optimal assignment strategy that can minimize the latency.

Case 3. f^* belongs to Category 1 (Line 29-Line 36). To assign f^* to the servers, we first select a set of servers from $\mathcal{E}(f^*)$ to distributedly carry out f^* and then select a server from $\mathcal{E}(f^*)$ to aggregate the distributed outputs. We have the following steps (Line 28 - Line 31) to assign f^*.

(1) Suppose $E_k \in \mathcal{E}(f^*)$ is selected to aggregate the distributed outputs.
(2) Solve the following program.

$$\begin{aligned} &min \quad T && (8)\\ &\text{s.t.} \quad T = \max\{\bar{t}_k^a,\ L_c\}\\ &t_k^r = \max\{\bar{t}_j(f^*) + l_{i,j}g_{f^*}(\sum\nolimits_{E_i\in\mathcal{E}^+} \alpha_{i,j})|E_j \in \mathcal{E}(f^*)\}\\ &t_k^a = \max\{t_k^r, t_k(f^*), \tau_k\}\\ &\bar{t}_k^a = t_k^a + L_j(f^*(\bigcup\nolimits_{E_j\in\mathcal{E}(f^*)} f^*(S_{i,j}(f^*))))\\ &\forall E_j \in \mathcal{E}(f^*):\\ &\quad t_j^r = \max\{\tau_{i,j} + l_{i,j}\alpha_{i,j}|E_i \in \mathcal{E}^+ \wedge S_i(f^*) \neq \emptyset\}\\ &\quad t_j(f^*) = \max\{t_j^r, \tau_j\}\\ &\quad \bar{t}_j(f^*) = t_j(f^*) + L_j(f^*(\bigcup\nolimits_{E_i\in\mathcal{E}^+} S_{i,j}(f^*))) \end{aligned}$$

The meanings of t_j^r, $t_j(f^*)$, and $\bar{t}_j(f^*)$ are the same as those in Program (7). $g_{f^*}(\sum_{E_i\in\mathcal{E}^+} \alpha_{i,j})$ is the size of $f^*(\bigcup_{E_i\in\mathcal{E}^+} S_{i,j}(f^*))$, and t_k^r is the time when E_k has received all the distributed outputs. t_k^a is the time when E_k can aggregate the outputs and $\bar{t}_k^a$ is the time when E_k finishes the aggregation. T is the latency after carrying out f^*. Obviously, Program (8) is also a convex optimization problem, and we can obtain the optimal solution of Program (8).
(3) The optimal solution of Program (8) is a strategy to assign f^*. Let A_k be the assignment strategy and T_k be the optimal latency of Program (8). The weight of A_k is $w_k = T_k$.

After the above steps, we have $|\mathcal{E}(f^*)|$ assignment strategies. Select the strategy with the minimum weight to assign f^*.

Finally (Line 37), after the above two phases, we can obtain a query processing plan $\Pi(Q)$.

In Step 2 of Phase II, the LMA algorithm constructs a feasible component for each service in F_Q and finally, the answer of Q is transmitted to the cloud. Obviously, $\mathit{\Pi}(Q)$ is a feasible query plan that can answer query Q.

Program (7) and Program (8) can be solved by many existing convex optimization methods [42]. Let $O(f_{CO})$ be the time complexity of the adopted method. The time complexity of the LMA algorithm is $O(|F_Q+1|f_{CO})$ where $|F_Q+1|$ is the number of the iterations.

VI. Theoretical Analysis

In this section, we will analyze the upper bound and lower bound of the MIN-QPL problem, and the approximation ratio of the LMA algorithm.

A. Upper Bound and Lower Bound of the MIN-OPL Problem

Given an instance of the MIN-QPL problem $I(G_Q(F_Q \cup S_Q, E), \mathcal{E}^+)$, we can construct the instances of C1-MIN-QPL and C2-MIN-QPL, denoted as I_{C1} and I_{C2} respectively.

Construction of I_{C1}. I_{C1} has the same E-DDAG G_Q and $\mathcal{E}^+$ as I. However, for each server E_i of I_{C1}, $E_i \cap F_Q = \emptyset$.

Construction of I_{C2}. The E-DDAG of I_{C2} $G_Q^{(2)}$ is a path of G_Q and the root node in G_Q is the last node of $G_Q^{(2)}$. For each service f in G_Q^2, we regard f as a Category 2 service.

Let $\mathit{\Pi}_{C1}(Q)$, $\mathit{\Pi}_{C2}(Q)$ and $\mathit{\Pi}(Q)$ be the optimal solutions of I_{C1}, I_{C2} and I respectively. We have the following theorems.

Theorem 3. *$L(\mathit{\Pi}_{C1}(Q))$ is the upper bound of $L(\mathit{\Pi}(Q))$, i.e., $L(\mathit{\Pi}_{C1}(Q)) \geq L(\mathit{\Pi}(Q))$.*

Proof. A feasible solution of I is to carry out all the services in the cloud. Therefore, we can consider $\mathit{\Pi}_{C1}(Q)$ as a solution of the MIN-OPL instance I, and $L(\mathit{\Pi}_{C1}(Q)) \geq L(\mathit{\Pi}(Q))$. □

Theorem 4. *$L(\mathit{\Pi}_{C2}(Q))$ is the lower bound of $L(\mathit{\Pi}(Q))$, i.e., $L(\mathit{\Pi}_{C1}(Q)) \leq L(\mathit{\Pi}(Q))$.*

Proof. $\mathit{\Pi}_{C2}(Q)$ only carries out a subset of the services in G_Q. Since $\mathit{\Pi}_{C2}(Q)$ is the optimal solution of I_{C2}, we have $L(\mathit{\Pi}_{C1}(Q)) \leq L(\mathit{\Pi}(Q))$. □

B. Approximation Ratio of the LMA algorithm

Given an E-DDAG $G_Q(F_Q \cup S_Q, E)$, we can partition F_Q into levels. The first level V_1 contains the root node f_r, $V_1 = \{f_r\}$. For every two levels V_i and V_{i-1}, we have $V_i = \{f | f\ \textit{has at least one parent node in}\ V_{i-1}\}$. For every two arbitrary $f, f' \in V_i$, f is neither the ancestor nor the child of f'. Let N_l be the number of the levels. We have $F_Q = \bigcup_{1 \leq i \leq N_l} V_i$. The following theorem proves the approximation ratio of the LMA algorithm.

Theorem 5. *Given the optimal solution $\mathit{\Pi}(Q)$ and the solution $\mathit{\Pi}_A(Q)$ obtained by the LMA algorithm, we have*

$$\frac{L(\mathit{\Pi}_A(Q))}{L(\mathit{\Pi}(Q))} \leq \max\{k_l^{max} k_f^{max} \delta_l, k_l^{max} \mathit{\Delta}(k_f^{max})\},$$

where $k_l^{max} = \max\{|V_i| | 1 \leq i \leq N_l\}$, $k_f^{max} = \max\{k_f | k_f = |\{E_i | f \in F_i \wedge E_i \in \mathcal{E}^+\}| \wedge f \in F_Q\}$, and $\mathit{\Delta}(x) \geq \max\{\frac{c_f(x|S|)}{c_f(|S|)} | f \in F_Q\}$.

Proof. Consider another LMA-based algorithm, which carries out services by levels. It has following steps.

Step 1. Let N_c be the current level and $N_c = N_l$.

Step 2. The services in the last level V_{N_l} are carried out.

Step 3. After all the services in V_{N_l} have been finished, then let $N_c = N_c - 1$.

Step 4. Process Step 1 - Step 3 in iteration until $N_c = 0$.

These steps will run repeatedly until all the services in F_Q are finished. Furthermore, in the LMA-based algorithm, a service can be finished if and only if the previous service has been finished already. Obviously, we have $L(\mathit{\Pi}_B(Q)) \geq L(\mathit{\Pi}_A(Q))$ where $\mathit{\Pi}_B(Q)$ and $\mathit{\Pi}_A(Q)$ are the solutions returned by the LMA-based algorithm and the LMA algorithm.

Without loss of generality, let $V_i = \{f_{i_1}, f_{i_2}, ..., f_{i_{k_i}}\}$ and $f_{i_1} = \arg\max\{c_f(|S_f|) | f \in V_i\}$. Consider a query plan $\mathit{\Pi}'$ that only carries out $f_{1_1}, f_{2_1}, ..., f_{N_{L1}}$ and treat them as Category 2 services. Suppose each server E_i in $\mathcal{E}^+$ contains a data set S_i that will be processed by f_{i_1}. We can assume that $S_j = \max\{S_i | E_i \in \mathcal{E}^+\}$. To carry out f_{i_1}, $\mathit{\Pi}'$ assigns the data sets needed by f_{i_1} to the servers, and $\mathit{\Pi}'$ needs at least $l^{min}|S_j| + \frac{c_{f_{i_1}}(|S_j|)}{C_k}$ time, where $l^{min} = \min\{l_{i,j} | E_i, E_j \in \mathcal{E}^+\}$ is the minimum communication latency and S_j is a piece of data assigned to server E_j. However, on the other hand, $\mathit{\Pi}_B(Q)$ needs at most $l^{max}|S| + \frac{c_{f_{i_1}}(|S|)}{C_{max}}$ time, where $S = \bigcup_{E_j \in \mathcal{E}^+} S_j$ is the whole data set that needs to be processed by f_{i_1} and C_{max} is the maximum computation capability since the LMA-based algorithm will choose the strategy with the minimum latency. Since $f_{i_1} = \arg\max\{c_f(|S_f|) | f \in V_i\}$, carrying out all the services in V_i needs at most $k_i(l^{max}|S| + \frac{c_{f_{i_1}}(|S|)}{C_{max}})$ time. We then have

$$\begin{aligned}\frac{k_i(l^{max}|S| + \frac{c_{f_{i_1}}(|S|)}{C_{max}})}{l^{min}|S_j| + \frac{c_{f_{i_1}}(|S_j|)}{C_k}} &\leq \frac{k_i(l^{max}|S| + \frac{c_{f_{i_1}}(|S|)}{C_k})}{l^{min}|S_j| + \frac{c_{f_{i_1}}(S_j)}{C_k}} \\ &\leq \frac{k_i(l^{max}k_f^{max}|S_j| + \frac{\mathit{\Delta}(S_j)c_{f_{i_1}}(|S_j|)}{C_k})}{l^{min}|S_j| + \frac{c_{f_{i_1}}(|S_j|)}{C_k}} \\ &\leq \max\{k_i k_f^{max}\delta_l, k_i \mathit{\Delta}(k_f^{max})\},\end{aligned}$$

where $\delta_l = \frac{l^{max}}{l^{min}}$, $\mathit{\Delta}(k) \geq \max\{\frac{c_f(k|S|)}{c_f(|S|)} | f \in F_Q\}$ and $k_f^{max} = \max\{k_f | k_f = |\{E_i | f \in F_i \wedge E_i \in \mathcal{E}^+\}| \wedge f \in F_Q\}$. Let $k_l^{max} = \max\{|V_i| | 1 \leq i \leq N_l\}$. We have

$$\begin{aligned}\frac{L(\mathit{\Pi}_B(Q))}{L(\mathit{\Pi}')} &\leq \frac{N_l(k_i(l^{max}|S| + \frac{c_{f_{i_1}}(|S|)}{C_{max}}))}{N_l(l^{min}|S_j| + \frac{c_{f_{i_1}}(|S_j|)}{C_k})} \\ &\leq \max\{k_l^{max}k_f^{max}\delta_l, k_l^{max}\mathit{\Delta}(k_f^{max})\}.\end{aligned}$$

Obviously, $L(\mathit{\Pi}(Q)) \geq L(\mathit{\Pi}')$. Then we have

$$\begin{aligned}\frac{L(\mathit{\Pi}_A(Q))}{L(\mathit{\Pi}(Q))} \leq \frac{L(\mathit{\Pi}_B(Q))}{L(\mathit{\Pi}(Q))} &\leq \frac{L(\mathit{\Pi}_B(Q))}{L(\mathit{\Pi}')} \\ &\leq \max\{k_l^{max}k_f^{max}\delta_l, k_l^{max}\mathit{\Delta}(k_f^{max})\}.\end{aligned}$$

□

VII. Simulations

A. Simulation Settings

In this section, we evaluate the performance of the LMA algorithm through extensive simulations. We run the LMA algorithm on different EDMSs. In these EDMSs, the number of edge servers varies from 5 to 30. Let C be the average computation capability of the edge servers. We assume that the value of C is in the range of $[10GHz, 40GHz]$ [21]. The computation capability of each edge server is randomly assigned based on the average value C. Furthermore, we assume that the computation capability of the cloud E_0 is five times larger than that of the edge servers. We assume that the data rate between every two servers varies from $200Mbps$ to $1000Mbps$ [43]. For each E-DDAG $G_Q(F_Q \cup S_Q, E)$, we assume that the number of data sources $|S_Q|$ and services $|F_Q|$ vary from 2 to 8 and 4 to 14, respectively. For each service $f \in F_Q$, the specific functions $c_f(\cdot)$ and $g_f(\cdot)$ are linear functions and they are randomly generated. Let $|S|$ be the average size of different types of data sets. In the simulations, we assume that $|S|$ varies from $20GB$ to $70GB$.

Through the simulations, we investigate the impact of different system parameters on the latency incurred by the LMA algorithm. The system parameters include the number of edge servers, the number of services, number of data sources, average size of each data set, average computation capability, and transmission latency.

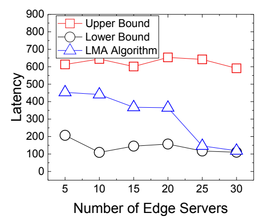

Fig. 8: Impact of $|\mathcal{E}|$.

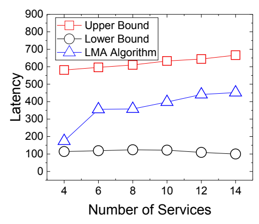

Fig. 9: Impact of $|F_Q|$.

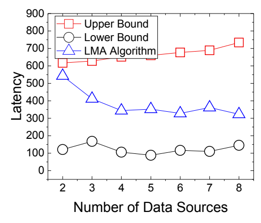

Fig. 10: Impact of $|S_Q|$.

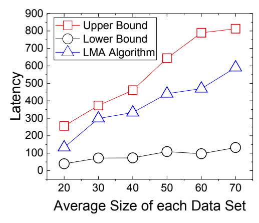

Fig. 11: Impact of $|S|$.

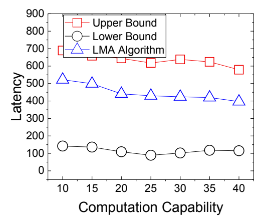

Fig. 12: Impact of C.

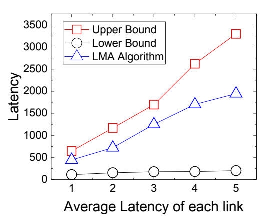

Fig. 13: Impact of l.

B. The Impact of Different System Parameters

In this section, we investigate the impact of different system parameters on the LMA algorithm and the upper bound and the lower bound of latency. The upper bound and lower bound of latency can be obtained based on Theorem 3 and Theorem 4. According to Theorem 3, the upper bound of latency is also the latency caused by the centralized query processing method.

1) The impact of the number of edge servers: In this group of simulations, we generate different EDMSs and vary the number of edge servers from 5 to 30. In these EDMSs, the average computation capability of each edge server is $20GHz$, the average data rate of each communication link is $1000Mbps$, the number of services is 10, the number of data sources is 4, and the average size of each data size is $50G$. We run the LMA algorithm on these EDMSs and calculate the upper bound and lower bound of the query processing latency. The results are shown in Fig.8. It can be seen that the latency incurred by the LMA algorithm is 50.3% better than the upper bound, and is close to the lower bound as the number of edge servers increases. If there are more edge servers in the EDMS, there are more computation resources. Therefore, when there are 30 edge servers, the performance of the LMA algorithm is close to the lower bound. However, when there are only 5 edge servers, the latency incurred by the LMA algorithm is 2.18 times larger than the lower bound. Moreover, since the centralized query processing method is not highly correlated with the number of edge servers, the latency's upper bound has no significant change when the size of $|\mathcal{E}|$ increases.

2) The impact of the number of services: In this group of simulations, we generate an EDMS with 10 edge servers, and vary the number of services in each E-DDAG from 4 to 14. The values of other system parameters are similar to the ones in the above group of simulations. We randomly generate hundreds of E-DDAGs with different numbers of services and carry them out based on the LMA algorithm. The average query processing latency of each type of E-DDAG is illustrated in Fig.9. We can see that the latency incurred by the LMA algorithm is 43.4% smaller than the upper bound, and is 3.05 times higher than the lower bound. In this case, the ratio between the latency incurred by the LMA algorithm and the optimal latency is much smaller than the approximation ratio provided in Theorem 5. Since the computation capability of the cloud is powerful, the upper bound slowly increases along with the increase of the number of services. On the other hand, when the number of services varies from 4 to 14, the latency incurred by the LMA algorithm increases from $174s$ to $372s$. Furthermore, since the lower bound latency is caused by a subset of services, the lower bound has no significant change when $|F_Q|$ increases.

3) The impact of the number of data sources: The number of data sources is another important parameter in E-DDAG. In this group of simulations, we vary the size of $|S_Q|$ from 2 to 8. The number of services is fixed to 10 and we generate hundreds of E-DDAGs based on the different numbers of data sources. The settings of other system parameters are similar to the ones in the previous simulations. We adopt the LMA algorithm to process these queries and illustrate the average query processing latency in Fig.10. It can be seen that the query processing latency incurred by the LMA algorithm is 42.7% lower than the upper bound. It is interesting to see that as the number of data sources increases, the query processing latency incurred by the LMA algorithm decreases. The reason is when the size of S_Q increases, the dependency among services becomes weaker, and a query is more feasible to be processed in a distributed manner.

4) The impact of the average size of each data set: The size of each data set determines the transmission latency between every two servers and is a key parameter that impacts query processing latency. In this group of simulations, we assume that there are four sensory data sets belonging to different data sources $\{S^{(1)}, S^{(2)}, S^{(3)}, S^{(4)}\}$, and we adjust the average size of these data sets $\frac{1}{4}\sum_{1\leq i\leq 4} S^{(i)}$ from $20GB$ to $70GB$. Moreover, the average computation capability of each edge server is $20GHz$, the average data rate of each communication link is $1000Mbps$, the number of services is 10, and the number of edge servers is 10. After running the LMA algorithm and calculating the upper bound and lower bound of the latency, the relation between size of data set and query processing latency is shown in Fig.11. We can see that when the average size of each data set increases, the upper bound and lower bound of the query processing latency incurred by the LMA algorithm increase simultaneously. When the size varies from $20GB$ to $70GB$, the query processing latency grows by 4.4 times. Since the LMA algorithm processes the query distributedly, the data size has a high impact. However, the LMA algorithm is still better than a centralized data processing method, and the performance of the LMA algorithm is 32% better than that obtained by a centralized data processing method (or the upper bound).

5) The impact of the average computation capability: The computation capability of each edge server determines the computation latency and also affects the query processing latency. In this group of simulations, we deploy ten edge servers $E_1, E_2, ..., E_{10}$, and the computation capability of each edge server is randomly generated based on an average value C where $C = \frac{1}{10}\sum_{1\leq i\leq 10} C_i$. The value of C varies from $10GHz$ to $40GHz$. Moreover, we assume that the computation capability of the cloud is five times higher than the average value. The number of services and data sources of each E-DDAG are 10 and 4, respectively. After running the LMA algorithm and calculating the upper bound and the lower bound of the latency, the simulation results are shown in Fig.12. We can see that when the computation capability increases, the computation latency of each server is reduced, and the query processing latency of the LMA algorithm, the upper bound and the lower bound of the latency all decrease. The query processing latency of the LMA algorithm decreases 24% when the computation capability increases from $10GHz$ to $40GHz$. Furthermore, the latency caused by the LMA algorithm is 29.7% less than the upper bound.

C. The impact of the average latency of each link

The data transmission rate has a high impact on query processing latency. In this group of simulations, we simulate the data transmission rate based on the 5G network [43]. We assume that the average data transmission rate varies from $200Mbps$ to $1000Mbps$, which means the latency of transmitting $1G$ data varies from $1s$ to $5s$. Based on different average data transmission rates, we generate different EDMSs and run the LMA algorithm on these EDMSs. The simulation results are shown in Fig.13. We can see that when the data rate decreases, the latency of the LMA algorithm and the upper bound increase significantly. However, since the services are assumed to be fully distributedly carried out, the lower bound increases slowly when the data rate decreases. Furthermore, the latency incurred by the LMA algorithm is 35.7% lower than the upper bound, and when the data rate increases from $200Mbps$ to $1000Mbps$, the latency of the LMA algorithm increases almost 5 times. We can conclude that the latency of each communication link has a high impact on the data processing latency.

VIII. Conclusion

In this paper, we investigate the distributed query processing problem in EDMS. We define the MIN-QPL problem which aims to deriving a query processing plan with the minimum query response latency. We prove that the MIN-QPL problem is NP-Hard. We first investigate two special cases of the MIN-QPL problem and prove that for these two cases, we can obtain an optimal solution for the MIN-QPL problem in polynomial time. Then we propose an approximation algorithm to solve the MIN-QPL problem and provide analysis that the algorithm has a feasible approximation ratio. We also illustrate the simulation results to evaluate the performance of the algorithm, and the simulation results imply that the algorithm is effective. In this work, we focus on how to reduce the latency, and in our future works, we will investigate the energy efficiency algorithms in the EDMS.

Acknowledgment

This work is partly supported by the National Science Foundation (NSF) under grant NOs. 2011845, 1741277, 1912753 and 1704287.

References

[1] X. Zheng and Z. Cai, "Privacy-preserved data sharing towards multiple parties in industrial iots," *IEEE Journal on Selected Areas in Communications*, 2020.

[2] B. Zhou, J. Li, X. Wang, Y. Gu, L. Xu, Y. Hu, and L. Zhu, "Online internet traffic monitoring system using spark streaming," *Big Data Mining and Analytics*, vol. 1, no. 1, pp. 47–56, 2018.

[3] D. Yu, Y. Zou, J. Yu, Y. Zhang, F. Li, X. Cheng, F. Dressler, and F. C. Lau, "Implementing abstract mac layer in dynamic networks," *IEEE Transactions on Mobile Computing*, 2020.

[4] C. Wang, C. Wang, Z. Wang, X. Ye, J. X. Yu, and B. Wang, "Deep-Direct: Learning Directions of Social Ties with Edge-based Network Embedding," vol. 31, no. 12, pp. 2277–2291, December 2019.

[5] Y. Xiao, Y. Jia, C. Liu, X. Cheng, J. Yu, and W. Lv, "Edge computing security: State of the art and challenges," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1608–1631, 2019.

[6] J. Shi, R. Zhang, and Y. Zhang, "Secure range queries in tiered sensor networks," in *IEEE INFOCOM 2009*. IEEE, 2009, pp. 945–953.

[7] ——, "A spatiotemporal approach for secure range queries in tiered sensor networks," *IEEE transactions on wireless communications*, vol. 10, no. 1, pp. 264–273, 2010.

[8] S. Yoon and C. Shahabi, "Exploiting spatial correlation towards an energy efficient clustered aggregation technique (cag)[wireless sensor network applications]," in *IEEE International Conference on Communications, 2005. ICC 2005. 2005*, vol. 5. IEEE, 2005, pp. 3307–3313.

[9] ——, "The clustered aggregation (cag) technique leveraging spatial and temporal correlations in wireless sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 3, no. 1, pp. 3–es, 2007.

[10] T.-Y. Fu, W.-C. Peng, and W.-C. Lee, "Parallelizing itinerary-based knn query processing in wireless sensor networks," *IEEE Transactions on knowledge and Data engineering*, vol. 22, no. 5, pp. 711–729, 2009.

[11] S. Cheng, Z. Cai, and J. Li, "Curve query processing in wireless sensor networks," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 11, pp. 5198–5209, 2015.

[12] S. Cheng and J. Li, "Sampling based (epsilon, delta)-approximate aggregation algorithm in sensor networks," in *2009 29th IEEE International Conference on Distributed Computing Systems*. IEEE, 2009, pp. 273–280.

[13] Z. Cai and Z. He, "Trading private range counting over big iot data," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 144–153.

[14] Internet of things - number of connected devices worldwide 2015-2025. [Online]. Available: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

[15] T. Zhu, T. Shi, J. Li, Z. Cai, and X. Zhou, "Task scheduling in deadline-aware mobile edge computing systems," *IEEE Internet of Things Journal*, 2018.

[16] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Trans. Communications*, vol. 64, no. 10, pp. 4268–4282, 2016.

[17] P. D. Lorenzo, S. Barbarossa, and S. Sardellitti, "Joint optimization of radio resources and code partitioning in mobile edge computing," *Computer Science*, 2013.

[18] C. You, K. Huang, H. Chae, and B. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wireless Communications*, vol. 16, no. 3, pp. 1397–1411, 2017.

[19] Y. Mao, J. Zhang, S. Song, and K. B. Letaief, "Power-delay tradeoff in multi-user mobile-edge computing systems," in *IEEE GLOBECOM*, 2016, pp. 1–6.

[20] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Trans. Signal and Information Processing over Networks*, vol. 1, no. 2, pp. 89–103, 2015.

[21] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, 2016.

[22] M. Molina, O. Muñoz, A. Pascual-Iserte, and J. Vidal, "Joint scheduling of communication and computation resources in multiuser wireless application offloading," in *25th PIMRC*, 2014, pp. 1093–1098.

[23] Y. Yu, J. Zhang, and K. B. Letaief, "Joint subcarrier and CPU time allocation for mobile edge computing," in *2016 IEEE Global Communications Conference, GLOBECOM 2016, Washington, DC, USA, December 4-8, 2016*, 2016, pp. 1–6.

[24] Z. Sheng, C. Mahapatra, V. C. M. Leung, M. Chen, and P. K. Sahu, "Energy efficient cooperative computing in mobile wireless sensor networks," *IEEE Trans. Cloud Computing*, vol. 6, no. 1, pp. 114–126, 2018.

[25] X. Cao, F. Wang, J. Xu, R. Zhang, and S. Cui, "Joint computation and communication cooperation for mobile edge computing," in *WiOpt*, 2018, pp. 1–6.

[26] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung, "Dynamic service migration and workload scheduling in edge-clouds," *Performance Evaluation*, vol. 91, pp. 205–228, 2015.

[27] R. Cziva, C. Anagnostopoulos, and D. P. Pezaros, "Dynamic, latency-optimal vnf placement at the network edge," in *Ieee infocom 2018-ieee conference on computer communications*. IEEE, 2018, pp. 693–701.

[28] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 10–18.

[29] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading dependent tasks in mobile edge computing with service caching," in *INFOCOM 2020*. IEEE.

[30] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

[31] D. Miao, J. Yu, and Z. Cai, "The hardness of resilience for nested aggregation query," *Theoretical Computer Science*, vol. 803, pp. 152–159, 2020.

[32] J. Li, M. Siddula, X. Cheng, W. Cheng, Z. Tian, and Y. Li, "Approximate data aggregation in sensor equipped iot networks," *Tsinghua Science and Technology*, vol. 25, no. 1, pp. 44–55, 2020.

[33] D. Yu, L. Zhang, Q. Luo, X. Cheng, J. Yu, and Z. Cai, "Fast skyline community search in multi-valued networks," in *BIG DATA MINING AND ANALYTICS*, 2020.

[34] Q.-S. Hua, Y. Shi, D. Yu, H. Jin, J. Yu, Z. Cai, X. Cheng, and H. Chen, "Faster parallel core maintenance algorithms in dynamic graphs," *IEEE Transactions on Parallel and Distributed Systems*, 2019.

[35] Z. He, Z. Cai, S. Cheng, and X. Wang, "Approximate aggregation for tracking quantiles and range countings in wireless sensor networks," *Theoretical Computer Science*, vol. 607, pp. 381–390, 2015.

[36] T. Shi, J. Li, G. Hong, and Z. Cai, "Coverage in battery-free wireless sensor networks," in *INFOCOM*. IEEE, 2018.

[37] T. Shi, S. Cheng, J. Li, and Z. Cai, "Constructing connected dominating sets in battery-free networks," in *INFOCOM*. IEEE, 2017, pp. 1–9.

[38] T. Shi, Z. Cai, J. Li, and H. Gao, "The energy-data dual coverage in battery-free sensor networks," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 338–347.

[39] T. Shi, C. Cheng, Z. Cai, and J. Li, "Adaptive connected dominating set discovering algorithm in energy-harvest sensor networks," in *INFOCOM*, 2016.

[40] X. Yang, X. Jia, M. Yuan, and D. Yan, "Real-time facial pose estimation and tracking by coarse-to-fine iterative optimization," *Tsinghua Science and Technology*, vol. 25, no. 5, pp. 690–700, 2020.

[41] L. Shi, Y. Wu, L. Liu, X. Sun, and L. Jiang, "Event detection and identification of influential spreaders in social media data streams," *Big Data Mining and Analytics*, vol. 1, no. 1, pp. 34–46, 2018.

[42] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[43] Dell'oro: 5g mobile backhaul + wdm equipment market grew 15[Online]. Available: https://techblog.comsoc.org/2018/12/04/delloro-5g-mobile-backhaul-wdm-equipment-market-grew-15-in-q3/

Zhipeng Cai received his PhD and M.S. degree in Department of Computing Science at University of Alberta, and B.S. degree from Department of Computer Science and Engineering at Beijing Institute of Technology. Dr. Cai is currently an Assistant Professor in the Department of Computer Science at Georgia State University. Prior to joining GSU, Dr. Cai was a research faculty in the School of Electrical and Computer Engineering at Georgia Institute of Technology. Dr. Cai's research areas focus on Networking and Big data. He is the steering committee chair of the International Conference on Wireless Algorithms, Systems, and Applications (WASA). Dr. Cai served/is serving on the editorial boards of several technical journals (e.g. IEEE Internet of Things Journal, IEEE Transactions on Knowledge and Data Engineering (TKDE), IEEE Transactions on Vehicular Technology (TVT)) He also chaired several international conferences (e.g. IEEE ICDCS 2019, IEEE IPCCC18). Dr. Cai is the recipient of an NSF CAREER Award.

Tuo Shi received his BS and MS degrees in computer science from Harbin Institute of Technology, China. He is currently working toward the PhD degree in the School of Computer Science and Technology at Harbin Institute of Technology. His research interests include sensor networks, battery-free networks and mobile edge computing. He has published papers in refereed journals and conferences, including IEEE/ACM Transactions on Networking (TNET), IEEE Internet of Things Journal (IoT-J), ACM Transactions on Sensor Network (TOSN), IEEE International Conference on Computer Communications (INFOCOM) and IEEE International Conference on Distributed Computing Systems (ICDCS). He is also the reviewer of distinguished journals, including TWC, IoT-J, TKDE and JOCO.