The Journal of Systems & Software 187 (2022) 111227

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

# Fault-tolerant scheduling and data placement for scientific workflow processing in geo-distributed clouds☆

Chunlin Li [a,b,c,*], Jun Liu [a], Min Wang [b], Youlong Luo [a]

[a] Department of Computer Science, Wuhan University of Technology, Wuhan 430063, PR China
[b] Institute of regulatory science, Beijing Technology and Business University, Beijing, PR China
[c] AnHui Key Laboratory of Detection Technology and Energy Saving Devices, AnHui Polytechnic University, Wuhu, PR China

ARTICLE INFO

*Article history:*
Received 23 September 2021
Received in revised form 10 December 2021
Accepted 13 January 2022
Available online 21 January 2022

*Keywords:*
Geo-distributed cloud
Data placement
Fault-tolerant scheduling
QoS

ABSTRACT

Scientific workflow processing in geo-distributed cloud is crucial for the scheduling of large-scale tasks and the massive data placement among tasks. However, the task execution time and energy consumption of data transmission are two urgent issues when a scientific workflow is processed in the different geo-distributed data centers. Aiming at the data placement problem, this paper proposes a Lagrangian relaxation method. This method considers the workload balance, storage capacity, data dependency, transmission bandwidth, and transmission cost to obtain the minimum time of data transmission time. Aiming at the task scheduling problems, a fault-tolerant scheduling strategy is proposed. The strategy optimizes the task scheduling mechanism by considering the task execution time and energy consumption. Finally, the performance of the proposed methods is evaluated via extensive experiments In terms of the data placement, the experiment results imply that the data transmission time of the proposed relaxation algorithm can averagely achieve up to 14.61%, 38.03%, and 39.57% reduction of over that of ILP-FDP algorithm, GA-DPSO algorithm, and GPDP algorithm, respectively. As for the fault-tolerant scheduling, the energy consumption of the TSPT algorithm is the lowest. Compared with the MTS algorithm, EODS algorithm and EWTS algorithm, the average gains of the proposed algorithm are 15.33%, 16.65%, and 28.96%, respectively. Compared with the benchmark algorithms, the task execution time of the TSPT algorithm can averagely reduce up to 12.78,18.85 and 25.65, respectively.



## 1. Introduction

In recent years, the explosive growth of data becomes a pressing problem for big data processing (Wu et al., 2017). To address the issue, cloud computing emerges as a promising solution. Cloud computing provides computing and storage services in a secure, reliable, low-cost, and highly scalable manner (Wan and Gu, 2020; Chauhan et al., 2019). With the development of cloud computing, data centers are usually distributed in different geographical areas. The cloud data centers can provide services close to users, reduce energy consumption and improve service stability (Hu et al., 2017a; Michailidou and Gounaris, 2019). Thus, the geo-distributed clouds are a new paradigm for providing geographical resources (Chen et al., 2019). However, some issues need to be addressed in the geo-distributed cloud (Hu et al., 2018; Jia et al., 2018). Firstly, the network resources between the geo-distributed data centers are limited. Therefore, the data needs to be placed in a reasonable geographic location (Li et al., 2018). In addition, tasks are hard to be scheduled in such a geo-distributed system. Before the tasks in that data center can be processed, the required data needs to be stored in a particular data center. Finally, due to the failures of server and software (Lu et al., 2017), a large number of backward tasks exist in the system. Therefore, for improving system performance significantly, the backward tasks need to be dealt with properly.

In this paper, the processing of data streams is divided into two phases: data stream creation and data stream execution. In both of these phases, the relationship between data placement and the task of fault-tolerant scheduling is closely linked. ① In the data stream creation phase, firstly, the Minimum-Cost-Transmission-Bandwidth (MCTB) method is proposed to decrease the cost of bandwidth. Subsequently, based on the data center capacity, workload, and data dependencies, the most relevant datasets are placed in the same data center when the data stream is created. In this way, the number of data transfers per data center is reduced. Finally, the Lagrangian relaxation algorithm is used to achieve system load balancing. At the same time, the transmission costs and the transmission time are significantly reduced. ②

☆ Editor: J.C. Duenas.

* Corresponding author at: Department of Computer Science, Wuhan University of Technology, Wuhan 430063, PR China.
*E-mail address:* chunlinli74@163.com (C. Li).

https://doi.org/10.1016/j.jss.2022.111227
0164-1212/

In the data stream execution phase, a replica creation strategy is introduced to ensure the fault tolerance in the distributed clouds. And improve the high performance of the cluster is obtained. Then, a Task Scheduling with Minimal Power and Execution Time (TSPT) algorithm is proposed to decrease the consumption cost and the responding time. Experimental results show that the strategy can significantly reduce the amount of data transmission between different data centers. Then the transmission cost, the transmission time, and the energy consumption are reduced.

The main contributions of this paper are summarized as follows.

(1) To reduce the data transmission time, a data placement algorithm is proposed that considers the workload, the remaining storage capacity, the data dependencies, the transmission bandwidth, and the transmission costs. The datasets are placed in load-balanced data centers at a lower cost and in less time.

(2) To reduce the total energy consumption and the average completion time, an integrated problem that combines replica creation with task scheduling is proposed. In the fault-tolerant scheduling method, the task cloning, anomaly detection, energy consumption, and the task deadline are taken into account. First, to guarantee the fault tolerance of the geo-distributed clouds and obtain high performance of Spark, a replica policy based on the speculative execution model is introduced. Then, a Task-Scheduling of TSPT strategy is proposed to reduce the energy consumption of tasks and task response time.

(3) The experiments show that the proposed method can significantly improve the transmission time and the system energy consumption during data placement. The proposed algorithms guarantee the load balance of the geo-distributed system.

The structure of this paper is shown as follows. Section 2 describes related works. The data placement scheme and the fault-tolerant scheduling scheme are introduced in Section 3. The related algorithms implementation is detailed in Section 4. The experimental results are shown in Section 5. The conclusion is discussed in Section 6.

## 2. Preliminaries

### 2.1. The related work of data placement

The issue of data placement in geo-distributed cloud systems is currently a hot topic of research. A large body of literature on data placement is analyzed as follows.

Data placement is an important issue that aims to reduce the data transmission cost between nodes in geo-distributed cloud systems, especially for data-intensive applications. Therefore, some existing works (Shao et al., 2019; Lin et al., 2019; Ikken et al., 2017) investigated data placement in distributed cloud environments to reduce access costs or transmission time. Oh et al. (Shao et al., 2019) proposed TripS to reduce the total cost. In the TripS system, the data placement strategy can be determined dynamically. Lin et al. (2019) proposed to optimize the data transmission time when placing data for a scientific workflow. Ikken et al. (2017) proposed an algorithm to find an optimal intermediate data placement with a cost-saving over the federated cloud datacenters, taking into account scientific user requirements, data dependency, and size.

In this system, the network resource requirement is one of the key areas of performance bottlenecks. Therefore, in distributed cloud environments, some studies (Ferdaus et al., 2017; Xu et al., 2018; Gao et al., 2017) investigated data placement to maximize network resource utilization and energy savings. For improving the network utilization of core switches and reducing the system energy consumption, Md et al. (Ferdaus et al., 2017) proposed a greedy heuristic algorithm that considers the network. In Xu et al. (2018), a data placement method based on Non-Dominated Sorting Genetic Algorithm II (NSGA-II) was designed to achieve high resource utilization and energy-saving rate. Yang et al. (Gao et al., 2017) used an adaptive hybrid strategy for the heterogeneous cloud to improve computing resource utilization and energy saving rate.

Khalajzadeh et al. (2017a), Amina et al. (2018), Yu and Pan (2015) and Khalajzadeh et al. (2017b) used a data placement method to address the data center cost, the transmission time, and the resource utilization through multi-objective optimization techniques. Hourieh et al. (Khalajzadeh et al., 2017a) proposed a new graph partitioning approach to minimize the monetary costs. Amina et al. (2018) proposed an efficient distributed cloud storage replication migration scheme to minimize the time required and ensure data availability. Yu and Pan (2015) proposed a correlated scheme that improves the efficiency of the cloud system. Khalajzadeh et al. (2017b) proposed a novel approach based on graph-partitioning to find a near-optimal data placement of replicas to minimize monetary cost while satisfying the latency requirement.

The existing data placement strategies in the above references have their advantages. However, the following shortcomings should not be overlooked. The aforementioned Shao et al. (2019), Lin et al. (2019) and Ikken et al. (2017) only optimize a single goal such as the transmission cost or transmission time for the data placement. However, to improve the overall performance of geo-distributed cloud systems, multi-objective optimized data placement needs to be employed. In Ferdaus et al. (2017), Xu et al. (2018) and Gao et al. (2017), the network resource utilization and the transmission time are analyzed. However, the load balancing of the geo-distributed environment is not considered. Therefore, the network congestion and the system throughput problems may not be significantly improved. Khalajzadeh et al. (2017a), Amina et al. (2018), Yu and Pan (2015) and Khalajzadeh et al. (2017b) do not consider the data dependencies and the residual capacity of the data centers. Although multi-objective optimized data placement techniques are used in the cloud computing system. As a result, data cannot be placed in advance. Then, the number of transfers and the power consumption in each data center cannot be reduced effectively. Comparing the differences between our work and previous studies, the advantages can be summarized as follows.

(1) A multi-objective optimized data placement scheme is used to balance the multiple objective functions dynamically, while the load balance is considered. Furthermore, the overall performance of the geo-distributed environment is significantly improved.

(2) By considering the data dependency and remaining capacity of the data center, the data placement method based on a multi-objective optimization strategy is used. In this way, when the data can be placed in advance, the number of data transfers and the transmission cost can be reduced.

### 2.2. Fault-tolerant scheduling

Task scheduling is a necessity for the geo-distributed cloud system. Many studies focus on improving the performance of task scheduling.

To reduce response times and avoid the impact of task scheduling failures as much as possible, Xu and Lau (2017), Liu et al. (2017a) and Liu et al. (2016a) proposed a replica management mechanism. In this mechanism, the data replicas are created dynamically, and the replica locations are adjusted. Xu and Lau (2017) combined the task cloning method with the task scheduling algorithms to reduce the impact of machine service variability, then the overall task completion time in different situations is minimized. Liu et al. (2017a, 2016a) investigated some

speculative detection execution strategies to improve system performance in the heterogeneous MapReduce distributed environment, such as LATE, MCP, ex-MCP, and ERUL.

Lu et al. (2018), Liu et al. (2016b, 2017b) and Tang et al. (2016) focused on the workload problem of task scheduling in the distributed cloud environment. Not only the energy consumption was saved, but also the network resource utilization or QoS was significantly improved in the above references. Li et al. (Lu et al., 2018) proposed a task scheduling algorithm based on the shortest path policy. In their proposed algorithm, the resources of the geo-distributed cloud are utilized fully to reduce energy consumption. Liu et al. (2016b) proposed an optimal task scheduling policy to reduce the average delay and the average power consumption. Liu et al. (2017b) address the issue of multi-task scheduling in distributed cloud environments. They focused on the workload balance and the QoS. Tang et al. (2016) present an EWTS method based on the initial state of scheduling tasks to obtain more energy reduction and maintain the QoS by meeting the deadlines.

Bibal Benifa and Dejey (2017), Cameron et al. (2004), Yi et al. (2010) and Li et al. (2019) investigated an integrated problem of combining replica creation with task scheduling to reduce both task completion time and task consumption. Bibal Benifa and Dejey (2017) proposed a scheduling strategy named efficient locality and replica aware scheduling (ELRAS) to reduce the execution time and improve the utilization of resources. Cameron et al. (2004) studied the impact of various task scheduling and data replication strategies. As a result, in terms of the resources of storage and computation, many scheduling algorithms considered both the file access cost of jobs and a load of computational resources. Yi et al. (2010) proposed a game theory based on a decentralized replication placement model and related algorithm to optimize the network load. Li et al. (2019) proposed a replica-aware task scheduling algorithm based on data replication. The experimental results showed that the proposed replica-aware task scheduling algorithm performed better than benchmark algorithms in terms of the node locality ratio and the job response time. Recently, as the joint optimization scheme, data placement combined with fault-tolerant scheduling has been studied in distributed cloud systems. And there were are already some corresponding literature.

Qin et al. (2018), Souli-Jbali et al. (2019) and Shi et al. (2021) combine data placement with fault-tolerant scheduling to improve the performance of distributed cloud computing. Yang et al. (Qin et al., 2018) proposed a fault-tolerant storage-based placement strategy to achieve load balance in data center networks. Subsequently, a priority-queue-based scheduling policy was designed to improve the efficiency of data access. Souli-Jbali et al. (2019) are interested in proving the impact of placing data in the cluster on the proposed fault tolerance protocol. And then job scheduling and data placement were combined to reduce the response time and improve the use of storage space. Shi et al. (2021) proposed a multi-domain random algorithm based on fault-tolerant domain (FTD) for data placement and task scheduling. And the results show that using FTDs not only effectively improves the data reliability of distributed storage systems but also promotes the effective utilization of storage space.

While existing task scheduling schemes or the joint schemes of data placement and task scheduling can reduce response time and energy consumption, and improve network resource utilization, the following shortcomings cannot be ignored. Xu and Lau (2017), Liu et al. (2017a) and Liu et al. (2016a) only consider replica creation based on task cloning methods. However, the replica technology adopted the load threshold is more suitable for task scheduling. Lu et al. (2018), Liu et al. (2016b, 2017b) and Tang et al. (2016) focus on the transmission consumption of fault-tolerant scheduling in the distributed cloud environment, but other energy consumption such as idle energy consumption, the work consumption of server, and the storage consumption should all be concerned. Otherwise, the total energy consumption of the geo-distributed cloud system cannot be accurately calculated. Bibal Benifa and Dejey (2017), Cameron et al. (2004), Yi et al. (2010) and Li et al. (2019) investigate task scheduling strategies based on replica creation and multi-objective optimization. However, a no-fault tolerance mechanism is introduced in the task scheduling scheme. As a result, the processing performance of the cluster may be significantly affected. Qin et al. (2018), Souli-Jbali et al. (2019) and Shi et al. (2021) proposed a joint optimization scheme of data placement and task scheduling. The fault tolerance mechanism was all considered to improve the performance in these joint schemes. However, the data placement schemes were implemented, the key factor of data dependencies in the same placed node was ignored. Therefore, the unnecessary data transmission was increased and the bandwidth resources were wasted. On the other hand, as the task was scheduled, the key factor of energy consumption was not considered, the transmission cost would be increased. Unlike the above studies, the advantages of our proposed task scheduling method are summarized as follows.

(1) Based on the load threshold, a replica creation method applicable to the task scheduling policy is proposed. A fault-tolerance mechanism is introduced into the task scheduling scheme. In this case, the proposed scheduling algorithm improves the system performance and resource utilization of the geo-distributed cloud system.

(2) A task scheduling method that considers the multi-objective optimization and the fault-tolerance mechanism is used. The key factors such as the idle energy consumption, the execution energy consumption, the transmission energy consumption, and the task duration are taken into account in the proposed method. In this way, the total energy consumption and the average completion time of task execution are reduced significantly.

## 3. Data placement optimization and fault-tolerant scheduling model based on the geo-distributed cloud environment

The architecture of the cloud system for fault-tolerant scheduling and data placement is introduced in Fig. 1. This system architecture includes a controller component and some clouds (Zhou et al., 2020). The central controller calculates the proper bandwidth between the data centers. The optimized strategy is obtained, while the optimization of the time is considered. The loud data centers are in different locations with good performance. Some nodes in the cluster with low performance results in slow execution of some tasks in the big data processing cluster. As a result, backward tasks tend to lengthen the task completion time. The speculative Spark-based execution method is used to determine the load state of the cluster in which the data resides. The task cloning method or the anomaly detection method is then selected to create replicas of the task. Finally, the task replica creation and scheduling (TSPT) algorithm is implemented for improving the electric energy and the average application completion time.

### 3.1. Optimal data placement model based on Lagrangian relaxation in geo-distributed cloud

Most geographically distributed clouds do not consider the load balancing and the capacity of the central cloud when the data is placed. To address this issue, a data placement strategy is proposed. First, a data placement model is developed. The optimization objective of this model is the data transmission time between cloud data centers. Meanwhile, the data transmission

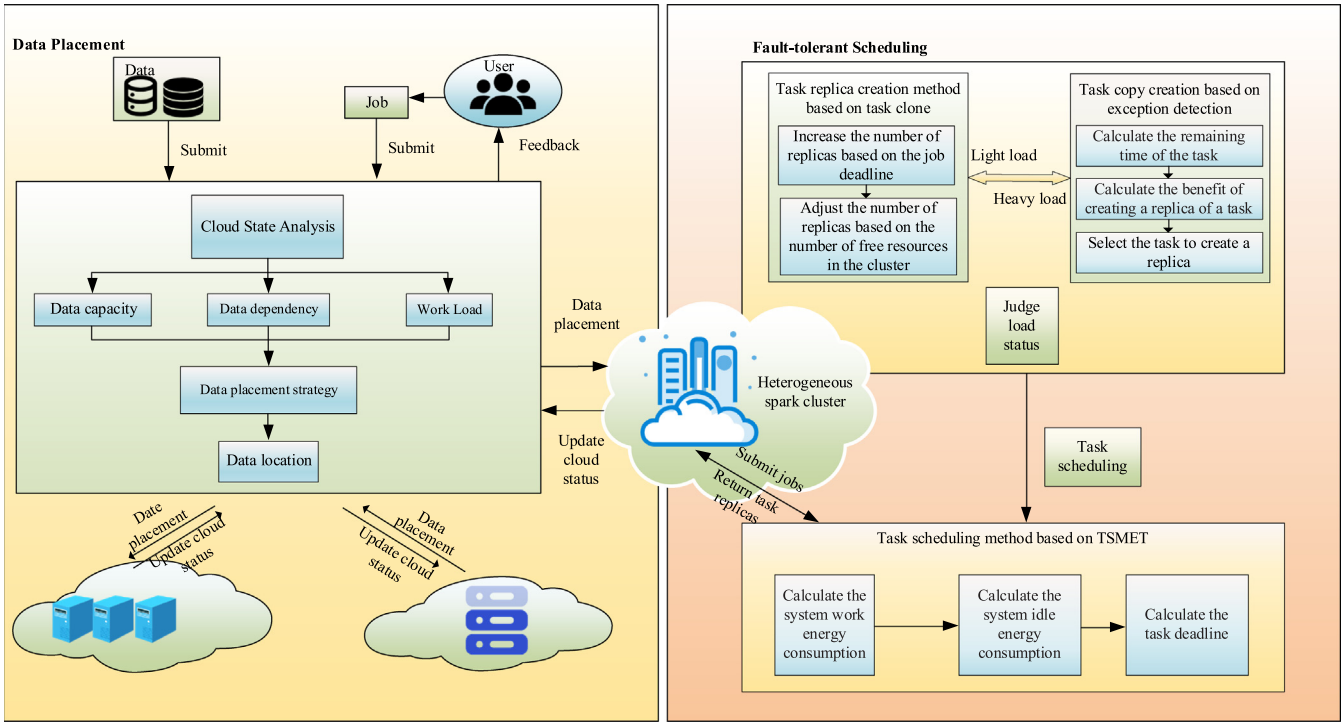


**Fig. 1.** The system architecture of data placement and fault-tolerant scheduling in the geo-distributed cloud environment.

cost, the cloud data center capacity, the data dependency, and the load balance are constraints. Then, the MCTB algorithm is implemented to obtain the optimal cost. Finally, the data placement solution is obtained by the proposed algorithm. The proposed method balances the system workload, improves the time overhead and the cost of the bandwidth.

In Fig. 2, a controller and multiple clouds exist in the system. During the phase of the data stream creation, according to the size of the data to be placed, the data information required for the job, and the bandwidth, the optimal cost of the bandwidth is calculated by the controller. Then, for obtaining the optimal time, the cost constraints, the data dependencies, the capacity constraints, and the load balancing constraints are used. Finally, data are placed by the optimal strategy.

### 3.1.1. Minimum transmission bandwidth cost solution based on MCTB model

In the system, data centers are not set up in the same location. Assuming that the geo-distributed cloud system $GC = \{GC_1, GC_2, \ldots, GC_I\}$ consists of $I$ clouds. The service capability and data capacity of each cloud are of great difference. The remaining capacity of each cloud is denoted as $\{RC_1, RC_2, \ldots, RC_I\}$. The numbers of physical machines in each cloud are $\{CS_1, CS_2, \ldots, CS_I\}$. Assuming that $J$ jobs are to be scheduled in $GC_i$, the jobs are denoted as $W = \{W_1, W_2, \ldots, W_j\}$. Therefore, assuming that $K$ data blocks need to be scheduled, and the dataset is denoted as $DB = \{DB_1, DB_2, \ldots, DB_k\}$. In the cloud data center $GC_i$, the total amount of data is less than $RC_i$, which represents the remaining capacity. The constraint is shown as follows Eq. (1).

$$RC_n \geq \sum_{j=1}^{J} \sum_{k=1}^{K} \delta_{ki} \cdot \chi_{kj} \cdot \varepsilon_{ki} \cdot DB_{k,i}, \tag{1}$$

where $\chi_{kj} \in \{0, 1\}$. Specially, when $\chi_{kj} = 1$, job $W_j$ needs data $DB_k$. Similarly, $\delta_{ki} \in \{0, 1\}$. When $\delta_{ki} = 1$ the job $W_j$ is scheduled to the cloud data center $GC_i$. $DB_{GC_k,GC_i}$ represents the number of data transferred from the cloud $GC_k$ to the cloud $GC_i$. $GC_k$ is the cloud which $DB_k$ is to be transferred to. $price_{GC_k,GC_i}$ represents the unit cost of transferring data. $F(DB_{k,i})$ is the frequency of user access to the transmitted data. Next, in each cloud, the cost of bandwidth $Trans_{cost}$ can be calculated as follows Eq. (2).

$$Trans_{cost} = \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{k=1}^{K} \delta_{ki} \cdot \chi_{kj} \cdot \varepsilon_{ki} \cdot F(DB_{k,i}) \cdot \frac{size(DB_{k,i})}{v} \cdot price_{GC_k,GC_i} \tag{2}$$

With the smallest cost of the bandwidth, the Minimum-Price-Transmission-Date (MCTB) algorithm based on the heuristics strategy is implemented for solving the data pre-placement issue. In the proposed strategy, based on the optimal cost of the bandwidth $mtc$ and $\gamma(\gamma \geq 1)$, the bandwidth cost limit is determined. The constraint is shown as follows Eq. (3).

$$Trans_{cost} \leq \gamma \cdot mtc. \tag{3}$$

For reducing the traffic overhead of data transmission across data centers, the data should be properly scheduled in advance. Assuming that the sent cloud is $a$, and the received data center is $b$, and the weight of the job from $a$ to $b$ can be expressed as follows.

$$Weight_b^a = \sum_{g}^{G} \sum_{j}^{J} (W_g \cdot W_j \cdot DB_{GC_k,GC_i}) \tag{4}$$

Assuming that the unit cost is determined, according to formula (4), the sum of the weights of the data center group sent out can be obtained as follows Eq. (5).

$$Weight_b = \sum_{b}^{I} (\varepsilon_{ki} \cdot Weight_b^a) \tag{5}$$

where $\varepsilon_{ki} \in \{0, 1\}$, $DB_{k,i}$ is the number of data transferred from the cloud $a$ to the cloud $b$. If $DB_{k,i}$ is not stored in the data center $b$, $\varepsilon_{ki}$ will be 1.

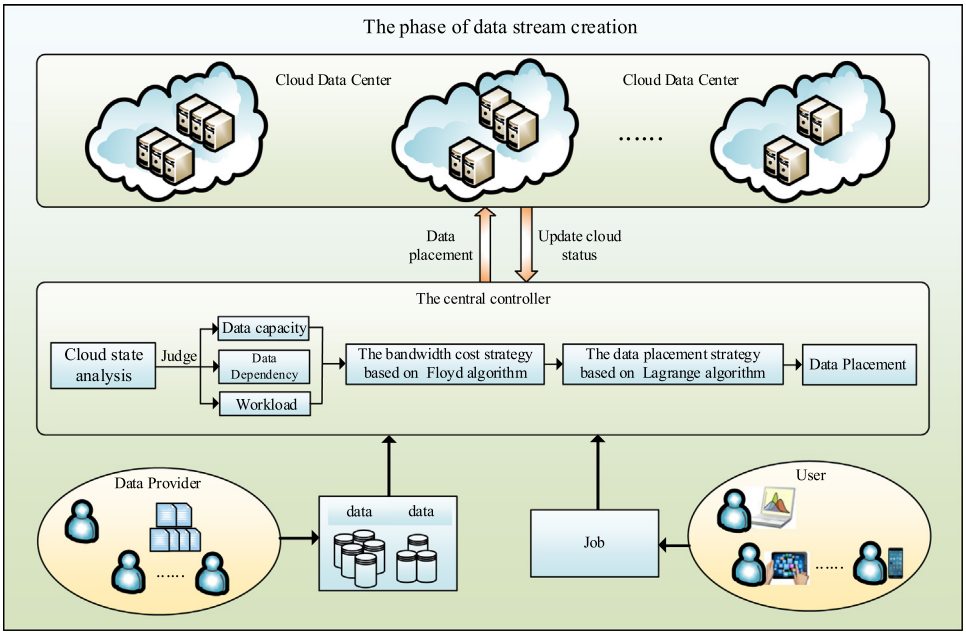


**Fig. 2.** Data placement process in the geo-distributed cloud system.

To obtain such a global minimum weight, the strategy of finding the minimum cut of the graph based on the stoer–wagner algorithm (Ferdaus et al., 2017) is introduced. Two adjacent points in the graph are either on one side of the cut or both sides. In the scenario where resources are allocated across data centers, for a node in the DAG graph of job, the nodes adjacent to it are either located in the same data center or other data centers.

When the MCTB algorithm is used to find the minimum bandwidth cost, a greedy algorithm based on heuristics is adopted. First, a sequence list in data centers exists. The sequence is used to determine the order in which data centers allocate resources. Then a task node of out-degree is selected to join the first data center group, and then the task node is deleted from the DAG.

Next, if a task node is selected to join the current data center and the cut value of the data center group i.e., the sum of all the out-degree can be smallest by this task node. Then, the task node is added to the current data center group, and the point is deleted from the DAG. The process is repeated until the tasks in the current data center group are full, then the same task allocation is performed for the next data center. After all the tasks are allocated to the last data center in this way, the task resource allocation strategy for the current data center sequence is obtained. The Weight based on the strategy is a greedy minimum value of the data center sequence. Since the unit price of the traffic for each cloud is not the same, and the number of resources leased by users in the data center is also different, each data center sequence gets a task resource allocation of the minimum weight based on the current data center sequence. To obtain the optimal allocation strategy of task resources, the various data center sequences need to be calculated. After the minimum allocation strategy and weight value corresponding to various data center sequences are obtained, the minimum weight value is selected as the ideal solution for the minimum traffic bandwidth overhead of the entire operation. Then, for the data block $DB_k$, the optimal cost of transmission is calculated as follows Eq. (6).

$$Trans_{cost}^{k,n} = \delta_{ki} \cdot \chi_{kj} \cdot \varepsilon_{ki} \cdot F(DB_{GC_k,GC_i}) \cdot \frac{size(DB_{GC_k,GC_i})}{v} \cdot price_{G_k,G_n}, \forall n \in N \tag{6}$$

In each cloud, the optimal cost of the bandwidth $mtc$ of transmitting the data is obtained as follows Eq. (7).

$$mtc = \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{k=1}^{K} Trans_{cost}^{k,n}. \tag{7}$$

### 3.1.2. Data placement based on Lagrangian relaxation

In this paper, the provider of the dataset includes users and service providers, the scale of the dataset may be very large. If we can reasonably place the data in advance, the bandwidth transmission cost can be significantly reduced. So the dataset can be placed on the data center that is most relevant to it, the correlation between dataset $DB_k$ and cloud center $GC_i$ is shown as follows Eq. (8).

$$GC\_dep = \sum_{j=1}^{J} \sum_{g=1}^{G} \eta_{gj} \left\{ W_g \cap W_j \right\} \times size_k \tag{8}$$

where $W_g$ denotes the job corresponding to the dataset $DB_k$ in the cloud data center $GC_i$, $W_j$ represents other jobs except for $W_g$, $k$ is the number of datasets in $GC_i$, $size_k$ is the size of all datasets in $GC_i$. $\eta_{gj}$ is an indicator function. If $W_g \cap W_j \neq \varnothing$, $\eta_{gj} = 1$, otherwise $\eta_{gj} = 0$.

Assuming that the CPU load of the processing data $DB_k$ is $\alpha_{cpu}^k$, the disk load is $\alpha_{disk}^k$, the memory load is $\alpha_{ram}^k$, $\lambda_1, \lambda_2, \lambda_3$ are the weight of CPU, disk, and memory load respectively, and $\lambda_1 + \lambda_2 + \lambda_3 = 1$. The geo-distributed cloud system is denoted as $GC$. According to the set of the pre-placement data, the load $WL$ of the system $GC$ is calculated as follows Eq. (9).

$$WL = \sum_{j=1}^{J} \sum_{k=1}^{K} \chi_{kj} \cdot \varepsilon_{ki} \cdot (\alpha_{cpu}^k \cdot \lambda_1 \cdot \alpha_{disk}^k \cdot \lambda_2 \cdot \alpha_{ram}^k \cdot \lambda_3) \tag{9}$$

Assuming that $CA$ is the total computing power of $GC$, the total computing resource usage of $GC$ is calculated as follows Eq. (10).

$$CR = WL/CA \times 100\% \tag{10}$$

A load of the cloud data center $GC_i$ is obtained as follows Eq. (11).

$$WL_i = \sum_{j=1}^{J}\sum_{k=1}^{K} \delta_{ki} \cdot \chi_{kj} \cdot \varepsilon_{ki} \cdot (\alpha_{cpu}^{k} \cdot \lambda_1 \cdot \alpha_{disk}^{k} \cdot \lambda_2 \cdot \alpha_{ram}^{k} \cdot \lambda_3). \quad (11)$$

Assuming that $CA_i$ is the capacity of computing of $GC_i$, the rate of the computing resource usage in $GC_i$ is calculated as follows Eq. (12).

$$CRO_i = WL_i / CA_i \times 100\% \quad (12)$$

In the geo-distributed cloud, to maintain the balance of the computing resources of each cloud data center, the computing resources $GC_i$ should be satisfied by the following condition Eq. (13).

$$\varpi_1 \cdot \frac{CS_i}{\sum_{k=1}^{I} CS_i} \cdot CRO_i \le WL_i \le \varpi_2 \cdot \frac{CS_i}{\sum_{k=1}^{I} CS_i} \cdot CRO_i \quad (13)$$

where $\varpi_1(\varpi_1 \le 1), \varpi_2(\varpi_2 \ge 1)$ are arguments of the load balance in $GC_i$.

When the job $W_j$ is executed, multiple data blocks need to be processed. When the job $W_j$ is scheduled to $GC_i$, all the data blocks are ready to be stored in $GC_i$, and the lost data blocks are ready to be scheduled from other clouds to the data center $GC_i$. The total transmission time is measured by the maximum transmission time in the transmitted data block. When the data is transferred, the transmission time $T_{ki}$ is calculated as follows Eq. (14).

$$T_{ki} = \max_{k \in N} \frac{\eta_{gj} \cdot \delta_{ki} \cdot \chi_{kj} \cdot \varepsilon_{ki} \cdot DB_{GC_k, GC_i}}{v}. \quad (14)$$

The optimization problem can be regarded as a linear programming problem. The Lagrangian relaxation method can be used to obtain the optimal solution. The Lagrangian relaxation can be formulated as follows Eq. (15).

$$\begin{aligned} &\min L(\gamma, \varphi, \phi) \\ &= \sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K} \delta_{ki} \cdot \chi_{kj} \cdot \varepsilon_{ki} \cdot \eta_{gj} \cdot CS_{GC_k, GC_i} - b_{GC_k, GC_i} + \\ &\sum_{i=1}^{I} \varphi_k \cdot \left( \sum_{j=1}^{J}\sum_{k=1}^{K} \delta_{ki} \cdot \chi_{kj} \cdot \varepsilon_{ki} \cdot \eta_{gj} \cdot CS_{GC_k, GC_i} - \right) + \\ &\phi \cdot \sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K} (\delta_{ki} \cdot \chi_{kj} \cdot \varepsilon_{ki} \cdot \eta_{gj} \cdot CS_{GC_k, GC_i} - \theta \cdot mtc), \end{aligned} \quad (15)$$

where $\varphi$ and $\phi$ are penalty factors. When objective Eq. (17) obtains the minimum value, for placing the original data, the best objective value is obtained for the objective function. For solving the Lagrangian lower bound issue, the core factor is to obtain the weighting parameters $\varphi$ and $\phi$. Finally, the sub-gradient method is used to obtain the optimal solution.

### 3.2. Task scheduling based on replica creation of fault-tolerant and TSPT model in geo-distributed cloud system

To reduce the delay of the entire application due to the abnormal executing of certain tasks, a fault-tolerant scheduling strategy for data flow applications in Spark clusters is proposed. The strategy not only improves the cluster processing performance but also achieves the goal of minimizing the job completion time and job resource consumption.

#### 3.2.1. The threshold of the load

As shown in Fig. 3, the research mainly includes replica creation and task scheduling. If the load is light, the speculative executing strategy based on the task cloning is adopted. Otherwise, the speculative executing strategy based on anomaly detection is adopted. Assuming that the node-set is $N = \{N_1, N_2, N_3, \ldots, N_U\}$ in the Spark system, the relevance among the job is ignored. Assuming that the jobs are set as $J = \{J_1, J_2, J_3, \ldots, J_H\}$one job comprises $m_h$ tasks and one task includes $r_h^n$ replicas. $\phi_g^h$ is set as the $j$th task of the $h$th job. Each node has only one processor and each processor only processes one task at any time. All the replicas are started in the meantime. $\tau_{aver}$ is the average completing time of a task $\phi_g^h$.

When the task arrives, the load status of each node in the server cluster is taken into account. The load status includes the CPU utilization, the network bandwidth, the memory utilization, the graphics card utilization, and the IO speed. In this paper, a dynamic load balancing algorithm is used to obtain the load threshold and determine whether the cluster workload is classified as light load or heavy load.

(1) Node hardware parameters

In a cluster, the CPU, the memory, the disk, the graphics card, and the network resources have different effects on the node load. Therefore, assuming that the impact factors for each parameter are $\chi_1$, $\chi_2$, $\chi_3$, $\chi_4$ and $\chi_5$, respectively. When the arrival task $\phi_g^h$ is processed, the utilization of the node $u$ is calculated as follows Eq. (16).

$$NL^u{}_g = \chi_1 C^u{}_g + \chi_2 M^u{}_g + \chi_3 D^u{}_g + \chi_4 V^u{}_g + \chi_5 N^u{}_g \quad (16)$$

where $C^u{}_g$, $M^u{}_g$, $D_g{}^u$, $V_g{}^u$ and $N_g{}^u$ are usages of the CPU, the memory, the disk, the graphics card, and the network resources when an arrived task is processed by the node $u$. The weight coefficient meets the following condition $\chi_1 + \chi_2 + \chi_3 + \chi_4 + \chi_5 = 1$.

In a time window, to count the CPU, the memory, the disk, and the network usage rates of all nodes in the system cluster, the calculation equations are shown as follows.

$$\begin{aligned} &ST_C = \sum_{u=1}^{U} C_u, ST_M = \sum_{u=1}^{U} M_u, \\ &ST_D = \sum_{u=1}^{U} D_u, ST_V = \sum_{u=1}^{U} V_u, ST_N = \sum_{u=1}^{U} N_u \end{aligned} \quad (17)$$

where $ST_C$, $ST_M$, $ST_D$, $ST_V$ and $ST_N$ respectively represent the total performance of the CPU, the memory, the disk, and the network resources in the cluster.

(2) Specific gravity of the load

The hardware parameters of each service node in the cluster system are inconsistent, and the system performance is also different. $NP_u$ is the load proportion of the node $u$, and is calculated as follows Eq. (18).

$$NP_u = \frac{\chi_1 \times C_g}{ST_C} + \frac{\chi_2 \times M_g}{ST_M} + \frac{\chi_3 \times D_g}{ST_D} + \frac{\chi_4 \times V_g}{ST_V} + \frac{\chi_5 \times N_g}{ST_N} \quad (18)$$

(2) Load weight

The load of the system node changes constantly with the client's request. The load weight corresponding to the node is also changed. The load weight of a node $u$ is calculated as follows Eq. (19)

$$NW_u = \frac{NP_u}{\sum_{u=1}^{U} NP_u}. \quad (19)$$

(3) The average completion time

The service cluster needs time to process the task. The shorter the business processing cycle, the stronger the system's ability to

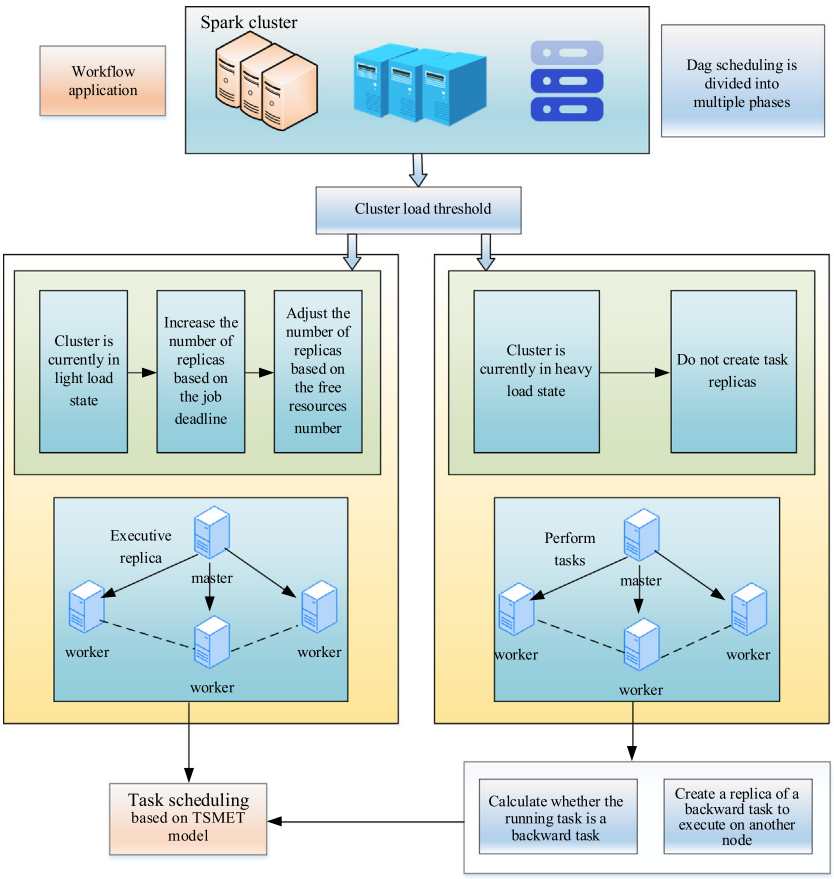


**Fig. 3.** Module structure of task replica creation for data flow in spark platform.

execute tasks. Assuming that the operating time of each node is $\tau(n_u)$, $\tau(n_u)$ is the shortest completion time of the executed task and $\tau_{\min}(n_u)$ is the shortest operating time of an executed task. The time set is $\tau_u$ for all the arriving tasks. The task completion time obeys the Pareto distribution. When $x \geq \tau_{\min}(n_u)$, $f_i(x) = 1 - (\tau_{\min}(n_u)/x)^{\mu}$, otherwise, $f_i(x) = 0$. According to the distribution function $f_i(x)$, the average completion time is calculated as follows Eq. (20).

$$\tau_u = [\tau(n_1), \tau(n_2) \ldots \tau(n_u)]^T,$$

$$\tau_{aver} = \int_0^{\infty} x df_i(x) = \int_0^{\infty} (1 - f_i(x))^{r^n_{m_h}} dx = \frac{\alpha r^n_{m_h} \tau_{\min}(n_u)}{\alpha r^n_{m_h} - 1} \quad (20)$$

When the cluster completes all jobs, the completion time is denoted as $\beta$ and is calculated as follows.

$$\beta = \sum_{h=1}^{H} \sum_{m=1}^{M} \sum_{n=1}^{N} r^n_{m_h} \tau_{aver}. \quad (21)$$

(4) The calculating of cluster load

In the distributed cluster, the arriving job set is denoted as $J = \{J_1, J_2, J_3, \ldots, J_W\}$. The set of task replicas corresponding to the job $J_W$ is $R = \{R_1, R_2, R_3, \ldots, R_N\}$. According to the current hardware status, the number of the arriving jobs, and the related task replicas, the cluster load is calculated as follows Eq. (22).

$$Load^N_W = \sum_{w=1}^{W} \sum_{n=1}^{N} \frac{NP_u \times NL_u}{NW_u} \times \frac{\tau(n_u)}{\tau_{aver}} \quad (22)$$

Where $NP_u$ is the load proportion of the node $u$, $NL_u$ is hardware usage of the node $u$, $NW_u$ is the load weight of the node $u$, $\tau(n_u)$ is the task completion time of the node $u$. The arriving task should have at least two replicas, because tasks without any additional replicas may delay the completion of the entire job. Therefore, $r^n_m \geq 2$, and $\tau_{aver}$ increases with the increase of $r^n_m$, the average rate of current cluster processing jobs is calculated as follows Eq. (23).

$$\frac{U}{\beta/H} = \frac{UH}{\sum_{h=1}^{H} \sum_{m=1}^{M} \sum_{n=1}^{N} r^h_m \tau_{aver}} \leq \frac{UH(2\mu - 1)}{4\mu \cdot \sum_{h=1}^{H} \tau_{\min}(n_u) m} = \delta. \quad (23)$$

Assuming that $\delta$ is the arrival rate threshold of the cluster load, where $1 < \mu < 2$. If the task arrival rate is not more than $\delta$, the system state is defined as light load. The inequality is shown as follows Eq. (24).

$$\sum_{u=1}^{U} L(node_u) \leq \delta \quad (24)$$

Otherwise, the system state is regarded as a heavy load. According to the different load states of the cluster, different replica creation methods should be used to reduce the impact of load on job completion time.

### 3.2.2. Speculative executing based on task cloning

If the current load arrival rate of the cluster is less than the threshold, some tasks are backed up by the task cloning method. To minimize the task response time, the number of task replicas should be achieved. The number of task replicas is $N$. The job deadlines are denoted as $DL = \{DL_1, DL_2, \ldots, DL_Z\}$ the response time of the job $J_i$ is denoted as $P_o$. Then, the condition is shown as follows Eq. (25).

$$P_o \leq DL_a, \forall a \in (1, Z). \quad (25)$$

When the replica is created, the available resources of the cluster should be considered. In addition, the cluster load rate cannot exceed the threshold. The number of the arriving job is denoted by $W$. The number of task replicas is denoted as $N$. Based on the equation(x), the cluster load is $Load_W^N$. By minimizing the job operating time and improving the resource utilization of the cluster, the optimization problem is formulated in detail Eqs. (26) and (27).

$$\min_{r_h^n} \sum_{n-1}^{H} P_o + \mu \cdot \sum_{n=1}^{H} \sum_{h=1}^{m_h} r_h^n \cdot t_g^h, \tag{26}$$

$$s.t. \quad (20), (23), (24), \tag{27}$$

where $\mu$ is the weighted parameter. For solving the above optimization problem, the optimized number of task replicas should be obtained. A two-dimensional array $A_{B\times C}$ is defined. Any element in the array $A_{B\times C}$ satisfies $a_{xy} \in \{0, 1\}$. If the cloning strategy of creating a few task replicas $R$ is accepted by the job $J_x$, it holds that $a_{xy} = 1$. Moreover, $1 \le x \le H$, $0 \le y \le R$. $H$ indicates the number of jobs, and $R$ is the largest number of replicas to be produced. As a result, any $a_{xy}$ in $A_{Q\times C}$ can be expressed as follows Eq. (28).

$$\forall x, \ a_{xy} = 1, a_{xy} \in \{0, 1\}, \tag{28}$$

As Fig. 3(b) is shown in Ananthanarayanan et al. (2013), when the number of task replicas is set as 4, the balance between the job straggling probability and the cluster workload can be achieved. For example, when the number of task replicas is set as 4, the job straggling probability is the lowest with the minimum number of task replicas. Based on this, the maximum number of task replicas is set to 4. Also, the relevant completion time of the job is steady. Based on the analysis above, the optimization goal is shown as follows Eqs. (29) and (30).

$$\min \sum_{x=1}^{H} \sum_{y=0}^{R} a_{xy} \cdot P_x + \mu \cdot \sum_{n=1}^{H} \sum_{h=1}^{m_h} r_h^n \cdot t_g^h, \tag{29}$$

$$s.t. \quad (21), (24), (25), (28). \tag{30}$$

### 3.2.3. Speculative executing based on anomaly detection

When the cluster load is bigger than the arriving rate $\delta$, the anomaly detection strategy is adopted. The speculative executing based on anomaly detection can monitor the processes of all tasks. And when a task is executed to be an anomaly or straggled, the backup task is to be started. As a result, anomaly detection is suitable under the circumstances of both low and high loads. Based on EWMA mode, the spark monitor system can calculate the speed of processing tasks of the node $node_u$. The speed of processing tasks is denoted as $v_{node(u)}$. And the specific process is described as follows. Every once in a while, the running node $u$ feedbacks the progress of the current task. Assuming that the feedback time is $tf_1$ and $tf_2$ respectively, the progress is $tp_1$ and $tp_2$ respectively. Then, at this stage, the current speed of the task is calculated as follows Eq. (31).

$$v_{node_u}(tf_2) = \eta \times v_{node_u}(tf_2) + (1-\eta) \times \frac{tp_2 - tp_1}{tf_2 - tf_1}, \eta = e^{\frac{tf_1 - tf_2}{\kappa}} \tag{31}$$

Where $(tp_2 - tp_1) / (tf_2 - tf)_1$ is the average processing speed during the two feedbacks, $\eta$ is the influencing factor of the speed at the previous moment. The interval between the two reporting tasks is uncertain. The longer the interval is, the influence on the speed of the previous moment is smaller. Because $tf_2 > tf_1$ the smaller $\kappa$ is, the smaller $\eta$ tends to be, which suggests that the task speed of the previous moment has less effect on it

As for the transformation of each type, the process can be divided into different stages. After each task is carried out, the complete task is recorded by the monitoring system. By analyzing to get the persisting time $T_{pr}$ during each phase, the average speed $V_{aver} = {}^1/_{T_{pr}}$ during each phase can be calculated to estimate whether the running tasks are possible anomaly tasks.

When all tasks of this type of job receive resources to get started, and when some tasks are completed, the selection and execution of speculative tasks can be started. At this time, the Spark monitoring system can get the information of both the running task speed and the completed tasks. To select the possible anomaly tasks from the running ones, the following two conditions need to be considered Eqs. (32) and (33):

$$V_{aver_{df}} \le V_{aver\_done} + V_{done\_sd} \tag{32}$$

$$V_{aver_{df}} \le V_{current\_aver} \tag{33}$$

where $V_{aver\_done}$ represents the average speed of the completed tasks during this phase, $V_{done\_sd}$ represents the standard deviation of the speed of the completed tasks, and $V_{current\_aver}$ represents the average speed of the running tasks during this phase. Inequality 2 indicates that when the current task speed lags far enough behind the completed task, the current task is regarded as an anomaly task. Inequality 3 means that when the current task belongs to the slower part of the running task, the current task is regarded as an anomaly task.

In a large-scale geo-distributed cloud system, the data center nodes are all heterogeneous. Both the total number of computers in the central cloud and the host performances are different. If we want to parameterize the nodes of each data center into sets $NS_i = \{data_i, error_i\ bw_i, deadline_i\ load_i\}$, $data_i$ is the amount of data entered when the node $i$ performs the task, $error_i$ stands for the error probability of the execution of the tasks, $bw_i$ is the bandwidth of the node $i$, $deadline_i$ is the deadline of the task execution, $load_i$ is the load of the node $i$. The value $price_i$ of the data center node is expressed as follows Eq. (34).

$$price_i = \omega \times bw_i + \rho_1 \times data_i + \rho_2 \times error_i + \rho_3 \times deadline_i + \rho_4 \times load_i \tag{34}$$

where $\omega > 0$, $\rho_1, \rho_2, \rho_3, \rho_4 < 0$, when the system needs to place a replica of the backup task at the selected node, all data center nodes send the $NS_i$ cluster to the master node Eq. (35).

$$Pro_{no\_back} = \varphi \times 0 - \varphi \times {}^1/_{v_{aver}} \tag{35}$$

where $\varphi$ is weight parameters of income and costs, respectively. If the backup task is not started on another node, only the resources of the current node are being occupied. Only when $price_i > Pro_{no\_back}$ the performance of the cluster processing jobs can be improved when a backup task is started. For the backup scheme that meets the above conditions, the backup startup scheme with the greatest benefits is selected.

### 3.2.4. Task scheduling strategy based on TSPT model

In this section, based on the speculative executing strategy, a TSPT algorithm is proposed. First, the target task is determined. According to the latest completion time, the submitted tasks are sorted in ascending order to obtain a new task queue. Time-urgent tasks always can be scheduled with priority. Second, at the head of the task queue, the minimum executing speed of the task is calculated, the virtual machine sequence is traversed, and find all virtual machines that meet the minimum executing speed of the current task are found. Third, when executing the current task, the energy consumption of each virtual machine is calculated, and the virtual machine with the least energy consumption is designated as the target virtual machine. Finally, the current task is assigned to the target virtual machine, and the task queue and virtual machine sequence are updated. Since the proposed

algorithm considers the target constraints, the energy consumption can be reduced, and the transmission time can be minimized. Meanwhile, tasks with higher urgency can be prioritized.

Assuming that the set of virtual machines is $VM = \{VM_1, VM_2, \ldots, VM_U\}$. The server can divide resources into idling and working states according to energy consumption. For the idle state, since the physical machine does not perform tasks, the energy consumption only includes the energy consumed by the system to maintain the operation of the corresponding state. However, when the physical machine is switched to executing state, the server needs to consume energy to maintain the corresponding state and work.

(1) Free Energy Consumption

The set of physical nodes in the cloud computing data center is defined as free energy consumption, which mainly includes the system consumption of CPU, graphics card, hard disk, memory, etc. The free energy consumption is related to the system itself and free time. The free energy consumption can be calculated as follows Eq. (36).

$$SC_e^{free} = \int_0^{T_e^{free}} P_e^{free} dt \tag{36}$$

where $S_e^{free}$ is the free energy consumption of server $S_e$, $P_e^{free}$ is the free power consumption (in W) of sever $S_e$, $T_e^{free}$ is the free time of sever $S_e$.

(2) Execution Energy Consumption

When a server $e$ is working, the energy consumption generated by the graphics card, motherboard, and fans is also included and marked as $OE_{lm}$. In the above case, the total energy consumption of the geo-distributed cloud system can be calculated as follows Eq. (37).

$$EC_{total} = \sum_{i=1}^{I}\sum_{e=1}^{E} SC_e^{free} + \sum_{i=1}^{I}\sum_{e=1}^{E}\sum_{m=1}^{M}(WE_{lm} + RWE_{lm} + ME_{lm} + OE_{lm}) \tag{37}$$

where $WE_{lm}$ is the energy consumption of computing resources, with the resources of CPU and memory, and graphics card included. $RWE_{lm}$ is the storage of energy consumption. When data is transmitted, $ME_{lm}$ refers to the energy consumption at the source node. At this time, the energy consumption is mainly related to the occupancy of network bandwidth.

(2) Minimum Executing Speed

Before the latest completion time, the minimum calculation speed of the CPU required for the task to be executed is calculated as follows Eq. (38).

$$MS_m = \frac{T_{size}^m}{T_{end}^m - T_{beginning}^m}. \tag{38}$$

(3) Real Executing Time

The time required for the task to be executed on the target virtual machine is calculated as follows Eq. (39).

$$RT_j^k = \frac{T_{size}^j}{V_{TC}^u}, \tag{39}$$

where $T_{size}^j$ represents the size of the $j$th task, $V_{TC}^u$ is the total computing power of the $u$th virtual machine, $RT_j^u$ is the actual executing time of the $j$th task on the $u$th virtual machine.

(4) Task Execution Time

The task execution time denotes the period from task submission to task completion, which is demonstrated as Eq. (40):

$$ET_j^k = T_{Beginning}^j + RT_j^k, \tag{40}$$

where $T_{Beginning}^j$ is the start time of the $j$th task, $ET_j^k$ represents the total time from submission to completion of the $j$th task.

(5) The average task execution time

The average task execution time can be denoted as follows Eq. (41).

$$CT_{aver} = \frac{\sum_{j=1}^{J} ET_j^k}{M}. \tag{41}$$

To minimize the total task execution time and system energy consumption, the task scheduling problem can be formulated as follows Eq. (42).

$$\begin{aligned} &\min\left(EC_{total} \times CT_{aver}\right) \\ &\text{s.t.} \quad RT_j^k \le T_{end}^j \\ &\forall j \in \{1, 2 \cdots J\}, \forall K \in \{1, 2 \cdots J\} \end{aligned} \tag{42}$$

## 4. Data placement optimization and fault-tolerant scheduling algorithm in a geo-distributed cloud environment

The relationship between data placement and task scheduling is shown in Fig. 4. In Fig. 4, the specific algorithm steps are described in detail. The whole process is divided into the phase of data stream creation and the phase of data stream execution. First, according to the workload and the data dependency, by using the proposed data placement algorithm, the data stream is created. Then, in the phase of data stream execution, the load threshold is calculated. Next, according to the load threshold, the current load is considered light or heavy. If the load is light, replicas are created based on the task cloning model. Otherwise, replicas are created based on the anomaly detection model. After replicas are created, according to the TSPT model, a task is scheduled. Finally, data placement results and task replicas are submitted to the heterogeneous spark cluster.

### 4.1. Data placement algorithm based on Lagrangian relaxation in geo-distributed cloud

The code of the data placement algorithm based on the relaxation scheme is shown in Algorithm 1. Firstly, the matrix of the shortest path between the vertices is achieved (Algorithm 1 Line 2∼5). And then the bandwidth cost is saved based on the algorithm of the shortest path between vertices (Algorithm 1 Line 9∼10). Secondly, the values of the penalty factors are updated according to the gradient method. Then, by using the updated penalty factors $\varphi_k$ and $\phi$, the problem of the solution is achieved. Subsequently, the calculated data placement scheme is obtained (Algorithm 1 Lines 11∼15). Then the time of content transmission is obtained (Algorithm 1 Lines 16). Assuming that $N$ is the data blocks number in the dataset, $num$ is the iterations number of the solution process.

### 4.2. Fault-tolerant scheduling algorithm description based on speculative executing and TSPT in spark cluster

#### 4.2.1. Task replica creation algorithm based on task clone

The task replica creation algorithm based on task cloning in the Spark system is shown in Algorithm 2. The algorithm is executed as follows. An initial number of the task replicas is set to 2 (Algorithm 2 Line 1). Second, the set of jobs to be executed is traversed. According to the deadline set, the timeout jobs are found. By considering the load arrival rate threshold and the maximum number of task replicas, the job that may time out is obtained, and the replicas of the job are increased (Algorithm 2 Line 2∼8). Third, the total number of replicas currently created is calculated (Algorithm 2 Line 9). Finally, based on the total amount

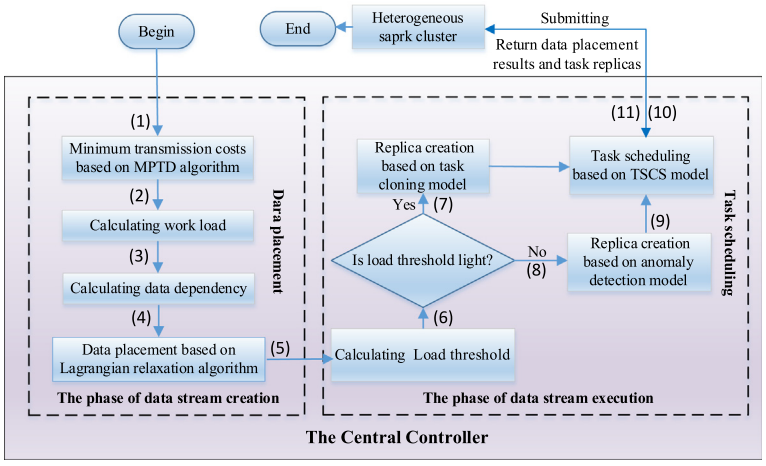


**Fig. 4.** The relationship between data placement and fault-tolerant scheduling.

**Algorithm 1: The optimized algorithm of data placement**

**Input:** $DG=(P,L)$

$\mathrm{L}(\delta,\varphi,\phi)$, $num$

**Output:** $\eta^{o}$, $\tau$

1 The adjacency matrix $R_{m\times m}$ is Initialized

2 **while** $(w=1;w\le m;w++)$ {

3 **while** $(u=1;u\le m;u++)$, $(v=1;u\le m;v++)$

4 **if** $R[u,v]>R[u,w]+R[w,v]$ **then**

5 $A[u,v]=A[u,w]+A[w,v]$

6 **end if**

7 **end while**

8 **end while}**

9 $bCost_{nk}=\gamma_{jk}\cdot\alpha_{jn}\cdot S_{G_n,G_k}\cdot a_{G_n,k}$, $\forall j\in J$

10 $opt=\sum_{k=1}^{K}\sum_{m=1}^{M}\sum_{n=1}^{N}bCost_{nk}$

11 $\varphi_k=0$, $\phi=0$, $\gamma^{*}=\mathrm{L}^{-1}(\delta,0,0)$

12 **while** the iteration number is less than $num$ and $\mathrm{L}^{-1}(\gamma,\varphi_k,\phi)$ does not convergence **do**

13 $\varphi_k$ and $\phi$ are updated by the sub-gradient method // The solution of the objective function is calculated.

14 $\gamma^{*}=\mathrm{L}^{-1}(\delta,\varphi_k,\phi)$

15 **end while**

16 $t=\max\dfrac{\delta_{mk}\cdot\alpha_{mn}\cdot D_{G_n,G_k}}{k}$ // The transmission time is calculated.

17 **return** $\eta^{o}$, $\tau$

of replicas and the replica number per task, the number of idle actuators in the cluster is calculated. When free executors exist, all the job sets are iterated through and the number of replicas is increased. Otherwise, the replica and the number of replicas are returned (Algorithm 2 Line 10∼16).

*4.2.2. Replica creation algorithm based on anomaly detection*

The task replica creation algorithm based on the anomaly detection in the Spark cluster is shown in Algorithm 3. Firstly, the set of jobs that are about to be executed is traversed. For the transformation operator, by using the estimated speed and the amount of data remaining, the remaining time is calculated. For the action operator, the remaining data amount and the estimated processing rate of the current phase node are used to calculate the remaining time. Then, by calculating whether the task is time-out or not, the task can be judged as the backward task (Algorithm 3 Line 1∼15). Second, for each backward task, the benefits of creating a replica and not creating a replica are calculated. Then, whether these backward tasks need to be backed up is judged by the benefit (Algorithm 3 Line 16∼24). Third, the task replica backup revenue is sorted (Algorithm 3 Line 25). Finally, the backward task with the largest backup benefit is selected,

**Algorithm 2: Task clone creation algorithm based on task cloning in Spark cluster**

**Input:** The job set $J$; The load threshold $\lambda$.
The number $R$ of currently idle executors.
The deadline set DL.

**Output:** $c[N]$ // The task replica set $c[i]$ is the replica number of the tasks in the $i$-th job.

1 $c[N] \leftarrow \{2,2,2,...,2\}, j \leftarrow 0$ // The initial value of replicas of all jobs is set to 2
2 **for each** $i \in [1,Q]$ **do**
3 **if** $\sum^{U} L(node_u) \le \delta$ **then** // According to Eq. 21, the $Load_W^N$ is calculated.
4 **if** $P_o \le DL_a$ and $r[n] < 4$ **then** // Jobs timed out are found, and the corresponding replicas are added.
5 $c[i]++$
6 **end if**
7 **end if**
8 **end for each**
9 *Total cloning tasks* $\leftarrow \sum_{h=1}^{H} m_h \bullet R[n]$
10 **while** $j < 4$ **do**
11 **for each** $i \in [1,H]$ **do** // The idle resources are made full use of.
12 **if** $sum\_cloning + m_i \le R \&\& c[n] < 4$ **do**
13 $c[i]++, sum\_cloning += m_i, j++$
14 **end if**
15 **end for each**
16 **end while**
17 **return** $c[N]$

**Table 1**
Server hardware configuration table.

| Server name | CPU cores | RAM | Disk | Role |
|---|---|---|---|---|
| node01 | 68 cores | 240 GB | 16 TB | Local server |
| node02∼node05 | 68 cores | 240 GB | 16 TB | Remote servers |

and a replica of the task is created to execute on other nodes (Algorithm 3 Line 26).

*4.2.3. The TSPT algorithm based on speculative executing*

In the Spark cluster, for data stream applications, Algorithm 4 represents the fault-tolerant scheduling algorithm based on speculative executing. First, the TSPT algorithm is implemented to schedule tasks. The submitted tasks are sorted in ascending order of the latest completion time, and a new task queue is obtained. Subsequently, all the virtual machines that can meet the minimum executing speed of the current task are selected. Meanwhile, the virtual machine with the least energy consumption and average time of executed task is determined as the target virtual machine (Algorithm 4Line 7∼14). Finally, the remaining time and the backup revenue that is judged as a slow task are calculated. The task with the most revenue backup is selected to create a replica and execute on other nodes (Algorithm 4 Line 16∼19).

## 5. Experiments

The combination of data placement and task scheduling can improve the performance of running data stream and reduce the response latency. Thus, the integrated model of data placement and task scheduling is introduced. As illustrated in Fig. 4, the data placement scheme and the task scheduling scheme are in the two dotted rectangles respectively. The main procedures of the data placement and task scheduling integration model are described as follows. In the phase of data stream creation, In the beginning, the user makes a request or data provider supplies to the data center. (1) In the central controller, the MCTB algorithm is implemented. (2) The workload will be calculated to make the system load balance. (3) The data dependency will be achieved to place related datasets into the same data center. (4) The Lagrangian relaxation algorithm is implemented. After data placement, the computing resources will be allocated to tasks. In the phase of data stream execution. (5) The selection method of replica creation is based on a load threshold. (6) The load threshold is used to judge the system status in terms of light and heavy. (7) If the load threshold is light, the replica creation algorithm based on task cloning will be executed. (8) If the load threshold is heavy, the replica creation algorithm based on anomaly detection will be executed. (9) The task scheduling algorithm based on the TSPT model will be executed. (10) The scheme of data placement and task scheduling will be submitted. (11) The data placement results and task replicas are returned.

### 5.1. Experiment environment

**(1) Experiment Setup**

The Spark platform and the Hadoop distributed platform are used to run our algorithms. JDK1.8.0 181, Scala 2.10.6, Eclipse 5.5.0, Apache Spark 1.7.0, and Apache Hadoop 2.6.0 make up the software environment. In the experiment environment, a

**Algorithm 3: Task replica creation algorithm based on anomaly detection in Spark cluster**

**Input:** *TaskInformation* // Information collection of running tasks

**Output:** *replica creation strategy* // The task of creating a replica

1 *speculative task* ← *null*

2 **for each** *task* ∈ *TaskInformation* **do**

4 **if** *task* is a *transformation task* **then**

5 $V_{aver} = 1/T_{...}$ // During each phase of task execution, the average speed is calculated.

6 **if (** $V_{aver_{sf}} \le V_{aver\_done} + V_{done\_sd}$ or $V_{aver_{sf}} \le V_{current\_aver}$ **) then**

7 *speculative task*+*task* = *task*

8 **end if**

9 **else**

10 $RT_{ac} = RT_{cp} + RT_{fp}$

11 **if (** $V_{aver_{sf}} \le V_{aver\_done} + V_{done\_sd}$ or $V_{aver_{sf}} \le V_{current\_aver}$ **) then**

12 *speculative task*+=*task*

13 **end if**

14 **end if** $Pro_{no\_back} = \varphi \times 0 - \varphi \times 1/V$

15 **end for**

16 **for each** *task* ∈ *speculative task* **do**

17 $price_i \leftarrow \omega \times bw_i + \rho_1 \times data_i + \rho_2 \times error_i + \rho_3 \times deadline_i + \rho_4 \times load_i$ //The node price is calculated.

17 **if (** $price_i > Pro_{no\_back}$ **) then**

18 A replica is created.

19 **else**

20 The replica is not created.

21 **end if**

24 **end for**

25 Tasks is in ascending order.

26 *replica creation strategy* ← *task whose price is largest*

27 return replica creation strategy, node

virtual machine cluster is developed on the local server, a virtual machine cluster is developed on the remote server, and a local cluster exists.

The local cluster environment is established by three local physical machines. Eleven virtual computers make up the cluster generated on the local server. On the remote servers, 29 virtual machines make up the remote cloud. The server hardware setup information is listed in Table 1.

Local clusters and remote clouds are in different geographical locations. Each cloud has one master node and several child nodes, and communication between different clouds is possible. The processes of the data placement are managed by the central controller. Fig. 5 depicts the experimental setting.

The configuration of the cloud data center is shown in Table 2.

**(2) Test Case**

Facebook is a world-famous social networking site, loved by people and widely used. So there is a strong practical significance for using Facebook dataset as a test case. In the geo-distributed cloud environment, for evaluating the effectiveness of the proposed data placement method, many related works (Khalajzadeh et al., 2016, 2020; Zhang et al., 2018) take the online social network Facebook as the dataset. In our experiment, the online social network Facebook on October 23, 2018, is taken as the dataset of the data placement strategy, (Anon, 2018a). This dataset consists of user networks from Facebook. Facebook data was collected from survey participants using the Facebook application. The dataset includes node features, circles, and ego networks. The data has been anonymized by replacing the Facebook-internal ids for each user with a new value. In addition, the feature vectors from this dataset are provided, so the interpretation of those features is obscured. More than 1000 data blocks ranging in [16 MB, 128 MB] are contained in this dataset. In each experiment, some data blocks are chosen to be transmitted to the whole system.

For the fault-tolerant scheduling algorithm based on speculation executing, the PageRank application and the Inverted-Index application are used as test benchmarks to verify the feasibility of the algorithm (Hu et al., 2017b). In the experiment of Hu et al. (2017b), the Google undirected graph data and the Wikipedia data are taken as the test data. The PageRank and the Inverted-Index are used to verify the performances of the algorithms. For

**Algorithm 4: The TSPT algorithm based on the speculative executing model**

**Input** : The set of jobs $J$ that arrive in the cluster, Information collection of running tasks *TaskInfo*, $T_{End}$
$T=\{T_1,T_2,...,T_m\}$, $VM=\{VM_1,VM_2,...,VM_u\}$

**Output** : Replica creation result **c**

1 Sort the task queue $T$ in ascending order by the deadline
2 **for each** $VM_u \in VM$ **do**
3 $V_{TC}^u \leftarrow$ Total computing power of u-th virtual machine
4. **end for**
5 **for each** $T_m \in T$ **do**
6 **for each** $VM_u \in VM$ **do**
7. **if** $MS_m \le V_{TC}^u$ **then**
8 $RT_j^u = T_{size}^j / V_{TC}^u$
9 $CT_{total} = \sum_{j=1}^{J} ET_j^u$
10 $EC_{total} \leftarrow$ The total energy consumption of the Geo-distributed Cloud System
11 $(EC_{total} \times CT_{total}) \leftarrow$ Objective function value // The remaining time is calculated by Eq. 31.
12 **end if**
13 $u$++
14 The virtual machine with the smallest $(EC_{total} \times CT_{aver})$ is selected.
15 The task is assigned to this virtual machine
15 **end for**
16 $m$++
17 **end for**
18 The exception task is executed with a task replica creation algorithm based on anomaly detection.
19 **return** Replica creation result **c**

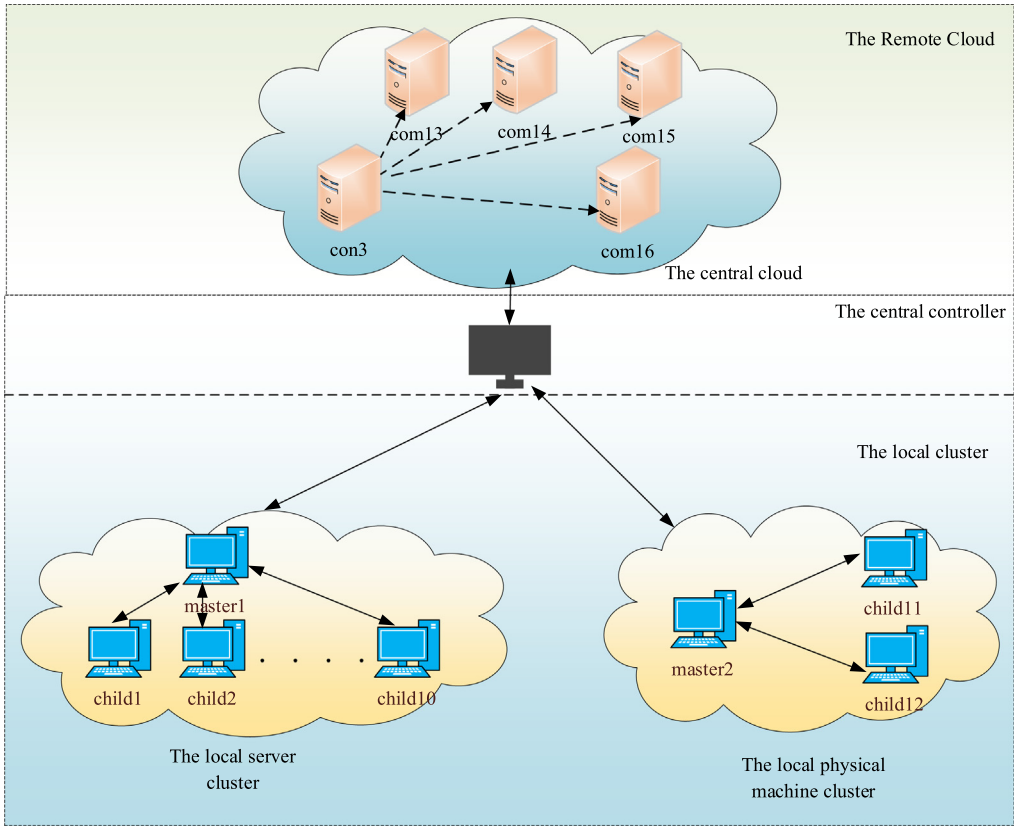


**Fig. 5.** Experimental environment.

**Table 2**
Experimental environment configuration table.

| Hostname | CPU core | RAM | Disk | Node IP |
|---|---|---|---|---|
| controller | 6 cores | 8 GB | 2 TB | 10.141.82.30 |
| master1 | 6 cores | 16 GB | 2 TB | 10.138.116.227 |
| worker1∼worker 10 | 6 cores | 8 GB | 1 TB | 10.168.203.20∼10.168.203.29 |
| master2 | 12 cores | 16 GB | 2 TB | 10.141.31.237 |
| worker 11∼worker 12 | 4 cores | 4 GB | 1 TB | 10.141.17.22∼10.141.68.126 |
| master3 | 12 cores | 16 GB | 2 TB | 10.141.16.172 |
| worker 13∼worker18 | 12 cores | 16 GB | 2 TB | 10. 141.123.83∼10. 141.128.88 |
| worker 19∼worker24 | 12 cores | 8 GB | 2 TB | 10. 141.128.89∼10. 141.128.94 |
| worker 25∼worker32 | 6 cores | 4 GB | 1 TB | 10. 141.128.95∼10. 141.128.102 |
| worker 33∼worker40 | 8 cores | 1 GB | 1 TB | 10.141.128.103∼10.141.128.110 |

the Page Rank, in the Google undirected graph data, the range of vertices is [82424, 284581], and the range of the edge numbers are [139081, 316312] (Leskovec et al., 2009; Anon, 2018b). For the Inverted-Index, 2 GB 128 GB of the Wikipedia data is used, and the range of the data size is [2 GB,128 GB] (Anon, 2018c).

**(3) Performance Metrics**

In multiple cloud storage environments, each data center is set up in a different geographical region. To facilitate subsequent access to data and improve the reliability of the system, the system stores user data in multiple data centers depending on the actual situation. For reducing the time overhead caused by transfers across regional data centers, the replicas are stored in multiple data centers. However, at the same time, the storage overhead increases. With the rapid increase in the number of users and increasingly frequent access to the cloud storage system, for cloud service providers, efficient data storage is of great importance. The storage overhead per file is defined as follows Eq. (43).

$$Cost_f^{sto} = N_f^r C_0^{sto} S_f \tag{43}$$

where $N_f^r$ is the number of replicas of the file $i$, $C_0^{sto}$ is the storage overhead per kilobyte, and $S_f$ is the size of the file $f$ in kb.

The storage overhead of a data center depends on the number of users and the number of replicas uploaded by users. As the number of users increases rapidly, the cost of data storage increases. The total storage overhead is a summation of all storage cases for each file. The total storage overhead is expressed as follows Eq. (44).

$$Cost^{sta} = \sum_{f=1}^{F} Cost_i^{sto} \tag{44}$$

Where $F$ is the number of files.

Typically, the network bandwidth available during file transfer is limited. When the network transfers multiple data files at the same time, congestion is likely to occur. The occurrence of congestion leads to an increase in access latency time. Assume that the occurrence of congestion per unit of time is an equal probability event, denoted as $P$. Therefore, the further away the user accesses the data, the greater the probability of congestion occurring. Based on the above assumptions, the expectation of congestion occurring is calculated as follows.

$$E = \frac{\min_{u\in\mathcal{U}, i\in\mathcal{I}} distance_{u,i}}{d_0} \cdot p \tag{45}$$

where $d_{u,i}$ is the distance from the user $u$ to the data center $i$, $\mathcal{U}$ is the set of users, $\mathcal{I}$ is the set of data centers, $d_0$ is the unit distance in kilometers.

If congestion occurs, the transmission rate of accessing replica decreases, and the access latency increases. The parameter $\varepsilon$ is set as the ratio of $v(t+1)$ to $v(t)$ when congestion occurs after the time $t$. The occurrence of congestion reduces the data transfer rate. Combined with the expected value of the occurrence of congestion, the transmission rate is defined as follows Eq. (46).

$$v = \varepsilon^E v_0 \tag{46}$$

where $v_0$ is the initial transmission rate, $\varepsilon = v(t+1)/v(t)$. $v(t)$ is the transmission rate of the replica at the time $t$. Therefore, the replica transmission time is related to the replica size and the distance from the user to the data center and is calculated as follows Eq. (47).

$$T = \frac{S_f}{v}. \tag{47}$$

where $S_f$ is the replica size of the file $f$.

The load balancing degree $LBD^{dp}$ takes into account the capacity and load of the whole system as well as the individual data centers and is calculated as follows Eq. (48).

$$LBD^{dp} = \min_{i\in\mathcal{I}} \left| \frac{C - S}{C_i \cdot S - S_i \cdot C} \right|, \tag{48}$$

where $C$ represents the total capacity of the geo-distributed cloud, $S$ is the total size of the dataset, $C_i$ represents the capacity of the $i$th cloud data center, and $S_i$ is the total size of the data transferred to the $i$th cloud data center. The higher the load balancing degree $LBD$, the more balanced the load of the whole system.

In addition, the CPU utilization and the disk utilization are taken as the performance metrics of the data placement method. The CPU utilization of the none-idle physical nodes is denoted as $U^{CPU} = \{U_1^{CPU}, U_2^{CPU}, \ldots, U_O^{CPU}\}$, where $O$ is the number of none-idle physical nodes. The disk utilization of the none-idle physical nodes is denoted as $U^{disk} = \{U_1^{disk}, U_2^{disk}, \ldots, U_O^{disk}\}$ The average CPU utilization is represented by the average CPU utilization of all non-idle physical nodes, and calculated as follows Eq. (49).

$$\overline{U^{CPU}} = \sum_{o=1}^{O} U_o^{CPU} \tag{49}$$

The average disk utilization is represented by the average disk utilization of all non-idle physical nodes and calculated as follows Eq. (50).

$$\overline{U^{disk}} = \sum_{o=1}^{O} U_o^{disk} \tag{50}$$

In the fault-tolerant scheduling algorithm, the completion time, the load balance degree, and the total energy consumption are taken as the performance metrics. In the formula (21), the completion time $\beta$ is defined.

The load balance degree $LBD^{fs}$ is calculated as follows Eq. (51).

$$LBD^{fs} = \frac{\sum_{u=1}^{U} t_u}{U * \beta} \tag{51}$$

where $U$ is the number of the nodes, $t_u$ is the completion time of the node $u$. $\beta$ is the completion time of the whole system, which is associated with the node that processes tasks the slowest.

The total energy consumption is calculated as follows Eq. (52).

$$E = \sum_{u=1}^{U} E_u \tag{52}$$

where $E_u$ is the energy consumption of the node $u$.

The QoS satisfaction rate $QoS$ is the percentage of the job execution time as a percentage of the total time. According to the system execution capability, the job deadline is adjusted. The higher the $QS$ stronger the cluster processing performance. $QoS$ is calculated as follows Eq. (53).

$$QoS = (N_{meet}/N_{total}) \times 100\% \tag{53}$$

where $N_{meet}$ is the number of jobs that meet the task deadline, and $N_{total}$ is the total number of the job.

**(4) Benchmark Algorithms**

For assessing the effectiveness of the data placement and fault-tolerant scheduling algorithms, the proposed algorithms are compared with the benchmark algorithms in the comparative experiments.

For evaluating the performance of the data placement algorithm based on Lagrangian relaxation (DPLR), the self-adaptive discrete particle swarm optimization algorithm with genetic algorithm operators (GA-DPSO) (Lin et al., 2019), the graph-partitioning based data placement algorithm (GPDP) (Khalajzadeh et al., 2017b), and the exact federation data placement algorithm based on integer linear programming model (ILP-FDP) (Ikken et al., 2017) is selected as the benchmark algorithms.

In the GA-DPSO algorithm, when the data is placed for a scientific workflow, the data transmission time is optimized. The GA-DPSO algorithm considers the characteristics of edge computing and cloud computing. The bandwidth between data centers (DCs), the number of edge data centers (DCS), and the storage capacity of edge data centers (DCs) are also taken into account in the GA-DPSO algorithm. To minimize the cost while satisfying the latency requirement, the GPDP algorithm is proposed to find a near-optimal data placement of replicas. The ILP-FDP algorithm considers the scientific user requirements, the data dependency, and the data size.

For verifying the performance of the proposed fault-tolerant scheduling algorithm, the fault-tolerant scheduling algorithm that considers the task replica creation and scheduling (TSPT) is compared with the efficient one-dimensional search algorithm (EODS) (Liu et al., 2016b), the multi-task scheduling algorithm (MTS) (Liu et al., 2017b), and the energy-efficient workflow task scheduling algorithm (EWTS) (Tang et al., 2016).

The EODS algorithm is proposed to find the optimal task scheduling policy. The computation tasks are scheduled according to the task buffer state, the state of the local processing unit, and the transmission state. In addition, the average delay of each task and the average power consumption at the mobile device are analyzed in the EODS algorithm. In the MTS algorithm, the task workload is modeled, the efficiency and quantity of service are considered. For reducing energy waste and maintaining the QoS while meeting the deadlines, the EWTS is proposed. In the EWTS algorithm, the running status of the relatively inefficient processors is optimized.

### 5.2. Results and analysis

#### 5.2.1. Data placement algorithm based on Lagrangian relaxation

For evaluating the performance of the proposed data placement algorithm, the total storage overhead, the data transmission time, the CPU utilization, and the disk utilization is taken as the performance metrics. To verify the advancement of the proposed data placement algorithm, the DPLR algorithm is compared with the GA-DPSO algorithm, the GPDP algorithm, and the ILP-FDP algorithm. Each experiment is repeated 20 times, and the average is taken as the final result.

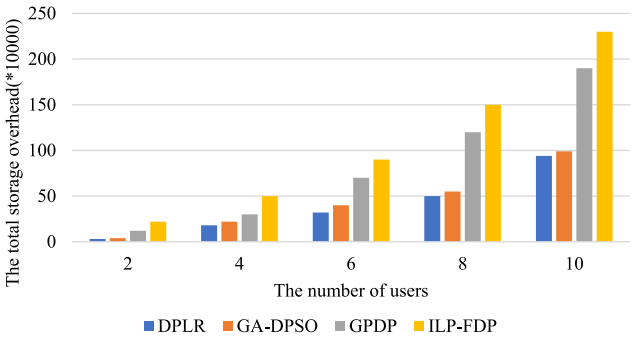


**Fig. 6.** The impact of different user numbers on the total storage overhead.

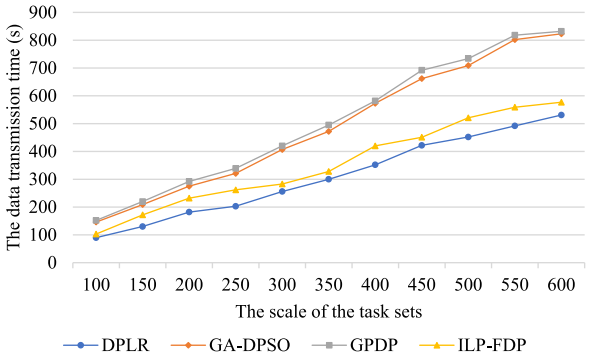


**Fig. 7.** The data transmission time varies from the scale of the task sets.

Fig. 6 depicts the impact of different user volumes on different metrics. The horizontal coordinate is the number of users using the cloud storage service. The range of the user number is set to 2000 to 10000 to test the performance of the DPLR algorithm.

When the number of users reaches 10,000, the total storage overhead of the DPLR algorithm is half that of the ILP-FDP algorithm. In Fig. 8, the storage overhead of the DPLR algorithm shows a linear increase. The DPLR algorithm has the lowest overhead of all the compared algorithms for eight different user counts.

In Fig. 7, the data placement algorithms are compared in terms of the data transmission time. The data transmission time varies from the scale of the task sets. The scale of the task sets is set from 100 to 600, and the experiment is conducted for each additional 50 tasks.

The DPLR algorithm has the least transmission time and the GPDP algorithm has the most. The GA-DPSO algorithm takes slightly less time to execute than the GPDP algorithm. The time growth rate of the DPLR algorithm is smaller than the other algorithms compared. The transmission time of the DPLR algorithm is reduced by approximately one-third compared to the GPDP algorithm, and the difference in time between the two algorithms is exaggerated as the number of nodes increases.

To minimize the cost while meeting the latency requirements, the GPDP algorithm is a new approach based on graphical partitioning for optimal replica data placement. However, the bandwidth and the data dependencies are not taken into account in the GPDP algorithm. The ILP-FDP algorithm takes the number of transfers as a criterion. In the ILP-FDP algorithm, based on the correlation between the data, the data is transferred to the

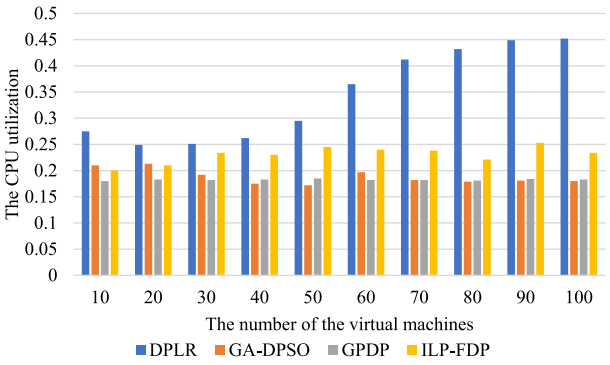


**Fig. 8.** The CPU utilization varies from the number of the virtual machines.

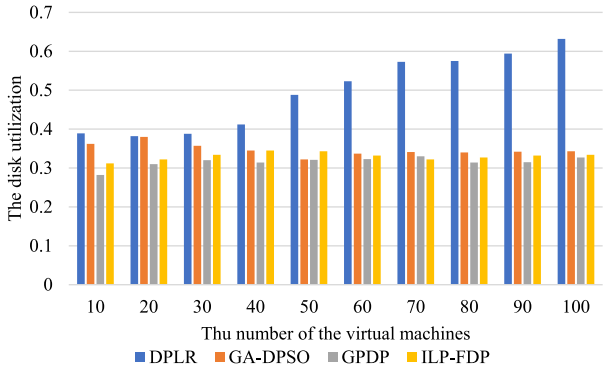


**Fig. 9.** The disk utilization varies from the number of the virtual machines.

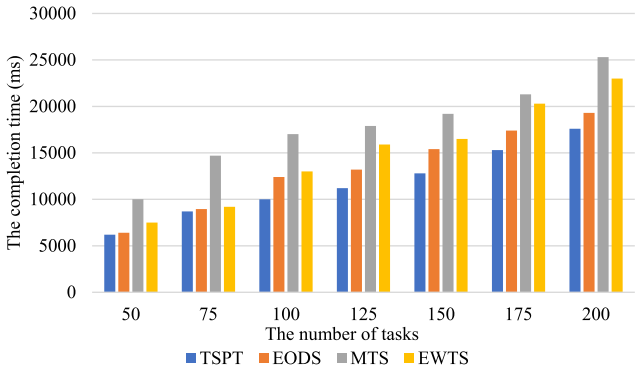


**Fig. 10.** The completion time varies from the number of tasks.

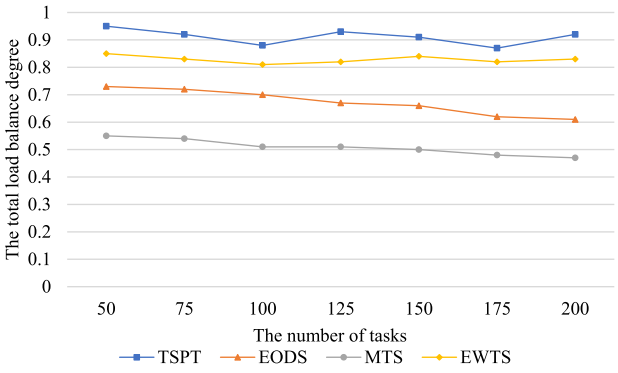


**Fig. 11.** The total load balance degree varies from the number of tasks.

node with which the data is most correlated. The ILP-FDP algorithm invokes the task with the node that has the least amount of transfers. But the ILP-FDP algorithm does not consider the bandwidth. The DPLR algorithm uses the minimum transmission time as a criterion and considers the impact of bandwidth and data dependency on transmission time. The data is placed at the node with the least transmission time. In addition, the number of data replicas of the DPLR algorithm is dynamically adjusted to enhance the data localization and reduce the data transmission requirements. Therefore, the proposed DPLR algorithm has the least transmission time.

In Figs. 8 and 9, the CPU utilization and disk utilization of physical nodes are compared after virtual machines are placed by different algorithms.

In Figs. 8 and 9, for different numbers of virtual machines, compared to the GA-DPSO, GPDP, and ILP-FDP algorithms, the DPLR algorithm in this paper results in improved CPU utilization and disk utilization of the physical nodes in the cloud data center. When the number of virtual machines increases to 100, the CPU utilization of the DPLR algorithm is approximately double that of the benchmark algorithms.

In terms of disk utilization, when the number of virtual machines is between 10 and 50, the change in disk utilization for each algorithm is not significant. Once the number of virtual machines exceeds 40, as the number of VMs increases, the disk utilization of the DPLR algorithm tends to increase significantly. When the number of VMs increases to 100, the disk utilization of the proposed DPLR algorithm is significantly higher than the disk utilization of the benchmark algorithms.

In the DPLR algorithm, the remaining capacity of the cluster in the cloud data center is considered. Whereas in the comparison algorithm, the load of the clusters in each data center is not taken into account as a metric that needs to be optimized. Therefore, the DPLR algorithm performs better than the benchmark algorithm in terms of resource optimization.

*5.2.2. Fault-tolerant scheduling algorithms based on speculative executing and TSPT*

In terms of the number of replicas generated, the completion time, the load balance degree, and the total energy consumption, the performance of the proposed fault-tolerant scheduling algorithm is verified. The TSPT algorithm is compared with the EODS algorithm, the MTS algorithm, and the EWTS algorithm for evaluating the performance of the proposed fault-tolerant scheduling algorithm.

In the experiment of the completion time, the number of tasks is set as 50, 100, 150, and 200 respectively. For each algorithm under each number of tasks, 10 experiments are conducted, and the average of the results is taken. In Fig. 10, the total completion time varies from the number of tasks.

In Fig. 11, in terms of total task completion time, the proposed TSPT algorithm performs better than the MTS algorithm and the EWTS algorithm. As the number of tasks increases, the performance differences between the TSPT algorithm and the benchmark algorithms become more pronounced in terms of total task completion time. As the number of tasks increases to 200, the TSPT algorithm has the lowest time overhead and significantly outperforms the benchmark algorithm. In the cloud computing environment, a large number of tasks often need to be processed. As a result, the TSPT algorithm is more suitable for processing large-scale data in a cloud environment than the benchmark algorithms.

In the formula (45), from the definition of load balance degree, the more synchronized the execution time of each node, the closer the value of load balance degree is to 1, and the more balanced the overall load of the system. In Fig. 13, under eight

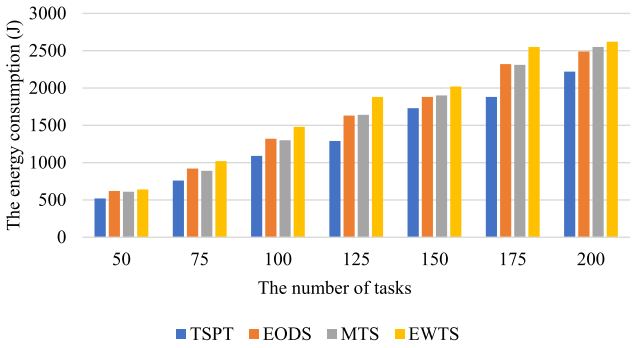


**Fig. 12.** The energy consumption varies from the number of tasks.

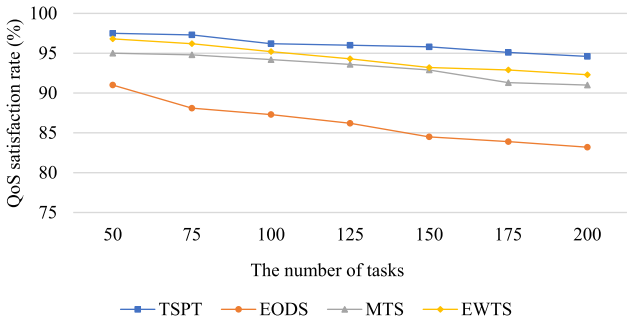


**Fig. 13.** Impact of job quantity on QoS satisfaction rate.

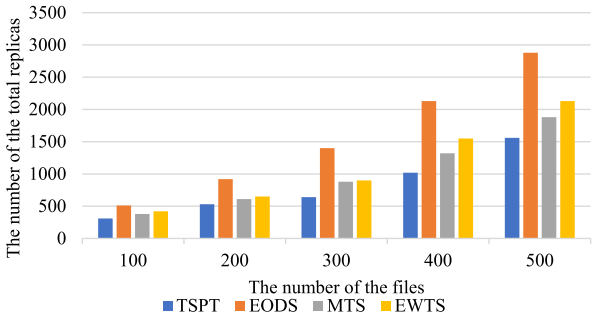


**Fig. 14.** The number of the total replicas varies from the number of files.

different task quantities, the total load balance degree of TSPT is higher than that of the other three benchmark algorithms and is the closest to 1. With the increase of the system load, the load balance curve of TSPT showed a trend of horizontal fluctuation, and the value of the load balance degree remained at about 0.9.

The above analysis shows that after the TSPT algorithm is executed, each task is evenly distributed to different nodes, and the time of each node to execute the assigned task is roughly the same. Therefore, the overall load of the system is relatively balanced. However, with the increase of tasks, the load balance degree of the EODS algorithm decreases. Because a load of each node in the cluster is not considered in the EODS algorithm, in terms of the load balance of the system, the difference between the EODS algorithm and the TSPT algorithm is more and more significant.

Fig. 12 shows the total energy consumption of the system under the four fault-tolerant scheduling algorithms when the number of tasks changes. From Fig. 14, the TSPT algorithm is better than the other three benchmark algorithms in terms of energy consumption. As the number of tasks increases, the energy consumption of all fault-tolerant scheduling algorithms increases. When the number of system tasks is 200, the total system energy consumption of the TSPT algorithm is 2220 joules, which is 10.8% lower than that of the EDS algorithm, 12.9% lower than that of the MTS algorithm, and 15.3% lower than that of the EWTS algorithm. Therefore, the TSPT algorithm multi-tasking environment has a better advantage.

As shown in Fig. 13, as the number of jobs increases, all four algorithms decreases. When the number of tasks is set as 125, the QoS satisfaction rate of the TSPT algorithm is 9.8% higher than that of the EODS algorithm. When the number of tasks is 200, the QoS satisfaction rate of the TSPT algorithm is 10.4% higher than that of the EODS algorithm. As the system load increases, the QoS satisfaction rate of the proposed TSPT algorithm is significantly better than the other three benchmark algorithms. When the system load is low and the cluster is under light load, the proposed TSPT and the EWTS algorithms create replicas before processing tasks, which reduces the impact of backward tasks on the system scheduling. When the cluster is under high load, the proposed TSPT algorithm and the MTS algorithm can detect the backward tasks and create replicas. Then the replicas of tasks on other nodes are restarted, and the efficiency of task scheduling is improved.

The performance of the TSPT algorithm is further verified by comparing the total number of files placed in the cloud storage center. Fig. 14 shows the total number of replicas varies from different amounts of data in the four scenarios. The TSPT algorithm has the smallest number of replicas, while the EODS algorithm uses the largest number of replicas. In the EODS algorithm, the number of replicas increases for reducing the transmission time of accessing replicas.

In our scheme, on one hand, the number of replicas is related to the deadline of the task execution and the load status of each node in the cluster. So for the TSPT algorithm, the number of replicas is increased according to the job's deadline and the load threshold in the task cloning-based scheme. On the other hand, more replicas are placed in different data centers, hence the frequent creation or deletion of replicas can be avoided. Because of the implementation of the above scheme, the TSPT algorithm can improve the resource utility based on the workload balance and the transmission cost can be reduced based on the policy of avoiding frequent transmission of replicas. And then the proposed algorithm considers the different types of energy consumption, the reduction of the total cost can be reflected more truthfully and accurately. Finally, the multi-objective optimized strategy is implemented by the proposed algorithm, the total cost and the task execution time are all reduced in the geo-distributed clouds. Furthermore, the proposed algorithm can improve QoS significantly and the user experiences are well guaranteed.

## 6. Conclusion and future work

In the environment of the geo-distributed cloud, the data placement and fault-tolerant scheduling algorithms are proposed. First, the existing data placement policy of the Spark platform does not consider the data dependencies and Multi-objective optimization, which leads to the longer data transmission time and lower system resource utilization. And then relaxation algorithm is proposed to decrease the data transmission based on system workload balance. After the data is placed, the job is scheduled to the cloud data center where the corresponding data exists. To reduce the energy consumption and execution time of scientific workflow processing, a task scheduling strategy for containerized Spark clusters under a heterogeneous environment is proposed. This strategy optimizes the task scheduling by considering the

actual performance of each physical node in the cluster, the fault-tolerant mechanism, and the severe power consumption is based on. Finally, the effectiveness of the proposed algorithm is verified by experiments. The performance of data processing is improved significantly. In the future, the scale of the experiment will be expanded to test and validate the algorithm. Meanwhile, the cost of renting a public cloud will be considered. In addition, more efficient task scheduling methods will be designed.

## CRediT authorship contribution statement

**Chunlin Li:** Designed the study, Developed the methodology, Performed the analysis, Wrote the manuscript, Collected the data, Revise the paper. **Jun Liu:** Designed the study, Developed the methodology, Performed the analysis, Wrote the manuscript, Collected the data, Revise the paper. **Min Wang:** Designed the study, Developed the methodology, Performed the analysis, Wrote the manuscript. **Youlong Luo:** Designed the study, Developed the methodology, Performed the analysis, Wrote the manuscript.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgement

The work was supported by the National Natural Science Foundation of China (NSFC) under grants (No. 62171330, No. 61873341), Open Research Fund Program of Institute of cosmetic regulatory science (CRS-2021-01), Open Research Fund of AnHui Key Laboratory of Detection Technology and Energy Saving Devices under grants (No. JCKJ2021A08), Key Research and Development Plan of Hubei Province (No. 2020BAB102).

## References

Amina, M., Ali, S.M., Faten, Z.M., et al., 2018. Efficient replica migration scheme for distributed cloud storage systems. IEEE Trans. Cloud Comput. 1.

Ananthanarayanan, G., Ghodsi, A., Shenker, S., et al., 2013. Effective straggler mitigation: attack of the clones. In: Networked Systems Design and Implementation. pp. 185–198.

Anon, 2018a. Online SNAP Datasets. s [2018-10-23] http://snap.stanford.edu/data/index.html.

Anon, 2018b. Online IBM ILOG CPLEX Optimizer. [2018-10-24]. https://googl/jyvDuV.

Anon, 2018c. Online PUMA Datasets. [2018-10-24]. https://engineering.purdue.edu/~puma/datasets.htm.

Bibal Benifa, J.V., Dejey, 2017. Performance improvement of MapReduce for heterogeneous clusters based on efficient locality and replica aware scheduling (ELRAS) strategy. Wirel. Pers. Commun..

Cameron, D.G., Millar, A.P., Nicholson, C., et al., 2004. Analysis of scheduling and replica optimization strategies for data grids using OptorSim. J. Grid Comput. 2 (1), 57–69.

Chauhan, S., Pilli, E.S., Joshi, R.C., et al., 2019. Brokering in interconnected cloud computing environments: A survey. J. Parallel Distrib. Comput. 133, 193–209.

Chen, L., Liu, S., Li, B., et al., 2019. Scheduling works across geo-distributed datacenters with max–min fairness. IEEE Trans. Netw. Sci. Eng. 6 (3), 488–500.

Ferdaus, M.H., Murshed, M., Calheiros, R.N., et al., 2017. An algorithm for network and data-aware placement of multi-tier applications in cloud data centers. J. Netw. Comput. Appl. 98, 65–83.

Gao, Y., Li, K., Jin, Y., 2017. Compact, popularity-aware and adaptive hybrid data placement schemes for heterogeneous cloud storage. IEEE Access 1.

Hu, Z., Li, B., Luo, J., 2017a. Time-and cost-efficient task scheduling across geo-distributed data centers. IEEE Trans. Parallel Distrib. Syst. 29 (3), 705–718.

Hu, Z., Li, B., Luo, J., 2017b. Time-and cost-efficient task scheduling across geo-distributed data centers. IEEE Trans. Parallel Distrib. Syst. 29 (3), 705–718.

Hu, Z., Li, B., Luo, J., 2018. Time- and cost- efficient task scheduling across geo-distributed data centers. IEEE Trans. Parallel Distrib. Syst. 47 (69), 705–718.

Ikken, S., Renault, E., Barkat, A., et al., 2017. Cost-efficient big intermediate data placement in a collaborative cloud storage environment. In: 2017 IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems. HPCC/SmartCity/DSS, IEEE, pp. 514–521.

Jia, L., Zhou, Z., Jin, H., 2018. Optimizing the performance-cost tradeoff in cross-edge analytics. In: 2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation. IEEE, pp. 564–571.

Khalajzadeh, H., Yuan, D., Grundy, J., et al., 2016. Improving cloud-based online social network data placement and replication. In: 2016 IEEE 9th International Conference on Cloud Computing. CLOUD, IEEE, pp. 678–685.

Khalajzadeh, H., Yuan, D., Grundy, J., et al., 2017a. Cost-effective social network data placement and replication using graph-partitioning. In: IEEE International Conference on Cognitive Computing. IEEE.

Khalajzadeh, H., Yuan, D., Grundy, J., et al., 2017b. Cost-effective social network data placement and replication using graph-partitioning. In: 2017 IEEE International Conference on Cognitive Computing. ICCC, IEEE, pp. 64–71.

Khalajzadeh, H., Yuan, D., Zhou, B.B., et al., 2020. Cost effective dynamic data placement for efficient access of social networks. J. Parallel Distrib. Comput. 141, 82–98.

Leskovec, Jure, Lang, Kevin J., Dasgupta, Anirban, et al., 2009. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. Internet Math. 6 (1), 29–123.

Li, C., Zhang, J., Tang, H., 2019. Replica-aware task scheduling and load balanced cache placement for delay reduction in multi-cloud environment. J. Supercomput. 75 (1), 2805–2836.

Li, W., Zhou, X., Li, K., et al., 2018. Trafficshaper: shaping inter-datacenter traffic to reduce the transmission cost. IEEE/ACM Trans. Netw. 26 (3), 1193–1206.

Lin, B., Zhu, F., Zhang, J., et al., 2019. A time-driven data placement strategy for a scientific workflow combining edge computing and cloud computing. IEEE Trans. Ind. Inf. 15 (7), 4254–4265.

Liu, Q., Cai, W., Shen, J., et al., 2017a. A speculative execution strategy based on node classification and hierarchy index mechanism for heterogeneous hadoop systems. In: 2017 19th International Conference on Advanced Communication Technology. IEEE Computer Society Press, Bongpyeong, pp. 889–894.

Liu, Q., Jin, D., Liu, X., et al., 2016a. A survey of speculative execution strategy in MapReduce. In: 2016 2th International Conference on Cloud Computing and Security. Springer, New York, pp. 296–307.

Liu, J., Mao, Y., Zhang, J., et al., 2016b. Delay-optimal computation task scheduling for mobile-edge computing systems. In: 2016 IEEE International Symposium on Information Theory. ISIT, IEEE, pp. 1451–1455.

Liu, Y., Xu, X., Zhang, L., et al., 2017b. Workload-based multi-task scheduling in cloud manufacturing. Robot. Comput.-Integr. Manuf. 45, 3–20.

Lu, X., Jiang, D., He, G., et al., 2018. Greenbdt: Renewable-aware scheduling of bulk data transfers for geo-distributed sustainable datacenters. Sustain. Comput. 20 (Dec), 120–129.

Lu, S., Rao, B.B., Wei, X., et al., 2017. Log-based abnormal task detection and root cause analysis for spark. In: 2017 IEEE International Conference on Web Services. ICWS, IEEE, pp. 389–396.

Michailidou, A.V., Gounaris, A., 2019. A fast solution for bi-objective traffic minimization in geo-distributed data flows. In: Proceedings of the 23rd International Database Applications & Engineering Symposium. pp. 1–10.

Qin, Y., Yang, W., Ai, X., et al., 2018. Fault tolerant storage and data access optimization in data center networks. J. Netw. Comput. Appl. 113 (7), 109–118.

Shao, Y., Li, C., Tang, H., 2019. A data replica placement strategy for IoT workflows in collaborative edge and cloud environments. Comput. Netw. 148, 46–59.

Shi, L., Wang, Z., Li, X., 2021. Novel data placement algorithm for distributed storage system based on fault-tolerant domain. J. Shanghai Jiaotong Univ. 26 (4), 463–470.

Souli-Jbali, R., Hidri, M.S., Ayed, R.B., 2019. Impact of replica placement-based clustering on fault tolerance in grid computing. Int. J. Web Eng. Technol. 14 (2), 151–177.

Tang, Z., Qi, L., Cheng, Z., et al., 2016. An energy-efficient task scheduling algorithm in DVFS-enabled cloud environment. J. Grid Comput. 14 (1), 55–74.

Wan, S., Gu, Z., 2020. Cognitive computing and wireless communications on the edge for healthcare service robots. Comput. Commun. 149, 99–106.

Wu, Y., Zhang, Z., Wu, C., et al., 2017. Orchestrating bulk data transfers across geo-distributed datacenters. IEEE Trans. Cloud Comput. 41 (99), 112–125.

Xu, X., Fu, S., Qi, L., et al., 2018. An IoT-oriented data placement method with privacy preservation in cloud environment. J. Netw. Comput. Appl. 124, 148–157.

Xu, H., Lau, W.C., 2017. Optimization for speculative execution in big data processing clusters. IEEE Trans. Parallel Distrib. Syst. 28 (2), 530–545.

Yi, K., Wang, H., Ding, F., 2010. Decentralized integration of task scheduling with replica placement. In: 2010 Ninth International Symposium on Distributed Computing and Applications To Business, Engineering and Science. pp. 332–336.

Yu, B., Pan, J., 2015. Location-aware associated data placement for geo-distributed data-intensive applications. In: 2015 34th IEEE Conference on Computer Communications. IEEE Computer Society Press, Hong Kong, pp. 603–611.
Zhang, L., Li, X., Khalajzadeh, H., et al., 2018. Cost-effective and traffic-optimal data placement strategy for cloud-based online social networks. In: 2018 IEEE 22nd International Conference on Computer Supported Cooperative Work in Design. CSCWD, IEEE, pp. 110–115.
Zhou, A., Wang, S., Wan, S., et al., 2020. LMM: latency-aware micro-service mashup in mobile edge computing environment. Neural Comput. Appl. (5).

**Chunlin Li** is a Professor of Computer Science in Wuhan University of Technology. She received the ME in Computer Science from Wuhan Transportation University in 2000, and Ph.D. in Computer Software and Theory from Huazhong University of Science and Technology in 2003. Her research interests include cloud computing and distributed computing.

**Jun Liu** received his BE degree in Department of Computer Science and Technology from China University of Geosciences in 2004 and ME degree in School of Computer Science from Wuhan University of Technology in 2011. He is a Ph.D. student in School of Computer Science and Technology from Wuhan University of Technology. His research interests include edge computing and big data.

**Min Wang** is associate professor in Department of Cosmetics of Beijing Technology and Business University. Her research interests include evaluation of the safety and efficacy of cosmetic raw materials.

**Youlong Luo**, He is a vice Professor of Management at Wuhan University of Technology. He received his M.S. in Telecommunication and System from Wuhan University of Technology in 2003 and his Ph.D. in Finance from Wuhan University of Technology in 2012. His research interests include cloud computing and electronic commerce.