IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 23 October 2022, accepted 31 October 2022, date of publication 4 November 2022, date of current version 9 November 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3219873

RESEARCH ARTICLE

JHTD: An Efficient Joint Scheduling Framework Based on Hypergraph for Task Placement and Data Transfer Across Geographically Distributed Data Centers

CHAO JING[1,2,3] AND PENGGAO DAN[1,2]
[1]School of Information Science and Engineering, Guilin University of Technology, Guilin 541004, China
[2]Guangxi Key Laboratory of Embedded Technology and Intelligent System, Guilin University of Technology, Guilin 541004, China
[3]Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic and Technology, Guilin 541004, China

Corresponding author: Chao Jing (jingchao@glut.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61802085 and Grant 62262011; in part by the Guangxi Provincial Natural Science Foundation, China under Grant 2020GXNSFAA159038 and Grant 2021GXNSFAA220074; and in part by the Guangxi Key Laboratory of Trusted Software Foundation, China, under Grant kx202011.

ABSTRACT As the explosive growth of the data volume, data center is playing a critical role to store and process huge amount of data. Traditional single data center can no longer to adapt into incredibly fast-growing data. Recently, some researches have extended the tasks such data processing to geographically distributed data centers. However, since the joint consideration of task placement and data transfer, it is complex and difficult to design a proper scheduling approach with the goal of minimizing makespan under the constraint of task dependencies, processing capability and network, etc. Therefore, our work proposes *JHTD*: an efficient joint scheduling framework based on hypergraph for task placement and data transfer across geographically distributed data centers. Generally, there are two crucial stages in *JHTD*. Initially, due to the outstanding of hypergraphs in modeling complex problems, we have leveraged a hypergraph-based model to establish the relationship between tasks, data files, and data centers. Thereafter, a hypergraph-based partition method has been developed for task placement within the first stage. In the second stage, a task reallocation scheme has been devised in terms of each task-to-data dependency. Meanwhile, a data dependency aware transferring scheme has been designed to minimize the makespan. Last, the real-world model China-VO project has been used to conduct a variety of simulation experiments. The results have demonstrated that *JHTD* effectively optimizes the problems of task placement and data transfer across geographically distributed data centers. *JHTD* has been compared with three other state-of-the-art algorithms. The results have demonstrated that *JHTD* can reduce the makespan by up to 20.6%. Also, various impacts (data transfer volume and load balancing) have been taken into account to show and discuss the effectiveness of *JHTD*.

INDEX TERMS Big data processing, geographically distributed data centers, joint scheduling framework, hypergraph, task placement, data transferring.

I. INTRODUCTION

With the advent of Big Data era, the rate of data generation is dramatically increasing. For example, Internet giants such as Google and Facebook crunch more than 10 PB of data a day [1]. As a result, it is essential to improve the efficiency of data processing in the face the huge amount of data. MapReduce [2] and Spark [3] have been widely adopted to deal with large amounts of data. These frameworks usually process data analytic jobs characterized by data-dependency awareness. These jobs can be divided into a set of dependent tasks. The execution of a task not only requires the outcome

The associate editor coordinating the review of this manuscript and approving it for publication was Alberto Cano.

of the parent tasks, but also the data. Normally, the data and tasks are stored within a data center, therefore the computing node selection is the only consideration; it is unnecessary to consider data transfer. However, due to the increase in the amount of data, the data is distributed between multiple data centers rather being kept in a single data center. Therefore, a joint optimization problem of task placement and data transfer has to be addressed to maximize performance.

Geographically distributed data centers (GDDCs) can be applied to improve the efficiency of processing large-scale jobs. When using GDDCs, several issues need to be considered. First, because the network bandwidth is the bottleneck for data transfer, the heterogeneity of link bandwidth must be taken into account during data transfer. Selecting suitable links to data centers can greatly reduce data transfer time. Second, the computing capability of each data center may vary; the higher the capability, the shorter the job processing time. Last, the privacy of data at each data center should be considered. Data transfer must obey the local legislation.

In this situation, literature that aims to improve the performance of GDDCs can be generally categorized into two groups. One attempts to reduce the size of data transfer across the data centers [4], [5], [6]. However, this group of studies cannot really enhance the efficiency of job processing. Since most of the data centers are heterogeneous, the network transmission time not only depends on the amount of data, but also on the link bandwidth [7], [8], [9]. As a result, solely limiting to amount of data transferred cannot improve the performance of GDDCs. The other group [10], [11] focuses on developing sophisticated strategies for task placement. Data analysis consists of multiple data-dependency tasks, whose processing time or energy cost [12] can be shortened by appropriately allocating the tasks. Different task placement strategies lead to different results. By far, previous studies have either focused on limiting the size of the transferred data or only focused on the placement of tasks.

In contrast to these studies, with the objective of optimizing performance via GDDCs, our study tackles the joint optimization problem of task placement and data transfer. To achieve this objective, we proposed a joint scheduling framework that considers task placement and data transfer based on a hypergraph (*JHTD*). We strictly formulated the joint optimization problem of task placement and data transfer between GDDCs. Since the key elements in our problem are data, tasks and data centers, we employed a hypergraph-based model to connect these elements. As a generalization of a graph, a hypergraph can connect multiple vertices with one hyperedge. Assigning data, tasks, data centers and other key elements to the vertices and hyperedges of the hypergraph can clearly express their dependencies, which helps us to better handle problems. So, we developed a two-stage scheduling framework (*JHTD*). The first stage was to devise a hypergraph-based partition method for task placement (*HPTP*) to minimize the amount of data transferring across data centers based on the various computing capabilities of data centers. The second stage was to devise a task allocation scheme to re-allocate tasks in terms of the lowest dependency (*TALD*) by computing the task-to-data dependency. Simultaneously, a data aware method with the highest dependency transferring scheme (*DHD*) was designed to minimize the data transfer completion time. Finally, we evaluated the performance of *JHTD* by comparing it with three other state-of-art algorithms.

The main contributions of our study are summarized as follows:

- We formulated a joint optimization problem of task placement and data transfer between GDDCs. To connect the data files, tasks, and data centers, we introduced and established a hypergraph-based model. The model of the distributed data center, data file, job, and network are setup, and the problem statement is given.
- We proposed a joint scheduling framework for task placement and data transfer that optimizes makespan. The proposed framework can be divided into two stages. At begin an initialized task placement problem is solved using a hypergraph partition method. Later, the tasks and data files are reallocated and transferred to the other data centers based on the awareness of dependency, accordingly.
- We performed experiments using real-world configurations from the China-VO project [13]. The proposed *JHTD* was compared with three other well-known algorithms: *Greedy*, *Hypegraph* [14], and $Fast-Newman$ [15]. The results show that *JHTD* has better performance in terms of data transfer volume and load balancing, and can reduce the makespan by 20.6% at most. Meanwhile, we also discussed and verified the effectiveness of *JHTD* under various impacts.

The remainder of this paper is organized as follows: in the next section, give the preliminary knowledge of hypergraph. We then review the current studies on optimizing performance of GDDCs. Section III formulates the joint problem of task placement and data transfer between GDDCs. Details of the models are also given. Section IV elaborates the proposed framework *JHTD*. Section V shows the evaluation and comparison results. Section VI discusses the performance of *JHTD* under various impacts. Section VII concludes our study and suggests future research prospects.

II. PRELIMINARY AND RELATED WORK

In this section, we give brief knowledge of hypergraphs, then we review three classes of previous work: Task-oriented scheduling, data-aware scheduling and hypergraph-based scheduling.

A. PRELIMINARY TO HYPERGRAPH

A hypergraph can be defined as a graph generalization, which is a discrete structure in finite sets. Unlike ordinary graphs where an edge can only connect two vertices, an edge in a hypergraph can connect any number of vertices. Consequently, the hypergraph can represent more

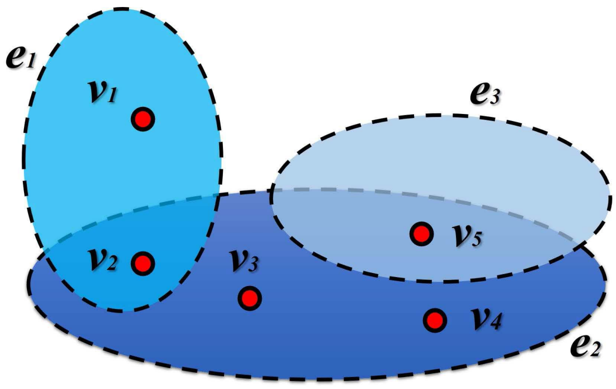


FIGURE 1. Illustration of a hypergraph, where $\mathcal{V} = \{v_1, v_2, v_3, v_4, v_5\}$ and $\mathcal{E} = \{e_1, e_2, e_3\}$. e_1 connects v_1 and v_2, while e_2 connects v_2, v_3, v_4, and v_5. At the same time, v_5 is also the only vertex of e_3.

complex relationships between vertices and edges. Due to its advantages, the hypergraph has many real-world applications in fields such as physics, chemistry, biology, and computer science. Hypergraphs have been extensively used in tasks including quantum entanglement [16], predicting molecular models [17], selecting genes [18], [19], predicting diseases [20]. In addition, hypergraph learning has also been used to build social networks [21], design recommendation systems [22], and [23] and [24] used it to extend graph neural network. Especially in the field of computer vision, where hypergraphs are used for person re-identification [25], hyperspectral image classification [26], and visual tracking [27]. Some studies have also applied hypergraph to high-performance computing [28] and task scheduling [28], [29], [30].

Figure 1 illustrates a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ with a two-tuple, where $\mathcal{V}$ is a set of the vertices, $\mathcal{E}$ consists of a set of hyperedges. Hyperedges e_1, e_2 and e_3 contain 2, 4, and 1 vertices, respectively, and v_2 overlaps with e_1 and e_2, so does v_5 with e_2 and e_3. The hypergraph can describe a complicated model, so it is adopted to define the joint optimization problem in our study.

B. RELATED WORK

In the framework of MapReduce and Spark, data analysis jobs typically consist of multiple dependent tasks. The execution of task requires data, as well as the data from its parent task according to the precedence. Several studies improved the performance of data analysis using the technique of scheduling. Based on the importance of data and tasks, we generally divide these works into three categories: task-oriented scheduling algorithms, data-aware scheduling algorithms, and hypergraph-based task scheduling algorithms.

1) TASK-ORIENTED SCHEDULING ALGORITHMS

The central idea of task-oriented scheduling algorithm is to find a way to allocate tasks to servers (computing nodes) that minimizes the completion time, which is a single-objective optimization problem. To address this problem, there are many classical approaches such as FIFO, greedy and maximum completion time (MCT [31]). Some heuristic algorithms can be exploited as Max-Min [32], Min-Min [33], and heterogeneous earliest finish time (HEFT), critical path on a processor (CPOP). Moreover, some traditional intelligent algorithms such as genetic, particle swarm, and ant colony [34] are adopted to find the optimal completion time. Not only that, but there are also some works that improve these algorithms, making them applicable to more scheduling scenarios. For example, [35] improved genetic algorithms through accelerating and optimizing the evolution processes, [36] presented an improved genetic algorithm by introducing the spectrum partitioning algorithm, and [37] proposed an adaptive granularity learning distributed particle swarm optimization (AGLDPSO) with the help of machine-learning techniques. However, these algorithms are straight-forward and confined to solving simple scheduling scenarios, and their results are random and susceptible to parameters. Since tasks with no precedence are naive and impractical, they neglect the factors of data-dependency on tasks. As the number of tasks increases, the traditional intelligent algorithms have a problem of being trapped in local optima with a lower convergence speed.

2) DATA-AWARE SCHEDULING ALGORITHM

Unlike task-oriented scheduling algorithms, data-aware scheduling algorithms consider the feature of data in improving performance [38]. Casanova et al., exploited file data locality to compute the cluster sufferage value. The higher value of sufferage means that it is needed for more tasks. To effectively reduce makespan, data files with high sufferage value have the priority to transfer [39]. Hu et al., devised a task scheduling algorithm titled Flutter [11]. To eliminate the impact of data movement, Flutter focuses on scheduling tasks to data between GDDCs to reduce the completion time. The close-to-files algorithm [40] schedules tasks with file data replication transfer to the least-loaded data center to balance the loading. Similarly, Convolbo et al., presented a data-replication aware scheduler [41] (DRASH), which partially replicates file data to data centers. Clearly, the advantage of this approach is that it generates less cost in terms of data transfer. Researchers in [42] formulated the issue of task scheduling as a community detection problem. By considering the dependencies between tasks, file data, and data centers, they designed a community-detection-based scheduling (CDS) algorithm. CDS relies on the task-to-data center relation to compute the value modularity that ignores the file data transferring cost.

Although data-aware scheduling algorithms benefit performance, scheduling scenarios have been simplified. Most studies consider a static scheduling condition, avoiding data movement between data centers. But, for online scheduling, it is difficult to prevent data transfer. Otherwise, the dependent tasks cannot be successfully executed.

3) TASK SCHEDULING ALGORITHM BASED ON HYPERGRAPH

Since the task, data, network link, and data center are four key factors in our joint optimization problem, to connect these three elements, many studies adopted the hypergraph method. For example, Ankita et al. used hypergraph to solve the data placement problem in geographically distributed clouds [43]. And Li et al. employed a hypergraph to divide the workflow, and scheduled tasks using the Dijkstra algorithm based on the Fibonacci heap [44]. The study [45] devote to task-placement-based hypergraph partitioning approach. The proposed algorithm can efficiently partition data items and place them in distributed data centers, but it has a limitation in data transfer. Yang et al. improved the traditional vertex-cut hypergraph partition into a net-cut hypergraph partition [46]; it effectively balanced the partitioning results but it is not suitable for large-scale data. In addition, [28] and [47] used hypergraphs to reduce data transfer to shorten the overall makespan but they neglected the actual execution of the task. From the above work, it is evident that the deficiency of current studies is that they simply integrate the idea of the hypergraph or cannot be applied to problem optimization in geographically distributed data centers. By adopting the hypergraph partition method, a task placement solution can be eventually obtained.

Besides, Yuen et al. improved the particle swarm algorithm [48], [49] and applied it to a signalized traffic problem to optimize the average vehicle delay and stop frequency. However, such algorithm gains optimized results that rely on the initialized parameters selection and tuning. Akbar et al. studied various limitations of tasks in different scenarios and optimized the network performance [50], [51]. Instead, for the geographically distributed data centers, tasks placement to the appropriate data center is not the only issue, more factors to be concerned data dependency, data transfer and processing capability on each data center.

Therefore, due to the defects of the current researches on GDDCs, it is necessary to propose a sophisticated algorithm that fully covers the joint optimization issues of task placement and data transfer.

III. PROBLEM FORMULATION

In this section, we formally describe the joint optimization problem. First, we set up the model of the system and job. Note that the job model has been extended with hypergraph to connect task, data transfer, and geographically distributed data centers. We then formally state the joint optimization problem for task placement and data transferring across GDDCs. To clarify the formulation, Table 1 summarizes some of the symbols and notation used in this article.

A. SYSTEM MODEL

There are $|M|$ number of geographically distributed data centers owned by a service provider, which can be represented as $C = \{C_1, C_2, \ldots, C_M\}$. Because of the heterogeneity

TABLE 1. Symbols and Notations used in this article.

Notation	Definition
$\|M\|$	The number of geographically distributed data centers.
$C = \{C_1, C_2, ..., C_M\}$	Data center set.
$P = \{P_1, P_2, ..., P_M\}$	CPU processing speed set.
P_j	CPU processing speed of data center $j, j \in M$.
$R_{(C_p,C_q)}$	Capacity of link $C_p, C_q, \forall p, q \in M$.
$\|F\|$	The total number of data files.
$f = \{f_1, f_2, ..., f_F\}$	Data file set.
f_i	Data file $f_i, i \in F$.
$S(f_i)$	The size of a data file f_i.
N_t	The number of jobs arriving at a certain time t.
$J = \{J_1, J_2, ..., J_{N_t}\}$	Job set.
N_k	The total number of tasks.
$T = \{T_1, T_2, ..., T_{N_k}\}$	Task set.
ET_l	The execution time of task T_l.
te_l	The size of task T_l.
WT_l	The waiting time of task T_l.
TA_l	The arrival time of task T_l.
TS_l	The starting time of task T_l.
$S(f_{i,l})^{C_1}$	The size of data file f_i required by Task T_l on data center C_1.
$N_{f_{i,l}}^{C_p}$	The number of data files f_i required by task T_l on data center C_p.
CDT_l	Cost of data file transferring time of task T_l.
CT_l	The completion time of task T_l.
$\mathcal{H} = (\mathcal{V}, \mathcal{E})$	Data center-Task-Data file relationship under hypergraph modeling.
$v_n, n \in \|M\| + \|N_k\|$	v_n is a vertex in the hypergraph $\mathcal{H}$.
$e_i, i \in F$	e_i is a hyperedge in the hypergraph $\mathcal{H}$.
λ_{e_i}	The number of partitions connected by hyperedge e_i.
$W_{C_l}, C_l \in C$	The weight for vertices.
$\bar{W}_C$	Average weight.
α	A scaling factor to tune the $\bar{W}_C$.
v^{us}	Unselected vertex.
v_{near}^{us}	Adjacent vertex of unselected vertex.
f_{max}^{l}	The largest data file in $items$.

of data centers, the processing capability of each data center is different from the other and is denoted by $P = \{P_1, P_2, \ldots, P_M\}$. These data centers are managed by a central controller and connected with heterogeneous network link. The bandwidth on each link is denoted as $R_{(C_p,C_q)}$, where $p, q \in M$. The data is encapsulated as a data file stored in data center. Assuming that the total number of data files is $|F|$, and formulated as $f = \{f_1, f_2, \ldots, f_F\}$. The size on each data file can be calculated by $S(f_i)$, $i \in F$. The system model is shown in Figure 2. When jobs are submitted, the central controller preprocesses the job and extracts the tasks from these jobs for task placement by a task placer. The data files are stored in data centers in advance. The main function of the task placer and file controller is to place tasks and transfer the data files to respective data centers.

B. JOB MODEL

Suppose at a certain time t, there are N_t number of jobs arriving, defined as $J = \{J_1, J_2, \ldots, J_{N_t}\}$. The job is presented as a direct acyclic graph (DAG) in Figure 3. Each vertex and edge represents a task and relationship, respectively. From the root task of T_1 to the last task T_7, the number of depth is 3 according to the precedence. Note

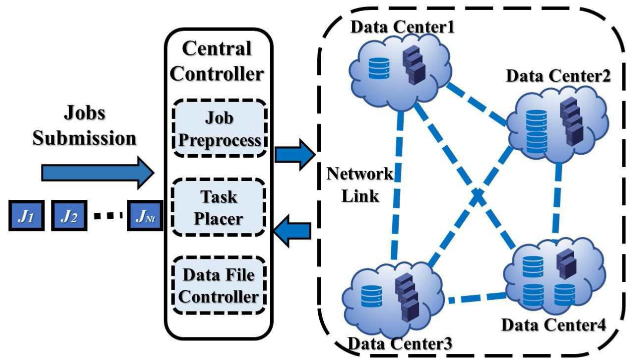


FIGURE 2. Illustration for System model. It consists of two parts, central controller and geographically distributed data centers, where the central controller contains job preprocess, task placer and data file controller.

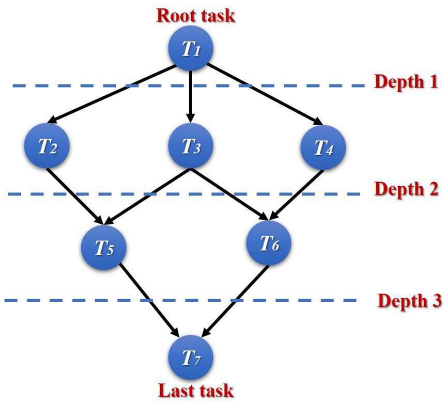


FIGURE 3. Illustration of direct acyclic graph (DAG) for a job. It represents a job containing 7 tasks, with three levels of depth, and tasks of the same depth have the same priority.

that the jobs are independent, but the tasks within each job are dependent on precedence and required data files, where the set of N_k task on $J_k, k \in N_t$ can be noted as $T = \{T_1, T_2, \ldots, T_{N_k}\}$. As the submitted jobs must be preprocessed by the central controller, each task is extracted with a size denoted as $te_l, l \in N_k$, where the size of each task includes two segments: the program file owned by task and the subset of data files from f required by T_l. The task T_l completion time CT_l consists of three parts: execution time ET_l, waiting time WT_l, and cost of data file transfer time CDT_l. Suppose that the task T_l is placed onto data center j with computing capability P_j, so, we can define (1) to calculate the execution time. ET_l is calculated as follows:

$$ET_l = \frac{te_l}{P_j} \tag{1}$$

The waiting time of T_l is the task arrival time TA_l, subtracted from the task starting process time TS_l so the task T_l waiting time can be defined as (2). WT_l can be calculated as follows:

$$WT_l = TS_l - TA_l \tag{2}$$

where TA_l is the arrival time of a job that embraces the task T_l.

As the task execution needs the data file, the data file transferring time for task T_l is computed using the size of data file required by T_l and bandwidth $R_{(C_p,C_q)}$. So, the cost of data file transferring time(CDT) can be defined as (3)

$$CDT_l = Max\left\{\frac{\sum_{i\in F} S(f_{i,l})^{C_1})}{R_{(C_1,C_q)}}, \frac{\sum_{i\in F} S(f_{i,l})^{C_2})}{R_{(C_2,C_q)}}, \ldots, \frac{\sum_{i\in F} S(f_{i,l})^{C_p})}{R_{(C_p,C_q)}}\right\} \tag{3}$$

where task T_l is allocated on the data center C_q, and the required data files are distributed on data center $\{C_1, C_2, \ldots, C_p\}$ with bandwidth $R_{(C_1,C_q)}, R_{(C_2,C_q)}, \ldots, R_{(C_p,C_q)}$, respectively. $S(f_{i,l})$ is to calculate the size of data file f_i required by T_l, and $S(f_{i,l})^{C_1}$ is the data files located on data center C_1. Because of the multi-port transferring model we used on the data center, the maximum data file transfer time is selected for CDT_l.

Therefore, the completion time of the task T_l is defined as (4),

$$CT_l = ET_l + WT_l + CDT_l \tag{4}$$

C. HYPERGRAPH MODEL

Through the description of system model and job model above, we observe that there is a complicated relationship between data center, task, and data file, respectively; ordinary graphs have limitations in accurately reflecting their connection and dependency. Therefore, based on the aforementioned models, we made further extension with respects to hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$. Figure 4 shows the difference between a normal graph and a hypergraph. From Figure 4(a), 4(b) data files f_1 and f_2 are distributed on data centers C_1 and C_2, respectively. And the task T_1 has a data-dependency on f_1 and f_2. Instead, while leveraging the hypergraph, these relationships can be explicitly represented in Figure 4(c). Under the hypergraph partition, each hyper edge f_1 and f_2 can only include one data center.

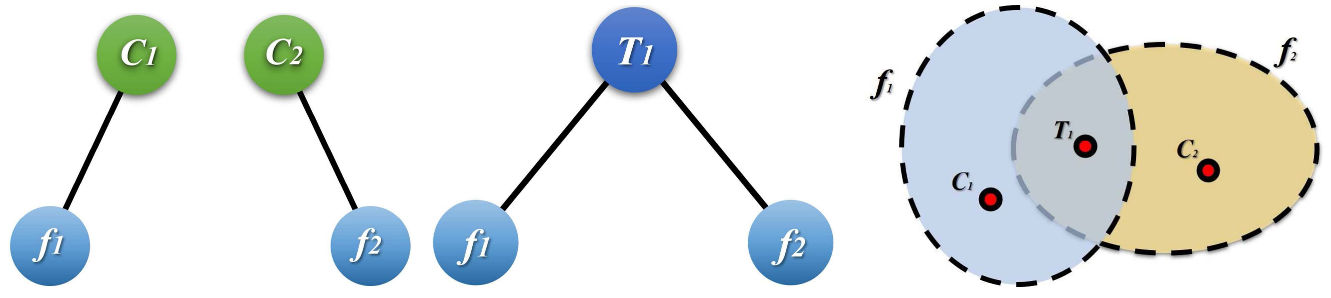
To guarantee the results mentioned above, we define the weight for vertices and hyper edge for the hypergraph partition. We set an initialized weight $W_{C_l}, C_l \in C$ on each data center. As (5), $\bar{W}_C$ is to compute the averaging weight of the data center. It is the sum of weights on each data center $W_{C_l}(C_l \in C)$ that divides the total number of data centers $|M|$, where α is a scaling factor to tune the $\bar{W}_C$.

$$\bar{W}_C = \frac{\sum_{C_l\in C} W_{C_l}}{|M| \times \alpha} \tag{5}$$

Then, the weight on each data center is recomputed as (6)

$$W'_{C_l} = \frac{W_{C_l}}{\bar{W}_C} \tag{6}$$

The heterogeneity of data center, W'_{C_l} is updated based on the processing capability. The basic updating rule is



(a) Relationship between data center and data file by ordinary graph (b) Relationship between task and data file by ordinary graph (c) Relationship between data center, data file, and task by Hypergraph

FIGURE 4. Illustration of difference between ordinary graph and hypergraph. (a) Indicates that data files f_1 and f_2 are located on data centers C_1 and C_2, respectively. (b) Indicates that task T_l requires data files f_1, f_2. (C) The above relationship is characterized using a hypergraph.

that the processing capability $P_j, j \in M$ in decreasing order is one-by-one assigned weights from W'_{C_l} in increasing order. When the updating is completed, the maximum weight must not be more than the sum of two least weights, otherwise the α must be tuned. Only that way can ensure that there is only one data center within a hyper edge.

D. PROBLEM STATEMENT

While submitting multiple jobs J at a certain time t, each job $J_k, k \in N_k$ constitutes numerous tasks, the makespan of a job is the time the last task is finished. The joint optimization problem of task placement and data transfer *JOP* can be stated as following:

Definition 1: with the system model, job model, and hypergraph-based model mentioned above, the objective is to find the minimum makespan of each job without compromising of the deadline $\mathcal{D}_{J_k}$ considering limited bandwidth, computing ability on each network link, and data center, respectively. Suppose that the makespan of J_k is symbolized as Γ_{J_k}, so the minimizing of Γ_{J_k} can be formally defined as (7a),

$$\begin{aligned}&\textit{Objective} \min\textit{imize}:\\ &\Gamma_{J_k} = \max\{CT_1^1, CT_2^1, \ldots, CT_{L1}^1\} + \max\{CT_1^2, CT_2^2, \\ &\quad \ldots, CT_{L2}^2\} + \ldots + \max\{CT_1^{\mathcal{L}}, CT_2^{\mathcal{L}}, \ldots, CT_{Ln}^{\mathcal{L}}\}\end{aligned} \tag{7a}$$

$$s.t., \quad \Gamma_{J_k} \leq \mathcal{D}_{J_k} \tag{7b}$$

$$\sum_{l \in N_t} x_{l,q} = 1, \textit{where } q \in M, x_{l,q} = 0 \textit{ or } 1 \tag{7c}$$

$$\Psi(T_l, C_q, f_1 \wedge f_2 \wedge \ldots \wedge f_{\hat{F}}) = 1, \quad \textit{where } \hat{F} < F \tag{7d}$$

From (7a), assuming that from root task to the last task, there is $\mathcal{L}$ depth that separates tasks due to the precedence, the sum of maximum task completion time on each depth $\{1, 2, \ldots, \mathcal{L}\}$ indicates the makespan of job J_k, $L1 + L2 + \ldots + Ln$ is the total number of tasks on J_k; Constraint (7b) means that each job has a deadline and the job J_k must be completed within the deadline $\mathcal{D}_{J_k}$; To successful complete a job J_k, constraint (7c) makes sure that each task can only be placed on one data center for processing, where $x_{l,q} = 1$ task T_l is allocated to data center C_q, otherwise not; (7d) is to check the satisfaction on data-dependency, where the data-dependency of task T_l has been satisfied with the set of data files $\{f_1, f_2, \ldots, f_{\hat{F}}\}$ on data center C_q, function (7d) equals to 1, otherwise not.

IV. PROPOSED JOINT SCHEDULING FRAMEWORK: *JHTD*

A. OVERVIEW

To address the problem of *JOP* defined in the last section, we propose an efficient joint scheduling framework for task placement and data transfer across GDDCs, i.e., *JHTD*. Figure 5. illustrates the flowchart of the *JHTD*. Initially, jobs are submitted to the system. The jobs are collected and preprocessed to extract the tasks according to the precedence of the tasks and required data files. Next, *JHTD* is going to implement two important stages. As we have built the hypergraph model in the last section, the first stage is mainly to obtain a basic feasible solution in terms of hypergraph partition to task placement *HPTP*. However, *HPTP* pays considerable attention on maintaining load balancing in the data center. It is inevitable to yield the file data transferring cost which increase the task completion time. In the second stage, based on the result of *HPTP*, with the aim of optimizing the completion time, there are two schemes developed: For the task reallocation, the task with the lowest data-dependency is reallocated to the data center with minimum completion time (*TALD*). From the aspect of data file, the data file with highest dependency is preferentially transferred (*DHD*). By repeatedly doing so, the minimum makespan can be eventually obtained.

B. JOB PREPROCESS

At a certain time t, while receiving a set of N_t jobs, the job preprocess is launched to process and extract tasks, so that the

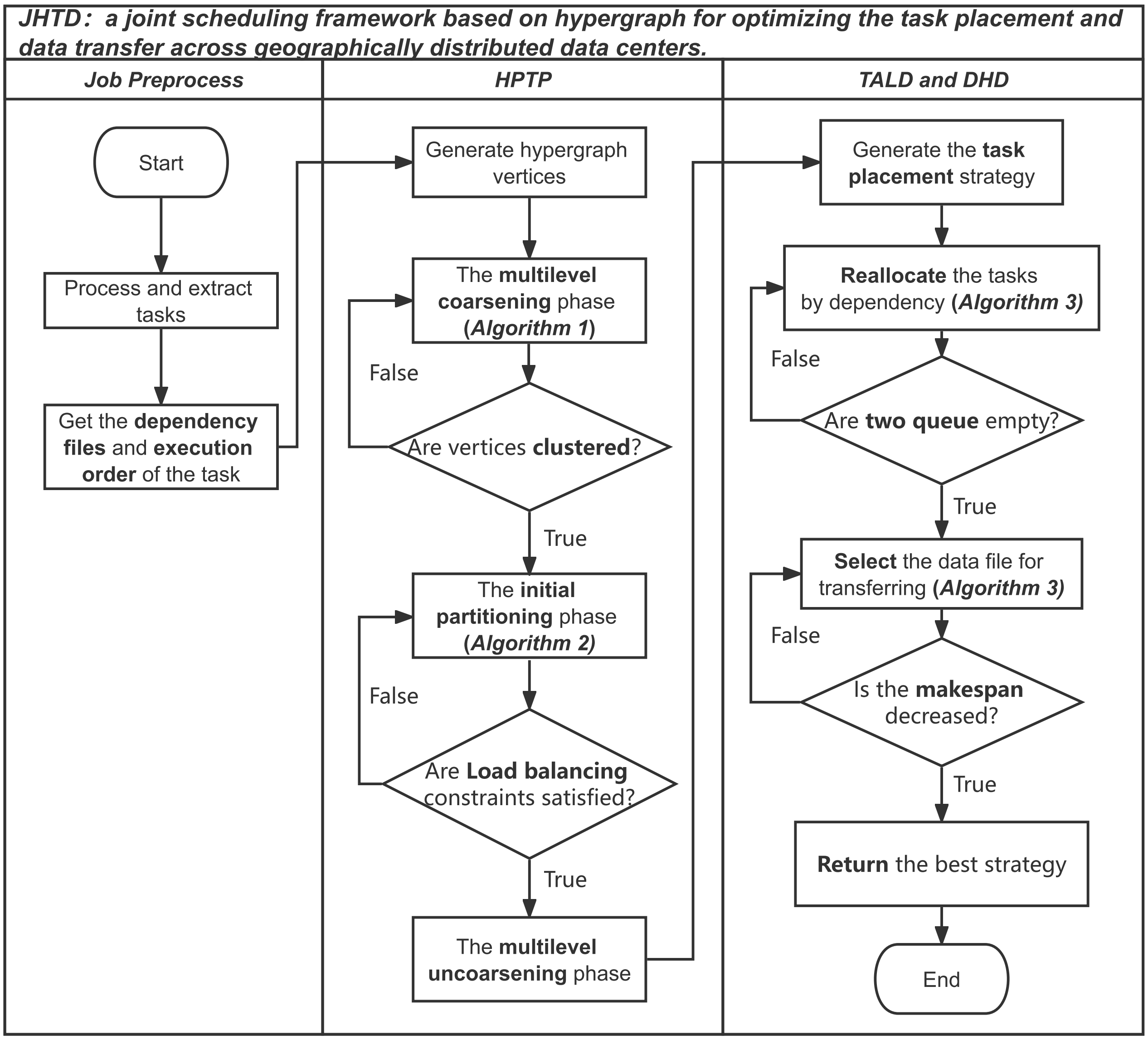

JHTD: a joint scheduling framework based on hypergraph for optimizing the task placement and data transfer across geographically distributed data centers.
Job Preprocess
HPTP
TALD and DHD
Start
Process and extract tasks
Get the dependency files and execution order of the task
Generate hypergraph vertices
The multilevel coarsening phase (Algorithm 1)
False
Are vertices clustered?
True
The initial partitioning phase (Algorithm 2)
False
Are Load balancing constraints satisfied?
True
The multilevel uncoarsening phase
Generate the task placement strategy
Reallocate the tasks by dependency (Algorithm 3)
False
Are two queue empty?
True
Select the data file for transferring (Algorithm 3)
False
Is the makespan decreased?
True
Return the best strategy
End

FIGURE 5. Flowchart of the proposed *JHTD*. It is a joint scheduling framework for task placement and data transfer including *Job Preprocess*, *HPTP*, *TALD* and *DHD*.

constraint and data files required by each task are clarified for later task placement and data transferring. According to the execution sequence of each task, the tasks can be extracted with precedence. Initially, we set the $depth(li)$ to record tasks with various precedence, the tasks within the same $depth(li)$ indicate that these tasks have no precedence, where the increment of li is 1 in each step. Initially, from the root task, as there is no parent vertex for the root task, li equals to 0, and the root task is reserved to $depth(0)$. Next, the child tasks reply on $depth(0)$ will be discovered. Among these tasks, if precedence on tasks is satisfied, the tasks are selected and classified into $depth(1)$. By iteratively doing so, all the tasks can eventually be extracted to the $depth(li)$ for task placement preparation and data transfer.

C. HPTP: HYPERGRAPH PARTITION BASED TASK PLACEMENT

As jobs are separated into tasks, the next step is to place these tasks onto data centers. We propose *HPTP* to get the initialized placement. *HPTP* deals with the hypergraph model we proposed in the previous section to transform the task placement problem into a hypergraph partitioning problem. There are many studies on the problem of hypergraph partitioning, which have developed many advanced hypergraph partitioning tools such as hMETIS [52], PaToH [53], Zoltan [54], and KaHyPar [55]. We use the most popular partition tool PaToh to deal with the problem. *HPTP* consists of three phases: multilevel coarsening, initial partitioning, and uncoarsening.

In the hypergraph model we have proposed, vertices represent the data center and tasks respectively, and the weight of vertices represents the processing capability and task size of the data center. Meanwhile, the hyper edge represents the data file, and the weight of hyper edge represents the size of the data file. We can calculate the baseline of the task completion. According to (4), it is easy to find that the actual calculation time is proportional to the size of data files $S(f_{i,l})$. So, we denote the number of data files f_i required by task T_l on data center C_p as $N_{f_{i,l}}^{C_p}$. The minimization cost function can be approximately regarded as the completion of the minimization makespan. We set the cost of e_i to the size of the file, i.e., $c(e_i) = S(f_i)$, and λ_{e_i} represents the number of partitions connected by hyperedge e_i. Therefore, we define the cost function $Cut(\mathcal{E})$ as below:

$$Cut(\mathcal{E}) = \sum_{e_i \in E} c(e_i)(\lambda_{e_i} - N_{f_{i,l}}^{C_p}). \tag{8}$$

where $Cut(\mathcal{E})$ is equal to the sum of the costs of the cut edges. The hypergraph partitioning problem can be defined as a task of dividing a hypergraph into two or more parts such that the $Cut(\mathcal{E})$ is minimized, when we complete the partitioning of the hypergraph, we actually get a task placement under load balancing. After expounding the problem of hypergraph partitioning, next we explain the three phases of partitioning. Multilevel coarsening phase, the purpose is to merge vertices to form multiple small-scale hypergraphs according to Algorithm 1. Then, initial partitioning phase uses Algorithm 2 to recursively partition of hypergraphs, and multilevel uncoarsening phase is to adjust the partition results.

1) THE MULTILEVEL COARSENING PHASE

The goal of multilevel coarsening initialization partitioning is to compress the hypergraph $\mathcal{H} = \mathcal{H}_1 = (\mathcal{V}_0, \mathcal{E}_0)$ into sufficiently small hypergraphs $\mathcal{H}_1 = (\mathcal{V}_1, \mathcal{E}_1)$,$\mathcal{V}_2 = (\mathcal{V}_2, \mathcal{E}_2)$,..., $\mathcal{H}_m = (\mathcal{V}_m, \mathcal{E}_m)$ satisfying $|\mathcal{V}_0| > |\mathcal{V}_1| > |\mathcal{V}_2| > \ldots > |\mathcal{V}_m|$. The coarsening at this stage is achieved by coalescing disjointed subsets of vertices of hypergraph $\mathcal{H}_i$ into multi-vertices, each vertex in $\mathcal{H}_i$ can form a single vertex of $\mathcal{H}_{i+1}$ after coarsening. Meanwhile, the weight of each vertex of $\mathcal{H}_{i+1}$ becomes equal to the sum of its constituent vertices of the respective multi-vertices in $\mathcal{H}_i$. The coarsening phase terminates when the number of vertices in the coarsened hypergraph reduces to below a pre-determined number. Two clustering methods have implemented in Patoh, namely matching-based and agglomerative clustering. We use the matching clustering, and its algorithm steps are shown in the Algorithm 1.

Each vertex u is assumed to constitute a singleton cluster $\mathcal{H}_u = u$ at the beginning of each coarsening level. Then, vertices are visited in a random order. If a vertex $u \in \mathcal{V}_i$ has not been matched, one of its unmatched neighbors is selected according to the weight of the vertex (i.e. the size of the task represented or the processing power of the data center). If such a vertex v exists, we merge the matching pairs

Algorithm 1 The Matching-Based Clustering Algorithm.

```
Input: $\mathcal{H}_0 = (\mathcal{V}_0, \mathcal{E}_0)$
Output: $\mathcal{H}_1 = (\mathcal{V}_1, \mathcal{E}_1)$,$\mathcal{H}_2 = (\mathcal{V}_2, \mathcal{E}_2)$,..., $\mathcal{H}_m = (\mathcal{V}_m, \mathcal{E}_m)$
1  for $u \in \mathcal{V}_i$ do
2  if $|\mathcal{H}_u| = 1$ then
3  if there is an eligible neighbor vertex $v$ then
4  $v$ add to $\mathcal{H}_u$, $\mathcal{H}'_u \leftarrow \mathcal{H}_u$
5  else $\mathcal{H}_u = u$
6  end if
7  else return $\mathcal{H}_u$
8  end if
9  return $\mathcal{H}'_u$
```

u and v into a cluster (lines 1-4). If there is no unmatched adjacent vertex of u, then vertex u remains unmatched, i.e., u remains as a singleton cluster(line 5). If u has already been clustered (i.e. $|\mathcal{H}_u| > 1$) it is not considered the source of a new clustering (line 7). Here, two vertices u and v are said to be adjacent if they share at least one$\mathcal{V}_i$, i.e., $\mathcal{V}[u] \cap \mathcal{V}[v] \neq \emptyset$.

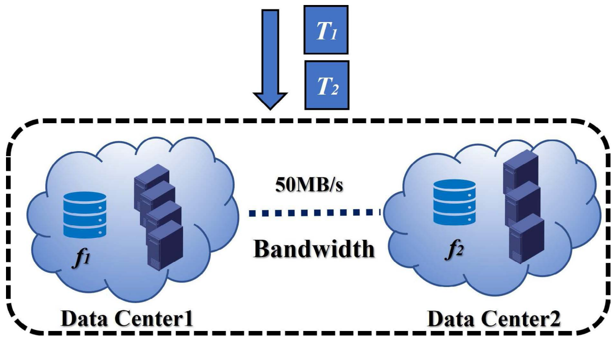
2) THE INITIAL PARTITIONING PHASE

$k - way$ hypergraph partition is achieved by recursive bisection (bidirectional partition) and a multi-level hypergraph bisection algorithm is used in each bisection step. In recursive dichotomy, a bisection of $\mathcal{H}_m$ is first obtained, and then each part of this bisection is further recursively partition. After $log_2 k$ steps, the hypergraph $\mathcal{H}_m$ is divided into k parts.

Algorithm 2 The Multilevel $k - way$ Hypergraph Partition Algorithm

```
Input: $\mathcal{H}_m = (\mathcal{V}_m, \mathcal{E}_m)$
Output: $k - way$ hypergraph partitioning $\Pi = \{\mathcal{H}_m^1, \mathcal{H}_m^2, \ldots, \mathcal{H}_m^3\}$
1  for $v^{us} \in \mathcal{V}_i$ do
2  add $v^{us}$ to priority queue //according to their FM gain;
3  if $v^{us}$ move to growing clusters;
4  cutsize reduce;
5  then select the highest gain vertex in priority queue;
6  while a vertex moves to a growing cluster
7  update the gain of $v_{near}^{us}$;
8  insert vertices that are not in the priority queue;
9  end while //reach the preset value;
10   return $\mathcal{H}_m^1$
11   end for
```

In the $k - way$ partitioning stage, we first generate a cluster around randomly selected vertices. In the rough course of the algorithm, selected and unselected vertices divide the $\mathcal{H}_m$ into two parts. Unselected vertices(v^{us}) connected to growing clusters are inserted into the priority queue according to their Fiduccia-Mattheyses(FM) gain [60] (line 2). If vertices are moved to growing clusters, the gain of unselected vertices corresponds to the cut-size reduction of the current bipartition (lines 3 and 4). The vertex with the highest gain is selected from the priority queue. When a vertex moves to a growing



(a) Two tasks T_1 and T_2 are going to submitted to data centers accordingly

	Unit
Processing capability of C_1	50MB per second
Processing capability of C_2	40MB per second
Size of T_1	50 MB
Size of T_2	60 MB
Required data file of T_1	f_1
Required data file of T_2	f_2
Size of f_1	40MB
Size of f_2	50MB
Link bandwidth	50MB per second

(b) List of relevant parameters on data center C_1 and C_2.

FIGURE 6. An example of scheduling multiple tasks across a geographically distributed data center by using *HPTP*.

cluster, the gain of its currently unselected neighbors(v_{near}^{us}) in the priority queue is updated, and those vertices that are not in the priority queue are inserted, and this cluster growing operation continues until a predetermined balancing the criteria (lines 5-9). At this time, the result of the first hypergraph partition is obtained and breadth-first search is used for the remaining graphs. The above process is repeated to obtain the hypergraph partition sequence (line 1and10).

3) THE MULTILEVEL UNCOARSENING PHASE

After multi-level $k - way$ hypergraph partition, the obtained k subgraphs need to be mapped back to the original hypergraph through multi-level fine-grained partition. Before each mapping, the Boundary FM (BFM) algorithm [61] is used to adjust the vertices between the k subgraphs, so that the hypergraph division result is further optimized and the load balancing constraints are satisfied. In the fine-grained process, the coarse-grained hypergraph $\mathcal{H}_m$ can be mapped back to the fine-grained hypergraph of the $\mathcal{H}_{m-1}$ layer, which is executed iteratively until the hypergraph is mapped to the $\mathcal{H}_0$ layer.

D. TALD AND DHD

Though *HPTP* can obtain a feasible solution, it has a limitation in gaining the optimal makespan. We hereby give a simple example to point out the defect of *HPTP* as shown in Figure 6. There are two data centers C_1 and C_2 with processing capability of 50 and 40, respectively. The two data centers are linked by a bandwidth of 50 MB/s. The data file f_1 is stored on C_1, and f_2 is located on C_2. The size of the data file is 40 MB and 50 MB respectively. At certain time, when tasks start to request data files. The task size of T_1 is 50 MB, which requires data file f_1, and the task size of T_2 is 40 MB, which requires data file f_2.

We give the results by *HPTP* listed in Table 2. Apparently, solely leveraging on *HPTP* generates a numerous cost by data file transfer that substantially results in longer makespan. Based on (5) and (6), the weight on C_1 and C_2 is computed as 4 and 5, respectively. In this case, due to the intuition that

TABLE 2. Comparison of *HPTP* and *Optimal* with data transfer and makespan.

	C_1	C_2	Data file Transfer	Makespan
HPTP	T_2	T_1	90 MB	2.2 s
Optimal	T_1	T_2	0 MB	1.5 s

HPTP depended on processing capability of the data center, the task T_1 is allocated on C_2, and task T_2 is placed onto C_1, causing a poor performance on gaining makespan.

Therefore, we also need to process the partition results. *TALD* is mainly based on the idea of Min-Min algorithm to iteratively optimize the scheduling scheme. In this stage the current task is allocated and the tasks in the data center with the longest total task completion time is continuously forwarded to other data centers to find the optimal allocation. We then use *DHD* to optimize the transferring strategy of data files. Algorithm 3 describes the specific process of task scheduling and data transferring.

The algorithm initially traverses the C_l data file list (line 2), finds the currently executable task te_l, joins the *ExecuteTask* queue (line 3), and adds the file to be transferred to *items*: *items* is a collection of files to be transferred, finds the file with the most occurrences from *items* and adds it to *MaxoccuringChars*. *DHD* compares the size of the data file in *MaxoccuringChars*, finds the smallest data file, then traverses the C_l list again, finds the C_l that stores this file, and adds the file to the transferring queue of the C_l (lines 4-5). At this time, the *ExecuteTask* pop, the execution time of the task is calculated, and the C_l is updated with the task list and transferring time on the above. Then, the completion time of the C_l at this time is calculated, and the next cycle is entered (lines 6-8); when the *ExecuteTask* queue and the *Transfer* queue are empty simultaneously, the cycle is ended.

TALD adopts the idea of Min-Min algorithm. Its main scheduling assigns and processes tasks in the fastest time. It allocates tasks to the data center with the shortest processing time to ensure the shortest time to complete

Algorithm 3 Task Allocation Scheme in Terms of the Lowest Dependency (*TALD*), and Data Aware With the Highest Dependency Transferring Scheme (*DHD*)

```
Input: task set based on hypergraph partitioning
Output: task scheduling strategy
1  hypergraph partition // According to Algorithm 1 and Algorithm 2;
2  for $T_l \in T$ do
3      add $te_l$ to $ExcuteTask$ queue;
4      add file of $te_l$ to $items$;
5      find $f^l_{max}$ and add it to $Transfer$ queue;
6      for $C_l \in C$ do
7          $CT_l = ET_l + WT_l + CDT_l$  //calculate the complete time according to formula (4);
8          $te_l$ and $f^l_{max}$ pop respectively;
9      end for
10 end for  //while $ExcuteTask$ queue and $Transfer$ queue are both empty;
11 calculate the respective completion times of $C_l$;
12 for $T_l \in C_l$ do
13     transfer $t_{mdd}$ to other data center  //find a suitable data center based on the Min-Min algorithm;
14     $minimize : \Gamma_{J_k}$  //minimize makespan according to formula (7a);
15 end for
16 return the task scheduling strategy  //if the makespan is successfully shortened, the scheduling is done.
```

the task. It compares the respective task completion times of C, and finds the longest. The task t_{mdd} on the long C_l is the task with the least dependence on the data file (lines 11-14). Transferring it can effectively reduce the completion time, schedule it to the C_l with the shortest complete time, and it will not cause too much impact on other tasks.

Finally, when the makespan is shortened, the allocation is completed, otherwise the allocation is cancelled, until the makespan cannot be further shortened, and the final task allocation is obtained (line 16).

E. COMPLEXITY ANALYSIS

The *JHTD* proposed in this paper mainly consists of two parts. (1) hypergraph-based partition method for task placement (*HPTP*). (2) task allocation scheme in terms of the lowest dependency (*TALD*) and data aware with the highest dependency transfer scheme (*DHD*). Since the coarsening phase takes a linear time complexity: $O(|V|)$ [56] and the Fiduccia-Mattheyses heuristic in top-level partitioning phase takes time $O(|E|)$ [57], so the time complexity of the k-way hypergraph partitioning algorithm we used is $O((|V| + |E|)log_2N)$. We assume that the number of tasks is N, and the number of data centers is constant M then the complexity of Algorithm 3 is $O(Nlog_2M)$. Therefore, the overall time complexity is $O((|V| + |E|)log_2N + Nlog_2M)$.

TABLE 3. List of configurations on data centers from China-VO.

Configuration	Beijing	Nanjing	Shanghai	Yunnan	Xinjiang
Storage(TB)	55	50	30	10	5
Number of Cores	900	690	850	350	160
Core rating(MIPS)	1330	1166	1200	1140	1000
Normalization	30	10	15	10	4

TABLE 4. Dataset and its characteristics.

Notation	Definition	Range
$S(f_i)$	The size of a data file f_i.	[1,10000]
te_l	The size of task T_l.	[200,5000]
N_k	The number of tasks.	[1,100]
$\|F\|$	The number of data files.	[1,400]
$R_{(C_p,C_q)}$	Capacity of link $C_p, C_q, \forall p, q \in M$.	[50,200]

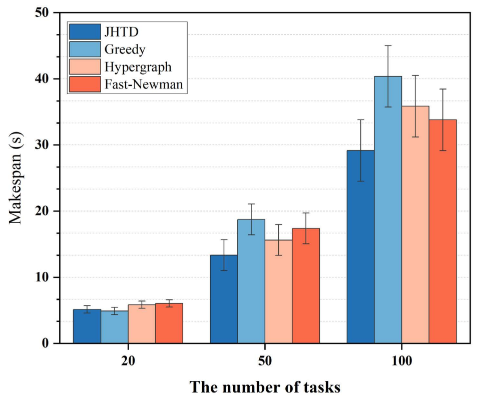
V. PERFORMANCE EVALUATION

In this section, we present our experimental setup and parameter settings in the geographically distributed data centers, and detailed experimental results on real workloads. We use a simulated cloud environment from the real configuration of the China-VO project [13] for evaluation, the data file and task size can be generated from this dataset. We repeated each experiment 10 times and took the average result as our final result.

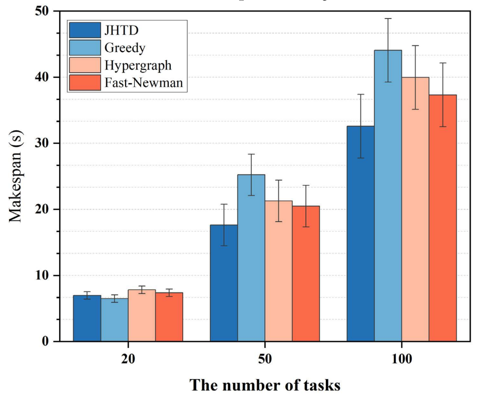
A. EXPERIMENTAL SETUP AND PARAMETERS SETTING

As the study work [13], we performed extensive simulations using real-world configurations and dataset from the China-VO project. The cloud system consisted of five data centers: Beijing (BJ), Nanjing (NJ), Yunnan (YN), Shanghai (SH), and Xinjiang (XJ), connected by high-capacity network links. Table 3 shows the configuration of these data centers. The computing capability and storage capacity vary between data centers. We normalized each data center capability combined by the number of cores and capability per core to calculate the hypergraph weights. The higher normalization value the a stronger the process capability. The data file transfer times for within data center were ignored because these are much smaller than the inter-data center transfer times.

The relevant dataset for our experiments is derived from [58], Table 4 shows its characteristics. the size of the data files randomly generated were in the range of 200 MB to 5,000 MB. The generated data files were randomly allocated to various locations of the data centers. For the task generation, according to Facebook's experience, more than 90% of the tasks are completed within 1000 seconds [59]. Therefore, the size of tasks randomly generated were in the range of [1,10000] based on the processing capability of the data center. In addition, we set the *Datadependency* to represent the dependency of the task on the number of data files, that is, when the task only needs one data file, *Datadependency* equals to 1. In our experiment, the *Datadependency* was set within [1,4]. The task with higher *Datadependency* makes the joint optimization issue more complex.

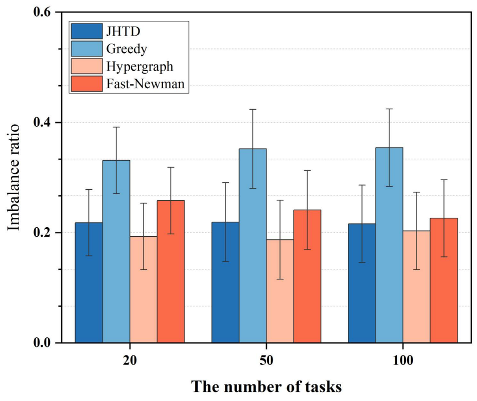


(a) $Datadependency \leq 2$

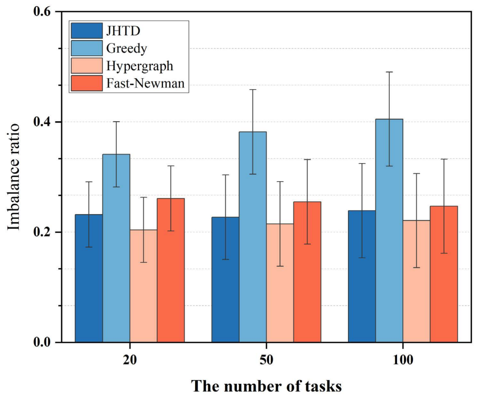


(b) $Datadependency > 2$

FIGURE 7. Comparison on the makespan of the proposed *JHTD* with *Greedy*, *Hypergraph*, and *Fast − Newman* on the different *Datadependency* (a) *Datadependency* ≤ 2, and (b) *Datadependency* > 2.



(a) $Datadependency \leq 2$



(b) $Datadependency > 2$

FIGURE 8. Comparison on the imbalance ratio of the proposed *JHTD* with *Greedy*, *Hypergraph*, and *Fast − Newman* on the different *Datadependency* (a) *Datadependency* ≤ 2, and (b) *Datadependency* > 2.

B. COMPARATIVE ALGORITHMS

We compared our proposed *JHTD* with other algorithms: *Greedy*, improved hypergraph partitioning algorithms *Hypergraph* [14], and community detection algorithm *Fast−Newman* [15]. All algorithms are implemented in Java. To obtain the results by *Hypergraph* partitioning algorithm, we adopted the partition toolkit (PaToH). The following are the brief details of the comparative algorithms:

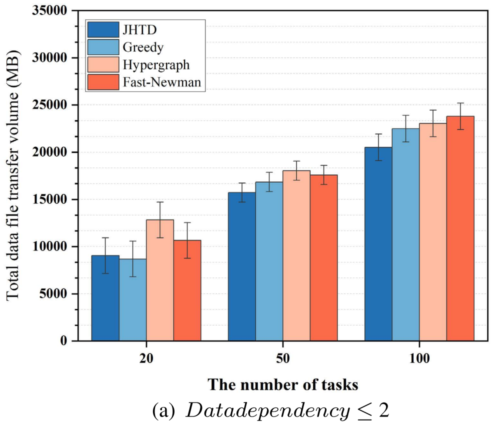
- *Greedy*: The greedy algorithm is a classic scheduling algorithm that is often used as a comparison algorithm. The main concept of *Greedy* is to select the data center with the shortest completion time for the current task and transfer the required data files for that task.
- *Hypergraph*: This algorithm assigns tasks to data centers using a hypergraph partitioning method, which evenly divides the tasks into balanced partitions according to the number of data centers.
- *Fast − Newman*: By considering the task scheduling problem as a community detection problem with the goal of reducing data transfer cost, this algorithm iteratively places tasks into communities while maximizing the modularity measure Q [47], where the greater modularity Q gains a better performance of community division.

C. RESULTS COMPARISON AND ANALYSIS

1) COMPARISON OF RESULTS

We compared the proposed *JHTD* with three other algorithms *Greedy*, *Hypergraph*, and *Fast − Newman* in terms of makespan under different *Datadependency*. As shown in the Figure 7, it is clear that the makespan of algorithms is increasing with the number of tasks and *Datadependency*. *Greedy* performs best when the number of tasks is small, especially the when the number of tasks is 20. Under a small-scale situation, the *Greedy* algorithm achieves an optimal task placement solution. In contrast, since *JHTD* has the ability of maintaining the load balance of the data centers, it cannot gain optimal results in this situation. However, *JHTD* still outperforms *Hypergraph* and *Fast − Newman*. *Hypergraph* outperforms *Fast − Newman* at low *Datadependency*. When *Datadependency* becomes greater than 2, the performance of *Hypergraph* gets worse than that of *Fast − Newman*. It shows the advantage of *Fast − Newman* in dealing with tasks with greater *Datadependency*.

In summary, as the number of tasks increase, our proposed *JHTD* has the best performance in achieving the shortest makespan under various *Datadependency*.



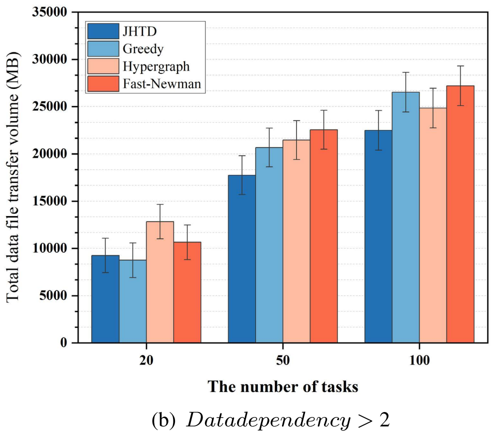
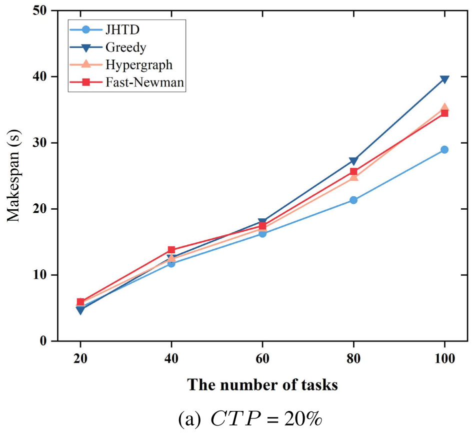


FIGURE 9. **Comparison on the total data file transfer volume of the proposed *JHTD* with *Greedy*, *Hypergraph*, and *Fast − Newman* on the different *Datadependency* (a) *Datadependency* ≤ 2, and (b) *Datadependency* > 2.**



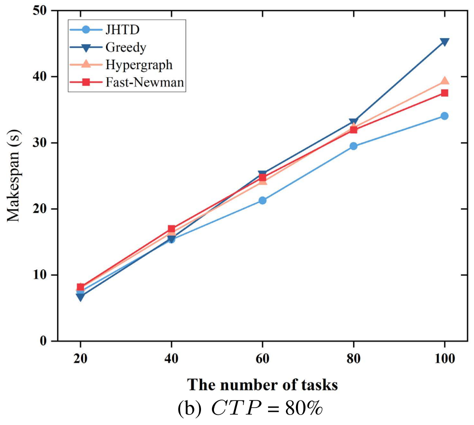


FIGURE 10. **Comparison on the makespan of the proposed *JHTD* with *Greedy*, *Hypergraph*, and *Fast − Newman* on the different *CTP* (a) *CTP* = 20%, and (b) *CTP* = 80%.**

2) IMPACT OF IMBALANCE RATIO

Data centers with stronger processing power can perform more tasks and reasonable load balancing can also reduce the waiting time of tasks. Therefore, we also evaluated the impact on the imbalance ratio on the data centers with the four algorithms in Figure 8. *JHTD* performed better than *Greedy* and *Fast − Newman* under different number of tasks with various *Datadependency*. However, *Hypergraph* outperformed our *JHTD* because the intuition of *Hypergraph* is to balance each hypergraph partition by compromising the makespan.

3) IMPACT OF TOTAL VOLUME OF DATA FILES BY TRANSFERRING

Because the transfer time of a data file partially relies on the size of the data file, the total volume of data files is the sum of each data file size to be transferred. It is indicated that a larger total volume of data files consumes higher time cost.

Figure 9 demonstrates the results on the total volume of data file transferred across data centers under various *Datadependency* by the four algorithms. From the results, the performance of *Hypergraph* is not good enough between the algorithms because the balance of the partition could increase the volume of the data file transferred. As the scale of the tasks increases, performances of *Hypergraph* and *Fast − Newman* are getting close. *Greedy* achieved optimal results with a small group of tasks. Apparently, *JHTD* has the least data transfer volume because of the two-stage processing of tasks and data file by *JHTD*, it succeeds in minimizing the volume of data file transferred.

4) IMPACT OF *CTP*, COMPLEX TASK PROPORTION

To further evaluate the performance of *JHTD*, since the size of task also has impact to the makespan results, based on the parameters we set, we classified task with sizes larger than 1000 and *Datadependency* > 2 as complex tasks. We defined a metric, complex task proportion (*CTP*), which is the complex task number divided by the total number of tasks.

In Figure 10, in the case of a small number of tasks, the makespan of each algorithm was almost the same. As the number of tasks increased, *JHTD* performed the best in makespan. *Greedy* had poor performance on complex tasks (a longer makespan). This situation worsened with the growth in the number of tasks as *Greedy* easily got trapped in local optimal solutions while dealing with large-scale problems. *Fast − Newman* showed better performance than

TABLE 5. Comparison on the performance of *JHTD*, *Greedy*, *Hypergraph*, and *Fast − Newman* under different *CTP* (the number of tasks is 100), *JHTD* with better performance has been highlighted in the table.

Metric	Method	The number of tasks is 100					
		CTP = 0%	*CTP = 20%*	*CTP = 40%*	*CTP = 60%*	*CTP = 80%*	*CTP = 100%*
Makespan(s)	*JHTD*	**28.04**	**28.67**	**29.83**	**31.68**	**33.93**	**36.24**
	Greedy	35.53	39.72	41.02	45.52	47.08	52.75
	Hypergraph	32.47	35.24	37.27	39.16	40.95	42.68
	Fast-Newman	32.96	34.56	36.11	38.58	39.14	40.42
Imbalance ratio	*JHTD*	0.2	**0.2**	0.23	0.25	0.28	0.3
	Greedy	0.33	0.36	0.41	0.43	0.44	0.47
	Hypergraph	0.19	0.2	0.22	0.23	0.27	0.28
	Fast-Newman	0.22	0.23	0.25	0.29	0.33	0.38
Total data file transfer volume(MB)	*JHTD*	**21350**	**21840**	**22540**	**22870**	**23810**	**24520**
	Greedy	22700	23360	25470	26080	27070	27850
	Hypergraph	23240	23650	24010	24760	25340	26100
	Fast-Newman	23710	24600	25840	26360	27530	28170

Hypergraph. The performance of *Fast − Newman* largely depended on the task number to maximize modularity metric Q. In contrast, with the aim of balancing each hypergraph partition, *Hypergraph* increased the amount of data file transfer, which resulted in the increase in makespan.

Overall, under the case of $CTP = 20\%$ and $CTP = 80\%$, *JHTD* had the most improvement of 20.6% and 15.2% compared to *Fast − Newman* accordingly.

5) COMPARISON OF TIME COMPLEXITY

We eventually compared the time complexity in Figure 11 by the running time of each algorithm. Since *Hypergraph* does not need to experience the stage of task reallocation, the running time totally depends on the partition result by the hypergraph; the time complexity is the smallest of the compared algorithms. *Fast − Newman* has the worst time complexity. This is because most time is consumed by the computation on the modularity measure Q. In comparison, though *JHTD* is a two-stage scheduling framework with task placement and data transfer, it still has a shorter running time than *Greedy* and *Fast − Newman*, due to the fact that *JHTD* does not need to traverse all tasks and data files.

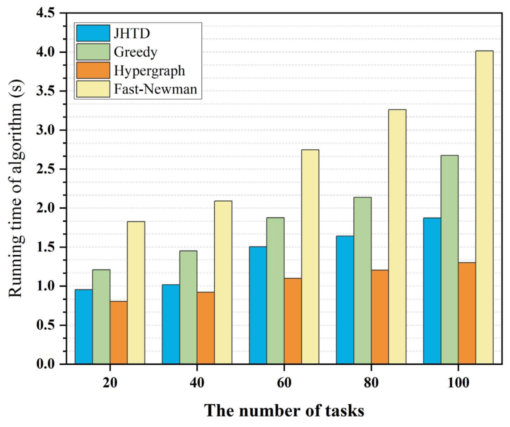


FIGURE 11. Comparison on the running time of the proposed *JHTD* with *Greedy*, *Hypergraph*, and *Fast − Newman* at different task scales.

VI. DISCUSSION

In the last section, the results have demonstrated the outperformance of *JHTD* while comparing with other algorithms. To further validate the extensibility of *JHTD* under various portion of complex tasks (*CTP*), we fixed the number of tasks to 100 and made the discussion of *JHTD* performance. As it is listed in Table 5, *JHTD* still shows a better performance in most cases. *Greedy* is difficult to deal with large-scale and complex tasks, and it does not perform well under multiple metrics. This is because it only makes the current optimal choice and cannot consider the impact of subsequent tasks. *Hypergraph* has good imbalance ratio, because this itself is its optimization goal, but because it focuses too much on the balance between data centers, resulting in increased data transfer, which requires a longer makespan. *Fast − Newman* always places tasks on data centers with more data files, so the execution time of tasks in the current data center is always short. However, as the number of data files required by the task increases, *Fast − Newman* will unbalance the load between the data centers and need to transmit a large number of data files, resulting in an increase in makespan. As a joint scheduling framework, *JHTD* not only fully considers the execution efficiency of tasks, but also considers the data file transfer between data centers, and slightly adjusts the workload between data centers, thereby greatly reducing the volume of makesapn and data transferring.

VII. CONCLUSION AND FUTURE WORK

This paper explored the joint optimization issue of task placement and data transfer across GDDCs. With the connection of task, data file and data center, we adopted hypergraphs to establish a system model and task model. On this basis, we proposed a joint scheduling framework, *JHTD*, to optimize task placement and data transfer problems and achieve maksepan minimization. In *JHTD*, we introduced and improved hypergraph partitioning techniques to partition tasks. In addition, we designed an algorithm to iteratively optimize the hypergraph partitioned results to minimize makespan. Finally, we conducted simulation evaluations of the proposed method on real-world datasets and compared it with well-known algorithms, *JHTD* balanced the workload between data centers well, and its data transfer volume was also the smallest. Most importantly, *JHTD* had the lowest makespan regardless of the complexity or size of the tasks. Moreover, we discussed the effectiveness of *JHTD* under various portions of complex tasks. The goal of minimizing maksepan was achieved.

There are several future avenues for our work. First, we will consider multiple requirements from tasks, e.g., the privacy of the data files required by the tasks, the deadline of the tasks, and the task's requirement on the capacity of the data center. Also, we will further establish the task model as spillable or non-spillable. At the same time, the network environment is also a worthy direction for exploration, such as flow control or task migration due to latency requirements among data centers. Furthermore, as GDDCs will continue to grow in size and multiple types of jobs emerge, we will integrate reinforcement learning-based methods into *JHTD* to improve adaptability and accelerate the speed of processing.

REFERENCES

[1] J. Parikh, "Keynote speech: Data infrastructure at web scale," *VLDB*, vol. 13, p. 855, Jan. 2013.

[2] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[3] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proc. USENIX Symp. Netw. Syst. Design Implement.*, 2012, pp. 15–28.

[4] K. Kloudas, M. Mamede, N. Preguiça, and R. Rodrigues, "Pixida: Optimizing data parallel jobs in wide-area data analytics," *Proc. VLDB Endowment*, vol. 9, no. 2, pp. 72–83, 2015.

[5] A. Vulimiri, C. Curino, P. Godfrey, T. Jungblut, J. Padhye, and G. Varghese, "Global analytics in the face of bandwidth and regulatory constraints," in *Proc. USENIX Symp. Netw. Syst. Design Implement.*, 2015, pp. 323–336.

[6] A. Vulimiri, C. Curino, P. Godfrey, K. Karanasos, and G. Varghese, "WANalytics: Analytics for a geo-distributed data-intensive world," in *Proc. Conf. Innov. Data Syst. Res.*, 2015, pp. 1087–1092.

[7] J. H. Wang, J. Wang, C. An, and Q. Zhang, "A survey on resource scheduling for data transfers in inter-datacenter WANs," *Comput. Netw.*, vol. 161, pp. 115–137, Oct. 2019.

[8] J. Zhang, M. Wang, J. Luo, F. Dong, and J. Zhang, "Towards optimized scheduling for data-intensive scientific workflow in multiple datacenter environment," *Concurrency Comput., Pract. Exper.*, vol. 27, no. 18, pp. 5606–5622, Dec. 2015.

[9] M. Selimi, L. Cerda-Alabern, L. Wang, A. Sathiaseelan, L. Veiga, and F. Freitag, "Bandwidth-aware service placement in community network micro-clouds," in *Proc. IEEE 41st Conf. Local Comput. Netw. (LCN)*, Nov. 2016, pp. 220–223.

[10] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low latency geo-distributed data analytics," in *Proc. ACM Conf. Special Interest Group Data Commun.*, Aug. 2015, pp. 421–434.

[11] Z. Hu, B. Li, and J. Luo, "*Flutter*: Scheduling tasks closer to data across geo-distributed datacenters," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun.*, Apr. 2016, pp. 1–9.

[12] T. Abbasi-Khazaei and M. H. Rezvani, "Energy-aware and carbon-efficient VM placement optimization in cloud datacenters using evolutionary computing methods," *Soft Comput.*, vol. 26, no. 18, pp. 9287–9322, Sep. 2022.

[13] L. Yin, J. Sun, L. Zhao, C. Cui, J. Xiao, and C. Yu, "Joint scheduling of data and computation in geo-distributed cloud systems," in *Proc. 15th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, May 2015, pp. 657–666.

[14] B. Yu and J. Pan, "A framework of hypergraph-based data placement among geo-distributed datacenters," *IEEE Trans. Services Comput.*, vol. 13, no. 3, pp. 395–409, May/Jun. 2020.

[15] M. E. J. Newman, "Fast algorithm for detecting community structure in networks," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 69, no. 6, p. 2, Jun. 2004.

[16] X. Gu, L. Chen, and M. Krenn, "Quantum experiments and hypergraphs: Multiphoton sources for quantum interference, quantum computation, and quantum entanglement," *Phys. Rev. A, Gen. Phys.*, vol. 101, no. 3, Mar. 2020, Art. no. 033816.

[17] P. Schwaller, R. Petraglia, V. Zullo, V. H. Nair, R. A. Haeuselmann, R. Pisoni, C. Bekas, A. Iuliano, and T. Laino, "Predicting retrosynthetic pathways using transformer-based models and a hyper-graph exploration strategy," *Chem. Sci.*, vol. 11, no. 12, pp. 3316–3325, 2020.

[18] S. Feng et al., "Hypergraph models of biological networks to identify genes critical to pathogenic viral response," *BMC Bioinf.*, vol. 22, no. 1, pp. 1–21, 2021.

[19] X. Zheng, W. Zhu, C. Tang, and M. Wang, "Gene selection for microarray data classification via adaptive hypergraph embedded dictionary learning," *Gene*, vol. 706, pp. 188–200, Jul. 2019.

[20] W. Shao, Y. Peng, C. Zu, M. Wang, and D. Zhang, "Hypergraph based multi-task feature selection for multimodal classification of Alzheimer's disease," *Computerized Med. Imag. Graph.*, vol. 80, Mar. 2020, Art. no. 101663.

[21] D. Yang, B. Qu, J. Yang, and P. Cudre-Mauroux, "Revisiting user mobility and social relationships in LBSNs: A hypergraph embedding approach," in *Proc. World Wide Web Conf.*, 2019, pp. 2147–2157.

[22] Q. Fang, J. Sang, C. Xu, and Y. Rui, "Topic-sensitive influencer mining in interest-based social media networks via hypergraph learning," *IEEE Trans. Multimedia*, vol. 16, no. 3, pp. 796–812, Apr. 2014.

[23] X. Liao, Y. Xu, and H. Ling, "Hypergraph neural networks for hypergraph matching," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 1246–1255.

[24] S. Bai, F. Zhang, and P. H. S. Torr, "Hypergraph convolution and hypergraph attention," *Pattern Recognit.*, vol. 110, Feb. 2021, Art. no. 107637.

[25] W. Zhao, S. Tan, Z. Guan, B. Zhang, M. Gong, Z. Cao, and Q. Wang, "Learning to map social network users by unified manifold alignment on hypergraph," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 12, pp. 5834–5846, Dec. 2018.

[26] F. Luo, L. Zhang, B. Du, and L. Zhang, "Dimensionality reduction with enhanced hybrid-graph discriminant learning for hyperspectral image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 58, no. 8, pp. 5336–5353, Aug. 2020.

[27] D. Du, H. Qi, L. Wen, Q. Tian, Q. Huang, and S. Lyu, "Geometric hypergraph learning for visual tracking," *IEEE Trans. Cybern.*, vol. 47, no. 12, pp. 4182–4195, Dec. 2017.

[28] O. Selvitopi, G. V. Demirci, A. Turk, and C. Aykanat, "Locality-aware and load-balanced static task scheduling for MapReduce," *Future Gener. Comput. Syst.*, vol. 90, pp. 49–61, Jan. 2019.

[29] C. Li, J. Tang, T. Ma, X. Yang, and Y. Luo, "Load balance based workflow job scheduling algorithm in distributed cloud," *J. Netw. Comput. Appl.*, vol. 152, Feb. 2020, Art. no. 102518.

[30] Y. Song, L. Wang, L. Xiao, W. Wei, R. Scherer, G. Qin, and J. Wang, "Hypergraph-partitioning-based online joint scheduling of tasks and data," *J. Supercomput.*, vol. 2022, pp. 1–30, Apr. 2022.

[31] K. Wang, Q. Zhou, S. Guo, and J. Luo, "Cluster frameworks for efficient scheduling and resource allocation in data center networks: A survey," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 3560–3580, 4th Quart., 2018.

[32] L. Chen, S. Liu, B. Li, and B. Li, "Scheduling jobs across geo-distributed datacenters with max-min fairness," *IEEE Trans. Netw. Sci. Eng.*, vol. 6, no. 3, pp. 488–500, Jul. 2019.

[33] J. Li, M. Qiu, Z. Ming, G. Quan, X. Qin, and Z. Gu, "Online optimization for scheduling preemptable tasks on IaaS cloud systems," *J. Parallel Distrib. Comput.*, vol. 72, no. 5, pp. 666–677, May 2012.

[34] P. Singh, M. Dutta, and N. Aggarwal, "A review of task scheduling based on meta-heuristics approach in cloud computing," *Knowl. Inf. Syst.*, vol. 52, no. 1, pp. 1–51, Apr. 2017.

[35] Y. Yun, E. J. Hwang, and Y. H. Kim, "Adaptive genetic algorithm for energy-efficient task scheduling on asymmetric multiprocessor system-on-chip," *Microprocessors Microsyst.*, vol. 66, pp. 19–30, Apr. 2019.

[36] G. Taheri, A. Khonsari, R. Entezari-Maleki, and L. Sousa, "A hybrid algorithm for task scheduling on heterogeneous multiprocessor embedded systems," *Appl. Soft Comput.*, vol. 93, pp. 106–202, Jan. 2020.

[37] Z.-J. Wang, Z.-H. Zhan, S. Kwong, H. Jin, and J. Zhang, "Adaptive granularity learning distributed particle swarm optimization for large-scale optimization," *IEEE Trans. Cybern.*, vol. 51, no. 3, pp. 1175–1188, Mar. 2021.

[38] S. Caino-Lores and J. Carretero, "A survey on data-centric and data-aware techniques for large scale infrastructures," *Int. J. Comput. Inf. Eng.*, vol. 10, pp. 459–465, Feb. 2016.

[39] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, "Heuristics for scheduling parameter sweep applications in grid environments," in *Proc. 9th Heterogeneous Comput. Workshop (HCW)*, Cancun, Mexico, May 2000, pp. 349–363.

[40] H. H. Mohamed and D. H. J. Epema, "An evaluation of the close-to-files processor and data co-allocation policy in multiclusters," in *Proc. IEEE Int. Conf. Cluster Comput.*, Los Alamitos, CA, USA, Sep. 2004, pp. 20–23.

[41] M. W. Convolbo, J. Chou, S. Lu, and Y. C. Chung, "DRASH: A data replication-aware scheduler in geo-distributed data centers," in *Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, New York, NY, USA, Dec. 2016, pp. 302–309.

[42] Y. Li, L. Zhao, C. Cui, and C. Yu, "Fast big data analysis in geo-distributed cloud," in *Proc. IEEE Int. Conf. Cluster Comput. (CLUSTER)*, Sep. 2016, pp. 388–391, doi: 10.1109/CLUSTER.2016.28.

[43] A. Atrey, G. Van Seghbroeck, H. Mora, F. De Turck, and B. Volckaert, "SpeCH: A scalable framework for data placement of data-intensive services in geo-distributed clouds," *J. Netw. Comput. Appl.*, vol. 142, pp. 1–14, Sep. 2019.

[44] C. Li, Y. Zhang, Z. Hao, and Y. Luo, "An effective scheduling strategy based on hypergraph partition in geographically distributed datacenters," *Comput. Netw.*, vol. 170, Apr. 2020, Art. no. 107096.

[45] B. Yu and J. Pan, "Location-aware associated data placement for geo-distributed data-intensive applications," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 603–611.

[46] W. Yang, L. Ma, R. Cui, and G. Wang, "Hypergraph partitioning for big data applications," in *Proc. IEEE SmartWorld, Ubiquitous Intell. Comput., Adv. Trusted Comput., Scalable Comput. Commun., Cloud Big Data Comput., Internet People Smart City Innov. (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, Oct. 2018, pp. 1705–1710.

[47] B. Cheng, X. Guan, H. Wu, and R. Li, "Hypergraph+: An improved hypergraph-based task-scheduling algorithm for massive spatial data processing on master-slave platforms," *ISPRS Int. J. Geo-Inf.*, vol. 5, no. 8, p. 141, Aug. 2016.

[48] M.-C. Yuen, S.-C. Ng, and M.-F. Leung, "A competitive mechanism multi-objective particle swarm optimization algorithm and its application to signalized traffic problem," *Cybern. Syst.*, vol. 52, no. 1, pp. 73–104, Jan. 2021.

[49] M.-C. Yuen, S.-C. Ng, and M.-F. Leung, "An improved competitive mechanism based particle swarm optimization algorithm for multi-objective optimization," in *Proc. 10th Int. Conf. Inf. Sci. Technol. (ICIST)*, London, U.K., Sep. 2020, pp. 209–218.

[50] A. Majidi, X. Gao, S. Zhu, N. Jahanbakhsh, J. Zheng, and G. Chen, "MiFi: Bounded update to optimize network performance in software-defined data centers," *IEEE/ACM Trans. Netw.*, early access, Jul. 28, 2022, doi: 10.1109/TNET.2022.3192167.

[51] A. Majidi, X. Gao, S. Zhu, N. Jahanbakhsh, and G. Chen, "Adaptive routing reconfigurations to minimize flow cost in SDN-based data center networks," in *Proc. 48th Int. Conf. Parallel Process.*, Aug. 2019, pp. 1–10.

[52] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Applications in VLSI domain," *EEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 7, no. 1, pp. 69–79, Mar. 1999.

[53] U. V. Catalyurek and C. Aykanat, "PaToH: Partitioning tool for hypergraphs," in *Encyclopedia of Parallel Computing*. Boston, MA, USA: Springer, 1999.

[54] K. D. Devine, E. G. Boman, R. T. Heaphy, R. H. Bisseling, and U. V. Catalyurek, "Parallel hypergraph partitioning for scientific computing," in *Proc. 20th IEEE Int. Parallel Distrib. Process. Symp.*, Apr. 2006, p. 10.

[55] S. Schlag, V. Henne, T. Heuer, H. Meyerhenke, P. Sanders, and C. Schulz, "k-way hypergraph partitioning via n-Level recursive bisection," in *Proc. 18th Workshop Algorithm Eng. Exp. (ALENEX)*, Jan. 2016, pp. 53–67.

[56] C. J. Alpert, J.-H. Huang, and A. B. Kahng, "Multilevel circuit partitioning," *IEEE Trans. Comput.-Aided Design Integr.*, vol. 17, no. 8, pp. 655–667, Aug. 1998.

[57] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proc. 19th Design Autom. Conf.*, 1982, pp. 241–247.

[58] L. Zhao, Y. Yang, A. Munir, A. X. Liu, Y. Li, and W. Qu, "Optimizing geo-distributed data analytics with coordinated task scheduling and routing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 2, pp. 279–293, Feb. 2020.

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

[60] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proc. 19th Design Autom. Conf.*, 1982, pp. 175–181.

[61] U. V. Catalyurek and C. Aykanat, "Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication," *IEEE Trans. Parallel Distrib. Syst.*, vol. 10, no. 7, pp. 673–693, Jul. 1999.

CHAO JING was born in 1983. He received the Ph.D. degree from the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China, in 2014. Since 2018, he has been an Associate Professor at the College of Information Science and Engineering, Guilin University of Technology, China. He has published over 50 academic papers in the international major journals and conferences. His research interests include cloud computing and big data processing, workflow scheduling on cloud data center, and deep reinforcement learning.

PENGGAO DAN was born in 1998. He is currently pursuing the master's degree majoring in computer science with the Guilin University of Technology, China. His research interests include cloud computing, and task scheduling and cost optimization based on geographically distributed data centers.