# Leveraging joint allocation of multidimensional resources for distributed task assignment

JIALONG LI[1,2], NAN HUA[1,2], KANGQI ZHU[1,2], CHEN ZHAO[1,2], GUANQIN PAN[3], YANHE LI[4], XIAOPING ZHENG[1,2], AND BINGKUN ZHOU[1,2]

[1] *Beijing National Research Center for Information Science and Technology (BNRist), Beijing 100084, China*
[2] *Department of Electronic Engineering, Tsinghua University, Beijing 100084, China*
[3] *School of Computer, Electronics and Information, Guangxi University, Nanning 530000, China*
[4] *Tsinghua Unigroup, Tsinghua University, Beijing 100084, China*
[*] *Corresponding author: xpzheng@mail.tsinghua.edu.cn*

*Compiled February 22, 2022*

**Edge computing has changed the landscape of telecommunication networks. Different from the cloud computing that thousands of servers are centralized in a remote site, computation and storage resources are deployed at the network edge in edge computing, reducing the end-to-end latency and the amount of transmitting data in the metro/backbone networks significantly. Due to the limited resource capacity in a single edge node and the requirements of distributed applications, some applications are supposed to be decomposed into multiple interdependent tasks and executed in distributed resource-constrained nodes. Assigning tasks to geographically distributed edge nodes is quite challenging because of the allocation of multidimensional resources (i.e., computation, storage, and transmission resources) as well as constraints of the interdependency between different tasks. Strategies that only take one factor into account for optimization will cause improper task assignments, leading to higher end-to-end latency and lower resource utilization efficiency. To solve this problem, we formulate a mathematical model aiming at minimizing the job completion time (JCT) by jointly considering the availability of multidimensional resources and the interdependency between different tasks. We obtain optimal results in the small topology by using the optimization software, which validates the correctness of the proposed mathematical model. Furthermore, we analyze the complexity and design a practical algorithm by narrowing the searching space in the large-scale topology. Simulation results present its effectiveness over greedy algorithms. Finally, we conduct a proof of concept experiment to validate the feasibility of the proposed strategy.**

http://dx.doi.org/10.1364/ao.XX.XXXXXX

## 1. INTRODUCTION

Benefit from the virtualization technology, cloud computing has made a great success and computation resources are easily accessible just like water and electricity [1]. However, some applications' performances are greatly influenced by the latency, and cloud computing is hard to meet the latency requirements. For example, the ideal latency for virtual reality (VR) should be less than 15–20 $ms$, or dissonance occurs. What's worse, due to the policy constraints and cost concerns, nowadays new data centers are built in farther places [2], making cloud computing harder to support these latency-critical applications. Edge computing is regarded as a promising architecture to provide extremely low-latency services, where computation and storage resources are deployed closer to the network edge [3, 4]. Compared with cloud computing that transmitting all data to the data center for centralized processing, edge computing could provide real-time processing at the network edge, which reduces the end-to-end latency significantly and enhances the prosperity of time-sensitive applications. In addition, the transmission resource bottleneck in the metro and backbone networks could be alleviated since most of the requests are processed locally. According to Gartner's prediction [5], at least 75% of the data will be generated and processed at the network edge by 2025.

Unlike the legacy network where data centers locate in a remote site and networks act as pipes that connect end users and data centers, edge computing empowers request processing and data storage within the network element. There are two main reasons why a request needs to be decomposed into multiple tasks and executed in distributed edge nodes. First, compared with the data centers containing thousands of servers, the computation and storage resources in a single computing node are limited [6]. Currently, the resource demand for a request is increasing dramatically, and it is beyond a single node's processing ability. The situation will be worse during peak traffic periods. Second, some applications need collaborations between different nodes, like federated learning (FL) [7, 8] and intelligent video surveillance (IVS) [9]. In FL, training data generated in different sources are not necessarily transmitted to the data center for data privacy concerns. Instead, training data are processed in a distributed way, where the intermediate results are transferred to a central server. As for IVS, the surveillance area for a camera is finite. An edge node where the camera is located needs real-time video analytics and shares information with the nearby nodes. In short, these applications rely on the coordination among multiple nodes, and edge computing will accelerate the emergence of such applications in turn.

Conducting task assignments to geographically distributed edge nodes is quite challenging. Two main factors make this problem complex. First, we should consider the constraints of tasks' characteristics. The interdependency among these tasks should be considered [10], and computation results transmission are required among these tasks. Furthermore, raw data are necessary during the execution process for some tasks. Second, the allocation of multidimensional resources (computation, storage, and transmission resources) is complex. It is crucial to take the availability of the resources into account as well as the resource deployment locations. The shortage of any resources will introduce the waiting time before task executions and data transmissions [11], leading to higher entire job completion time (JCT) and lower resource utilization efficiency.

To tackle this problem, we design a strategy based on the joint allocation of transmission, computation, and storage resources (JTCS) when assigning multiple tasks to distributed physical nodes. The contributions of this work are summarized as follows.

1. To the best of our knowledge, it is the first work on joint allocation of multidimensional resources for distributed tasks mapping, taking the interdependency among tasks, constraints of multidimensional resources, and propagation delay into consideration.

2. We formulate the JCT optimization problem into an integer linear program (ILP) model. The optimal results in a small topology by using the optimization software are obtained. In this way, the correctness of this model is validated. We also analyze the time complexity of the proposed model in detail.

3. We further design a practical algorithm by narrowing the searching space in the large-scale topology. We prove the effectiveness of our strategy by numerical results.

4. We investigate JCT and transmission resource utilization performances under various conditions. Raw data transmission time, maximum computation resource capacities, number of raw data storage nodes, and physical link distances are set as variables and well-studied in the simulations.

5. Finally, we conduct a proof of concept experiment to verify the feasibility of our approach. Experiment results demonstrate that lower JCT could be realized by joint allocation of multidimensional resources.

The rest of the paper is organized as follows. Previous works on task assignments and multidimensional resource allocations are reviewed in Section 2. In Section 3, distributed task model and multidimensional resources allocation are illustrated. Section 4 presents the mathematical model and its complexity in detail. A benchmark algorithm is also proposed for comparison. Performances are evaluated in small and large-scale topologies in Section 5 and Section 6, respectively. Experiment setup and results are shown in Section 7. Section 8 concludes this work.

## 2. RELATED WORK

The coordination between tasks assignment and computation resources, radio frequency, and caching, is already studied in wireless communications. Reference [12] explored the synergy between communication, caching, and computation in wireless scenarios. The simulation results conclusively present that appropriate deployment of computation and caching resources should be determined by the application requirements. Authors in Reference [13] tried to minimize the computation latency by optimizing the task assignment jointly with radio and computation resource allocation, subject to the constraints of device energy consumption and computation capacities. To reduce the calculation complexity, they designed a heuristic scheme based on the greedy task assignment. The proposed strategy performs better than the benchmarks without considering joint task assignments and resource allocations. However, the interdependency among tasks is not considered, which is very critical in task assignments. The interdependency is included in the mathematical model introduced by Reference [14]. This paper took into account the application-specific profile, availability of computation resources, link connectivity, and energy consumption when finding optimal task assignments. Latency performances are improved by 16% compared with a previously published heuristic scheme, which validates the effectiveness of the designed scheme.

In terms of the collaborative allocation of multidimensional resources in optical networks, Reference [15] proposed a strategy by joint adjusting the size of allocated computation and transmission resources. In this way, multiple computation results generated by distributed tasks could be transmitted to the next node simultaneously, avoiding the waiting time among these results. However, waiting time reduction is limited, and its performances are highly dependent on the tasks' interdependency. Reference [16] designed a multi-state adjustment algorithm for multidimensional resource allocation. The task mapping procedures are adjusted according to the availability of the transmission resources and the location of the raw data. However, this strategy is essentially based on greedy methods and could not obtain the theoretical optimal value. Furthermore, this work does not take the availability of computation resources into account. Reference [17] introduced the resource utilization balance factor to allocate multidimensional resources evenly. Unexpected long waiting time caused by the shortage of resources could be avoided. However, the raw data are not considered in the authors' model. Authors in Reference [18] pointed out that compared with transmitting the raw data, it is a better choice to assign the task to the node where the raw data are stored. According to the simulation results, JCT is diminished by more than 25%, while the amount of transmitting raw data is reduced by 75%. However, the transmission resource restriction is not considered in the model, influencing the simulation results greatly.

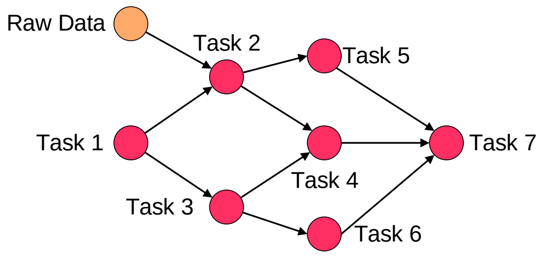


**Fig. 1.** DAG model.

## 3. DISTRIBUTED TASK MODEL AND MULTIDIMENSIONAL RESOURCES ALLOCATION

### A. Distributed Task Model

In this section, we first introduce how to model the interdependency among distributed tasks. A job is decomposed to multiple distributed computation tasks, which could be modelled by directed acyclic graph (DAG) [19]. A typical DAG illustrating the interdependency among 7 tasks is given in Figure 1. A one-way arrow connects two tasks. The task at the beginning is called forward task, and the end is the backward task. Task 2 is the forward and backward tasks of Task 5 and Task 1, respectively. Note also that there may be more than one forward or backward task. For example, Task 4 has two forward tasks, i.e., Task 2 and Task 3. The backward task execution could only start after executions of all forward tasks are finished and corresponding computation results are transmitted to the backward task. For instance, only receiveing all the computation results generated by Task 4-6, the execution of Task 7 could start.

Topological sorting algorithm is widely used to get a linear ordering of a DAG. For every forward task $u$ and backward task $v$, $u$ comes before $v$ in the ordering. According to the linear ordering, we could obtain a mapping order of tasks. Take Figure 1 as an example, sequence 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 is a feasible ordering when ruuning the algorithm. This sequence could provide a task execution order, which does not violate the constraints of forward and backward tasks. The complexity of the algorithm is $O(|M|+|Q|)$ when the DAG consists of $M$ tasks and $Q$ links. Topological sorting algorithm will be adpoted when we design the benchmark in Section 4.

In addition, we also consider the raw data constraint. Some tasks, such as Task 2 in Figure 1, require raw data to complete the execution. Take the face identification system as an example. The face identification system could be divided into five components, i.e., faical image acquisition, face detection, faical image preprocessing, feature extraction, and system database [20, 21]. System database locates in edge nodes and stores user identity information. So, user identity information is required in the face detection task. Note that if the task is not assigned to the node where database is stored, the database should be transmitted to the node where the task is executed. We should point out that not all edge nodes will be used for raw data storage due to the data privacy concern [22, 23]. In Section 6, we also study the impact by different number of raw data storage nodes.

### B. Multidimensional Resources Allocation

Figure 2 is a schematic diagram of multidimensional resources. It can be seen from the figure that distributed task executions require the participation of three types of resources, namely storage, computation, and transmission resources. Computation resources are used for task executions. Computation resources are further composed of the central processing unit (CPU) and random-access memory (RAM). Benefit from the virtualization technology, the edge node could be shared by multiple users simultaneously. The allocation of computation resources is achieved by configuring virtual machines (VM) with different specifications. By specifying the size of CPU and RAM, we could set up VMs to meet various requirements of task executions. Three VMs are running on node 6 in Figure 2, namely VM1, VM2, and VM3. Task 3 is executed by one of these VMs.

The storage resources are used for storing raw data, which are necessary for some task executions. Note that for data privacy concerns, not all nodes are used to store raw data. If raw data are not stored in the node where the task is processed, the task could not start execution until the raw data from other nodes are transmitted. As the figure shows, the raw data are transmitted to the node where Task 2 is executed.

As for transmission resources, we assume that the optical networks connecting edge nodes are based on optical time slice switching (OTSS) technology [24]. OTSS is a novel all-optical switching technology and could realize transparent connections. Previous experiments and simulations proved the deployment feasibility in metro networks [25]. The optical channel in time-domain is divided into repetitive OTSS frames, and an OTSS frame is further slicing into multiple minimum optical time slices (MOTS). Transmitting data will occupy one or more continuous MOTSs in a selected wavelength. So an MOTS has two states, i.e., available and unavailable states. In this way, the availability of transmission resources could be measured by the MOTS states in time domain.

### C. Components of JCT

As Figure 3 shows, JCT includes computation results transmission time ($T_{cotr}$), task execution time ($T_{exec}$), waiting time ($T_{wait}$), and propagation latency ($T_{prop}$). The first two are identical even applying different strategies, while the last two should be minimized. Waiting time is induced by the shortage of transmission/computation resources when transmitting data or executing tasks. A shorter path leads to lower propagation latency.

## 4. MATHEMATICAL MODEL AND JTCS STRATEGY

### A. Mathematical Model

In this section, we first present the mathematical model, whose goal is to minimize the JCT. The purpose of Equation (1) is to minimize the average JCT of all jobs. $A$ is the set of jobs. Take one of the jobs $a$ as an example. Note that $a_p$ is one of the tasks of job $a$. $\theta^{a_p}_{(n,i)}$ is a binary decision variable, which is one if task $a_p$ uses computation resources in the $i_{th}$ time slice in node $n$. When $\theta^{a_p}_{(n,i)} = 1$, max($i$) means the last task of job $a$ is executed in time slice $i$. That is to say, max (i) is the JCT of job $a$. We add all JCT values and calculate the average JCT.

**Given parameters**

Table 1 shows the given parameters of ILP model.

**Decision variables**

Table 2 summarizes the decision variables of ILP model.

**Optimization**

$$\textbf{Minimize:} \quad F = \sum_{a \in A} (\max(i), \quad \theta^{a_p}_{(n,i)} = 1, a_p \in a, n \in N, i \in S)/A \tag{1}$$

**Constraints**

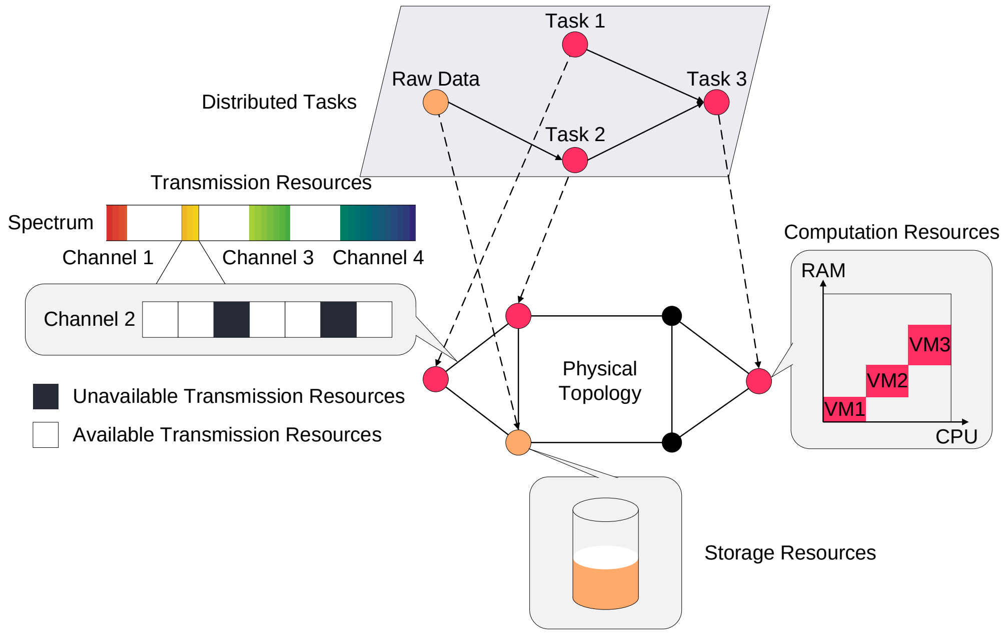


**Fig. 2.** Distributed tasks mapping and multidimensional resources allocation.

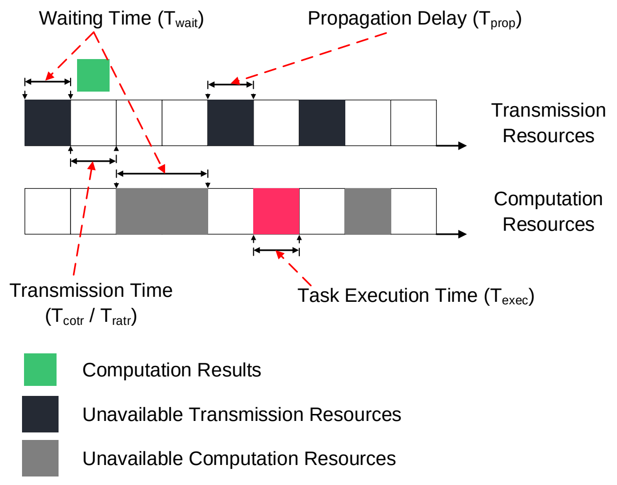

Waiting Time (T_wait)
Propagation Delay (T_prop)
Transmission Resources
Computation Resources
Transmission Time (T_cotr / T_ratr)
Task Execution Time (T_exec)
Computation Results
Unavailable Transmission Resources
Unavailable Computation Resources

**Fig. 3.** JCT consists of transmission time, task execution time, waiting time, and propagation delay.

$$x_n^{a_p} + x_n^{a_q} \leqslant 1, \quad \forall n \in N, a_p \neq a_q, a_p, a_q \in a, a \in A \tag{2}$$

$$\sum_{n\in N} x_n^{a_p} = 1, \quad \forall a_p \in a, a \in A \tag{3}$$

$$T^{(b,a_p)} = \begin{cases} T^{(b,a_p)}, & x_n^{a_p} = 1, b \neq n \\ 0, & x_n^{a_p} = 1, b = n \end{cases} \quad a_p \in a, a \in A, n \in N \tag{4}$$

Equations (2-4) are about mapping constraints between distributed tasks and physical nodes. Equation (2) demonstrates that a node could not support two tasks from the same job. Equation (3) indicates that a task is executed by one and only one node. Equation (4) shows that data transmission is avoided when data storage and task execution are served in the same node.

$$\sum_{i\in S} \theta_{(n,i)}^{a_p} = \begin{cases} C^{a_p}, & x_n^{a_p} = 1 \\ 0, & x_n^{a_p} = 0 \end{cases} \quad a_p \in a, a \in A, n \in N \tag{5}$$

$$\sum_{a\in A} \sum_{a_p \in a} x_n^{a_p} \cdot \theta_{(n,i)}^{a_p} \cdot cpu^{a_p} \leqslant CPU, \quad \forall n \in N, \forall i \in S \tag{6}$$

$$\sum_{a\in A} \sum_{a_p \in a} x_n^{a_p} \cdot \theta_{(n,i)}^{a_p} \cdot ram^{a_p} \leqslant RAM, \quad \forall n \in N, \forall i \in S \tag{7}$$

Equation (5) denotes the computation resources allocation when a task is assigned to a node. Equations (6-7) show the amount of computation resources is limited. Equations (6) and (7) imply that the usage of CPU and RAM resources should be less than the maximum capacity, respectively. Computation resources will be released after the task execution is finished.

$$\pi_{(e_1,w)}^{(a_p,a_q)} = \pi_{(e_2,w)}^{(a_p,a_q)}, \quad \forall w \in W, e_1, e_2 \in l^{(a_p,a_q)}, a_p, a_q \in a, a \in A \tag{8}$$

$$\sum_{i\in S} y_{(e,w,i)}^{(a_p,a_q)} = \begin{cases} T^{(a_p,a_q)}, & \pi_{(e,w)}^{(a_p,a_q)} = 1 \\ 0, & \pi_{(e,w)}^{(a_p,a_q)} = 0 \end{cases} \quad \forall e \in l^{(a_p,a_q)}, \; a_p, a_q \in a, a \in A, w \in W \tag{9}$$

$$\max(i) - \min(i) = T^{(a_p,a_q)} - 1, \quad y_{(e,w,i)}^{(a_p,a_q)} = 1, \forall e \in l^{(a_p,a_q)}, \; a_p, a_q \in a, a \in A, w \in W, i \in S \tag{10}$$

$$\sum_{e_1,e_2 \in l^{(a_p,a_q)}} (\min(j) - \min(i)) = \lceil D^{l^{(a_p,a_q)}}/c/t \rceil, \quad y_{(e_1,w,i)}^{(a_p,a_q)} = 1, \; y_{(e_2,w,j)}^{(a_p,a_q)} = 1, e_1 \neq e_2, a_p, a_q \in a, a \in A, w \in W, i, j \in S \tag{11}$$

**Table 1.** Given parameters

| Input | Description |
|---|---|
| $G(N,E)$ | network topology, where $N$ and $E$ represent the set of nodes and fiber links, respectively. |
| $A$ | set of DAGs. |
| $a$ | a DAG, and $a \in A$. |
| $W$ | set of wavelengths in each fiber link. |
| $c$ | the speed of light in the fiber. |
| $(a_p, a_q)$ | computation results transmission between tasks $a_p$ and $a_q$, where $a_p$ is the forward task and $a_q$ is the backward task. $a_p, a_q \in a$. |
| $T^{(a_p,a_q)}$ | number of time slices required by transmission $(a_p, a_q)$. |
| $(b, a_p)$ | raw data transmission for task $a_p$, where raw data are stored in node $b$ initially. $b \in N$. |
| $T^{(b,a_p)}$ | number of MOTSs required by transmission $(b, a_p)$. |
| $C^{a_p}$ | number of MOTSs required by execution $a_p$. |
| $t$ | the length of a MOTS. |
| $S$ | set of MOTSs in one frame. |
| $l^{(a_p,a_q)}$ | lightpath used by computation results transmission $(a_p, a_q)$. |
| $l^{(b,a_p)}$ | lightpath used by raw data transmission $(b, a_p)$. |
| $D^l$ | physical length of lightpath $l$. |
| $D^e$ | physical length of edge $e$, $e \in l$. |
| $CPU$ | CPU capacity of each node. |
| $RAM$ | RAM capacity of each node. |
| $cpu^{a_p}$ | CPU resource requirements for task $a_p$. |
| $ram^{a_p}$ | RAM resource requirements for task $a_p$. |

Equations (8-11) show constraints for optical networks based on OTSS. Equation (8) is wavelength continuity constraint. For a transmission request, the same wavelength should be used in each link. Equations (9)-(10) guarantee the time slices continuity and equation (11) shows the propagation latency of optical time slices.

$$z^{(b,a_p)}_{(e,w,i)} - \theta^{a_p}_{(n,i)} = \begin{cases} 1, & z^{(b,a_p)}_{(e,w,i)} = 1 \\ -1, & \theta^{a_p}_{(n,i)} = 1 \\ 0, & else \end{cases} \quad \forall e \in l^{(b,a_p)}, \tag{12}$$
$$a_p \in a, a \in A, n \in N, w \in W, i \in S$$

$$\min(j) - \min(i) \geqslant \lceil D^{l^{(b,a_p)}}/c/t \rceil + T^{(b,a_p)}, \quad z^{(b,a_p)}_{(e,w,i)} = 1, \tag{13}$$
$$\theta^{a_p}_{(n,j)} = 1, e \in l^{(b,a_p)}, a_p \in a, a \in A, n \in N, w \in W, i,j \in S$$

**Table 2.** Decision variables

| Variables | Description |
|---|---|
| $x^{a_p}_n$ | binary decision variable, which is one if the task $a_p$ is executed in node $n$. |
| $y^{(a_p,a_q)}_{(e,w,i)}$ | binary decision variable, which is one if results transmission $(a_p, a_q)$ uses the $i_{th}$ time slice on wavelength $w$ in edge $e$. |
| $z^{(b,a_p)}_{(e,w,i)}$ | binary decision variable, which is one if data transmission $(b, a_p)$ uses the $i_{th}$ time slice on wavelength $w$ in edge $e$. |
| $\pi^{(a_p,a_q)}_{(e,w)}$ | binary decision variable, which is one if results transmission $(a_p, a_q)$ uses wavelength $w$ in edge $e$. |
| $\theta^{a_p}_{(n,i)}$ | binary decision variable, which is one if the task $a_p$ uses computation resources in the $i_{th}$ time slice in node $n$. |

Equations (12)-(13) indicate that a task could not be executed before the data transmission. If raw data are not stored in the node where the task is processed, the task execution does not start until the raw data are transmitted to the node.

$$\theta^{a_p}_{(n,i)} - y^{(a_p,a_q)}_{(e,w,i)} = \begin{cases} 1, & \theta^{a_p}_{(n,i)} = 1 \\ -1, & y^{(a_p,a_q)}_{(e,w,i)} = 1 \\ 0, & else \end{cases} \quad \forall e \in l^{(a_p,a_q)}, \tag{14}$$
$$a_p, a_q \in a, a \in A, n \in N, w \in W, i \in S$$

$$\min(j) - \min(i) \geqslant \lceil D^{l^{(a_p,a_q)}}/c/t \rceil + C^{a_p}, \quad \theta^{a_p}_{(n,i)} = 1, y^{(a_p,a_q)}_{(e,w,j)} = 1,$$
$$e \in l^{(a_p,a_q)}, a_p, a_q \in a, a \in A, n \in N, w \in W, i,j \in S \tag{15}$$

$$y^{(a_p,a_q)}_{(e,w,i)} - \theta^{a_q}_{(n,i)} = \begin{cases} 1, & y^{(a_p,a_q)}_{(e,w,i)} = 1 \\ -1, & \theta^{a_q}_{(n,i)} = 1 \\ 0, & else \end{cases} \quad \forall e \in l^{(a_p,a_q)}, \tag{16}$$
$$a_p, a_q \in a, a \in A, n \in N, w \in W, i \in S$$

$$\min(j) - \min(i) \geqslant \lceil D^{l^{(a_p,a_q)}}/c/t \rceil + T^{(a_p,a_q)}, \quad y^{(a_p,a_q)}_{(e,w,i)} = 1,$$
$$\theta^{a_p}_{(n,j)} = 1, e \in l^{(a_p,a_q)}, a_p, a_q \in a, a \in A, n \in N, w \in W, i,j \in S \tag{17}$$

Equations (14)-(17) are relations among computation results, forward and backward tasks. Equations (14)-(15) ensure computation results transmission starts after the forward task execution is finished. Equations (16)-(17) infer that the backward task will not be executed until all computation results from its forward tasks are transmitted.

$$y^{(a_p,a_q)}_{(e,w,i)} + y^{(a_u,a_v)}_{(e,w,i)} \leqslant 1, \quad e \in l^{(a_p,a_q)}, e \in l^{(a_u,a_v)}, a_p, a_q \in a_1, \tag{18}$$
$$a_u, a_v \in a_2, a_1, a_2 \in A, a_1 \neq a_2, w \in W, i \in S$$

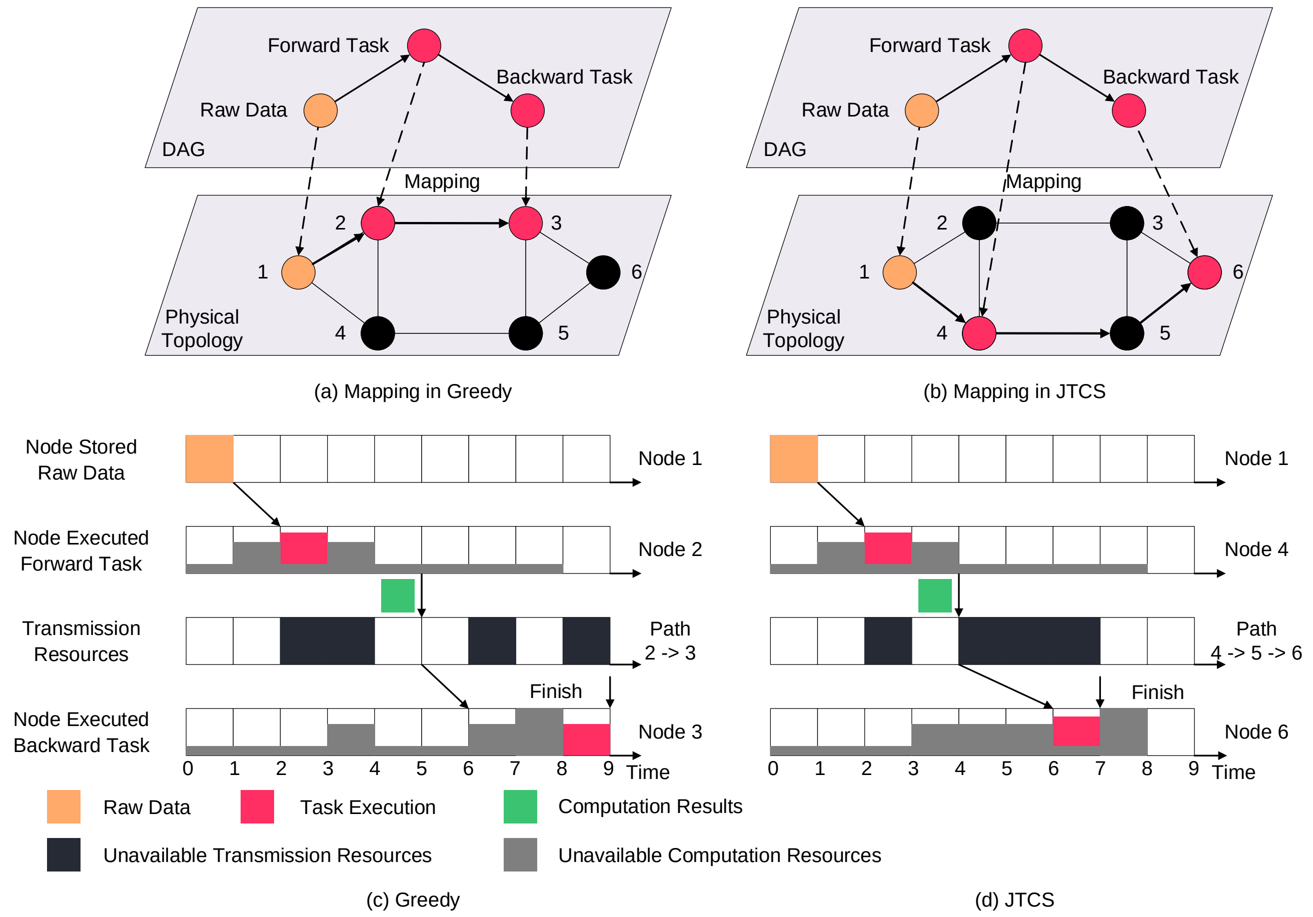


**Fig. 4.** Greedy and JTCS.

$$z_{(e,w,i)}^{(b,a_p)} + y_{(e,w,i)}^{(a_u,a_v)} \leqslant 1, \quad e \in l^{(b,a_p)}, e \in l^{(a_u,a_v)}, a_p \in a_1,$$
$$a_u, a_v \in a_2, a_1, a_2 \in A, a_1 \neq a_2, w \in W, i \in S \tag{19}$$

$$z_{(e,w,i)}^{(b,a_p)} + z_{(e,w,i)}^{(b,a_u)} \leqslant 1, \quad e \in l^{(b,a_p)}, e \in l^{(b,a_u)}, a_p \in a_1,$$
$$a_u \in a_2, a_1, a_2 \in A, a_1 \neq a_2, w \in W, i \in S \tag{20}$$

Equations (18)-(20) represent the constraints between different jobs. Equation (18) ensures that computation results from two different jobs do not use the same MOTS for transmission. Equation (19) guarantees that computation results and raw data transmission from two jobs occupy different MOTSs. Equation (20) ensures that transmission resource conflicts are avoided when transmitting raw data from two jobs.

## B. Greedy Strategy

Before we discuss JTCS, we first design a greedy method based on minimal distance (Greedy) [26], which is shown in Figure 4(a) and (c). Greedy tries to minimize the propagation delay, so as to minimize JCT. To this end, the backward task is deployed on the physical node as close as possible to the node where the forward task is mapped. The process of Greedy is displayed as follows.

1. Running topological sorting algorithm to obtain a linear ordering of the DAG;
2. Selecting a node randomly [1]; Running breadth-first search (BFS) algorithm to obtain a node sequence by taking this node as the root;
3. Mapping the tasks from the linear ordering to the nodes from the node sequence in order;
4. For the computation result transmission, Dijkstra algorithm is applied to obtain the shortest path. After the path is determined, wavelengths and MOTSs should be allocated according to the transmission resource requirement.

Here we give an example to illustrate the process of Greedy, which is shown in Figure 4(a) and (c). Figure 4(a) demonstrates the mapping process, while Figure 4(c) presents the computation and transmission resource allocation. The topology in Figure 4(a) contains 6 nodes. Two tasks are depicted in the DAG. The forward and backward tasks are mapped to Node 2 and 3, respectively. The raw data required by the forward task locate in Node 1. Assuming that the distance between any two nodes directly connected is identical, and the propagation delay in the distance is one time slice. Based on the assumption, we know that the distance between Node 2 and Node 3, which execute the forward and backward tasks respectively, is the shortest. However, the shortest distance does not guarantee the minimal

[1]Note that for the first task of a job, it does not have any forward task for references. So we will just select a node randomly and assign it to the first task. And the other tasks will choose nodes based on their physical distances to the node which is chosen by the forward task.

entire JCT, which is depicted in Figure 4(c). After the raw data are transmitted to Node 2 where the forward task is mapped, Node 2 could provide enough computation resources for the forward task execution from Time 2 to Time 3. So the forward task is executed immediately. After the forward task execution is finished at Time 3, the generating computation results need to be transmitted to Node 3 where the backward task is executed. However, the time slice within Time 3 to Time 4 is already occupied. So the computation results are buffered for another one time slice before their transmissions. The transmission resources within Time 4 to Time 5 are occupied for transmitting the results. Note that the propagation delay between Node 2 and Node 3 is one time slice. When the results arrive at Node 3 at Time 6, Node 3 is heavy-load and could only execute the backward task at Time 8. The overall JCT achieved by Greedy is 9 time slices, among which 3 time slices are waiting time.

We can see that the computation result transmission and the task execution are buffered due to improper task mapping and resource allocations. If the interdependency and the availability of multidimensional resources are taken into consideration jointly, we could find the optimal mapping of the DAG and allocate resources properly, shortening the entire JCT.

## C. JTCS Strategy

The ILP model tries to optimize the average JCT with all jobs simultaneously, which will introduce a huge searching space. It is hard to be solved by using optimization software with a large number of jobs. Instead, JTCS adopts the one-by-one optimization policy. The optimal solution of the subsequent job depends on the network state where the optimal resource allocation has been determined for the previous job. In this way, we could obtain suboptimal results in an acceptable time. The following process shows how JTCS finds a mapping for a job. All jobs will be processed one by one in this way.

1. Traversing all feasible task and node mapping combinations;
2. Dijkstra algorithm is applied to obtain the shortest path for the computation result transmission. After the path is determined, wavelengths and MOTSs should be allocated according to the transmission resource requirement;
3. Calculating the corresponding JCT. Selecting the mapping combination with the minimum JCT.

An example of JTCS is presented in Figure 4(b) and (d). In Figure 4(b), the forward and backward tasks are mapped to Node 4 and Node 6, respectively. We could observe that the path between Node 4 and Node 6 (Path 4 -> 5 -> 6) is double the length of that between Node 2 and Node 3 (Path 2 -> 3), where tasks are mapped in Greedy. From Figure 4(d), we learn that the computation results could be transmitted immediately when the forward task execution is finished at Time 3. Note that the propagation delay between Node 4 and Node 6 is 2 time slices, and the result arrive time is Time 6. When the computation results arrive at Node 6 where the backward task is mapped, Node 6 could provide enough computation resources for the task execution right away. No waiting waiting is caused under such circumstances. The total JCT of JTCS is 7 time slices, which is shorter than that of Greedy. Even though the propagation delay in JTCS is one time slice longer than Greedy, JTCS could achieve better performances than the latter.

In conclusion, the method based on minimizing the propagation latency could not get the optimal JCT. In some cases, it will introduce unwanted waiting time due to the lack of available resources. However, in JTCS, the raw data storage location and the distribution of feasible resources are considered comprehensively. The unnecessary waiting time could be avoided, leading to the JCT reduction.

## D. Another Benchmark: Flutter

Flutter is a method proposed in Reference [18]. This method assigns the task to the node where raw data are stored. However, in our simulation settings, only one task requires raw data for each job. For the other tasks that do not demand raw data, we need to design a strategy on how to assign them. So we try to assign these tasks by borrowing the idea from the Greedy method. In this benchmark, the first task of the job will not be assigned randomly. Instead, we will try to assign the first task to every node. Then we apply the Greedy method and see if the task is assigned to the raw data storage node. If so, we will choose this assignment. If not, we will just adopt the assignment that the Greedy method provides. The process of Flutter is as follows.

1. Running topological sorting algorithm to obtain a linear ordering of the DAG;
2. Selecting a node randomly; Running BFS algorithm to obtain a node sequence by taking this node as the root;
3. Mapping the tasks from the linear ordering to the nodes from the node sequence in order; Checking the task is assigned to the node where raw data are stored. If not, turn to Step 2 and select a new node as the root;
4. For the computation result transmission, Dijkstra algorithm is applied to obtain the shortest path. After the path is determined, wavelengths and MOTSs should be allocated according to the transmission resource requirement.

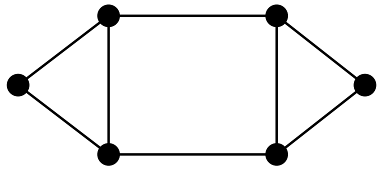
## E. Complexity Analysis

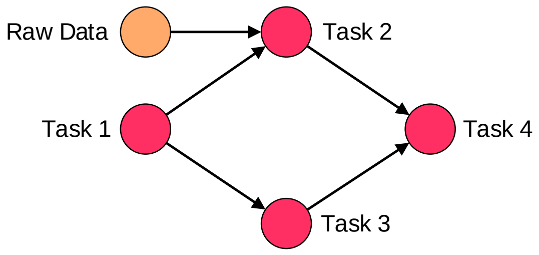
We first discuss the complexity of Greedy. Assuming that each job consists of $M$ tasks, and there are $Q$ computation results generated in a job. The topology contains $N$ nodes and $E$ links. $W$ wavelengths are deployed in each link, and one OTSS frame is divided into $S$ MOTSs. Topological sorting and BFS are running first, whose time complexities are $O(|M|+|Q|)$ and $O(|N|+|E|)$, respectively. These two algorithms are running individually, so the complexity is $O(|M|+|Q|) + O(|N|+|E|) = O(|M|+|Q|+|N|+|E|)$. Now the mapping is determined, and we need to allocate computation and transmission resources. The complexity of the computation resource allocation is $O(|S|)$. As for the transmission resource allocation, Dijkstra algorithm is first applied to obtain the shortest paths of $Q$ computation results, whose complexity is $O(|Q||N|^2)$. Then the complexity of finding continuous MOTSs in $W$ wavelengths is $O(|W||E||S|^2)$. The complexity of allocating computation and transmission resources for a mapping is $O(|S|+O(|Q||W||E||S|^2|N|^2) = O(|Q||W||E||S|^2|N|^2)$. So, the total complexity is $O((|M|+|Q|+|N|+|E|)(|Q||W||E||S|^2|N|^2))$.

Then we analyze the complexity of JTCS. For a given job, mapping $M$ tasks to $N$ nodes at first, which contains $\binom{N}{M}$ mapping combinations. For one of the combinations, the complexity is $O(|Q||W||E||S|^2|N|^2)$ according to previous analysis. So, the total complexity is $O(\binom{N}{M}|Q||W||E||S|^2|N|^2)$.

## 5. SIMULATION EVALUATION IN THE SMALL TOPOLOGY

### A. Simulation Parameters


(a) The small topology.



(b) The job described by the DAG model.

**Fig. 5.** (a) The small topology with 6 nodes and 8 links. (b) The job described by the DAG model.

**Table 3.** Parameters in the small topology

| Parameters | Value |
|---|---|
| the small topology | 6 nodes, 8 links |
| distance between two adjacent nodes | 20 *km* |
| number of wavelengths in each link | 4 |
| length of an OTSS frame | 10 *ms* |
| number of MOTS in an OTSS frame | 100 |
| computation result transmission time ($T_{cotr}$) | 100 / 200 $\mu s$ |
| task execution time ($T_{exec}$) | 100 $\mu s$ |
| raw data transmission time ($T_{ratr}$) | 300 $\mu s$ |
| maximum CPU capacity in each node | 100 |
| maximum RAM capacity in each node | 100 |
| required CPU capacity for a task | 5 - 50 |
| required RAM capacity for a task | 5 - 50 |
| number of jobs | 1 - 600 |

We evaluate the performances in a small mesh topology at first. The topology contains 6 nodes and 8 links, which is depicted in Figure 5(a). Figure 5(b) shows the job model used in the simulation. The job model consists of 4 tasks, one of which requires raw data for execution. It is worth noting that the task that requires raw data is selected randomly, which means Node 1 to 4 are all possible to be chosen and only one node is selected for each job. Other parameters are given in Table 3. The distance between two adjacent nodes is 20 *km*, and the number of wavelengths deployed in each link is 4. The length of an OTSS frame and a MOTS are 10 *ms* and 100 $\mu s$, respectively. In other words, an OTSS frame is divided into 100 MOTSs. The time required by computation result transmission ($T_{cotr}$), task execution ($T_{exec}$), and raw data transmission ($T_{ratr}$) is 100/200 $\mu s$, 100 $\mu s$, and 300 $\mu s$, respectively. In terms of computation resources, the maximum capacities of CPU and RAM are both set to 100. And the required CPU and RAM resources for a task are randomly selected between 5 to 50. Note that these two values are chosen independently. Finally, the number of jobs ranges from 1 to 600.

### B. Number of Jobs: 1 - 15

To verify the correctness of the proposed mathematical model, we use the optimization software CPLEX Studio to find optimal solutions when the number of jobs ranges from 1 to 15. The software is running on the server with 3.5GHz CPU (Intel Core i9-9900X processor) and 64G RAM. Figure 6(a) demonstrates the average JCT under computation result transmission time of 100 $\mu s$ and task execution time of 100 $\mu s$.

The ILP's curve is divided into two parts: Finished and Unfinished part. Optimal results are obtained and drawn in the Finished part. For the Unfinished part, final optimal results are not available due to the long solving time. So the results in the Unfinished part could be regarded as upper bounds. JTCS's performances are almost the same compared with ILP's curve in the Finished part, and better than the upper bounds in the Unfinished part. From the figure, we could learn that the Greedy curve increases almost linearly, while JTCS could realize lower JCT performance than Greedy. Similar outcomes could be observed in Figure 6(b), where $T_{cotr}$ is 200 $\mu s$.

### C. Number of Jobs: 50 - 600

Then we evaluate the performances with a larger number of jobs, i.e., 50 to 600. Here we also show Flutter's curves. In Figure 7(a) and (b), it can be seen that as the number of jobs increases, the average JCT and the proportion of transmission resource utilization are also higher, respectively. The reason is that a longer waiting time is inevitable when finding available resources for more jobs, leading to a higher JCT. And more jobs introduce more computation results and raw data to be transmitted, resulting in a greater proportion of transmission resource utilization.

In Figure 7(a), we find that JTCS realizes the lowest JCT among three algorithms, and JTCS reduces the average JCT by more than 50% compared with the Greedy algorithm. JTCS tries to minimize the JCT by jointly taking the availability of multi-dimensional resources into consideration, which obtains better performances than Greedy and Flutter. For Flutter, it also outperforms Greedy since raw data transmissions are avoided. Assigning the task to the data storage nodes will avoid the raw data transmission, and it also has a higher probability that finding the minimum JCT under such circumstances. The ratio that the task is assigned to the data storage node in Flutter is almost 100%. If fewer transmission resources are occupied under the same conditions, it is easier to find idle transmission resources when transmitting data. In this case, waiting time could be reduced, resulting in a lower JCT for Flutter.

Figure 7(b) presents how many transmission resources are needed under conditions of the different number of jobs. It is worth noting that transmission resources are used for transmitting computation results and raw data. And computation results are identical no matter what methods you apply in our

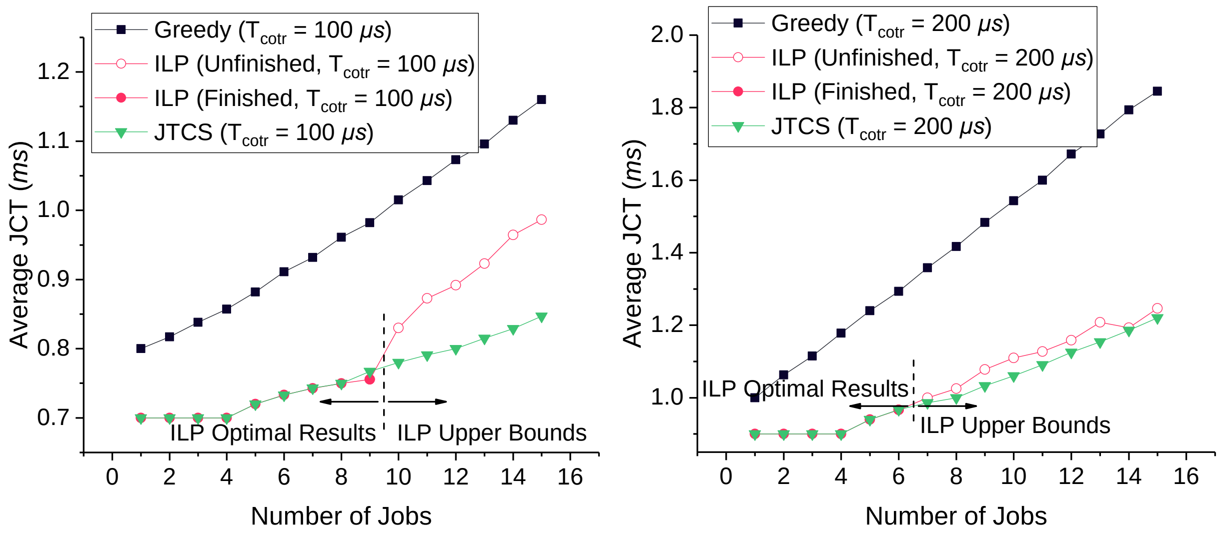


**Fig. 6.** Average JCT calculated by CPLEX Studio under different computation result transmission time ($T_{cotr}$). Other parameters are the same: $T_{exec}$ = 100 μs, $T_{ratr}$ = 300 μs. (a) $T_{cotr}$ = 100 μs. (b) $T_{cotr}$ = 200 μs.

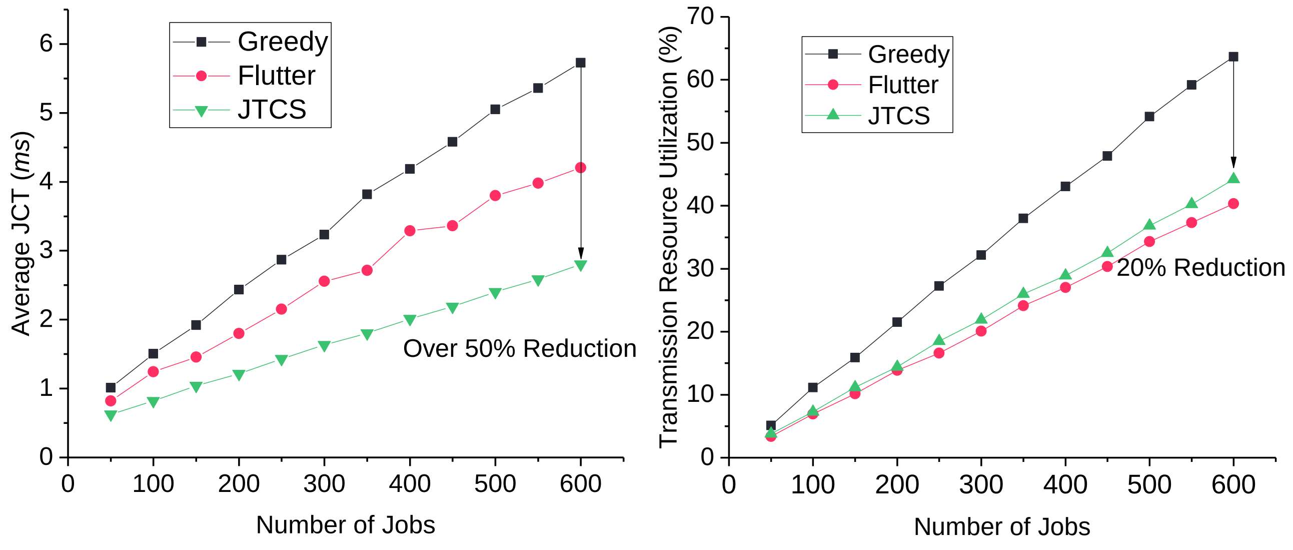


**Fig. 7.** Performances with Greedy, Flutter and JTCS in the small topology. Other parameters: $T_{cotr}$ = 100 μs, $T_{exec}$ = 100 μs, $T_{ratr}$ = 300 μs. (a) Average JCT. (b) Transmission resource utilization.

simulation settings. So the differences in transmission resource utilization mainly come from the different amounts of transmitting raw data.

From Figure 7(b), we learn that more transmission resources are required as more jobs are generated, and JTCS saves about 20% of the resources than Greedy. This is due to the reduction of raw data transmission. There is no surprise that Flutter uses the least transmission resources than JTCS and Greedy since most of the raw data transmissions are avoided. It is worth noting that JTCS will not always assign the task to the node where demanding raw data are located. Instead, it will consider the constraints of multidimensional resources and try to find the lowest JCT. It is because avoiding the raw data transmission does not guarantee the minimum JCT for a job. For example, a task is assigned to the data storages nodes. Unfortunately, this node lacks computation resources at this moment, and the task has to wait for a long time before its execution. If the waiting time is longer than the time consumption in transmitting raw data, it is a better choice to assign the task to another node and transfer the raw data to that node. Overall, JTCS would rather transfer the raw data to obtain a smaller JCT in some cases. That is the reason why it uses more transmission resources than Flutter.

Here we explain why JTCS uses fewer transmission resources than the Greedy algorithm. For all the mapping results that JTCS finds, the ratio that the task is assigned to the data storage node is higher than that of Greedy. That is to say, JTCS realizes the reduction of required transmission resources, though its purpose is to obtain the minimum JCT. As we mentioned in the previous paragraph, reducing the raw data transmission is beneficial to find a shorter JCT.

**Table 4.** Parameters in the large topology

| **Parameters** | **Value** |
|---|---|
| the large topology | 38 nodes, 59 links |
| distance between two adjacent nodes | 20 / 100 *km* |
| number of wavelengths in each link | 16 |
| length of an OTSS frame | 100 *ms* |
| number of MOTS in an OTSS frame | 1000 |
| computation result transmission time ($T_{cotr}$) | 100 - 300 $\mu s$ |
| task execution time ($T_{exec}$) | 100 - 800 $\mu s$ |
| raw data transmission time ($T_{ratr}$) | 100 - 400 $\mu s$ |
| maximum CPU capacity in each node | 80 / 100 |
| maximum RAM capacity in each node | 80 / 100 |
| required CPU capacity for a task | 1 - 10 |
| required RAM capacity for a task | 1 - 10 |
| number of jobs | 100 - 1500 |

## 6. SIMULATION EVALUATION IN THE LARGE-SCALE TOPOLOGY

### A. Simulation Parameters

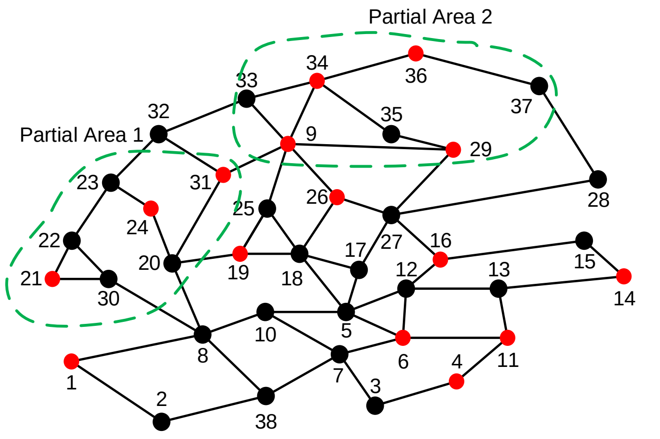


**Fig. 8.** The large topology with 38 nodes and 59 links.

In this section, we verify the effectiveness of JTCS on a large-scale topology. This topology consists of 38 nodes and 59 links, which is depicted in Figure 8. Unlike the previous simulations in the small-scale topology, storage node restrictions are introduced here. Among the 38 nodes, only 15 nodes can store the raw data, which are marked in red in the figure. In other words, the red nodes in the graph could store raw data and process task execution, while the black nodes could only be used for computation.

Due to the large scale of the network topology, it is very difficult for JTCS to find the optimal resource allocation for a job in the entire network. According to Section 4, the complexity is mainly determined by the number of nodes in the topology. To tackle this problem, we narrow the searching space. Only part of the nodes will be considered instead of searching a whole topology. These nodes form partial areas in this paper. This approach is called region dividing method (RDM). When RDM is adopted to allocate resources for a job, all tasks are only mapped in the partial area, and nodes outside the area will be ignored. Then we discuss how to determine a partial area for a job. At first, we search all the nodes and select one with the lowest waiting time for the first task's execution. Usually there are more than one node with the same waiting time, and we just choose one randomly. Then, running BFS algorithm by taking this node as the root. We could obtain a sequence of nodes and select the top $k$ nodes to form a partial area.

In our simulation, $k$ is set to 7. Figure 8 presents two partial areas, namely Partial Area 1 and 2. Partial Area 1 contains 7 nodes, among which Node 21, 24, and 31 are used for both raw data storage and task execution. Node 9, 29, 33, 34, 35, 36, and 37 locate in Partial Area 2, and nodes with the storage capability are Node 9, 29, 34, and 36. Narrowing the searching area reduces the complexity and enhances the availability of RDM.

Other parameters are summarized in Table 4. The distance between two adjacent nodes is set to 20 *km*. In Subsection D, we study the influences by longer physical link length, and the distance is set to 100 *km* for comparison. The number of deployed wavelengths in each link is 16. The OTSS frame length and MOTS length are 100 *ms* and 100 $\mu s$ respectively, meaning an OTSS frame is divided into 1000 MOTSs. As for the parameters of demanding resources, the time for result transmission, task execution, and raw data transmission ranges from 100-300 $\mu s$, 100-800 $\mu s$, and 100-400 $\mu s$, respectively. In terms of computation resources, the maximum capacities of CPU and RAM are both set to 100. The value is changed to 80 for studying the impacts of reducing maximum capacities. The CPU and RAM resources required by a task are randomly chosen between 1 to 10. As mentioned before, The CPU and RAM resources requirement of a task is independent of each other. In the simulation, 100 to 1500 jobs are generated.

### B. Performances under different raw data transmission time

Figure 9 demonstrates the results with the raw data transmission as a variable. D1 and D2 denote that the raw data transmission time is 400 $\mu s$ and 100 $\mu s$, respectively. The computation result transmission time and task execution time are set to 300 $\mu s$ and 300 $\mu s$, respectively. From Figure 9(a), we can find that higher raw data transmission time leads to longer average JCT. This is because higher raw data transmission time leads to a greater probability that the start of the corresponding task execution will be delayed, which affects the JCT of the entire job.

Figure 9(b) shows that larger raw data transmission time will take up more transmission resources, and the performances of RDM are still better than Greedy under the same condition.

### C. Performances under different maximum computation resource capacities

Figure 10 shows the results where the maximum computation resource capacities are 100 and 80 (represented by CR1 and CR2), respectively. The computation result transmission time, task execution time, and raw data transmission time are set to 100 $\mu s$, 100 $\mu s$, and 300 $\mu s$, respectively. Figure 10(a) indicates that the average JCT of Greedy-CR2 is longer than that of Greedy-CR1. And the JCT of RDM-CR2 is longer than that of RDM-CR1, too. This is because the maximum computation resource capacity of CR2 is lower than that of CR1, which may

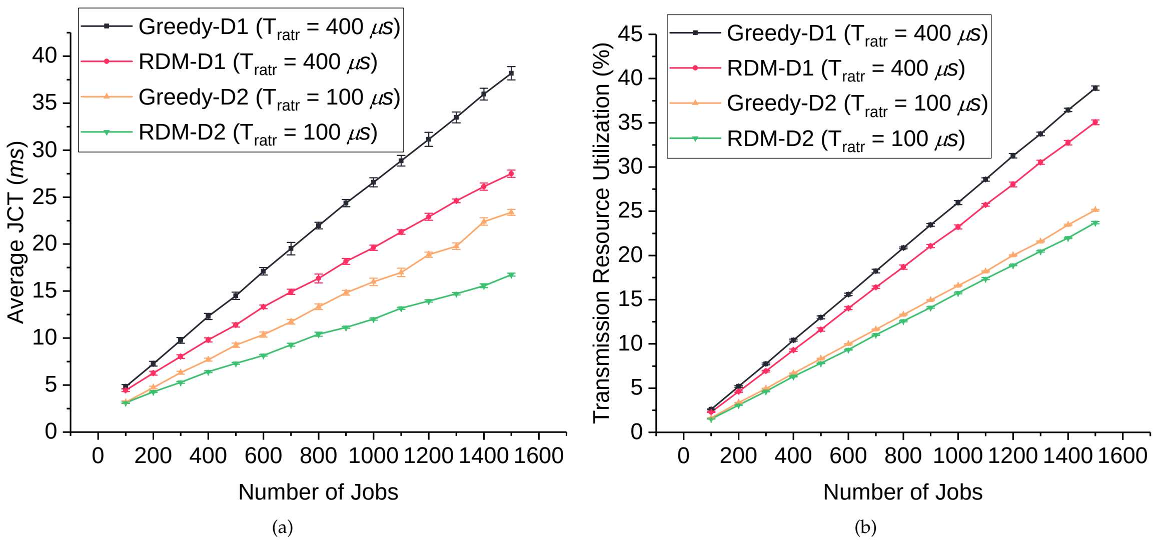


**Fig. 9.** Performances under raw data transmission time 400 and 100 $\mu s$. (a) Average JCT. (b) Transmission resource utilization.

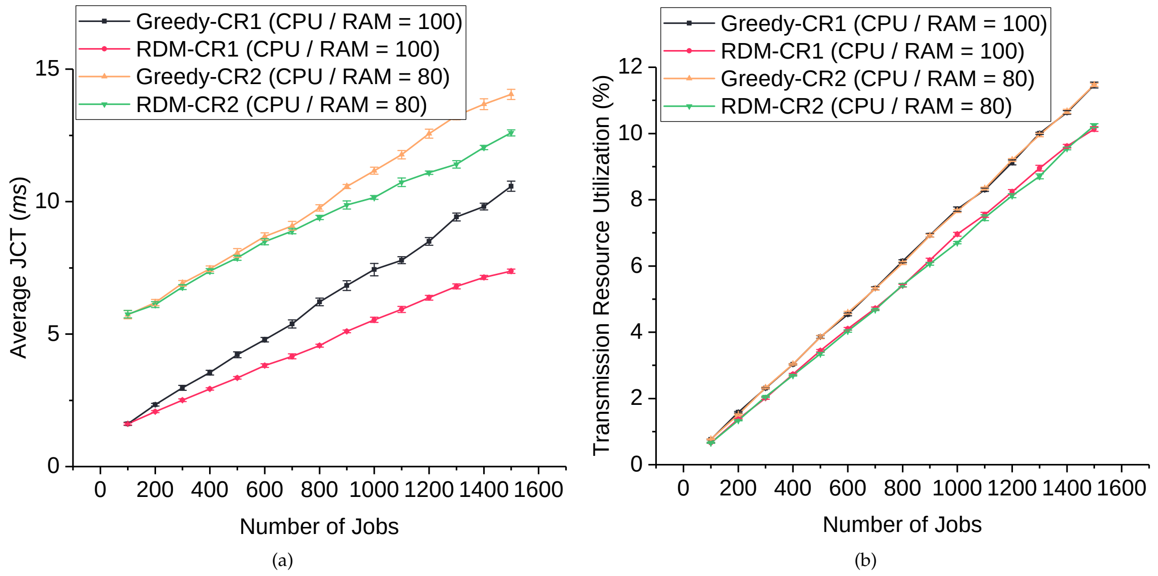


**Fig. 10.** Performances under maximum computation resource capacities 100 and 80. (a) Average JCT. (b) Transmission resource utilization.

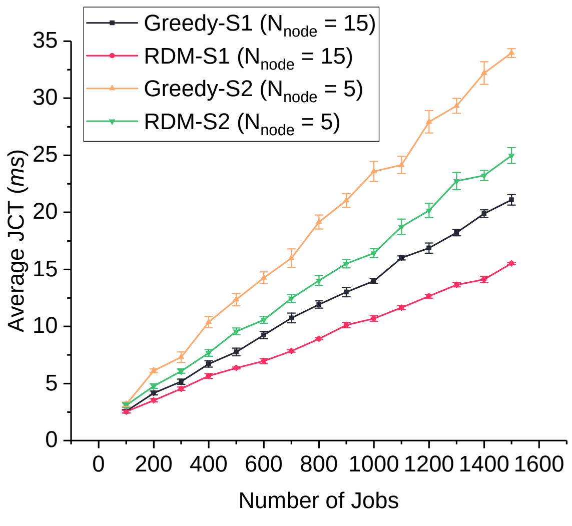
result in a longer waiting time to have available computation resources for task execution. Assuming that the computation resources in each node are infinite, any task can be allocated to the required computation resources without delay, eliminating the waiting time before the start of task execution. However, the computation resources in the edge node are limited. It is particularly critical to allocate resources appropriately to reduce the waiting time.

Furthermore, from Figure 10(a), we could notice that RDM-CR2 is partially overlapped with Greedy-CR2 when the number of jobs is fewer than 600. That is to say, the performances of RDM-CR2 are only slightly better than Greedy-CR2. The reason is that when the amount of computation resources is reduced in each node, i.e., dropping from 100 to 80, the number of solutions that finding a feasible JCT will decrease. Under such circumstances, it is more difficult for RDM-CR2 to find a much lower optimal JCT than the one Greedy-CR2 finds. In addition, the difference between two JCTs obtained by RDM-CR2 and Greedy-CR2 respectively are mainly realized by reducing the waiting time. However, the waiting time induced by the resources shortage is relatively short when the number of jobs is small. So the differences between two JCTs are not significant.

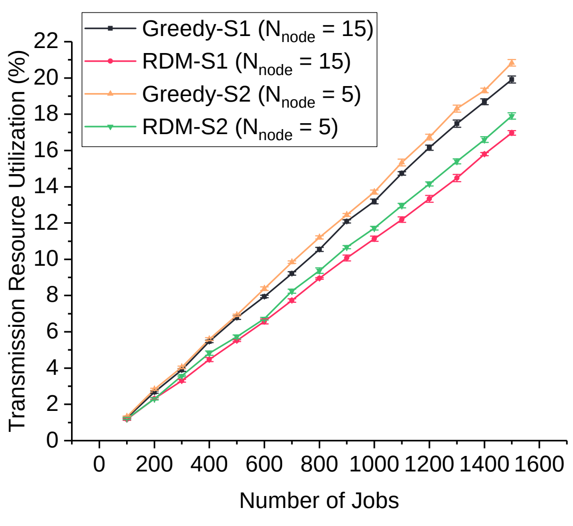
From Figure 10(b), we could see that the curves of Greedy-CR1 and Greedy-CR2 are overlapped, and it is the same case for curves of RDM-CR1 and RDM-CR2. It is because lowering the computation resource capacity will not affect the transmission resource utilization. Only the amount of computation results and raw data will influence the demanding transmission resources.

## D. Performances under different number of raw data storage nodes

We study the performances in Figure 11 under the constraints of the different number of raw data storage nodes. In the pre-

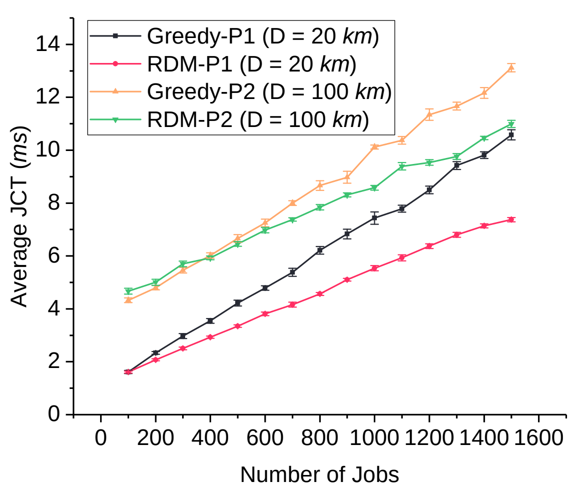


(a)

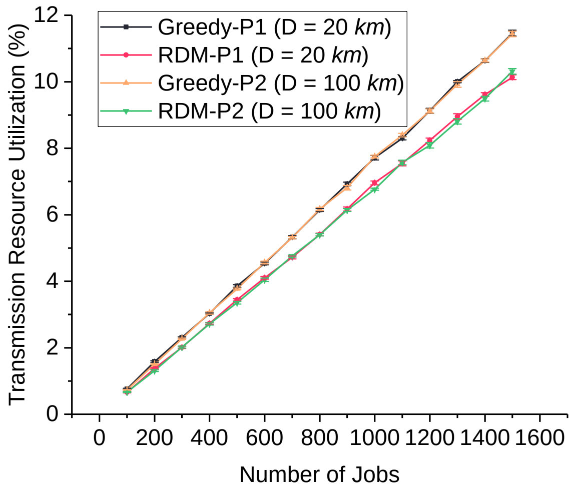


(b)

**Fig. 11.** Performances under number of raw data storage nodes 15 and 5. (a) Average JCT. (b) Transmission resource utilization.



(a)



(b)

**Fig. 12.** Performances under physical link distances 20 and 100 *km*. (a) Average JCT. (b) Transmission resource utilization.

vious simulation settings, the number of nodes used for storing raw data is 15. As a comparison group, we reduce the number to 5. The five storage nodes are Node 1, 9, 16, 24, and 31. The five storage nodes are a subset of the previous 15 nodes. S1 and S2 represent the number of storage nodes are 15 and 5, respectively. As for other simulation parameters, the computation result transmission time, task execution time, and raw data transmission time are set to 100 $\mu s$, 100 $\mu s$, and 300 $\mu s$, respectively. It can be seen from Figure 11(a) that the average JCT of Greedy-S2 is higher than that of Greedy-S1, and it is the same case with RDM-S2 and RDM-S1. Fewer storage nodes will reduce the probability that the task is executed in the node where the required raw data is stored. Under such circumstances, more raw data transmission is needed, resulting in higher average JCT. We can conclude that increasing the number of storage nodes could reduce the amount of raw data transmission and lower the average JCT. However, due to the constraints of the deployment environment and data privacy considerations, not all nodes are suitable for data storage. From Figure 11(a), we could also learn that RDM achieves better results than Greedy.

Figure 11(b) shows that more transmission resources are demanded when the number of nodes for storing raw data is reduced. It is because more raw data transmission is required for the execution of the corresponding task. According to our previous analysis, the amount of raw data transmission will influence the demanding transmission resources.

## E. Performances under different physical link distances

Finally, the performances under different physical link distances are studied here. P1 and P2 denote the distances between two adjacent nodes are 20 $km$ and 100 $km$, respectively. The computation result transmission time, task execution time, and raw data transmission time are set to 100 $\mu s$, 100 $\mu s$, and 100 $\mu s$, respectively. Figure 12(a) shows that RDM-P1 realizes better performances than Greedy-P1 when the distance between two adjacent nodes is 20 $km$. When the distance increases to 100 $km$, the results are pretty interesting. If the number of jobs is less than 400, the average JCT of Greedy-P2 is better than that of RDM-P2. However, The average JCT difference is slight and less than 0.3 $ms$. On one hand, the increment of physical

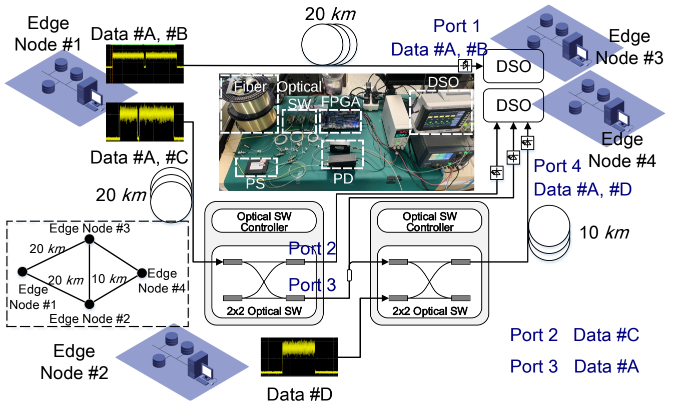


**Fig. 13.** Experiment setup.

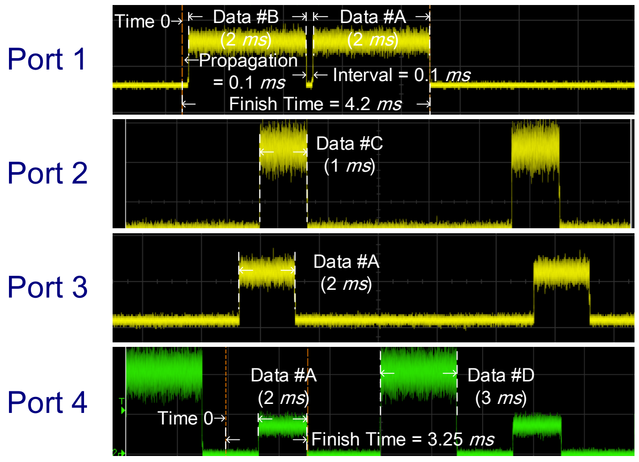


**Fig. 14.** Experimental results.

link distances leads to higher propagation delay. Generally, the optical signals propagate at a speed of 200 $km/ms$. The entire propagation delay will reach several milliseconds if a few physical links are traversed. Greedy is based on searching minimal distances. The longer the physical link distance is, its advantages are more significant. When the number of jobs is small, the available computation and transmission resources are abundant, and the waiting time induced by the shortage of available resources is relatively small. Under such circumstances, the waiting time reduced by the optimization of RDM is relatively tiny. On the other hand, RDM could only search for optimal solutions in a partial area due to the size of the network, which further reduces its advantages. Based on the above two reasons, Greedy-P2 performs slightly better than RDM-P2 when the number of jobs is small. However, as the number of jobs increases, the average JCT of Greedy-P2 exceeds that of RDM-P2. The reason is that as the number of jobs increases, the available resources begin to decrease and the waiting time becomes more prominent. The propagation delay remains unchanged, thus weakening the advantages of Greedy-P2. It is worth noting that the range of the metro network is generally less than 100 *km*, and the distance between two adjacent nodes is smaller than this value. Therefore, Greedy is unlikely to perform better than RDM under real scenarios.

The transmission resource utilization is shown in Figure 12(b). The required transmission resources remain unchanged for different distances since it will not change the amount of computation results and raw data transmission. And RDM realizes less transmission resource utilization than Greedy due to the joint optimization of transmission, computation, and storage resources allocation.

## 7. EXPERIMENTAL VALIDATION

We conduct a proof of concept experiment to validate the effectiveness of our strategy. Four physical nodes, i.e., Node #1 - #4, are emulated in this experiment. The topology contains these four nodes is presented at the left side of Figure 13. The distance between Node #1 and #3 is 20 *km*, while the distances between Node #1 and #2, Node #2 and #4 are 20 *km* and 10 *km*, respectively. The DAG only contains two tasks, forward and backward tasks. We assume that the forward task is mapped to Node #1, and the mapping node for the backward task is determined by Greedy method and JTCS, respectively. In this case, the mapping node for the backward task is Node #3 when adopting Greedy method. And the backward task will be executed in Node #4 when JTCS is applied. We could learn that the distance between Node #1 and #3 (20 *km*) is shorter than Node #1 and #4 (30 *km*). Note that Node #2 is just a bypass node when computation results are transmitted from Node #1 to #4.

Data #A represents the computation results of the forward task, and Data #B - #D indicate computation results from other jobs. As Figure 14 shows, the interval between Data #A and #B is 0.1 *ms*. At time 0, the execution of the forward task is finished and the computation results are waiting for available transmission resources. In Greedy's case, Data #A has to wait for 2 *ms* since Data #B occupies the time slot, as Port 1 in Figure 13 and Figure 14 show. The total time for computation results transmission is 4.2 *ms*, which includes 2 *ms* waiting time, 2 *ms* transmission time, 0.1 *ms* propagation delay and 0.1 *ms* guard interval. As for JTCS's case, Data #C occupies the time slot when the execution of the forward task is finished. However, the length of Data #C is only 1 *ms*. Data #A waits for 1 *ms* and starts transmission. Data #D is added in Node #3, and it does not collide with Data #A in the time domain, as Port 4 shows. The total time for computation results transmission is 3.25 *ms*, which includes 1 *ms* waiting time, 2 *ms* transmission time, 0.15 *ms* propagation delay and 0.1 *ms* guard interval (the guard interval is between Data #A and #C).

From the above experimental results, we could know that the waiting time dominates the total time for computation results transmission. Even though the propagation delay in JTCS is longer than that in Greedy method, JTCS still achieves a shorter time.

## 8. CONCLUSION

In this work, we study the task assignment problem subject to multidimensional resource allocations and interdependency among multiple tasks. To this end, we mathematically formulate the task assignment problem and solve this optimization model by adopting the optimization solver in a small topology. We further design an efficient algorithm by reducing the searching space, which could be implemented in practical networks. Simulation shows that our strategy could achieve lower JCT and higher resource utilization efficiency by wisely assigning tasks to geographically distributed nodes. Its performances and effectiveness are also investigated under various scenarios. A proof of concept experiment is conducted to validate the feasibility of the proposed approach.

In our future work, we will evaluate the strategy under state-of-the-art network technologies, like reconfigurable data center networks. Reconfigurable networks in References [27, 28] could provide much flexibility when allocating multidimensional resources. Combining the joint allocation strategy with reconfigurable network technologies will prove its compatibility and flexibility.

## FUNDING

This work was supported in part by the National Key R&D Program (No.2020YFB1805602), and the National Natural Science Foundation of China (No.61871448).

## DISCLOSURES

The authors declare no conflicts of interest.

## REFERENCES

1. Q. Zhang, L. Cheng, and R. Boutaba, “Cloud computing: State-of-the-art and research challenges,” J. Internet Serv. Appl. **1**, 7–18 (2010).
2. Datacenter Hawk, “Why data centers are moving to the suburbs,” (2015).
3. E. Ahmed, A. Ahmed, I. Yaqoob, J. Shuja, A. Gani, M. Imran, and M. Shoaib, “Bringing computation closer toward the user network: Is edge computing the solution?” IEEE Commun. Mag. **55**, 138–144 (2017).
4. Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, “Mobile edge computinga key technology towards 5G,” ETSI white paper **11**, 1–16 (2015).
5. Gartner, “What edge computing means for infrastructure and operations leaders,” (2018).
6. M. Satyanarayanan, “The emergence of edge computing,” Computer **50**, 30–39 (2017).
7. T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, “Federated learning: Challenges, methods, and future directions,” IEEE Signal Process. Mag. **37**, 50–60 (2020).
8. B. Shariati, P. Safari, A. Mitrovska, N. Hashemi, J. Fischer, and R. Freund, “Demonstration of federated learning over edge-computing enabled metro optical networks,” in *2020 European Conference on Optical Communications (ECOC),* (IEEE, 2020), pp. 1–4.
9. W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” IEEE Internet Things J. **3**, 637–646 (2016).
10. C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, “Serendipity: Enabling remote computing among intermittently connected mobile devices,” in *Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing,* (2012), pp. 145–154.
11. J. Li, N. Hua, C. Zhao, Y. Li, X. Zheng, and B. Zhou, “Towards low-latency distributed tasks collaboration by joint optimization of transmission, computation and storage resources allocation in edge computing,” in *Asia Communications and Photonics Conference,* (Optical Society of America, 2020), pp. M4A–210.
12. S. Andreev, O. Galinina, A. Pyattaev, J. Hosek, P. Masek, H. Yanikomeroglu, and Y. Koucheryavy, “Exploring synergy between communications, caching, and computing in 5G-grade deployments,” IEEE Commun. Mag. **54**, 60–69 (2016).
13. H. Xing, L. Liu, J. Xu, and A. Nallanathan, “Joint task assignment and resource allocation for D2D-enabled mobile-edge computing,” IEEE Transactions on Commun. **67**, 4193–4207 (2019).
14. Y.-H. Kao, B. Krishnamachari, M.-R. Ra, and F. Bai, “Hermes: Latency optimal task assignment for resource-constrained mobile computing,” IEEE Transactions on Mob. Comput. **16**, 3056–3069 (2017).
15. J. Zhang, L. Cui, Z. Liu, and Y. Ji, “Demonstration of geo-distributed data processing and aggregation in mec-empowered metro optical networks,” in *2020 Optical Fiber Communications Conference and Exhibition (OFC),* (IEEE, 2020), pp. 1–3.
16. Y. Sahni, J. Cao, and L. Yang, “Data-aware task allocation for achieving low latency in collaborative edge computing,” IEEE Internet Things J. **6**, 3512–3524 (2018).
17. H. Yang, J. Zhang, Y. Zhao, J. Han, Y. Lin, and Y. Lee, “Cross stratum optimization for software defined multi-domain and multi-layer optical transport networks deploying with data centers,” Opt. Switch. Netw. **26**, 14–24 (2017).
18. Z. Hu, B. Li, and J. Luo, “Flutter: Scheduling tasks closer to data across geo-distributed datacenters,” in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications,* (IEEE, 2016), pp. 1–9.
19. Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, “A survey on mobile edge computing: The communication perspective,” IEEE Commun. Surv. & Tutorials **19**, 2322–2358 (2017).
20. P. Hu, H. Ning, T. Qiu, Y. Zhang, and X. Luo, “Fog computing based face identification and resolution scheme in internet of things,” IEEE transactions on industrial informatics **13**, 1910–1920 (2016).
21. P. Mach and Z. Becvar, “Mobile edge computing: A survey on architecture and computation offloading,” IEEE Commun. Surv. & Tutorials **19**, 1628–1656 (2017).
22. N. Semmler, M. Rost, G. Smaragdakis, and A. Feldmann, “Edge replication strategies for wide-area distributed processing,” in *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking,* (2020), pp. 1–6.
23. C. Wang, Y. He, F. R. Yu, Q. Chen, and L. Tang, “Integration of networking, caching, and computing in wireless systems: A survey, some research issues, and challenges,” IEEE Commun. Surv. & Tutorials **20**, 7–38 (2017).
24. N. Hua and X. Zheng, “Optical time slice switching (OTSS): An all-optical sub-wavelength solution based on time synchronization,” in *Asia Communications and Photonics Conference,* (Optical Society of America, 2013), pp. AW3H–3.
25. J. Li, N. Hua, Z. Zhong, Y. Yu, X. Zheng, and B. Zhou, “Flexible low-latency metro-access converged network architecture based on optical time slice switching,” IEEE/OSA J. Opt. Commun. Netw. **11**, 624–635 (2019).
26. Y. Zhang and M. Xie, “A more accurate delay model based task scheduling in cellular edge computing systems,” in *2019 IEEE 5th International Conference on Computer and Communications (ICCC),* (IEEE, 2019), pp. 72–76.
27. X. Xue, F. Yan, K. Prifti, F. Wang, B. Pan, X. Guo, S. Zhang, and N. Calabretta, “ROTOS: A reconfigurable and cost-effective architecture for high-performance optical data center networks,” J. Light. Technol. **38**, 3485–3494 (2020).
28. X. Guo, X. Xue, F. Yan, B. Pan, G. Exarchakos, and N. Calabretta, “DACON: A reconfigurable application-centric optical network for disaggregated data center infrastructures,” J. Opt. Commun. Netw. **14**, A69–A80 (2022).