Burst Load Evacuation Based on Dispatching and Scheduling In Distributed Edge Networks

Shuiguang Deng, *Senior Member, IEEE*, Cheng Zhang, Chang Li, Jianwei Yin, Schahram Dustdar, *Fellow, IEEE*, and Albert Y. Zomaya, *Fellow, IEEE*

Abstract—Edge computing, a fast evolving computing paradigm, has spawned a variety of new system architectures and computing methods discussed in both academia and industry. Edge servers are directly deployed near users' equipment or devices owned by telecommunications companies. This allows for offloading computing tasks of various devices nearby to edge servers. Due to the shortage of computing resources in edge computing networks, they are often not as sufficient as the computing resources in a cloud computing center. This leads to the problem of service load imbalance once the load in the edge computing network increases suddenly. To solve the problem of "load evacuation" in edge environments, we introduce a strategy when the number of service requests for mobile devices or IoT devices increases rapidly within a short period of time. Therefore, to prevent poor QoS in edge computing, service load should be migrated to other edge servers to reduce the overall delay of these service requests. In this article, we have introduced a strategy with two stages during the burst load evacuation. Based on an optimal routing search at the dispatching stage, tasks will be migrated from the server in which the burst load occurs to other servers as soon as possible. Subsequently, with the assistance of the remote server and edge servers, these tasks are processed with the highest efficiency through the proposed parallel structure at the scheduling stage. Finally, we conduct numerical experiments to clarify the superiority of our algorithm in an edge environment simulation.

Index Terms—Edge computing, routing search, online scheduling

1 Introduction

In comparison to cloud computing, edge computing has numerous extensions to today's network architectures. Edge computing refers to a framework that facilitates low latency of edge servers deployed close to the sources of data or applications that deployed on users' devices [1], [2]. Edge computing is defined as a computing pattern along the path with any resources of computing and network between users and remote cloud center[3]. The edge network can be any functional part from users' side to the center of the cloud server. These parts consist of different entities playing an important role to integrate traditional networks. They provide computing, storage, cache, and transmission to support different services of application in real-time, dynamic and intelligent service for clients in the edge network [4]. Unlike traditional cloud computing (centralized servers), algorithms and strategy selection in edge computing pushes the computing and intelligence closer to actual activity generated by users. It requires services computing of central servers moving to the edge. Thus, there are differences related to multi-heterogeneous processing, bandwidth capacity, resources utilization, and privacy protection[5], [6], [7].

Emerging patterns of service utilization require increasingly more computing capabilities provided by end users. Offloading addresses problems of users' devices regarding storage resources, computing performance, and energy efficiency. Recent studies have introduced this technique consisting of the offloading algorithm, strategy for offloading and the offloading system [1].

Offloading educes to the new problem in edge environments with the number of end users increase continuously. As the rapid growth in IoT and mobile devices, limited computation and network resources among end users in a specific edge network will become more and more common. Due to the heterogeneous of edge networks, the traffic load in the network will endure nonuniform and dynamic burst load. all the time[8]. Some recently published researches introduce that the bursty traffic in radio access network (C-RAN) architecture is an important problem and difficult to solve[9], however, edge network combining edge servers and remote cloud center is proposed as a popular technique nowadays in C-RAN for 5G [10]. It's a challenge to tackle the problem when burst load occurs at the edge environment. To reduce the latency and to maintain an acceptable QoS of edge networks, we focus on the problem of the burst load evacuation when offloading is working in an edge environment.

There are a number of occasions where load burst may occur. Some examples include sudden shopping crowds in shopping malls, crowds on festival streets, traffic roads during rush hour, and crowds in touristic areas. Such scenes have obvious regional characteristics or special temporal characteristics. For example, in vehicle edge computing network infrastructure, it's a fact that edge servers deployed

- *Shuiguang Deng, Cheng Zhang, Chang Li, and Jianwei Yin are with the College of Computer Science, Zhejiang University, Hangzhou 310027, PR China. E-mail: {dengsg, coolzc, 21921187, zjuyjw}@zju.edu.cn.*
- *Schahram Dustdar is with the Distributed Systems Group, TU Wien, Vienna 1040, Austria. E-mail: dustdar@dsg.tuwien.ac.at.*
- *Albert Y. Zomaya is with the School of Computer Science, The University of Sydney, Sydney 2006, Australia. E-mail: albert.zomaya@sydney.edu.au.*

Manuscript received 9 June 2020; revised 10 Jan. 2021; accepted 13 Jan. 2021. Date of publication 18 Jan. 2021; date of current version 19 Feb. 2021. (Corresponding author: Shuiguang Deng.) Recommended for acceptance by F. Wolf. Digital Object Identifier no. 10.1109/TPDS.2021.3052236

with small-cell BS close to a vehicle may not be enough to satisfy the computation demands burst of each user end (vehicles)[11].

Besides, lots of IoT/mobile devices with ability to connect to Internet are deployed in these areas. A variety of applications such as intelligent sensors, healthcare systems and smart home/city devices are widely running constantly. Numerous devices could generate bursty traffic if edge servers are supporting to process services from IoT/mobile devices[12]. For instance, a large number of mobile users would cause the sudden influx of requests for their services in crowded sports events, concerts or other scenes. However, with the popularity of the Internet of things, protocols which are applied in IoT or mobile devices only focus on the information transmission[13]. These protocols such as HTTP, MQTT, AMQP, etc, have no ability to achieve an optimal solution to solve the bursty traffic problem caused by a huge amount of requests from clients. It's necessary to adopt a proper mechanism to tackle the evacuation of burst load.

For the distributed mobile edge system, a key lesson from studies of solving load balance is that the scheduling of the tasks in the network should follow the proposed algorithm of optimization under constraints of the resources of network links and edge servers[1]. The network equipments like routers and switches are fixed assets in edge environment, in other words, the network links are typically assumed to have a fixed rate. Similarly, the computational resource of each edge server is fixed too. As more and more users are added to the edge network, these limited resources can easily become a performance bottleneck as users attempt to race against each other. It is of great importance to design an efficient evacuation and scheduling mechanism when burst load occurs at the edge server under limited resources.

To cope with bursty data arrivals of multiuser application offloading in an edge environment, scholars designed an algorithm for the objective of minimized overall users' queue latency[14]. Authors in[15] propose an orchestrator for mobile augmented reality for edge computing, the major challenge is the difficulty of what extent the latency can be reduced. Data-intensive services in edge computing will face that load balancing conditions change constantly between edge servers [16], based on the limited resource of edge servers, the author proposed a scheme based on a genetic algorithm. Under resource constrained distributed edges environment, authors in [5] focus on the edge provisioning of computational resources to ensure stable latency of complex tasks for each mobile. Therefore, in order to get an optimal solution of the evacuation policy, network latency and computational latency are mostly considered in most of edge network systems.

In addition, edge networks do not exclude the cloud center networks, instead, recent studies have shown a promising trend that edge computing and traditional cloud computing come together to form a new architecture, and this heterogeneous edge computing system will consist of multiple layers[17]. Nevertheless, edge servers require the support of the remote server. For one thing, it solves the problem of lack of computing resources on the users' devices, and for another, with the help of edge servers for preprocess of computing tasks, it also reduces the latency due to reducing the computing cost at the remote server. For example, in the field of security cameras for road traffic control or indoor monitoring, before recording and sending the video, edge servers need to compress the data before transmission. This ensures maximum efficiency of migrated the data in a limited capacity of links to the remote server for future processing. Particularly, in face recognition, edge servers may need to access the portrait data from the database of the remote server, or some intelligence prediction computing algorithms need to be updated in real time for the model, framework and parameters[18]. Application of intelligent computing can make good use of the edge computing mode[19]. To mitigate the latency of mobile augmented reality(AR) applications, a hierarchical computation architecture consisting of several layers such as edge layer and cloud layer is widely adopted[15].

But the computation workload of each task is not easy to observe, those heavy computation workloads are hard to know before their completions. For instance, the latency caused by the computation of object recognizer and renderer components is difficult to evaluate in advance for mobile augmented reality(AR)[15]. Besides, tasks from the users in the edge environment can be an arbitrary sequence when arriving servers. Strategies without any assumption on the distribution of task arrivals are appropriate for the real situations. From the perspective of the deployment of edge servers, idle edge servers can often be used as supplements, and the edge servers can join other edge servers when a burst load of tasks occurs, thereby helping to evacuate and process these tasks. But computational resource provisioning problem in multi-layers is a particular challenge for burst load, because different factors can affect the performance of the provisioning policy such as workload of edge server, transmission latency between different layers, etc.

Through analysis, three factors that affect the overall latency of the burst load while implementing the evacuation are the link capacity, the network transmission speed, and the computational latency. Load balancing in a multi-layer edge network is of paramount importance in edge environments. Actually, the evacuation of the burst load is a particular case of load balancing, moreover, the main challenges of this technique can be considered include:

- limitation of the link bandwidth in the edge network
- limited edge network transmission speed
- efficiency in an online policy-based distributed edge network

In this paper, we address the aforementioned challenges and propose an efficient algorithm to minimize evacuation latency. To solve the burst load evacuation problem, therefore, we focus on the three layers architecture of the edge environment, namely the collaborated migration layer, the computing sharing layer and the remote assisted layer shown in Fig. 1. We denote the dispatching as the process that tasks migration from the edge server which burst load occurs to other edge servers, and scheduling as the sequence processing of tasks at each edge server and the remote server. Considering the limitations of link capacity and transmission speed, we have proposed an optimal algorithm to deal with the tasks in burst load among edge servers and the remote server. Thus, we aim to minimize the delay of all tasks after the dispatching and scheduling stages in this paper. Our contributions can be summarized as follows.

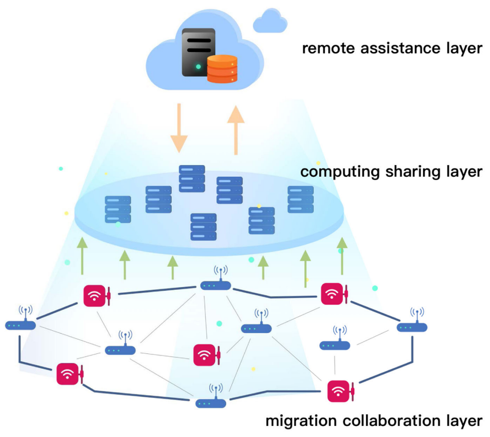

Fig. 1. The edge environment.

1) We propose an efficient strategy to address burst loads in edge environments. In this paper, we divide an edge computing network into three layers, namely the migration collaboration layer, the computing layer and the remote assisted layer. We present a fast evacuation strategy dealing with load evacuation. This provides a substantial extension for edge computing frameworks allowing them to deal with heavy traffic load in time.
2) We address the problem of how to evacuate the burst load to other servers for load balancing reasons. This will be done in minimal time once the burst load occurs in an edge server, offering a solution to migrate the burst load when the resources of the edge network are limited.
3) Within the computing sharing layer of edge servers, we offer a scheduling algorithm to make tasks parallel process in edge environments. The formulated problem is solved by re-entrant line scheduling without looking at all the task processing time in advance.
4) We have proposed our algorithm based online processing without knowledge of the task state in advance at the scheduling stage. It will enable in-situ processing of the latency-sensitive task. With the computing capability of edge servers in an edge network, tasks are processed in parallel under free computing resources in the network.

2 Related Work

Many task migration solutions are based on the limitations of links to achieve an optimal solution. In [20], the author considers a vehicle control system in a multi-layered edge network, and design a policy that switches edge server when burst load happens. Stability and dynamic control are introduced in the edge environment too, a centralized joint flow scheduling algorithm is proposed[21]. Besides, some scholars [22] also consider the disaster incident occurring at the edge environment and design strategies to handle the data deluge. The connection of links in the edge network is very complicated, making it extremely difficult to find an optimal path among the network links. These works above consider links capacity and transmission rate of tasks, but the edge environment cannot migrate tasks to other edge servers with the assistance of the remote server in the target of the optimization for the whole network to release the burst load.

Some scholars in [23] adopt method that asymptotically schedule the tasks based history knowledge of network. For achieving an optimal routing, a parallel invocation protocol toward multiple edge servers is designed in [24], which will both reduce the end-to-end latency and make latency more predictable. [25] focuses the optimization of network on bandwidth allocation to minimize the average file transmission delay, which can match radio resource with the traffic load distribution caused by content placement. Limited resources of IoT nodes and the shortage of coverage bring great challenges to the quality of service (QoS) of IoT, the author in [26] proposed an energy-balanced and obstacle-adaptive mobile edge node dispatch algorithm to solve this the traffic load problem based on breadth first search (BFS). In [27], the authors proposed a temporal task scheduling algorithm to efficiently dispatch tasks while meeting the delay bounds in Hybrid Clouds. But the proposed algorithms in the studies above has not consider the online scheduling when tasks are processed among edge servers and the remote server. Most of the task scheduling methods make assumptions on the knowledge of tasks before they are released to each edge server.

The burst load needs to be evacuated immediately. The author in [28], adopts game theory to formulate the distributed computation offloading algorithm, but each device involves redundant decision updates to achieve an optimal offloading solution, it might not be appropriate for tasks evacuation in a short time. Some works of edge computing have investigated computation peer offloading taking into account the transmission delay between links[29], a strategy of joint edge servers is proposed to solve the tasks offloading problem[30], similar to edge computing, some researcher has investigated cooperation between some nearby cloudlets to improve application performance[31] considering dispatching for tasks, but all of these studies implement the proposed algorithms by Lyapunov optimization which is based on a long term average technique and can not ensure to get an immediate decision to evacuate the burst load of finite tasks as soon as possible.

In this paper, to find an optimal routing for each task we adopt a routing search method to allocate each task in sequence, then, we propose an online algorithm for scheduling once the tasks are migrated to each edge server, we don't need the knowledge of tasks in advance to optimize the makespan of all tasks in the scheduling stage.

3 System Model

Our edge network system consists of multiple edge servers, as shown in Fig. 1. Denote the index set of tasks as $\mathcal{N} \triangleq \{1, 2, \ldots, N\}$ and the index set of edge servers by $\mathcal{M} \triangleq \{1, 2, \ldots, M\}$. All edge servers are linked in the form of a certain topology. Each edge server is operating to receive tasks from other edge servers, this kind of task migration starts from a heavy load edge server to another with less load. The workload of edge servers will drastically rise when a huge amount of requests initiated by the users flood into them, under this condition, a burst load occurs. Assume each edge server can offer computing resources for tasks

TABLE 1
Summary of Key Notations

Notation	Description
$\mathcal{N}$	The index set of tasks in a burst load
$\mathcal{M}$	The index set of edge servers
$\tilde{t}^s_{i,j}, \tilde{X}_{i,j}, \tilde{t}^c_{i,j}$	Start time, computation delay and completion time of the ith task for the pre-process at the jth server
$\hat{t}^s_{i,j}, \hat{X}_{i,j}, \hat{t}^c_{i,j}$	Start time, computation delay and completion time of the ith task for remote-assistance at the remote server corresponds to the jth edge server
$\bar{t}^s_{i,j}, \bar{X}_{i,j}, \bar{t}^c_{i,j}$	Start time, computation delay and completion time of the ith task the final-process at the jth server
$I^{\dagger}_i(t)$	The indicator for the ith task whether waitting in the burst load at the tth time slot
$I^{\ddagger}_{i,k}(t)$	The indicator for the ith task whether being migrated on the kth link at the tth time slot
$I^{\S}_{i,j}(t)$	The indicator for the ith task whether being executed on the j edge server at the tth time slot
T^i_1	The latency of the ith task waitting in the burst load
T^i_2	The latency of migration of each task i
T^i_3	The latency of task i at a specific edge server and the remote server for processing
$\tilde{e}_i$	The CPU cycles needed in pre-process
$\bar{e}_i$	The CPU cycles needed in final-process
f_j	The frequency of CPU at each server j
l_k	The latency of migration due to link k
c_k	The capacity of link k

during the burst load happens in the computing sharing layer [32], and connects to the remote server directly. Each task will consume this resource at different levels. Here, in the migration collaboration layer, we assume there are N tasks at one specific edge network which need to be evacuated as soon as possible, with a total of M edge servers deployed in this network, which could be allocated to execute tasks for this workload evacuation [1]. After allocating all of the tasks in the burst load, we can decide each task to wait in the burst load or migrate into another edge server immediately. Furthermore, the evacuation will be finished until the last task is completed. We aim to minimize the makespan of the tasks evacuation.

The different stages of tasks in the edge network are shown in Fig. 2. For example, some tasks might be suspended after a burst load happens at an edge server from t_0 to t_1 because of the limited capacity of links. Then these tasks will be migrated from the original edge server to another under the dispatching algorithm with latency from t_1 to t_2. At the following stage, these tasks will be prepared in an edge server to be scheduled by our scheduling algorithm, and finally, these tasks will be completed at t_3. Moreover, from the above analysis, it is shown that the migration of tasks only occurs at the migration collaboration layer. The computing layer and remote assisted layer are responsible for the computation of tasks in burst load cooperatively. For ease of reference, we list the key notations of our system model in Table 1.

3.1 Task Latency in Burst Load

When a burst load occurs at an edge server, it will accumulate a large number of tasks in a short period. These tasks from users' requests are waiting at the edge server to be dispatched for migration to another edge server. Our proposed evacuation policy starts at the moment when tasks in the burst load accumulated to some extent which can be preset according to the system manager. Thus, our designed mechanism only focuses on how to deal with all tasks involved at once after the burst load has formed. Tasks are cached in the task pool, the caching phase takes up a period until they can start the migration. We denote the indicator of a task latency in burst load as $I^{\dagger}_i(t)$,where $I^{\dagger}_i(t) \in \{0, 1\}$. $I^{\dagger}_1(t) = 1$ means the ith task is waiting in the task pool of the edge server at the tth time slot, otherwise $I^{\dagger}_i(t) = 0$. Once a burst load occurs in the edge network, all of the tasks are caching in the edge server until they can be allowed to migrate. The latency time of the ith task denoted as:

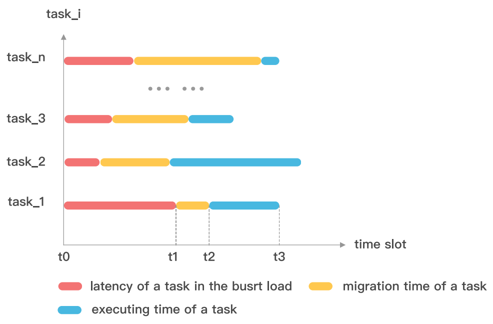

Fig. 2. The flowchart of tasks when evacuation.

$$T^i_1 = \sum_{t>0} I^{\dagger}_i(t), \tag{1}$$

where T represents the time when evacuation finishes.

3.2 Task Migration Latency Between Edge Servers

As compared with traditional networks, several edge servers that are close to each other might be linked and constructed as a local area network. We assume that each edge server is linked by several servers to form a migration collaboration layer. After the burst load occurs, tasks can be migrated through these links according to our optimal strategy. The system of an edge network can monitor the information of links status as well as available resources of each edge server. Under our evacuation scheme, a task can migrate along with its optimal routing from one edge server to another in order to reduce the workload on the original server shown in Fig. 3.

Denote the index set of links as $\mathcal{K} = \{1, \ldots, K\}$, and the index set of available routing in the edge network as $\mathcal{R} = \{1, \ldots, R\}$. Before a task is migrated to some server, the migration cost is known in this edge server network. An available routing in the edge environment consists of several links, each link latency is denoted as $l_k, k \in \mathcal{K}$[33].

Each link allows to pass a limited number of tasks in a time slot t during migration, denote the number as $c_k(t)$ and the maximum number as c^{max}_k, thus satisfying

$$c_k(t) = \sum_{i \in N} task^i_k(t) \le c^{max}_k, t > 0, k \in \mathcal{K}, \tag{2}$$

where $task^i_k(t) = 1$ means the ith task is migrated through the kth link, otherwise $task^i_k(t) = 0$.

The task migration should be working on the available links in our edge network between edge servers. In other words, each task after waiting in the burst load should start the migration. We denote the ith task's migration state as $I^{\ddagger}_{i,k}(t) = 1$ which means the ith task is being migrated on the kth link at the tth time slot, otherwise $I^{\ddagger}_{i,k}(t) = 0$. Furthermore,

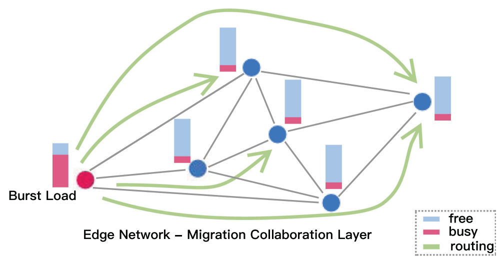

Fig. 3. The migration of the burst load.

denote a vector as

$$\mathbf{I}_i^{\ddagger}(t) = \left[I_{i,1}^{\ddagger}(t) \quad \ldots \quad I_{i,|\mathcal{K}|}^{\ddagger}(t) \right]^{\mathbf{T}}, \tag{3}$$

where

$$I_{i,k}^{\ddagger}(t) \in \{0,1\}, i \in \mathcal{N}, k \in \mathcal{K}, t > 0. \tag{4}$$

Since a task at a time slot is only migrating through one specific link or not. It can not be duplicated or split, and should satisfy the following constraint:

$$\sum_{k \in K} I_{i,k}^{\ddagger}(t) \in \{0,1\}, k \in \mathcal{K} \tag{5}$$

Because temporary congestion may occur at the nodes of some links, we can estimate the latency of migration of each task as:

$$T_2^i \geq l_1 + l_2 + \ldots + l_k, k \in \mathcal{K} \tag{6}$$

3.3 Task Processing Latency at the Edge Server and Remote Server

Once the task arrives at an edge server, after our dispatching strategy, the edge network system will arrange the task's processing to achieve a minimization of the makespan for all tasks. In general, the latency in this stage includes three parts as pre-process latency, remote-assistance latency, and final-execution latency as shown in Fig. 4.

Denote the indicator of the task execution by

$$I_{i,j}^{\S}(t) \in \{0,1\}, i \in \mathcal{N}, j \in \mathcal{M}, \tag{7}$$

which means the ith task is being executed on the j edge server for the tth time slot if $I_{i,j}^{\S}(t) = 1$, otherwise $I_{i,j}^{\S}(t) = 0$. We denote the start time of the ith task after arrived the jth edge server as $\tilde{t}_{i,j}^s$, the final completion time of the ith task at the jth server as $\bar{t}_{i,j}^c$. The computation delays of the ith task are denoted as $\tilde{X}_{i,j}, \hat{X}_{i,j}, \bar{X}_{i,j}$ at the steps of pre-process,

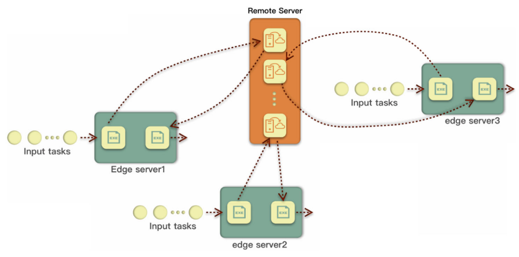

Fig. 4. Task scheduling for executing and requesting data.

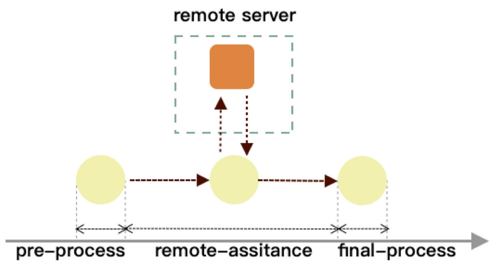

Fig. 5. Task processing sequence.

remote-assistance and final-process respectively. Denote the CPU cycles needed in pre-process and final-process part of the ith task as $\tilde{e}_i$ and $\bar{e}_i$ respectively[30]. In our scheduling stage, we have tasks that are processed in sequence for each task as shown in Fig. 5. Denote the frequency of CPU at each edge server is f_j where $j \in \mathcal{M}$. The latency of ith task generated by remote-assistance is denoted as $\hat{X}_{i,j}$, where j is the edge server which processed the task i. The latency for buffering a task at servers is defined as ξ_i. Therefore, we can calculate the execution time of a task through the formulation by processing the task at a single edge server:

$$T_3^i = \frac{\tilde{e}_i}{f_j} + \hat{X}_{i,j} + \frac{\bar{e}_i}{f_j} + \xi_i, \tag{8}$$

where $i \in \mathcal{N}, j \in \mathcal{M}$, Computing work of each phase of a task on a certain edge server should never stop until the work is finished. However, recall that a server can only process one job at most at a specific time point. The task execution should satisfy the constraints as follows:

$$\sum_{j \in \mathcal{M}} I_{i,j}^{\S}(t) \leq 1, i \in \mathcal{N}, t > 0, \tag{9}$$

$$\mathbf{I}_i^{\S}(t) = [I_{i,1}^{\S}(t), I_{i,2}^{\S}(t), \ldots, I_{i,M}^{\S}(t)] \tag{10}$$

$$\mathbf{rank}\left(\sum_{t=t'}^{t''} \mathbf{I}_i^{\S}(t) \right) = 1, t > 0, \tag{11}$$

$$t' \in \{\tilde{t}_{i,j}^s, \bar{t}_{i,j}^s, \hat{t}_{i,j}^s\}, t'' \in \{\tilde{t}_{i,j}^c, \bar{t}_{i,j}^c, \hat{t}_{i,j}^c\}, \tag{12}$$

the rank of the matrix should be 1 which means that the ith task is executed only on one specific edge server, neither allows being separated nor executed in several edge servers during the same step. Because the system in both dispatching and scheduling stages are dynamic, once the task is migrated to the edge server the edge server will start its processing if there exists free computation resource that is not occupied by any other task. Therefore, the task processing time should satisfy the constraints in phases of pre-process, remote-assistance, and final-process as:

$$\begin{aligned} \hat{t}_{i,j}^s &\geq \tilde{t}_{i,j}^c \\ \bar{t}_{i,j}^s &\geq \hat{t}_{i,j}^c, \end{aligned} \tag{13}$$

where $i \in \mathcal{N}$.

4 Problem Analysis

Tasks will evacuate from the burst load along optimal routes to get a minimal makespan. Our proposed strategy is based on the knowledge of connections about links in the edge environment. In the edge network, there may be a large number of tasks that require cloud-edge cooperation, such as the machine learning model of cloud-edge collaboration mentioned in 1, the processing time of the model itself may be uncertain, and the data transmission latency between the edge network and the cloud center server is also uncertain. Affected by the transmission latency of the edge-cloud, it is difficult to determine its transmission delay. Therefore, due to the uncertainty in task processing, it is difficult for us to know in advance how much time the task needs to spend while during the processing [34].

For simplicity, the uncertainty consists of transmission latency between the edge server and the remote server, the latency due to the processing at the edge server and the remote server. As shown in Fig. 5, the latency caused by the remote-assistance begins from the moment the task's pre-process completes and stops at the moment the task arrives back to the edge server to which the task belongs.

We define the latency of the ith task from the burst load starts to its completion as $a_i, i \in M$. There are three stages throughout a task's lifetime T_1^i, T_2^i, T_3^i corresponding to the task latency in the burst load, the task migration latency among edge servers, and the total processing latency of the task respectively. We need to arrange the strategy of each task in the burst caching phase, migration phase, and processing phase.

Since a task might be migrated from the original edge server to another edge server, we should decide which link is scheduled for each task to migrate at each time slot, and which edge server is scheduled to execute each task. Denote the total latency of a task as:

$$a_i = T_1^i + T_2^i + T_3^i, i \in \mathcal{N}. \tag{14}$$

Our target is to minimize the maximum task duration in all tasks. Thus, to satisfy the evacuation requirements, we are making the whole workload of users' tasks complete as soon as possible. To minimize the makespan of all tasks in a burst load, we define the formulation of the problem as follows:

$$\begin{aligned} \mathcal{P}_1 : \quad & \min_{\mathbf{I_i^\dagger(t)}, \mathbf{I_i^\ddagger(t)} \mathbf{I_i^\S}(t)} \quad \max(a_1, \ldots, a_N) \\ & s.t. \quad (2), (5), (6), (9), (11), (13) \\ & \qquad \tilde{t}_{i,j}^s \geq T_1^i(t) + T_2^i(t) \\ & t \; > \; 0, i \in \mathcal{N}, j \in \mathcal{M}, \end{aligned} \tag{15}$$

where $\max(a_1, \ldots, a_N)$ means the most time-consuming task in the edge computing network.

We can divide the task evacuation problem $\mathcal{P}_1$ into two parts. Furthermore, in this article, our two stages are regarded as two independent stages. At the dispatching stage, the task is continuously migrated from the server where the burst load occurs to the rest of the edge server through the link of the edge network. Then, at the scheduling part, these tasks through the edge server pre-process, the remote central server cooperates with the processing (remote-assistance), and finally completes the last step of processing(final-process) on the edge server, and the task evacuation process is finally completed. Therefore, we break down the problem into two subproblems as $\mathcal{P}_2$ and $\mathcal{P}_3$.

The *collaborated migration layer* is composed of topological connection links of edge servers in the edge network. They form a network with a limited number of nodes and links, and each server shares these links as a possible task evacuation path. Tasks from the burst load will accumulate at the specific edge server, we propose the OPTD4.1 algorithm to get an optimal decision to evacuate tasks from the server to others in the collaborated migration layer with constraints of capacity and migration rate. The *computing sharing layer* consists of each node in the collaborated migration layer at the dispatching stage. In general, each node in the collaborated migration layer represents one edge server in the computing sharing layer. Edge servers in the computing sharing layer receive tasks that are migrated from the collaborated migration layer. Edge servers share their computation resources to process these tasks from the burst load evacuation. The *remote assisted layer* is used to assist edge servers to process tasks so that to cooperate with the edge servers and expand more functions of the edge computing. We propose the 4.2 to schedule the tasks between edge servers and a remote server to get an optimized latency at the scheduling stage.

4.1 Dispatching With Routing Search

Suppose the connection of links in a specific edge environment is known. We denote $\mathcal{R} = \{1, 2, \ldots, R\}$ as a set of the available evacuation routings. We let each task of the burst load correspond to a evacuation routing to form a dispatching solution set as $x = \{r_1, r_2, \ldots, r_N\}$ where r_i represents the index of a routing which the ith task chooses, $i \in \mathcal{N}, r_i \in \mathcal{R}$. A feasible x represents the solution of our algorithm for optimal routing in the task dispatching stage as shown in 4.1.

Furthermore, based on the analysis of the original problem, we can come up with the basic problem of the optimal routing for tasks dispatching stage. Denote $\mathcal{P}_2$ as

$$\begin{aligned} \mathcal{P}_2 : & \min_{x \in \mathcal{X}} \max(r_1, \ldots, r_N) \\ & s.t. (5), (6), (9), (11), (2). \end{aligned} \tag{16}$$

To find the optimal routing, we investigate the total evacuation time of all tasks with the makespan criterion. Denote y as the makespan among all tasks during the dispatching stage in the evacuation. The optimal routing search for task dispatching is presented in the algorithm of OPTD4.1. We apply a method called derivative-free optimization[35], [36] based on computational learning theory[37] to search the routing for each task. An optimization method is proposed to search for a promising solution in our algorithm with a sampling model. Under a particular hypothesis, this algorithm is to discriminate good solutions from bad ones. In each iteration in $\mathcal{T}$, applying the updated hypothesis, a new optimal solution might be generated from this space.ted hypothesis, a new optimal solution might be generated from this space.

Denote those hypothesis are function mappings as a set $H = \{h : \mathcal{X} \to \mathcal{Q}\}$, where $\mathcal{Q}$ is the optimal solutions. We aim to find a suitable hypothesis h, $h \in H$ which can construct an optimal solution space to sample the a feasible solution of routings allocation for each task. Denote $y = c(x)$ as the makespan of the tasks after the dispatching stage. Assume an initial solution set as $S_0 = \{(x_0, y_0), \dots, (x_{\mathcal{U}}, y_{\mathcal{U}})\}$ with a size of $\mathcal{U}$, and $\mathcal{R}$ as a set of the available routings which is sorted in ascending order of latency while a task migrates through. The value of x^i means the routing which the ith task chooses, where $x^i \in \mathcal{R}$.

The sliding window moves to the right every W tasks in the solution space of the task, where $W \leq N$. We randomly generate a number λ for each iteration t. If λ is less than α which is the parameter, we first traverse the tasks in the window, and we proceed in a heuristic strategy to allocate a smaller latency routing for the task, otherwise, a random routing is allocated to the task. Once the evacuation routing is selected for the task, we fix the evacuation routing under the task index on I, where I is denoted as the set of indexes of tasks in the burst load. Allocate the task index i with a feasible evacuation routing, then all allocated tasks are denoted as the set I_f, and I_v represents the remaining task index.

After all the tasks in the window are traversed, we compare the newly constructed solution $\tilde{x}$ and $\tilde{y}$ with the minimum value of the solution in the previous iteration. If $\tilde{x}$ is better($\tilde{y}$ is smaller) then the accepted search space represented by the hypothesis h_t(the updated I_f and I_v) generated by this iteration. Space is used as the sampling space for the next round of solutions. Otherwise, according to the 16th line in the algorithm, a global search is introduced to reconstruct h_t. Finally, we sample by hypothesis space to get y_h, then compare with y_t, $\tilde{y}$ to get y_{min}, and then start the next round of sampling.

Algorithm 1. Optimal Routing for Task Dispatching (OPTD)

```
1:  Generate the window size $W$, the initial solution size $\mathcal{U}$,the
    parameter of routing selection $\gamma$
2:  Initial $S_0 = \{(x_0, y_0), \dots, (x_{\mathcal{U}}, y_{\mathcal{U}})\}, V_t = \emptyset$
3:  for $t = 1$ to $\mathcal{T}$ do
4:    $(x_t, y_t) = (x_{min}, y_{min}), I_v = I, I_f = \emptyset$
5:    for $i = (t-1) * W$ to $t * W$ do
6:      if $\lambda < \alpha$ then
7:        $\tilde{x}_t^i = \lceil x_t^i / \gamma \rceil$, $I_v = I_v \setminus \{i\}, I_f = I_f \cup \{i\}$
8:      else
9:        $\tilde{x}_t^i = rand(1, |\mathcal{R}|)$
10:     end if
11:   end for
12:   $\tilde{y}_t = c(\tilde{x}_t)$
13:   if $\tilde{y}_t < y_t$ then
14:     construct $h_t$ from the updated $I_v$ and $I_f$
15:   else
16:     randomly choose $i$ with the size of $|I_f|$ to replace the
        elements in $I_f$, $I_v = I \setminus I_f$, construct $h_t$
17:   end if
18:   for $t = 1$ to $\mathcal{U}$ do
19:     Sample $x_h$ from $h_t$, $y_h = c(x_h), V_t = V_t \cup (x_h, y_h)$
20:   end for
21:   $y_{min} = \min\{y_h, y_t, \tilde{y}_t\}, (x_h, y_h) \in V_t$
22: end for
23: return $(x_{min}, y_{min})$
```

Every iteration after search the possible region constructed by the h_t, then sampling an optimal solution is a high probability event. We define α as a threshold to choose a low latency routing for a task i. The makespan of each routing to select the lesser routing heuristically, after modified α and $\mathcal{T}$ to get a better optimal solution. In the beginning of the algorithm, we initial feasible solution that each task randomly choose an edge server to evacuate and an fast routing with probability as $P = \frac{\sum_{l_k \in r_i} l_k}{\sum_{r \in \mathcal{R}_m} \sum_{l_k \in r} l_k}$, where $r_i \in \mathcal{R}$, $\mathcal{R}_m$ is the set of routings for evacuations to the mth edge server. Denote wsz is a parameter to control the window size in each iteration, we set window size $W = \lceil \frac{N}{wsz} \rceil$ in each iteration t. We set iteration number as $\mathcal{T} = \frac{N}{iter}$, where $iter$ is a parameter to control the iterations. Assuming the solution of tasks dispatching with size of N, after a simple analysis of 4.1, the overall complexity of the algorithm is therefore $\mathcal{O}(N\mathcal{T})$

Lemma 1. *Given $\epsilon > 0$ and $0 < \delta < 1$, suppose after sampling from h_t we get $\hat{x}, \hat{y}$ s.t. $\mathbf{P_h}((\hat{y} - y^*) > \epsilon) < \delta$, where y^* is optimized solution. Inspired by[35], the complexity of algorithm's upper bound4.1 holds as:*

$$\mathcal{O}\left(\frac{1}{\tilde{P}_h} * ln \frac{1}{\delta}\right). \tag{17}$$

Proof.

$$\begin{aligned} &\mathbf{P_h}((\hat{y} - y^*) > \epsilon) = (1 - \mathbf{P_x})^{\mathcal{U}} \prod_{t=1}^{\mathcal{T}} (1 - \mathbf{P_{h_t}})^{\mathcal{U}} \\ &\leq \exp(-(1 - \mathbf{P_x})^{\mathcal{U}}) \prod_{t=1}^{\mathcal{T}} \exp(-(1 - \mathbf{P_{h_t}})^{\mathcal{U}}) \\ &= \exp\big(-(1 - \mathbf{P_x})\mathcal{U} - (1 - \tilde{\mathbf{P_h}})\mathcal{U}\mathcal{T} \big) \\ &\leq \exp\big(-\tilde{\mathbf{P_h}}\mathcal{U}\mathcal{T} \big) \leq \delta, \end{aligned} \tag{18}$$

where P_x represents the the 19th line in OPTD algorithm that samples from the routes space $\mathcal{R}$. Denote the average probability of h_t as $\tilde{\mathbf{P_h}} = \frac{1}{\mathcal{T}} \sum_{t=1}^{\mathcal{T}} \mathbf{P_{h_t}}$. And with the help of the formulation $1 - x \leq \exp(-x)$, thus, we get $\mathcal{U}\mathcal{T} \in \mathcal{O}(\frac{1}{\tilde{P}_h} ln \frac{1}{\delta})$, thus we proves the lemma. □

Lemma 2. *Given $\epsilon > 0$ and $0 < \delta < 1$, denote $|D|$ as the total sample size of algorithm4.1, we use the form from [38], we set a tolerance τ, if $(\hat{y} - y^*) < \tau$, if we accept this value $\hat{y}$, denote $\mathbf{I}(\hat{y}) = 1$ otherwise, reject and $\mathbf{I}(\hat{y}) = 0$. For any hypothesis h_t: if the number of sample x returns y_t, where $x \in \mathcal{D}s.t.$ empirically $\hat{\mathbf{I}}(\hat{y}) = 1$, hence the number of feasible solutions is n_{opt}, then there are $|\mathcal{D}| - n_{opt}$ samples s.t. $\hat{\mathbf{I}}(\hat{y}) = 0$. Therefore, the probability $\mathbf{P_r}\big((1 - \frac{n_{opt}}{|\mathcal{D}| - n_{opt}}) < \epsilon\big) \geq 1 - \delta$ holds if*

$$|\mathcal{D}| > \frac{1}{\epsilon}\left(ln|H| + ln\left(\frac{1}{\delta}\right)\right), \tag{19}$$

where $H = \{h_1, \dots, h_{\mathcal{T}}\}, h_t \in H$. It concludes that the lower bound of the samples is $\frac{1}{\epsilon}(ln|H| + ln(\frac{1}{\delta}))$.

Proof. Under a hypothesis h_t, after $\mathcal{T}$ iterations we will get the probability, we will get n_{opt} optimal solutions, the rejected solutions' size is $|\mathcal{D}| - n_{opt}$. Suppose there is no rejected solution under the hypothesis h_t, then the

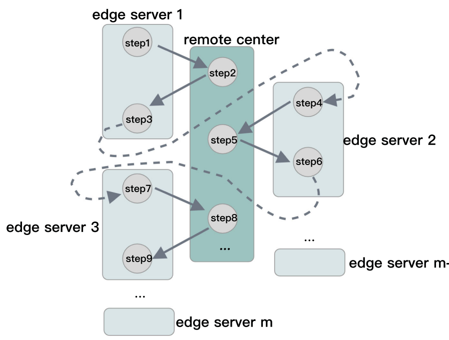

Fig. 6. Task scheduling for executing and requesting data.

probability is $\mathbf{P_r}_{h_t}(1-\frac{n_{opt}}{|\mathcal{D}|-n_{opt}}=0)\leq(1-\epsilon)^{|\mathcal{D}|}$. For any hypothesis h_t where $h_t\in H\ s.t.$ under h_t $\mathbf{P_r}_{h_t}(1-\frac{n_{opt}}{|\mathcal{D}|-n_{opt}}=0)$, define the event that the number of n^t_{opt} at tth iteration is 0 as $event(h_t)$. With the help of formulation $1-x\leq \exp(-x)$ and uniion bound rule, we can get

$$\begin{aligned}&\mathbf{P_r}\big(evnet(h_1)\cup evnet(h_2)\cup,\ldots,\cup event(h_t)\big)\\&\leq\sum_{h_t\in H}\mathcal{P}_{h_t}(n^t_{opt}=0)\leq\sum_{h_t\in H}(1-\epsilon)^{|\mathcal{D}|}\\&\leq|H|(1-\epsilon)^{|\mathcal{D}|}\leq|H|exp(-|\mathcal{D}|\epsilon)\leq\delta,\end{aligned}\tag{20}$$

thus

$$ln|H|exp(-|\mathcal{D}|\epsilon)\leq ln\delta,\tag{21}$$

thus we prove the lemma. □

4.2 Scheduling With Online Policy

The online scheduling algorithm in this paper considers how to arrange the order of distributed execution of tasks on edge servers. We adopt distributed edge servers in the edge network to realize a parallel strategy in scheduling. After pre-process a task by the edge server, this task is migrated to the remote server for remote-assistance. Meanwhile, other edge servers may migrate their task to the remote server too. Our target is to optimize the strategy to achieve a minimization makespan of all tasks from the burst evacuation load. Thus, we adopt a method to make the scheduling of tasks as a re-entrant scheduling problem [39]. as shown in Fig. 6. All edge servers can be seen as a whole pipeline structure. This will greatly save time and increase the overall efficiency of the system, so that the time to process all tasks is relatively short. Furthermore, we don't need the knowledge of the tasks, which means we don't need the latency that the tasks will generate under the computation resources.

Under the re-entrant line scheduling problem. Every edge server needs to deliver tasks with the remote server which needs to process tasks from all the edge servers as shown in Fig. 6. In general, the total consuming time of the remote server will be much greater than any edge server in the edge environments. Thus, we simply assume the remote server as the bottleneck phase of the edge system denote as btn[39].

As shown in Fig. 6, to construct a re-entrant line scheduling, we denote $step_1,step_2,\ldots,step_9$ correspond to each step in edge servers and remote server. For example, $step_1, step3$ represent the pre-process and final-process in the 1st edge server and the steps of $step_2, step_5, step_8$ represent btn_1, btn_2, btn_3 of the remote server. We regard all the steps as a cycle start from a task starting at $step_1$ to a task completion at $step_9$, if there are 3 edge servers and one remote server. As shown in Fig. 6, dotted lines connect each edge server with the remote server, the sequence constructs the virtual processing of tasks representing a cycle of re-entrant line scheduling. When some task completes at the final processing step of each edge server, one cycle in Algorithm 4.2 finishes. For example if there are only 3 edge servers with a btn server, the 4th task in $step_1$, the 3th task in $step_2$ and the 2nd task in $step_3$ is one cycle in our algorithm shown in 7. Thus, we denote the set of tasks' index dispatched to the jth edge server as $\mathcal{N}_j$, where

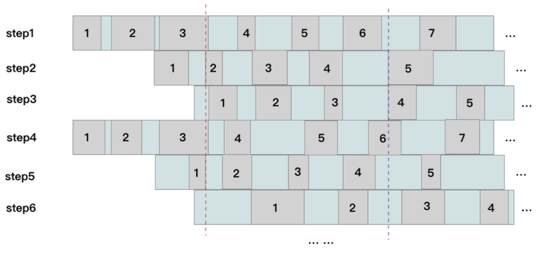

Fig. 7. Task scheduling for in pipeline.

$$\mathcal{N}=\mathcal{N}_1\cup\mathcal{N}_2,\ldots,\cup\mathcal{N}_M.\tag{22}$$

The pipeline structure has good parallel efficiency, and the distributed edge system can maximize the utilization of the limited computation resources. T_{ini} is denoted later. In a parallel structure the start times of the task execution should satisfy the constraints as follows:

$$\tilde{t}^s_{1,j}=T_{ini},\tag{23}$$

$$\tilde{t}^s_{i,j}\geq\bar{t}^c_{i-1,j}\tag{24}$$

$$\hat{t}^s_{i,j}\geq\tilde{t}^c_{i,j},\tag{25}$$

$$\bar{t}^s_{i,j}\geq\hat{t}^c_{i,j}, i\in\mathcal{N}_j\setminus\{1\}, j\in\mathcal{M}.\tag{26}$$

The task execution in each edge server in our parallel strategy is followed by the btn server(remote server). Each cycle starts at the start time of the first task executed by the btn server, edge servers are not allowed to start ahead of the cycle. As shown in Fig. 7, $step_2$ and $step_5$ represent the steps in btn server, the btn server always executes tasks without any delay. Suppose the $step_3$ and $step_1$ represent the ith server, if the 3th task at the $step_3$ completes, the ith server can't start the task 6 at step immediately, because the btn server has not started a new cycle in which the 4th task has not started yet. Therefore the btn server pace all edge servers in the computing sharing layer. Hereafter, i is denoted the unique sequence number of tasks in all servers, which means the ith task stands for the task arrived each server that is indexed by i by this server. The start time of the task in each edge server should satisfy the constraint as follows:

$$\tilde{t}^s_{i,1}=\max\{\bar{t}^c_{i-1,M},\hat{t}^s_{i,j}\}.\tag{27}$$

We must make sure to execute a task at the edge server at a certain step after its migration to this edge server has been done. In the same view of the task migration stage on any link, the task is only allowed to start its migration after it has migrated at the previous hop of routing it belonged. We call this special buffer safety stock. Safety stock is a term used by logistics[40], it is used in case of mitigating risk of stockouts of the tasks. Safety stocks are used in our system to make each step of the edge server could be feasible and efficient at the scheduling stage, denote index of tasks as $\tilde{F}_j = \{1, 2, \ldots, |\mathcal{N}_j|\}$ for pre-process, $\hat{F}_j, \bar{F}_j$ for remote-assistance and final-process, as the safety stock at step d of the jth edge server as the part in the left of red dotted line shown in Fig. 7. Each step in edge servers now has an offset before the edge server starts to process the task. To get a feasible schedule, those safety stocks in each step should be set up before the step starts work. It is similar to the have some delay in each step when the pipeline structure begins as shown in 6. We denote the initialization delay for this kind of delay at the beginning in the pipeline organization as T_{ini}.

Proposition 1. *The initialization latency in the scheduling stage is lower bounded by*

$$\max_{j\in\mathcal{M}} \left\{ \sum_{i\in\tilde{F}_j} \tilde{t}^c_{i,j}, \sum_{i\in\bar{F}_j} \bar{t}^c_{i,j} \right\}. \tag{28}$$

Proof. Each task in $\tilde{F}_j$ or $\bar{F}_j$ should be processed in advance at the j edge server. However, after the task arrive at each edge server or remote server, it may not be processed immediately because the execution of a previous task does not complete or the cycle of the btn server does not start. Furthermore, $t^c_{i,j} = t^s_{i,j} + X_{i,j}$ and $t^s_{i,j} \geq t^c_{i-1,j}$, the arrive time of task i at each step of the$j$ edge server may not be exactly the start time of $t^s_{i,j}$ due to dispatching stage, where $t^s_{i,j} \in \{\tilde{t}^s_{i,j}, \hat{t}^s_{i,j}, \bar{t}^s_{i,j}\}, t^c_{ij} \in \{\tilde{t}^c_{i,j}, \hat{t}^c_{i,j}, \bar{t}^c_{i,j}\}$, $X_{i,j} \in \{\tilde{X}_{i,j}, \hat{X}_{i,j}, \bar{X}_{i,j}\}$. We can conclude that the maximum value $\max_{j\in\mathcal{M}}\{\sum_{i\in\tilde{F}_j} \tilde{t}^c_{i,j}, \sum_{i\in\bar{F}_j} \bar{t}^c_{i,j}\}$ of each edge server shold be the start time of the algorithm4.2 in our the system as T_{ini}.

T_{ini} is the cost time of processing in the algorithm4.2. After initialization of the scheduling, the task starts time at each step should be given by T_{ini} at the scheduling stage. Then, the start time of all the first steps to work at each edge server is the same. As stated above, all edge servers parallel start their work according to the algorithm of Initialization Scheduling4.2. Denote $\tilde{b}_j$ and $\bar{b}_j$ as sets to cache the task from previous step for pre-process and final-process, where $j \in \mathcal{M}$. And $\hat{b}_j$ is denoted the task pool at a remote server that cache the tasks from the jth edge server, particularly, $\hat{b}_j$ represents the task pool at the jth server in order to receive the arrived tasks from the collaborated migration layer. Processing each task in the edge server should follow the First-Come-First-Served(FCFS) rule in the task pool.

Suppose tasks left in the step of final-process at the edge server after the initialization scheduling is $\mathcal{N}_j$. Thus, based on the analysis of the original problem $\mathcal{P}_1$ and $\mathcal{P}_2$, after tasks arrive at the edge server by arranging each task to a routing, we can get the problem of optimal parallel scheduling for tasks as $\mathcal{P}_3$ which is the same problem of $\mathcal{P}_1$ as

$$\mathcal{P}_3 : \min_{j\in\mathcal{M}} \max\{\bar{t}^c_{|\mathcal{N}_j|,j}\} \\ s.t. (23), (24), (25), (26), (27), \tag{29}$$

where $\max\{\bar{t}^c_{|\mathcal{N}_j|,j}\}$ means the maximum value in the set of task completion time at each step of all edge servers. In general, after we solve the problem of $\mathcal{P}_2$, we can get the routing arrangement for each task. Then based on the solution of $\mathcal{P}_2$, it's easy to solve the problem of $\mathcal{P}_3$ to achieve the makespan of tasks' evacuation.

Algorithm 2. Initialization Scheduling

```
1: while the $i$th task arrive at $j$th edge server
   or $|\tilde{b}_j| \neq 0$ or$|\bar{b}_j| \neq 0$ do
2:   start the arrived task followed FCFS rule at the first step
     of each edge server
3:   push the task to $\hat{b}_j$
4:   if the $(|\hat{F}_j| - 1)$th task has completed at step of
     pro-processing and the $btn$ remote server is not busy then
5:     start the arrived task followed FCFS rule
6:     push the task to $\hat{b}_j$
7:   end if
8:   if the $(|\bar{F}_j| - 1)$th has task completed at step of
     final-process and the $j$th edge server is not busy then
9:     break the while loop, calculate the stop time $\bar{t}^c_{|\bar{F}_j|,j}$
10:  end if
11: end while
12: when all the edge servers stop the initialization,
    $T^j_{ini} = \max(\bar{t}^c_{|\bar{F}_j|,j}, \hat{t}^c_{|\hat{F}_j|,j})$.
13: $T_{ini} = \max(T^1_{ini}, T^2_{ini}, \ldots, T^{|\mathcal{M}|}_{ini})$
```

To solve the problem of $\mathcal{P}_3$, we have designed a mechanism that event-driven scheduling is provided, as shown in the right part of the red dotted line. In the algorithm of OPST, we design a heuristically online strategy that makes btn server as busy as possible, we don't use any processing value in advance to make any favorable permutation of tasks. Edge servers may process some tasks at time t when some previous task completes or the remote server starts a new cycle. We denote the indicator set $s_j(t) = \{0, 1\}, t > 0, j \in \mathcal{M} \cup \{btn\}$ as these edge servers and the remote server. If $s_j(t) = 1$ means the jth edge server is busy processing some task, otherwise $s_j(t) = 0$. Denote the cycles started by the remote server as $d_{j'}$, the edge server as $d_j, j \in \mathcal{M}$. **startCycleEn** is set of the indicator where $startCycleEn_j = 1$ represent the j edge server is allowed to process the task, otherwise $startCycleEn_j = 0$, $j \in \mathcal{M}$. Actually, the scheduling algorithm is distributed parallel implemented by each edge server and the remote server. The algorithm is a kind of event-driven mechanism, thus we can deduce that the overall complexity of the algorithm is, therefore, $\mathcal{O}(1)$.

Denote the index of set of running edge servers at time slot t as RS,where $t > 0$. $\tilde{p}_j$, $\hat{p}_j$, $\tilde{p}_j$ represent the step at edge server j for pre-process, remote-assistance and final-process respectively, where $j \in \mathcal{M}$, j' stands for the remote server.

Lemma 3. *The lower bound of $\mathcal{P}_3$ is the maximum processing time of edge servers. Denote the lower bound as*

$$LB_1 = \max_{j\in\mathcal{M}} \left\{ \tilde{t}^s_{1,j} + \sum_{i\in\mathcal{N}_j} (\tilde{X}_{i,j} + \bar{X}_{i,j} + \hat{X}_{i,j}) \right\}. \tag{30}$$

Proof: The makespan of the task evacuation includes the execution time of the task at the edge server and the latency generated by the remote assistance. If some task is in the buffers of steps, the latency of those tasks in buffer certainly is delayed. Therefore the lower bound must be greater than or equal to the overall latency of tasks arrived at any edge servers and the remote server. □

Lemma 4. *Considering the average time of the task's completion at each edge server[41], we can get the lower bound of $\mathcal{P}_3$ as*

$$LB_2 = \Theta_M + \Gamma_M. \tag{31}$$

Proof. From the perspective of each edge server, the makespan of all tasks can be denoted as the time when the last task in the edge server is finished processing.

It's obvious to conclude that the average completion time of each edge server must be less or equal to the makespan of all tasks. Thus $\max(A) \geq \frac{\sum_{j\in\mathcal{M}} \tilde{t}^s_{1,j}}{|\mathcal{M}|} + \sum_{j\in\mathcal{M}} \sum_{i\in\mathcal{N}_j} \frac{\tilde{X}_{i,j}+\bar{X}_{i,j}+\hat{X}_{i,j}}{|\mathcal{M}|}$. Denote $\Theta_M = \frac{\sum_{j\in\mathcal{M}} \tilde{t}^s_{1,j}}{|\mathcal{M}|}$ where $j \in \mathcal{M}$. Denote $\Gamma_M = \frac{\sum_i \sum_j \left(\tilde{X}_{i,j}+\bar{X}_{i,j}+\hat{X}_{i,j}\right)}{|\mathcal{M}|}$, Therefore, $\Theta_M + \Gamma_M$ is one of the lower bound in our algorithm. □

Lemma 5. *In lemma.4, we assume each edge server has tasks allocated, inspired by[41], it's interesting to discuss if our algorithm has not allocated tasks to some edge server for optimal scheduling. We can get the lower bound as*

$$LB_3 = \max(\mu, \tau), \tag{32}$$

where $\tau = \tilde{t}^s_{1,j'} + \sum_{i\in\mathcal{N}_{j'}} \tilde{X}_{i,j'} + \bar{X}_{i,j'} + \hat{X}_{i,j}$, and suppose that the edge server j' will perform at least $|\mathcal{N}_{j'}| = \frac{1}{|\mathcal{M}|} \cdot |\mathcal{N}|$ tasks, τ is the latency generated by tasks at the j'th edge server.

$$\mu = \begin{cases} \Theta_{\mathcal{M}\setminus j} + \Gamma_{\mathcal{M}\setminus j} & if \quad |\mathcal{N}| - |\mathcal{N}_j| \geq |\mathcal{M}| - 1 \\ \bar{\Theta}_{\mathcal{N}\setminus\mathcal{N}_j} + \Gamma_{\mathcal{M}\setminus j} & if \quad |\mathcal{N}| - |\mathcal{N}_j| < |\mathcal{M}| - 1 \end{cases}. \tag{33}$$

Proof. Under the result of our optimal scheduling algorithm, if the jth edge server carries out $\mathcal{N}_j$ tasks that $|\mathcal{N}| - |\mathcal{N}_j| < |\mathcal{M}|$, the number of tasks executed on a server may be 0. As mentioned in lemma.4, $\Gamma_{\mathcal{M}\setminus j} = \frac{\sum_{i\in\mathcal{N}\setminus\mathcal{N}_j} \sum_{j\in\mathcal{M}\setminus j} \left(\tilde{X}_{i,j}+\bar{X}_{i,j}+\hat{X}_{i,j}\right)}{|\mathcal{M}|-1}$ Denote $\bar{\Theta}_{\mathcal{N}\setminus\mathcal{N}_j} = \frac{\sum_{j\in\mathcal{M}\setminus j} \tilde{t}^s_{1,j}}{|\mathcal{N}|-|\mathcal{N}_j|}$ we can easily know the lower bound is μ, Therefore, the lower bound value must be one of the maximum in the set of completion times of edge servers, we denote the set as $\{\mu, \tau\}$. Thus it concludes the lemma. □

5 Experiments

In this section, we implement a small prototype system to prove the efficiency of our proposed algorithm and framework to perform an empirical evaluation. In contrast to other algorithms, the optimal path search algorithm we proposed can always meet the shortest global migration and execution time. The parallel pipeline algorithm that we propose to execute tasks can always execute all tasks in the most efficient way between the remote server and the edge server so that the overall evacuation latency of the task is minimized.

5.1 Experiment Settings

In this section, we will verify the effectiveness of our proposed algorithms. We will first introduce a baseline method for comparison. Then show the impact of our parameters by varying certain variables in the experiments. The experiment was run on a machine with Intel Xeon E5-2620 v4@2.10GHz2 CPU and 64G of memory and python3.8 is applied. We conduct our simulation as follows. We assume that the burst load occurs at one of all edge server, whereby the $freq$ of each edge server is uniformly distributed with a mean of 2. We randomly sampled 5000 tasks from google cluster data to conduct the simulation [42]. Cycles needed in the pre-process, remote-assistance, and final-process of each task is generated by tasks in the google cluster. The cycles of each task is a result of a linear combination of resource request for CPU cores, resource request for RAM, and resource request for local disk space from the data sets. We set the maximum number of task allowed to migrate in each link as $c^{max}_k = 8$, thus, suppose the link capacity is uniformly distributed as $U(1, 8)$ with the mean of 4, the task migration latency in each link l_k is uniformly distributed as $U(40, 150)$ with the mean of 95. We set $alpha = 0.9$, $iter = 4$, $wsz = 3$, $\gamma = 3$, $|V_t| = 20$ in dispatching stage.

Algorithm 3. Optimal Parallel Scheduling for Task (OPST)

```
1: For every distributed edge server, the event driven
   algorithm is implemented parallel
2: Initial $d_{j'} = 0, d_j = 0,$ **startCycleEn** $= \vec{0}$ Monitor valuse of
   $\hat{b}_j(t)$, $\bar{b}_j(t)$, $j \in \mathcal{M}$, $t \geq 0$
3: while ($t = \hat{t}^s_{i,1}$ or $t = \bar{t}^c_{i,j}$ or $t = \tilde{t}^c_{i,j}$) do
4:   if $t = \hat{t}^s_{i,1}$ then
5:     $d_{j'} = j_{j'} + 1$
6:     Start the task from $\tilde{b}_j(t)$ in FIFO order at the first step of
       the $j$ server, $j$ in $\mathcal{M} \setminus RS$
7:     if $s_j(t) = 1$ then
8:       $startCycleEn[j] = 1$
9:     end if
10:  end if
11:  if $t = \hat{t}^c_{i,j}$ then
12:    Start the task at the $\hat{p}_{j+1}$ step of the remote server in FIFO
       order from $\hat{b}_{j+1}(t)$
13:  end if
14:  if $t = \tilde{t}^c_{i,j}$ then
15:    Start the task at step $fi$ of the $j$th server in FIFO order
       from $\bar{b}_j(t)$
16:  end if
17:  if $t = \bar{t}^c_{i,j}$ and $startCycleEn[j] = 1$ and $d_{j'} > d_j$ then
18:    Start the task from $\tilde{b}_j(t)$ in FIFO order at first setp of $j$th
       server, increase $d_j$ as $d_j + 1$
19:    if $d_{j'} = d_j$ then
20:      $startCycleEn[j] = 0$
21:    end if
22:  end if
23: end while
```

5.2 Baseline Policies

In addition to the optimal algorithm we proposed, we also performed three comparison algorithms for comparison, called "speed priority" and "capacity priority" at the dispatching stage, "server greedy" at the scheduling stage.

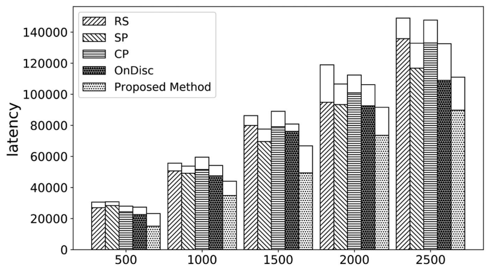

Fig. 8. Latency under different number of tasks.

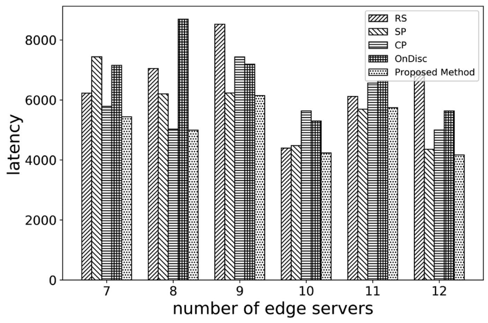

Fig. 9. Latency of tasks under different number of edge servers.

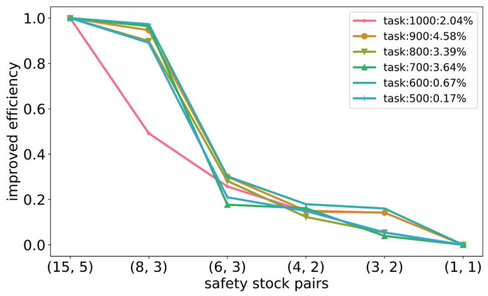

Fig. 10. Performence under different size of safety stock pairs.

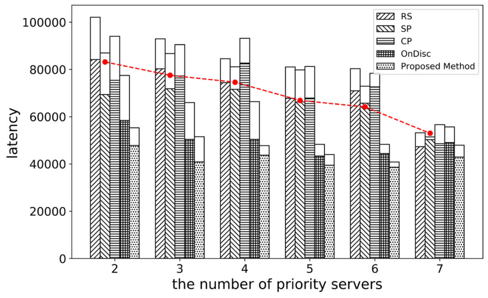

Fig. 11. Latency of tasks under SG algorithm.

– *Random Selection(RS)*: In the migration collaboration layer, the tasks in the burst load that need to be migrated uniformly select the link to an edge server. For example, assign routing r to $task_i$, where $i \in \mathcal{N}, r \in \mathcal{R}$, r is randomly selected from $\mathcal{R}$.
– *Speed Priority(SP)* : In the migration collaboration layer, the tasks in the burst load that need to be transferred always select the link with less migration time as the transfer path with a certain probability. For example,assign r to $task_i$, where $i \in \mathcal{N}, r \in \mathcal{R}$, r, is selected from $\mathcal{R}$ with probability of $Pr = \frac{v_k}{\sum_{z\in|\mathcal{R}|} v_z}$, where $v_z = \frac{1}{\sum_{k\in\{1,2,\cdots,|r_z|\}} l_k}$
– *Capacity Priority(CP)* : In the migration collaboration layer, the task in the burst load that needs to be transferred always selects the link with the larger capacity as the transfer path with a certain probability. For example,assign r_z to $task_i$, where $i \in \mathcal{N}, r_z \in \mathcal{R}$, r_z, is selected from $\mathcal{R}$ with probability of $Pr = \frac{c_z}{\sum_{z\in|\mathcal{R}|} c_z}$, where $c_z = \sum_{k\in\{1,2,\cdots,|r_z|\}} c_k$.
– *Server Greedy(SG)*: In the computing sharing layer, a number of edge servers always choose to occupy the resource in the remote assisted layer until the task is completed.
– *OnDisc*: It is an online algorithm dealing with the dispatching and scheduling of tasks among edge servers and the remote server efficiently [43]. It can be easily implemented in distributed systems especially edge environments.

5.3 Simulation Results

The analysis above has led the conclusion of algorithm efficiency that outperform the baselines

Optimal Search Routing in Dispatching. We demonstrate the performance with benchmark algorithms of RS, SP, CP, and our proposed algorithm respectively, when the number of tasks increases from 500 to 2500 in Fig. 8. Different tasks represent the levels of the burst load under the same condition of the network and servers. The total length of each bar indicates the makespan after the tasks evacuation during both dispatch and scheduling simultaneously, and the shaded part indicates the dispatching latency.

Obviously, as the number of tasks increases under the same condition of the network and servers, the total evacuation time required also increases, especially the time of all tasks in finding the routing to evacuate increases significantly. Due to the limited link bandwidth and the network transmission speed, the migration time of tasks plays an important role in total latency. It can be easily understood that the time of task migration in the routing may often occupy a large part, because each network link migrates a limited number of tasks per unit time, and the migration speed of tasks in the routing is limited. However, in contrast, under the proposed algorithm, we show that the latency of task migration is significantly less than the comparison algorithm because these tasks evacuated earlier are already executed by the edge server and the remote server. After the execution is completed, the computing resources are released could be used by the tasks which arrive later, and the subsequent tasks can be executed at a faster speed under our pipeline strategy without delay too much time. Besides, it can be observed from Fig. 12 that our algorithm has good convergence performance in different levels of burst loads. The total number of tasks divided by $iter$. When the number of tasks is increased from 500 to 2500, the proposed algorithm can always achieve good results after about 100 iterations. The experimental result demonstrates the effectiveness of our proposed method when the number of servers increases shown in Fig. 9 in the case of 100 tasks.

Distributed Parallel Strategy for Scheduling. In the scheduling stage of tasks, we first constructed the initialization of pipeline 7. The purpose of initialization is to build a buffer

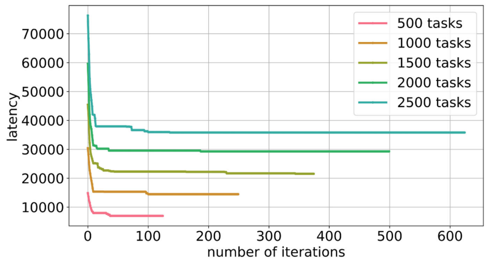

Fig. 12. Iterative process of routing search.

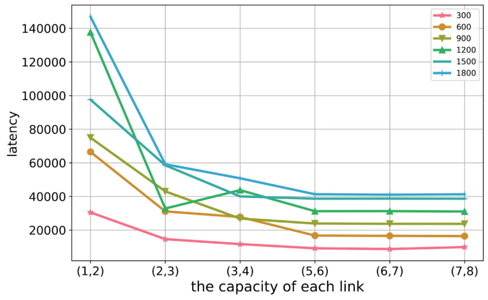

Fig. 13. Latency under different capacity of links.

for each step in the pipeline network to boost efficiency. If some edge servers in the pipeline structure adopt a greedy strategy and continue which means it will occupy the assistance resources of the remote server until they complete all tasks allocated to them. As shown in the Fig. 11, assuming that there are 11 edge servers in the edge network, the number of greedy strategy server increases from 2 to 7 under the same 2000 tasks. Obviously, the total execution time shows a downward trend (dotted line) whether under the OPTD algorithm or other comparison algorithms at dispatching as shown in this figure. The value of the right-most of the red dot is 36.2 percent lower than the left-most red dot which the dot is the average of all value of the four algorithms under the same greedy strategy of edge servers. The result reveals that no matter the number of how many edge servers that apply to greedily occupy the computational resource of the remote server, our proposed algorithm always achieves the best performance.

Besides, the migration time of all tasks in the network and the final execution completion time are getting closer and closer. It can be shown that these tasks can always be executed quickly after reaching the server, which reduces both migration and waiting latency. When the number of servers in the greedy strategy is close to the total number of servers, it is close to the pipeline optimal strategy. Because this conclusion is consistent with the total latency of these tasks that decrease while increasing the number of greedy servers in the edge environment.

Fig. 10 demonstrates the total task evacuation time will show an approximately linear relationship as the total number of evacuation tasks changes under different pipeline buffer strategies are adopted in the edge environment. In the Fig. 10, the normalization of Min-Max Normalization is implemented as

$$\zeta_{num} = \frac{latency - latency_{min}}{latency_{max} - latency_{min}}, \tag{34}$$

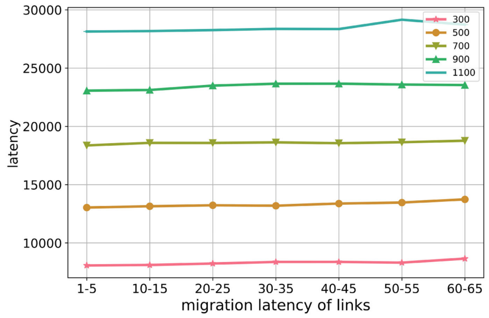

Fig. 14. Latency under different migration cost of links.

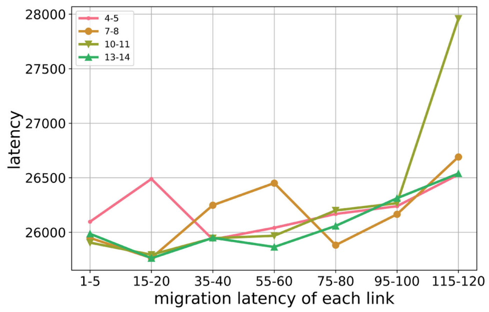

Fig. 15. Latency under different capacity versus migration delay of links.

we can easily konw that when the number of tasks 900, as the safety stock strategy to make $(\bar{F}_j, \tilde{F}_j)$ select from (15,1) to $(1,1)$, is used to stand for the improved percentage ζ_{num} under different task numbers where $num \in \{500, 600, \ldots, 1000\}$ the buffer strategy reduces the overall task evacuation time by about 4.5 percentage points, which is the best among all the different number of task evacuation. Moreover, under the same number of task, the larger the size of the buffer, the smaller the task evacuation latency

Impact of Parameters in the Edge Environment. Inspired by [44], we choose the most important parameters to design and implement the simulation below. First of all, the link capacity and the speed of task migration are key factors in our proposed algorithm. Obviously, different parameters of an edge network will affect the evacuation of tasks, but we want to explore how these parameters impact the latency of all tasks. The simulation results are mainly shown in Figs. 13, 14 and 15.

As the capacity of links in the migration collaboration layer increases, the total latency required for all tasks in this edge network to evacuation always converges to a stable time. As shown in the Fig. 13, we can observe that under different number of tasks, the time required for task evacuation will first decrease rapidly as the link capacity increases. But when the link capacity increases to a certain value, they have little effect on the total time for task evacuation. The total time of task evacuation tends to converge to a stable value.

Therefore, when a sudden burst load occurs in an edge network, appropriately increasing the network capacity helps to improve the task evacuation latency, but it should also be noted that increasing the network capacity does not always reduce the total time for evacuation. As shown in the Fig. 14, the migration cost in the edge network is used as an independent variable. When we increase the migration

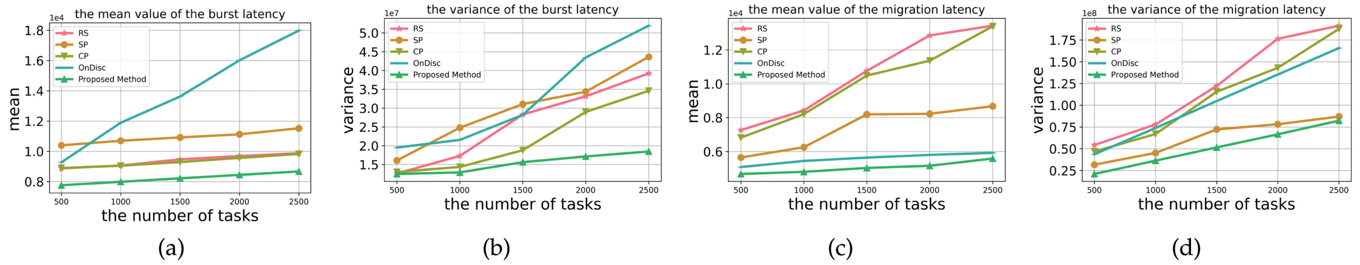

Fig. 16. Dispatching performance.

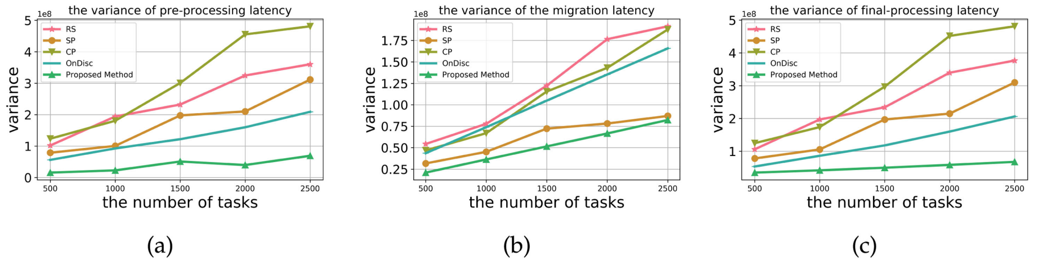

Fig. 17. Scheduling performance.

cost of the link, the time required for the evacuation will always maintain a stable change within a large range.

Considering that there are two stages of tasks in the edge environment, namely, dispatching and scheduling, tasks at the two stages of tasks are executed simultaneously, then each task migration and execution processes overlap in time. Therefore, we observe that the task evacuation time will not change significantly with an appropriately increased migration cost, but once the migration cost reaches a certain value, the task evacuation time will rise significantly.

The time for the evacuated tasks to be migrated to other servers generates a large latency, which often exceeds the pipeline execution time required for task scheduling and execution. It leads to the inefficiency of the pipeline structure. Edge servers need to wait for the task to arrive and then execute, under these circumstances the pipeline structure of edge servers are always idle. Therefore, in the edge network, once we can first ensure all the tasks can be efficiently executed parallel on servers, even if the task migration cost in the network is not small, the total evacuation time can be maintained at a relatively small value.

As shown in the Fig. 15, we keep the number of tasks unchanged but change the network parameters of the task during evacuation. When we also use the migration cost as the horizontal axis, we can observe that under different link capacities in the range with a relatively small rate, the total time for the task evacuation increases slowly, but once a certain threshold is reached, the rate of the curve growth increases significantly.

We can conclude that if the migration cost and capacity of an edge network are limited, we can also achieve a stable evacuation latency. The total evacuation delay is composed of two parts, dispatching and scheduling. Once we can guarantee the efficiency of scheduling, we can get a little inspiration is that the infrastructure investment of the edge network does not necessarily need to select large-capacity and high-speed equipment. Once a burst load happens at the edge environment, as long as the implementation of the dispatch and scheduling strategies are reasonably guaranteed, the total time of the task evacuation can always be maintained within a reasonable range.

Fariness. In the evacuation of all tasks, we can observe from the experimental results that under our optimal routing search algorithm, the average and variance of the delay of each task in the burst phase and migration phase are the smallest compared with the benchmark algorithms, as shown in Fig. 16. Experimental results also show that variance values of latency in stages of pre-process, remote-assistance and final-process are the smallest compared with the benchmark algorithms as shown in Fig. 17.

Our proposed algorithm can give an optimal routing for each task, it will reduce tasks congestion to a great extent in the migration collaboration layer, the time variance of each task is small means that each task can share the resources of links in the migration collaboration more fairly. Similarly, the smaller variance of the execution time of each task in every phase means that they can get computation resources relatively more equal in the computing sharing layer and remote assisted layer. Therefore, both network and computation resources are distributed fairly to each task after the burst load occurs.

6 Conclusion

In this paper, we propose a framework for an edge environment that consists of three layers: the migration collaboration layer, the computing sharing layer, and the remote assisted layer. We divide the evacuation of the burst load into two stages: dispatching and scheduling. While an optimal method to migrate the burst load is given when the resources of the edge network are limited, based on the online algorithm, a strategy dealing with task processing is proposed without knowledge of the task state in advance at the scheduling stage. Applying our proposed algorithm OPTD4.1 and POTS4.2, burst load evacuation can be efficiently processed throughout these three layers. The experimental results prove that these algorithms incorporate both efficiency and equity for tasks generated by users. Our proposed algorithms can be easily integrated into distributed systems in edge environments. In future work, we are interested in a more complex system if there is a complicated processing flow of tasks. In practice, the value of different types of latency of a task may be estimated inaccurately, with other uncertainties not related to this paper, we need to design more effective strategies to solve the evacuation problem.

Acknowledgments

This work was supported in part by the National Science Foundation of China under Grant U20A20173 and Grant 61772461, and in part by the Natural Science Foundation of Zhejiang Province under Grant LR18F020003.

References

[1] Y. Mao *et al.*, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surv. Tut.*, vol. 19, no. 4, pp. 2322–2358, 2017.

[2] F. Lyu *et al.*, "LEAD: Large-scale edge cache deployment based on spatio-temporal wifi traffic statistics," *IEEE Trans. Mobile Comput.*, 2020.

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

[4] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.

[5] S. Deng *et al.*, "Optimal application deployment in resource constrained distributed edges," *IEEE Trans. Mobile Comput.*, to be published, doi: 10.1109/TMC.2020.2970698.

[6] H. Zhao, S. Deng, Z. Liu, J. Yin, and S. Dustdar, "Distributed redundancy scheduling for microservice-based applications at the edge," *IEEE Trans. Serv. Comput.*, to be published, doi: 10.1109/TSC.2020.3013600.

[7] S. Deng *et al.*, "Dynamical resource allocation in edge for trustable Internet-of-Things systems: A reinforcement learning method," *IEEE Trans. Ind. Inform.*, vol. 16, no. 9, pp. 6103–6113, Sep. 2020.

[8] S. Wang, X. Zhang, Z. Yan, and W. Wenbo, "Cooperative edge computing with sleep control under nonuniform traffic in mobile edge networks," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4295–4306, Jun. 2019.

[9] I.-A. Chousainov, I. Moscholios, P. Sarigiannidis, A. Kaloxylos, and M. Logothetis, "An analytical framework of a C-RAN supporting bursty traffic," in *Proc. IEEE Int. Conf. Commun.*, 2020, pp. 1–6.

[10] W.-C. Chien, C.-F. Lai, and H.-C. Chao, "Dynamic resource prediction and allocation in C-RAN with edge artificial intelligence," *IEEE Trans. Ind. Inform.*, vol. 15, no. 7, pp. 4306–4314, Jul. 2019.

[11] Y. Liu, H. Yu, S. Xie, and Y. Zhang, "Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 11, pp. 11 158–11 168, Nov. 2019.

[12] S. H. Rastegar, A. Abbasfar, and V. Shah-Mansouri , "Rule caching in SDN-enabled base stations supporting massive IoT devices with bursty traffic," *IEEE Internet Things J.*, vol. 7, no. 9, pp. 8917 –8931, Sep. 2020.

[13] C. Gündoğran, P. Kietzmann, M. Lenders, H. Petersen, T. C. Schmidt, and M. Wählisch, "NDN, COAP, and MQTT: A comparative measurement study in the IoT," in *Proc. 5th ACM Conf. Inf.-Centric Netw.*, 2018, pp. 159–171.

[14] M. Molina, O. Muñoz, A. Pascual-Iserte , and J. Vidal, "Joint scheduling of communication and computation resources in multiuser wireless application offloading," in *Proc. IEEE 25th Annu. Int. Symp. Pers. Indoor Mobile Radio Commun.* , 2014, pp. 1093–1098.

[15] J. Ren, Y. He, G. Huang, G. Yu, Y. Cai, and Z. Zhang, "An edge-computing based architecture for mobile augmented reality," *IEEE Netw.*, vol. 33, no. 4, pp. 162–169, Jul./Aug. 2019.

[16] Y. Chen, S. Deng, H. Ma, and J. Yin, "Deploying data-intensive applications with multiple services components on edge," *Mobile Netw. and Appl.*, vol. 25, pp. 426–441, 2019.

[17] J. Ren, D. Zhang, S. He, Y. Zhang, and T. Li, "A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet," *ACM Comput. Surv.*, vol. 52, no. 6, pp. 1–36, 2019.

[18] E. Li, Z. Zhou, and X. Chen, "Edge intelligence: On-demand deep learning model co-inference with device-edge synergy," in *Proc. Workshop Mobile Edge Commun.*, 2018, pp. 31–36.

[19] S. Deng *et al.*, "Toward mobile service computing: Opportunities and challenges," *IEEE Cloud Comput.*, vol. 3, no. 4, pp. 32–41, Jul./Aug. 2016.

[20] K. Sasaki, S. Makido, and A. Nakao, "Vehicle control system for cooperative driving coordinated multi-layered edge servers," in *Proc. IEEE 7th Int. Conf. Cloud Netw.*, 2018, pp. 1–7.

[21] Y. Sarikaya, H. Inaltekin, T. Alpcan, and J. S. Evans, "Stability and dynamic control of underlay mobile edge networks," *IEEE Trans. Mobile Comput.*, vol. 17, no. 9, pp. 2195–2208, Sep. 2018.

[22] H. Trinh *et al.*, "Energy-aware mobile edge computing and routing for low-latency visual data processing," *IEEE Trans. Multimedia*, vol. 20, no. 10, pp. 2562–2577, Oct. 2018.

[23] X. Lyu *et al.*, "Optimal schedule of mobile edge computing for internet of things using partial information," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2606–2615, Nov. 2017.

[24] P. Romano and F. Quaglia, "Design and evaluation of a parallel invocation protocol for transactional applications over the Web," *IEEE Trans. Comput.*, vol. 63, no. 2, pp. 317–334, Feb. 2014.

[25] C. Wang, F. R. Yu, C. Liang, Q. Chen, and L. Tang, "Joint computation offloading and interference management in wireless cellular networks with mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 66, no. 8, pp. 7432–7445, Aug. 2017.

[26] X. Deng, M. Xu, L. T. Yang, M. Lin, L. Yi, and M. Wang, "Energy balanced dispatch of mobile edge nodes for confident information coverage hole repairing in IoT," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4782–4790, Jun. 2019.

[27] H. Yuan, J. Bi, W. Tan, M. Zhou, B. H. Li, and J. Li, "TTSA: An effective scheduling approach for delay bounded tasks in hybrid clouds," *IEEE Trans. Cybern.*, vol. 47, no. 11, pp. 3658–3668, Nov. 2017.

[28] 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, Oct. 2016.

[29] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1619–1632, Aug. 2018.

[30] H. Zhao, S. Deng, C. Zhang, W. Du, Q. He, and J. Yin, "A mobility-aware cross-edge computation offloading framework for partitionable applications," in *Proc. IEEE Int. Conf. Web Serv.*, 2019, pp. 193–200.

[31] W. Fang, X. Yao, X. Zhao, J. Yin, and N. Xiong, "A stochastic control approach to maximize profit on service provisioning for mobile cloudlet platforms," *IEEE Trans. Syst., Man, Cybern.: Syst.*, vol. 48, no. 4, pp. 522–534, Apr. 2018.

[32] S. Deng, L. Huang, J. Taheri, and A. Y. Zomaya, "Computation offloading for service workflow in mobile cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 12, pp. 3317–3329, Dec. 2015.

[33] M. S. Elbamby, M. Bennis, and W. Saad, "Proactive edge computing in latency-constrained fog networks," in *Proc. Eur. Conf. Netw. Commun.*, 2017, pp. 1–6.

[34] J. Liu *et al.*, "Online multi-workflow scheduling under uncertain task execution time in iaas clouds," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2019.2906300.

[35] Y. Yu, H. Qian, and Y.-Q. Hu, "Derivative-free optimization via classification," in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016, vol. 30, no. 1.

[36] S. Gao, M. Zhou, Y. Wang, J. Cheng, H. Yachi, and J. Wang, "Dendritic neuron model with effective learning algorithms for classification, approximation, and prediction," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 2, pp. 601–614, Feb. 2019.

[37] M. J. Kearns, U. V. Vazirani, and U. Vazirani, *An Introduction to Computational Learning Theory*. Cambridge, MA, USA: MIT press, 1994.

[38] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*. Cambridge, MA, USA: MIT Press, 2012.

[39] J. Dai and G. Weiss, "A fluid heuristic for minimizing makespan in job shops," *Operations Res.*, vol. 50, no. 4, pp. 692–707, 2002.

[40] E. Gebennini, R. Gamberini, and R. Manzini, “An integrated production–distribution model for the dynamic location and allocation problem with safety stock optimization,” *Int. J. Prod. Econ.*, vol. 122, no. 1, pp. 286–304, 2009.

[41] I. Kacem, S. Hammadi, and P. Borne, “Lower bounds for evaluating schedule performance in flexible job shops,” *Perform. Metrics for Intell. Syst. Workshop*, pp. 347–363, 2002.

[42] C. Reiss, J. Wilkes, and J. L. Hellerstein, “Google cluster-usage traces: Format+ schema,” Google Inc., White Paper, pp. 1–14, 2011.

[43] H. Tan, Z. Han, X.-Y. Li, and F. C. Lau, “Online job dispatching and scheduling in edge-clouds,” in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.

[44] J. Wang and T. Kumbasar, “Parameter optimization of interval type-2 fuzzy neural networks based on PSO and BBBC methods,” *IEEE/CAA J. Automatica Sinica*, vol. 6, no. 1, pp. 247–257, Jan. 2019.

Shuiguang Deng (Senior Member, IEEE) received the BS and PhD degrees in computer science, in 2002 and 2007, respectively. He is currently a full professor with the College of Computer Science and Technology in Zhejiang University, China. He previously worked at the Massachusetts Institute of Technology, in 2014, and at Stanford University, in 2015 as a visiting scholar. His research interests include edge computing, service computing, mobile computing, and business process management. He serves as an associate editor for the journal *Computing*, *Knowledge and Information Systems*, *IEEE Access,* and *IET Cyber-Physical Systems: Theory Applications*. During the past ten years, he has published more than 100 papers in journals and refereed conferences. In 2018, he was granted the Rising Star Award by IEEE TCSVC. He is a fellow of IET.

Cheng Zhang received the MS degree in electrical engineering from Zhejiang University, China, in 2013, Currently, he is working toward the PhD degree in computer science and technology at Zhejiang University. His research interests include edge computing and edge intelligence.

Chang Li is currently working toward the postgraduate degree in computer technology. His research interests includ edge computing, distributed system, and middleware software.

Jianwei Yin received the PhD degree in computer science from Zhejiang University (ZJU), in 2001. He was a visiting scholar with the Georgia Institute of Technology. He is currently a full professor with the College of Computer Science, ZJU. Up to now, he has published more than 100 papers in top international journals and conferences. His current research interests include service computing and business process management. He is an associate editor of the *IEEE Transactions on Services Computing*.

Schahram Dustdar (Fellow, IEEE) is a full professor of computer science (informatics) with a focus on Internet Technologies heading the Distributed Systems Group at the TU Wien. He is chairman of the Informatics Section of the Academia Europaea (since December 9, 2016). From 2004–2010 he was honorary professor of Information Systems at the Department of Computing Science, the University of Groningen (RuG), The Netherlands. From December 2016 until January 2017 he was a visiting professor at the University of Sevilla, Spain, and from January until June 2017 he was a visiting professor at UC Berkeley, USA. He is a member of the IEEE Conference Activities Committee (CAC) (since 2016), the Section Committee of Informatics of the Academia Europaea (since 2015), a member of the Academia Europaea: The Academy of Europe, Informatics Section (since 2013). He is the recipient of the ACM Distinguished Scientist Award (2009) and the IBM Faculty Award (2012). He is an associate editor of *IEEE Transactions on Services Computing*, *ACM Transactions on the Web*, and *ACM Transactions on Internet Technology,* and on the editorial board of *IEEE Internet Computing*. He is the editor-in-chief of *Computing* (an SCI-ranked journal of Springer).

Albert Y. Zomaya (Fellow, IEEE) is currently the chair professor of high performance computing & networking at the School of Computer Science, University of Sydney. He is also the director of the Centre for Distributed and High Performance Computing.He published more than 600 scientific papers and articles and is a author, co-author, or editor of more than 20 books. He served as the editor-in-chief of the *IEEE Transactions on Computers*(2011–2014). Currently, he is the editor-in-chief of *ACM Computing Surveys* and serves as associate editor for several leading journals. He delivered more than 190 keynote addresses, invited seminars, and media briefings and has been actively involved, in a variety of capacities,in the organization of more than 700 national and international conferences. He received the IEEE Technical Committee on Parallel Processing Outstanding Service Award (2011), the IEEE Technical Committee on Scalable Computing Medal for Excellence in Scalable Computing (2011), and the IEEE Computer Society Technical Achievement Award (2014). He is a chartered engineer, a fellow of AAAS, IET (United Kingdom), and an elected member of Academia Europaea. His research interests include the areas of parallel and distributed computing and complex systems.