



An optimized hierarchical MapReduce framework in supercomputing Internet environment

Yalin Zhu¹ · Youquan Chang² · Jiapeng Zhang^{1,3} · Yingjie Song¹ · Zhuo Tang^{1,4}

Received: 20 August 2024 / Accepted: 11 February 2025 / Published online: 28 March 2025
© The Author(s) 2025

Abstract

Distributed computing frameworks play a crucial role in supporting compute-intensive applications in the era of big data. The growing demand for computing resources has spurred the interconnection of data centers, leading to the formation of supercomputing Internet. MapReduce is a popular distributed computing framework designed for large independent clusters. The original MapReduce framework deployed on supercomputing Internet performs inefficiently due to redundant geo-distributed reduce operations. Nonetheless, its abstraction remains significant potential. This paper proposes an enhanced MapReduce framework for geo-distributed supercomputing Internet to minimize the necessity for data transmission across data centers. Leveraging hierarchical scheduling techniques, the framework optimizes data locality to mitigate network latency and bandwidth consumption during reduce operations, thereby reducing overall job execution times. The paper introduces a mathematical model for task scheduling within supercomputing Internet and formally describes the data transmission process among data centers. In the job scheduling phase, our framework facilitates efficient overlap of transferring and computing through pre-selected data centers. Meanwhile, in the data transmission phase, the framework aggregate data to reduce the frequency of transmission, thus alleviating the adverse effects on transmission of hierarchical network architecture. Comparative analysis with existing methods demonstrates the efficacy of the proposed framework in addressing similar computational challenges. Empirical evaluations underscore the effectiveness of our method in practice.

Keywords MapReduce · Scheduling · Supercomputing Internet

1 Introduction

As big data technology continues to evolve, MapReduce (Dean and Ghemawat 2008) remains a foundational framework in the field of big data processing. Many subsequent big data frameworks have inherited the core principles of MapReduce, which continues to excel in batch processing tasks and scenarios requiring high fault tolerance. However, the current implementation of MapReduce, due to its architectural design and implementation specifics, is primarily suited for operations within a single data center (Dean and Ghemawat 2004). This is mainly reflected in the fact that MapReduce can efficiently utilize the internal network when processing intermediate data in a single-cluster environment. However, in a multi-cluster environment, this efficiency significantly decreases. For instance, Hadoop can only ensure high-performance computing when computational resources are configured as homogeneous nodes within a cluster, interconnected via high-speed network links, and all data is stored within a single data center rather than distributed

✉ Jiapeng Zhang
zhangjp@hnu.edu.cn

✉ Zhuo Tang
ztang@hnu.edu.cn

Yalin Zhu
zhuyalin@hnu.edu.cn

Youquan Chang
changyouquan@hn.chinamobile.com

Yingjie Song
yjsong0044@hnu.edu.cn

¹ College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China

² China Mobile Communications Group Hunan Co., Ltd., Changsha 410001, China

³ The Ministry of Education Key Laboratory of Fusion Computing of Supercomputing and Artificial Intelligence, Hunan University, Changsha 410082, China

⁴ Shenzhen Research Institute of Hunan University, Hunan University, Shenzhen 518000, China

across multiple data centers (Heintz et al. 2014). The supercomputing Internet, which integrates the strengths of supercomputing and Internet technologies, interconnects large-scale computing resources to provide powerful computing and data processing capabilities (Chen 2024). This advancement is instrumental in driving scientific research, industrial applications, and social development. Nevertheless, as the supercomputing Internet expands, the classic MapReduce framework struggles to accommodate scenarios involving multiple interconnected data centers.

Since MapReduce is primarily designed for operation within a single data center, using the classic MapReduce framework in the context of the supercomputing Internet presents several challenges. First, large-scale data transfers can lead to network congestion, thereby degrading the overall performance of tasks. Second, communication overhead across data centers can cause delays in task scheduling, impacting the overall efficiency of MapReduce tasks. Third, improper task allocation may result in unnecessary data transfers. Therefore, this paper addresses the problem of scheduling a batch of MapReduce jobs in the supercomputing Internet with the goal of minimizing total execution time.

Existing architectures capable of handling large-scale distributed data processing across multiple data centers primarily include G-Hadoop (Wang et al. 2013) and MDC-Hadoop (Li et al. 2021). G-Hadoop is a classic implementation of MapReduce. During the Map phase, G-Hadoop assigns tasks to idle slots with data locality (Xue et al. 2015). If no nodes meet the requirements, the remaining map tasks are assigned to nodes with available idle slots. Nodes for executing reduce tasks are selected randomly. The limitation of G-Hadoop is in the allocation of map and reduce tasks, which can result in unnecessary data transfers. MDC-Hadoop, on the other hand, adopts a new algorithmic framework for scheduling MapReduce tasks, consisting of three phases: Map task scheduling, Reduce data center determination, and Reduce task scheduling. During the Map and Reduce task scheduling phases, tasks are optimally assigned to available containers using the classic Hungarian algorithm, based on specific evaluation criteria. In the Reduce data center determination phase, all intermediate data generated by map tasks are transferred to a selected data center, and reduce tasks are scheduled only within that data center.

MDC-Hadoop has addressed some of the limitations of G-Hadoop, but it still encounters issues such as high data transfer overhead and prolonged job execution times, particularly when handling small to medium-sized jobs. Firstly, during the map task scheduling phase, if tasks are scheduled based solely on optimal locality without considering data center levels, this can lead to significant data transfer overhead during the reduce phase due to data transfers between data centers. Secondly, the MDC-Hadoop architecture

initiates the Reduce data center determination phase only after the completion of the map task scheduling phase. Once the data center is determined, the transfer of intermediate data begins. The reduce task scheduling phase commences only after all intermediate data has been transferred. This sequential process of task execution and data transfer is not conducive to minimizing overall job completion time. Figure 1 illustrates an example of job scheduling in MDC-Hadoop, designed to visually highlight the potential shortcomings of the MDC-Hadoop framework when handling job scheduling in a supercomputing Internet environment. Consider three geographically distributed data centers (DC_1 , DC_2 , and DC_3).

Job J_1 has three map tasks awaiting allocation: T_{11} , T_{12} , and T_{13} , with their required data blocks distributed across different data centers, for example, the data required by T_{11} , denoted as d_{11} , is stored in DC_1 . T_{11} is allocated to a node in DC_1 , T_{12} to a node in DC_2 , and T_{13} to a node in DC_3 . If MDC-Hadoop selects DC_2 during the Reduce data center determination phase, intermediate data from DC_1 and DC_3 must be transferred to DC_2 . However, if we initially choose to execute job J_1 entirely in data center DC_2 , only one cross-data center data transfer is needed, moving data block d_{11} to DC_2 . While this data transfer occurs, the cluster in DC_2 can start executing tasks T_{12} and T_{13} using the existing data blocks. These two processes can run in parallel, reducing the overall job completion time, Fig. 2 illustrates the transmission time in the optimized job scheduling process. Furthermore, we can optimize the processing time of data block d_{11} after it is transferred to DC_2 through specific strategies, as shown in Fig. 2c. Additionally, the MDC-Hadoop architecture does not account for issues such as network latency, bandwidth congestion, and the costs associated with frequent data transfers. These shortcomings of the MDC-Hadoop architecture have motivated us to modify the G-Hadoop architecture from a different perspective to address the researched problems.

In summary, existing MapReduce frameworks encounter challenges related to long data transfer times and extended job execution durations within the supercomputing Internet environment. To tackle these issues, we propose a new scheduling framework. This architecture utilizes a hierarchical scheduling technique to minimize data transfer across data centers, ensuring reasonable resource allocation and efficient utilization. Additionally, our architecture reduces the frequency of data transfers during replication by aggregating data, thereby mitigating the impact of the layered network architecture on data transmission. The main contributions of this paper are summarized as follows:

- Based on G-Hadoop, a new hierarchical architecture for data analysis in supercomputing networks, called SIE-Hadoop, has been developed. This architecture improves

Fig. 1 Job scheduling example in MDC-Hadoop

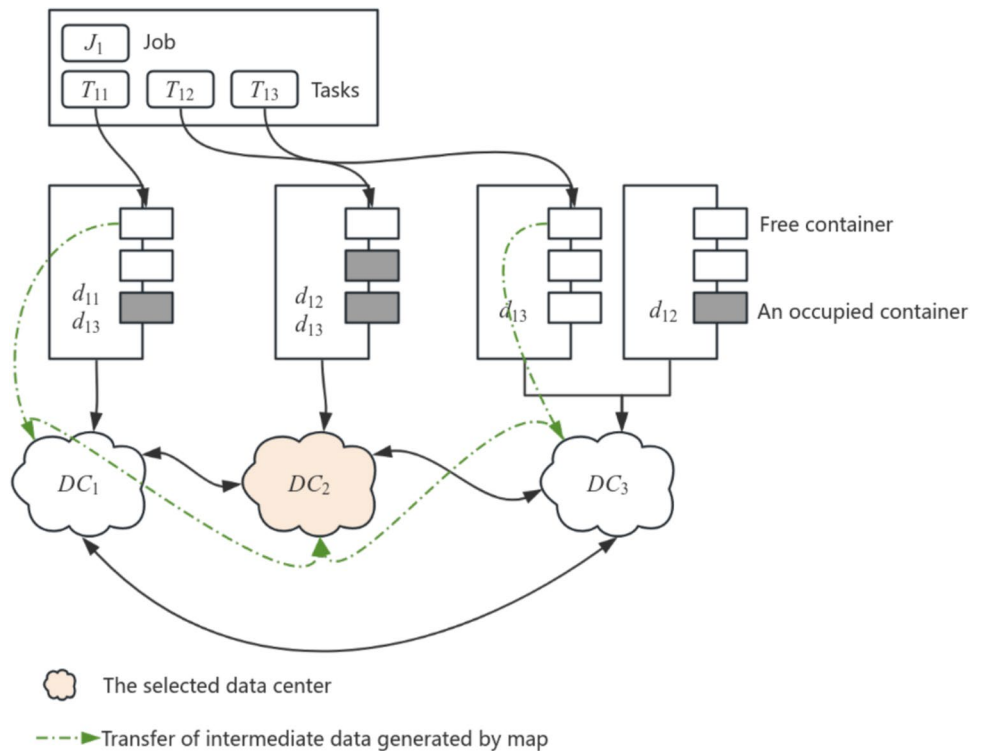
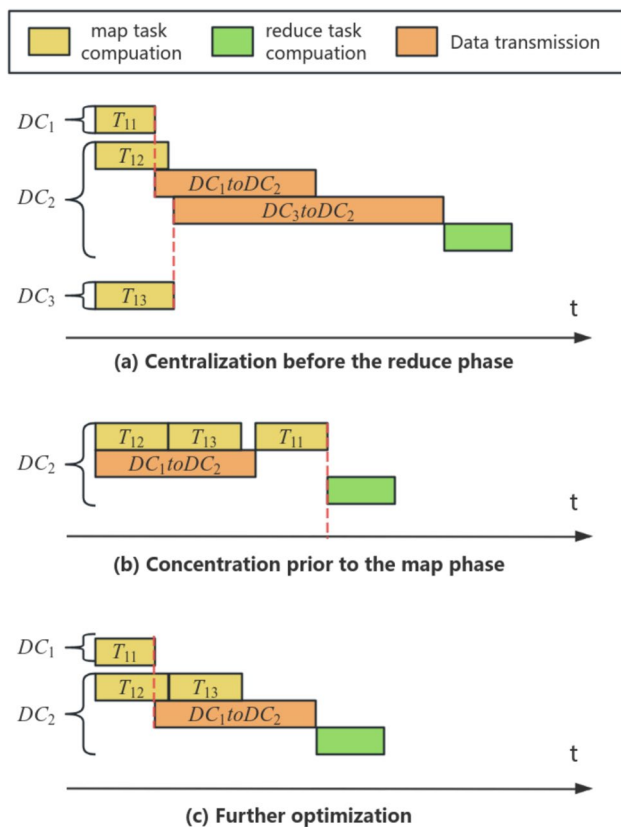


Fig. 2 Optimization of transfer time in job scheduling

task scheduling efficiency, optimizes resource utilization, and reduces data transmission costs.

- We mathematically modeled the problem considering different levels of data transmission and the heterogeneity of nodes within the supercomputing Internet environment.
- We proposed an algorithmic framework for scheduling MapReduce tasks across distributed data center nodes. This framework comprises two phases: job scheduling and task scheduling. It aims to maximize the overlap between computation time and transfer time, thus reducing overall job execution time. This approach, akin to pipeline processing, effectively hides data transfer latency during map task execution.

The remainder of this paper is organized as follows: Sect. 2 reviews related work. Section 3 introduces the new architecture and presents a mathematical model of the problem. Section 4 details our job scheduling algorithm. Experimental evaluations are discussed in Sect. 5. Finally, Sect. 6 concludes the paper.

2 Related work

This section explains the necessity of distributed computing frameworks as a key technology for implementing the supercomputing Internet. It then discusses the impact of different approaches to handling small and medium-sized jobs using

distributed computing frameworks in the supercomputing Internet on job processing speed.

2.1 Distributed computing frameworks under the supercomputing Internet

The supercomputing Internet connects supercomputing centers through high-speed networks, forming a unified resource platform that can provide services to the public (Sciences Bulletin 2024). This platform enables the integration and sharing of computing resources across regions. Such resource integration significantly enhances resource utilization efficiency and provides powerful computational capabilities and storage space for large-scale data processing.

Distributed computing frameworks, such as Hadoop (2024) and Apache Spark (2024), are the core technologies for distributed computing and big data processing. MapReduce is the core computing engine of Hadoop, while Apache Spark can be seen as an improvement and extension of traditional MapReduce. Other MapReduce implementations can also be used in various architectures, such as CUDA (He et al. 2008), FPGA platforms (Shan et al. 2010), multiple virtual machines (Ibrahim et al. 2009). Net environments (Jin and Buyya 2009), and stream runtime environments (Pallickara et al. 2009). As a fundamental model of distributed computing, it is essential to optimize the MapReduce framework to adapt to the supercomputing Internet. The work by Cardosa et al. (2011) shows that the classic Hadoop MapReduce implementation generally performs poorly in geographically distributed environments with high network heterogeneity.

Therefore, it is necessary to make reasonable improvements to the MapReduce architecture to adapt it to the integration with the supercomputing Internet. This would enable more efficient utilization of the computational and storage resources provided by the supercomputing platform, significantly enhancing the capabilities of big data processing and high-performance computing.

2.2 Existing optimization methods for Hadoop

As data volumes continue to increase and the demand for geographically distributed data processing becomes inevitable, a significant amount of research has focused on improving the performance of MapReduce in handling geographically distributed data. Building on the traditional Hadoop framework, several new cross-data-center architectures have been developed. G-Hadoop (Wang et al. 2013) implements node scheduling and data processing tasks across multiple data centers, providing a more fault-tolerant data processing environment and achieving high throughput data processing performance. However, this framework does not consider data locality during the reduce phase, resulting in a large

amount of unnecessary data transfer. MDC-Hadoop (Li et al. 2021) is a new architecture improved based on G-Hadoop. This architecture takes task data locality into account. Due to the strict sequential order of its three algorithm phases, resources cannot be fully utilized. CoGRS (Hammoud et al. 2012) proposed a framework that places reduce tasks as close as possible to the intermediate data, but it does not consider data locality during the map phase. H2F (Tomarchio et al. 2018) is a layered MapReduce-based framework, but it suffers from performance bottlenecks due to extensive data transfer. The ChEsS framework (Zacheilas and Kalogeraki 2016) is a method for scheduling MapReduce jobs in multi-cluster environments. This framework takes into account the performance/budget trade-off to minimize costs, but it does not treat each data center as part of an overall supercomputing Internet. A method proposed by others (Afrati et al. 2015), called Meta-MapReduce, reduces communication costs by using metadata much smaller than the original data to compute the output, but it does not explicitly indicate that it is designed for geographically distributed environments. Zhang et al. (2016) proposed a new framework and key-based scheme to enable Hadoop usage across data centers, but their improvements are limited to the reduce phase. Zhang et al. (2014) modified the Hadoop algorithm to improve job performance in cloud-based multi-data-center computing environments, but their algorithm changes involve HDFS data placement strategies. Ryden et al. (2014) proposed an edge-based nebula architecture that supports geographically distributed MapReduce, which also involves data placement strategies. Other works (Gadre et al. 2011; Kim et al. 2011; Mattess et al. 2013) mainly consider reducing job completion time from the perspective of computing resources, which differs from our work.

Unlike existing work, we have constructed a new MapReduce architecture for data analysis in the supercomputing Internet, focusing on the overlap of transmission and computation time during the Map phase to optimize the overall job execution time. Additionally, we improve the data locality of tasks by selecting a single data center for MapReduce jobs and reduce data transmission frequency in the layered network architecture by aggregating data.

3 SIE-Hadoop system architecture

This section presents an improved system architecture and provides a mathematical description of the problem based on this architecture.

3.1 System architecture design

We designed a new Hadoop hierarchical architecture for Supercomputing Internet Environments (SIE-hadoop) based

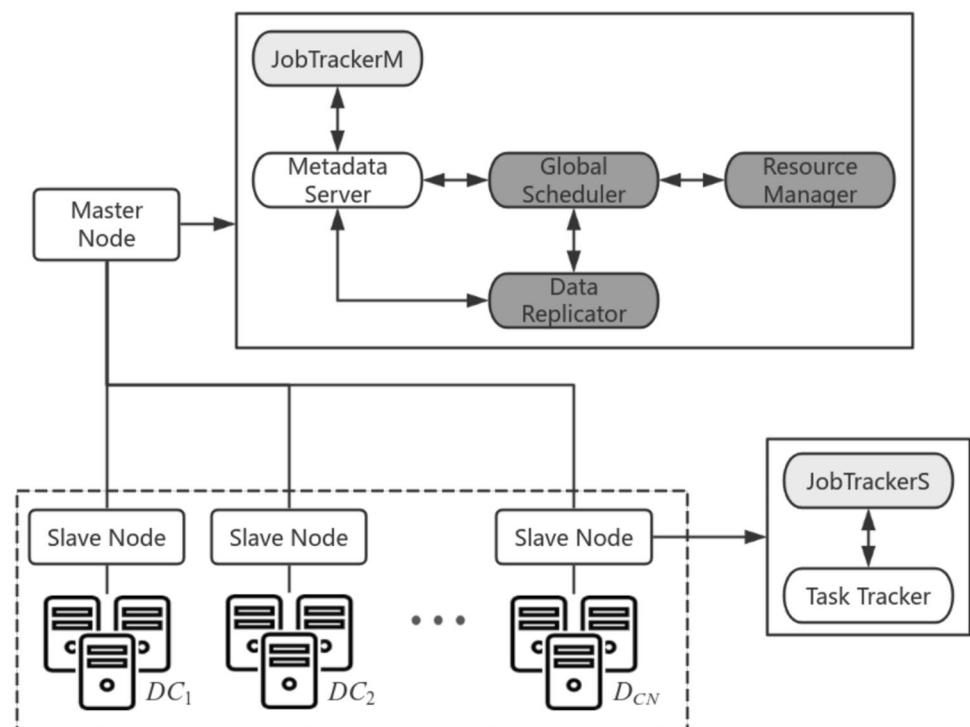
on G-Hadoop to better accommodate large-scale distributed data processing across multiple data centers. Figure 3 shows the architecture of SIE-hadoop picture. The communication model of our architecture adopts a master–slave model similar to G-Hadoop. The master node is responsible for global control, and all data centers are interconnected with the master node through the network. As shown in Fig. 3, our architecture differs from G-Hadoop by adding three components to the master node: the Global Scheduler, Resource Manager, and Data Replicator.

In our architecture, the metadata server is similar to that in G-Hadoop and is responsible for managing files distributed across multiple data centers. It provides a global view of all files within a data center and records all file metadata, such as data size, file format, and access paths. Unlike the JobTracker in G-Hadoop, which is a single entity, our architecture divides the JobTracker into master and slave components, located on the master node and worker nodes, respectively. The master JobTracker is responsible for splitting jobs into smaller tasks, while the Global Scheduler handles the scheduling of these tasks across different data centers. The slave JobTracker is responsible for scheduling and executing jobs assigned by the Global Scheduler among different nodes within the cluster. The Resource Manager component collects relevant information from various data centers to assist the Global Scheduler in making scheduling decisions, such as the load status of each cluster and

the number of available containers. The Global Scheduler is responsible for scheduling the current job to an appropriate data center for execution and synchronizing job-related information to the slave JobTracker of the cluster to manage the job tasks. The DataReplicator component is responsible for replicating data between data centers based on scheduling decisions. Transferring large amounts of data from various source locations to a centralized data center can result in excessive latency and potentially daunting bandwidth costs (Rabkin et al. 2013). Current cloud providers have significantly expanded the popular three-tier network architecture in their data centers (Ahmad et al. 2012). At the access layer (i.e., the bottom layer), each machine connects to one or two access switches. In the aggregation layer, each access switch connects to one (or two) aggregation switches. Finally, each aggregation switch connects to multiple core switches. The Fig. 4 shows a two-tier topology, which is usually rooted in one of the core switches. Therefore, the DataReplicator component takes into account the impact of the layered network architecture on data transmission when performing data replication. For example, the diagram below depicts a scenario where node d_1 needs to receive data from nodes d_2 , d_3 , and d_4 . In actual transmission, two rounds of aggregation can be used, as shown in Fig. 4, to reduce transmission frequency and thus achieve lower network overhead.

After a job is submitted to the master JobTracker, it is divided into separate map and reduce tasks. All data

Fig. 3 The proposed SIE-Hadoop architecture



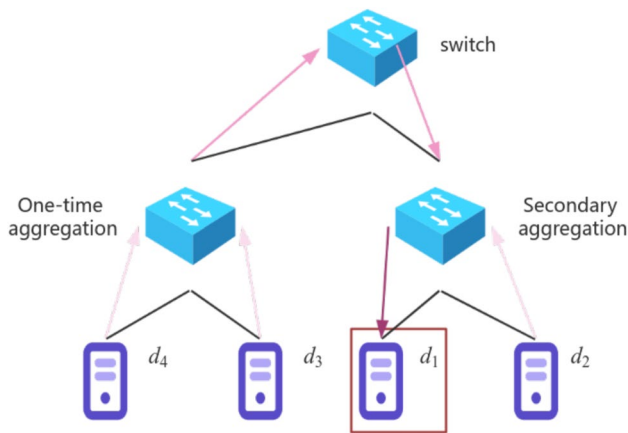


Fig. 4 Example of reducing transfer frequency through data aggregation

information for each job is stored in the metadata server. During each heartbeat, the ResourceManager acquires information about available containers in each data center to enable the Global Scheduler to allocate jobs to the appropriate data centers. The selected data center's slave JobTracker then schedules the tasks across the nodes in its cluster, while the DataReplicator is responsible for copying any missing data blocks of the job to suitable locations within the selected data center.

3.2 Mathematical modeling

We make the following assumptions: (i) Within the same data center, the communication overhead between different nodes on the same rack differs from the communication overhead between nodes on different racks. (ii) Computing resources are heterogeneous. (iii) The number of containers in each node is fixed, and a container can handle only one task at a time; a task is handled by only one container at a time. (iv) The output of Map tasks is stored on the node that processes them. (v) Each data block corresponding to a task

has 3 replicas. (vi) The reduce phase cannot begin until the map phase is completed. When selecting a data center for job execution, only communication overhead due to data transfer and cluster load are considered. (vii) Task failures are not considered. The Table 1 lists some important variable declarations.

Let α be the number of data centers. We initialize the network topology of the distributed data processing system across data centers as the set of data centers $\{DC_0, DC_1, \dots, DC_{\alpha-1}\}$. For any data center DC_l , there are n_l nodes. The k th node $N_{l,k}$ contains $n_{l,k}$ containers. $N_{l,k,g}$ is the g th container in $N_{l,k}$. The processing speed of $N_{l,k,g}$ is $v_{l,k,g}$. Since computing resources are typically heterogeneous, different containers have different processing speeds.

There are γ jobs that need to be processed through these data centers, and the set of jobs is $\{J_0, J_1, \dots, J_{\gamma-1}\}$. Each job J_r consists of a set of map tasks T_r^{map} and a set of reduce tasks T_r^{reduce} . $T_{r,i}^{\text{map/reduce}}$ represents the i th map or reduce task of the r th job. Each map task is responsible for one data block, and the data blocks required for each job are distributed across multiple data centers. The number of data blocks that job J_r needs to process is b_r . In HDFS, each data block has multiple replicas, so the i th data block $B_{r,i}$ of job J_r can be represented as $\{B_{r,i0}, B_{r,i1}, \dots, B_{r,i(\alpha-1)}\}$. Typically, the replication factor $\lambda = 3$. Each job's required data is distributed across multiple data centers.

J_r is a small to medium-sized job, and its required data is distributed across different data centers. Without intervention, the map tasks of the job will be scheduled to different clusters based on data locality, while the reduce tasks of the job will need the output data of almost all map tasks. In this case, the communication overhead caused by cross-data center transmission will far exceed the execution time of the job itself. Therefore, scheduling job J_r to a single data center from the beginning will make the execution time of J_r overlap with part of the data transmission time, thereby reducing the data transmission generated during the job execution process.

Table 1 Important variable declarations

Symbol	Description
$DC_l, 0 \leq l \leq \alpha$	The l th data center
$N_k, 0 \leq k \leq \beta$	The k th node
$n_{l,k}$	The number of containers on the k th node in the l th data center
$v_{l,k,g}$	The processing speed of the g th container in $n_{l,k}$
$J_r, 0 \leq r \leq \gamma$	The r th job
$T_r^{\text{map/reduce}}$	The set of map or reduce tasks for the r th job
b_r	The number of data blocks required to process job J_r
$C_{l,l'}$	The communication cost of replicating data blocks from data center l' to l

Before execution, select a suitable data center DC_l for all tasks in J_r , including map tasks and reduce tasks. The global scheduler needs to copy the data blocks required by the map tasks in J_r but not in DC_l from other data centers to DC_l . Let $C_{l,l'}$ be the communication overhead required to copy the data blocks from cluster l' to cluster l . Assuming that J_r is scheduled to be executed in data center DC_l , the time required to transfer the data from data center $DC_{l'}$ to DC_l is defined as follows:

$$T_{l,l'}^{tran} = \sum_{B_{r,i} \in DC_{l'}}^{br} |B_{r,i}| C_{l,l'}. \quad (1)$$

Assume that DC_l is the selected data center, then the data transmission time during the operation adjustment phase is defined as:

$$t_l^{tran} = \max_{0 \leq l' \leq \alpha, l' \neq l} T_{l,l'}^{tran}. \quad (2)$$

Since the map task does not have a pre-assigned task, so the map stage and data transmission stage of J_r can be carried out at the same time. $t_{r,i}^{map/reduce}$ and $c_{r,i}^{map/reduce}$ represent the start time and data transmission time of $T_{r,i}^{map/reduce}$. Assume $T_{r,i}^{map}$ is allocated to container $S_{l,k,g}$, the completion time of the task is:

$$f_{r,i}^{map} = t_{r,i}^{map} + c_{r,i}^{map} + \frac{|B_{r,i}|}{v_{l,k,g}}. \quad (3)$$

The completion time of the map phase is the time when the last map task in the job is completed. $F_r^{map} = \max\{f_{r,i}^{map} | 1 \leq i \leq |T_r^{map}|\}$.

All tasks of job J_r are scheduled to be executed on data center DC_l , so as soon as the map phase ends, the reduce phase begins. Similar to the map tasks, assuming $T_{r,i}^{reduce}$ is allocated to container $S_{l,k,g}$, the completion time for this task is:

$$f_{r,i}^{reduce} = t_{r,i}^{reduce} + c_{r,i}^{reduce} + \frac{I_{r,i}^{reduce}}{v_{l,k,g}}. \quad (4)$$

In this equation, $I_{r,i}^{reduce}$ represent the input data size of $T_{r,i}^{reduce}$. Because the reduce phase only starts after the map phase has finished, $t_{r,i}^{reduce}$ occurs after F_r^{map} .

Thus, the completion time of the reduce phase is $F_r^{reduce} = \max\{f_{r,i}^{reduce} | 1 \leq i \leq |T_r^{reduce}|\}$.

The goal is to minimize the total execution time of all MapReduce jobs in the geographically distributed data centers, which can be expressed as:

$$\min F = \min(\max F_r^{reduce}). \quad (5)$$

Due to a large number of MapReduce jobs being submitted to the system over a period, forming a complex DAG, and if all tasks are handled by a single data center with homogeneous computational resources and negligible transmission delays, the problem becomes NP-hard (Hartmann and Briskorn 2022). In other words, since even the special case of the problem is NP-hard, the general case is also NP-hard. Therefore, our algorithm aims to find an approximate optimal solution to this problem.

4 Algorithm design

This section presents a MapReduce task scheduling algorithm in the supercomputing Internet environment. The algorithm is divided into job scheduling and task scheduling algorithms. Map tasks and reduce tasks ultimately need to be scheduled to containers for execution. The job scheduling algorithm primarily functions within the GlobalScheduler and DataReplicator modules of the architecture, assisting the GlobalScheduler in selecting a suitable data center for the job and providing data transfer solutions for the DataReplicator. The task scheduling algorithm mainly operates within the JobTracker module of the architecture, helping it make task scheduling decisions for the data center it is in.

4.1 Job scheduling algorithm design

In a distributed computing environment, the input data for jobs is typically divided into equally sized data blocks, with each data block potentially having multiple replicas across cluster nodes. For MapReduce jobs, the job is divided into map and reduce phases that proceed sequentially. During the reduce phase, the input for each reduce task usually comes from the output of all map tasks. As a result, reduce tasks do not benefit from data locality, which is particularly pronounced in distributed data processing systems across multiple data centers. Therefore, our architecture selects a data center for each job at the start and replicates any missing data blocks to that data center.

Selecting the appropriate data center to minimize job completion time requires considering several factors: the workload of the target cluster, the data transfer duration, and the expected execution time of the job in the target cluster. Higher workloads may cause longer delays for other tasks

or jobs. During the map phase, tasks are processed in parallel, so the target data center can process the data blocks it already has while waiting to receive the required data that is in transit. The expected execution time of the job in the target cluster depends on the performance of the containers in that cluster.

First, calculate the average processing speed of containers in the target data center. In the target data center DC_l , there are $\sum_{k=0, N_k \in DC_l}^{\beta} \mathbf{n}_{l,k}$ containers, and the total processing speed of these containers is $\sum_{k=0, N_k \in DC_l}^{\beta} \sum_{g=0}^{n_{l,k}} \mathbf{v}_{l,k,g}$. Therefore, the average processing speed of containers in the target data center is $\bar{v}_l = \frac{\sum_{k=0, N_k \in DC_l}^{\beta} \sum_{g=0}^{n_{l,k}} \mathbf{v}_{l,k,g}}{\sum_{k=0, N_k \in DC_l}^{\beta} \mathbf{n}_{l,k}}$.

To measure the workload of the target data center DC_l , in this algorithm, we define the processing time of the remaining tasks that have been scheduled to the target data center DC_l as t_l^{remain} , which can be determined by the following formula:

$$t_l^{\text{remain}} = \sum_{J_r \in J_l} \left(\sum_{T_{r,i} \in \tilde{T}_r^{\text{map}}} \frac{|B_{r,i}|}{\bar{v}_l} + \sum_{T_{r,i} \in \tilde{T}_r^{\text{reduce}}} \frac{I_{r,i}^{\text{reduce}}}{\bar{v}_l} \right). \quad (6)$$

where J_l is the set of jobs already assigned to DC_l , $\tilde{T}_{r,i}^{\text{map/reduce}}$, $J_r \in J_l$, represents the map or reduce tasks that have not been completed in J_l . Only when the map stage of job J_r is completed, the reduce tasks of J_l are added to $\tilde{T}_{r,i}^{\text{map/reduce}}$.

To measure the duration of the data transmission required for job J_r to be scheduled to DC_l , in this algorithm, we define the duration of the transmission time as t_l^{tran} . Since the transmission time across data centers is much longer than the transmission between racks and nodes within a single data center, in the data transmission phase, only the communication overhead across data centers is considered. The data transmission process starts simultaneously in each data center, so only the largest one needs to be taken. Its value can be determined by the (2) formula.

To balance the execution time of job J_r in the target data center DC_l , in this algorithm, we define the expected execution time as t_l^{pre} . Since map tasks are independent and do not rely on other data centers for further processing, to reduce the execution time of the job, it is possible to let a part of the waiting data blocks from other data centers be executed locally first, and then directly transmit the intermediate data after

execution. The time required for the data outside the data center DC_l' to be transmitted to DC_l can be calculated by formula (1), the time for the data corresponding to the map task to be processed at the data center DC_l' is $\frac{\sum_{B_{r,i} \in DC_{l'}}^{b_r} |B_{r,i}|}{\bar{v}_{l'}}$, while the

maximum number of map tasks that can be executed locally while ensuring that the total transmission time does not

increase is calculated as $[N_{l'} - 1]$, where $N_{l'}$ is calculated as $\frac{T_{l,l'}^{\text{tran}} \times \sum_{B_{r,i} \in DC_{l'}}^{b_r} |B_{r,i}|}{\sum_{B_{r,i} \in DC_{l'}}^{b_r} |B_{r,i}|} = T_{l,l'}^{\text{tran}} \times \bar{v}_{l'}$.

Due to the presence of some data blocks of job J_r within the target data center itself, the expected execution time should be divided into the expected execution time of the map tasks on the current data center $t_l^{\text{pre-cur}}$ and the expected execution time of the remaining map tasks and subsequent reduce tasks that are waiting for data block transmission $t_l^{\text{pre-else}}$:

$$t_l^{\text{pre}} = t_l^{\text{pre-cur}} + t_l^{\text{pre-else}}, \quad (7)$$

where $t_l^{\text{pre-cur}}$ is calculated as $\sum_{B_{r,i} \in DC_l}^{b_r} b_r \frac{|B_{r,i}|}{\bar{v}_l}$, and $t_l^{\text{pre-else}}$ is calculated as $\sum_{B_{r,i} \notin DC_l}^{b_r} \frac{|B_{r,i}|}{\bar{v}_l} - \sum_{l'=0, l' \neq l}^a [N_{l'} - 1] \times \frac{|B_{r,i}|}{\bar{v}_{l'}} + \sum_{T_{r,i} \in \tilde{T}_r^{\text{reduce}}} \frac{I_{r,i}^{\text{reduce}}}{\bar{v}_l}$.

The final completion time t_l^{finish} of the job in DC_l depends on the workload of the target cluster, the transmission time required for data, and the expected execution time of the job in the target cluster. Because the transmission time may be greater than the remaining execution time of the data center plus the expected execution time of the current job on the target data center's data block, and the map task can be executed independently without waiting for other data to be transmitted, so t_l^{finish} has two cases:

If the data transmission time is greater than the remaining execution time of the target data center tasks plus the expected execution time of the current job on the target data center's data block, then $t_l^{\text{finish}} = t_l^{\text{tran}} + t_l^{\text{pre-else}}$.

If the data transmission time is less than or equal to the remaining execution time of the target data center tasks plus the expected execution time of the current job on the target data center's data block, then $t_l^{\text{finish}} = t_l^{\text{remain}} + t_l^{\text{pre}}$.

Choose the data center that can minimize the completion time of the job to schedule the current job. The proposed job scheduling algorithm is described in Algorithm 1.

Algorithm 1 Job Scheduling Algorithm

Require: Job to be scheduled: J_r

- 1: Initialize selected data center index: $\text{target} = 0$
- 2: Initialize job completion time set: $T_{\text{finish}} = \emptyset$
- 3: Initialize minimum completion time cost: $\text{cost} = \infty$
- 4: **for** $i = 0; i < \alpha; i++$ **do**
- 5: Calculate t_i^{remain} using Equation (6);
- 6: Calculate t_i^{tran} using Equation (2);
- 7: Calculate t_i^{pre} using Equation (7);
- 8: **if** $t_i^{\text{tran}} > t_i^{\text{remain}} + t_i^{\text{pre-cur}}$ **then**
- 9: $T_{\text{finish}}(i) = t_i^{\text{tran}} + t_i^{\text{pre-else}};$
- 10: **else**
- 11: $T_{\text{finish}}(i) = t_i^{\text{remain}} + t_i^{\text{pre}};$
- 12: **end if**
- 13: **end for**
- 14: **for** $i = 0; i < \alpha; i++$ **do**
- 15: **if** $\text{cost} > T_{\text{finish}}(i)$ **then**
- 16: $\text{target} = i;$
- 17: **end if**
- 18: **end for**
- 19: **return** $\text{target};$

To illustrate the process of selecting data centers in the job scheduling described above, consider the following example. Given a job that needs to be scheduled, we need to choose the appropriate data center. As shown in Fig. 5, there are two data centers and one job to be scheduled. Each data center has some incomplete tasks, and the data transfer time required for scheduling the job to different data centers varies, as does the processing speed of tasks.

According to the Eqs. (6) (2) (7), calculate and compare, it is obtained that the completion time of the scheduling operation of job to DC_1 is $t_1^{\text{finish}} = t_1^{\text{tran}} + t_1^{\text{pre-else}}$, and the completion time of the scheduling operation of job to DC_2 is $t_2^{\text{finish}} = t_2^{\text{remain}} + t_2^{\text{pre}}$. Through comparison, it is found that $t_1^{\text{finish}} > t_2^{\text{finish}}$, so the choice is to schedule job to DC_2 .

4.2 Task scheduling algorithm design

In our architecture, partial map tasks and all reduce tasks of each job are executed in the selected data center.

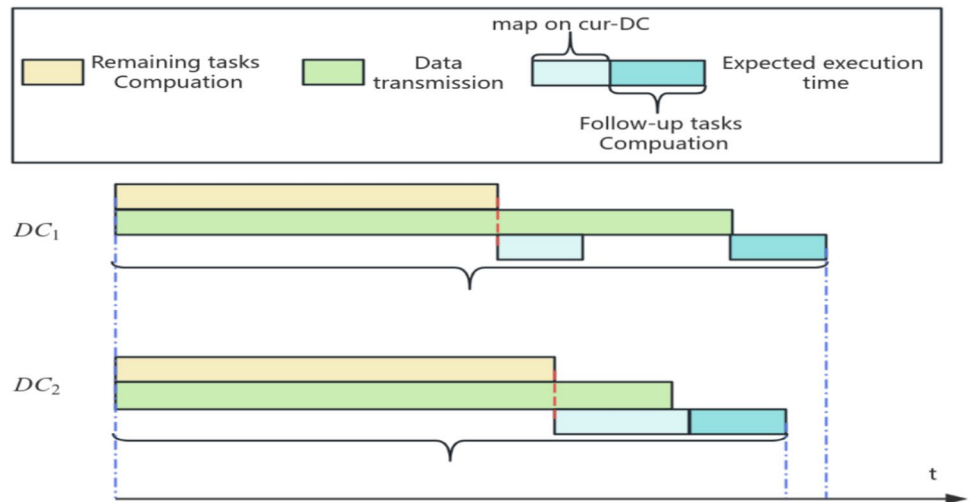
The primary factors affecting the job's execution time are the data transfers across nodes or racks during the scheduling of map and reduce tasks. Clearly, we can optimize the problem by reducing cross-node/rack data transfers by leveraging task locality. Many studies have proposed useful task scheduling algorithms to improve task locality (Naik et al. 2019; Pan et al. 2018; Selvitori et al. 2019; Duan et al. 2016; Tang et al. 2016; Guo et al. 2012; Liu et al. 2018; Huang

et al. 2016). However, since the communication patterns for the map and reduce phases differ, existing research has only optimized the task locality for one of these phases, which significantly limits the performance improvement of the job. To maximize map task locality within the same data center while also providing some task locality for reduce tasks, we refer to the optimal locality-aware task scheduling algorithm based on bipartite graph modeling from Fu et al. (2020).

The algorithm uses a weighted bipartite graph to model the scheduling of Map and Reduce phase tasks uniformly. It then formulates the task scheduling problem as minimizing the overall communication cost based on the communication cost matrix for each phase, as illustrated in Fig. 6.

Then, the optimal task scheduling problem is transformed into a Minimum Weighted Bipartite Matching problem and solved using the Hungarian algorithm (Kuhn–Munkres) (Kuhn 1955), Fig. 7 shows a simple example. When necessary, a complete bipartite graph is constructed by adding virtual Executors or tasks.

The Minimum Weighted Bipartite Matching (MM) problem involves selecting the maximum number of non-adjacent edges such that the total weight of the selected edges is minimized. For the constructed weighted bipartite graph, selecting an edge indicates assigning a task to an executor. Choosing non-adjacent edges ensures that the selected edges represent different tasks and executors. In other words, this means assigning each task to only one executor, and each executor can execute only one task at a time. Additionally,

Fig. 5 Job scheduling example

by maximizing the number of non-adjacent edges, the number of scheduled tasks is maximized. MM also minimizes the total weight of the selected edges, thereby minimizing the total communication cost for all scheduled tasks. Therefore, the MM problem in a bipartite graph is equivalent to the optimal task scheduling problem.

This algorithm improves the data locality of map tasks while maximizing the data locality of reduce tasks. It minimizes the overall communication cost of jobs executed within a single data center, thereby accelerating job execution.

5 Experimental evaluation

This section utilizes real-world communication latency and multi-center communication architecture to set up experimental scenarios. Based on these real-world settings, we design simulation data and experimentally validate the proposed algorithm.

5.1 Experimental setup

In this section, we first tested the communication response time in real-world scenarios, with the results presented in Table 2. The actual response times between different nodes form the basis of this experimental part. We also designed an experimental simulation environment based on real data. In the experimental environment, we set up 3, 5, 10 computing centers, connected by a wide-area network (WAN). Each computing center has 5, 10, 15 racks, with every two racks sharing a switch. Each router connects to 10 switches. Each rack contains two computing nodes, with nodes in the same rack directly connected via network cables, and each computing node has five containers.

In this environment, we generated communication latency data using a random algorithm based on the maximum and minimum values of actual communication latency. For example, the communication latency between nodes within the same rack ranges from 0.16 to 0.23 ms, while the latency between nodes in different centers ranges from 24 to 77 ms. The processing speed of each container follows a uniform distribution in $U(10, 30)$ MB per second.

To evaluate performance, we chose to conduct experimental tests using the widely used WordCount task within the MapReduce framework. The number of test jobs is set as follows: $\gamma \in \{10, 20, 30\}$ each job has a random number of map tasks ranging from 50 to 150, with 5 reduce tasks. The input data size for map tasks is set at 256 MB, and the input data size for reduce tasks depends on the output of the map tasks, set at 512 MB.

Due to the limited existing research on algorithms for the considered problem, we have only compared the proposed

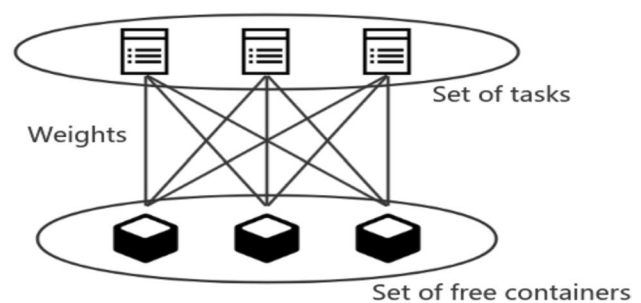


Fig. 6 Weighted bipartite graph construction. If scheduling map tasks, the weight in the graph is calculated as the network distance between the data block required by the task and the selected container, multiplied by the size of the data block. If scheduling reduce tasks, the weight in the graph is calculated as the sum of the network distances between the partitions corresponding to the reduce task from all the intermediate data produced by the map tasks and the selected container, multiplied by the size of the partitions

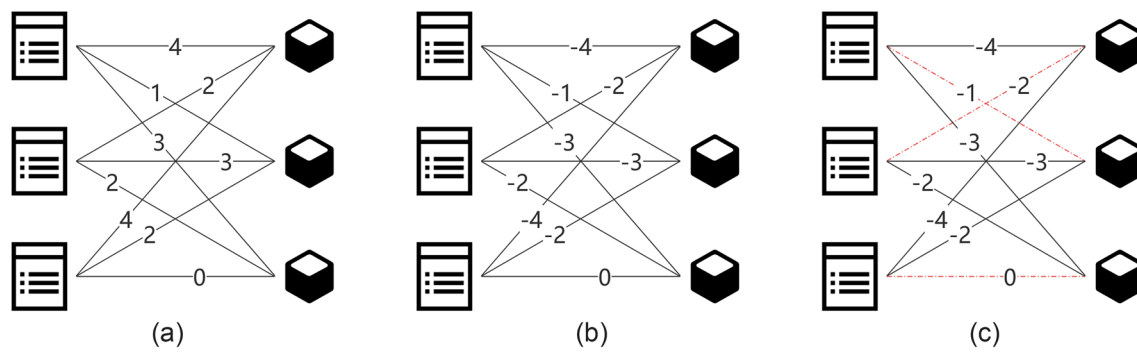


Fig. 7 Example of solving the minimum weighted bipartite matching problem. **a** Shows the matching between three tasks and available containers, with the numbers on the lines representing their weights. The Hungarian algorithm (Kuhn–Munkres) typically solves the maxi-

mum matching problem; to compute the minimum weighted bipartite matching, all weights need to be negated, as shown in **b**. **c** Shows the final solution, with the red lines indicating the selected matches (colour figure online)

algorithm with the MDC-Hadoop algorithm and adaptive algorithms for similar problems:

G-Hadoop: Does not specifically choose a data center for each job. During the map phase, tasks are assigned based on data locality. In the reduce phase, the containers executing tasks are randomly selected.

MDC-Hadoop: For each job, tasks are assigned based on data locality during the map phase. During the shuffle phase, a data center is selected, and in the reduce phase, tasks are executed in the selected data center.

Among the above algorithms, G-Hadoop is an adaptive algorithm for the current problem. MDC-Hadoop is an improvement over the G-Hadoop architecture.

5.2 Comparison of algorithms

First, evaluate the performance of the relevant algorithms, and then compare the performance improvements of the algorithms during the data transmission phase.

Figures 8 and 9 depict the relationship between the average execution time of a single job and the average execution time of the whole job when the number of data centers changes and the data center size is constant but the number of jobs changes for the comparison algorithm respectively. The results show that compared with the MDC-Hadoop algorithm, our proposed algorithm has shorter execution time both from the perspective of single job execution time and the overall job execution time. However, G-Hadoop has poor performance due to the higher communication overhead incurred by not fixing the data center in the reduce phase. As the number of data centers increases, the task execution slows down because the experiment doesn't use enough data, but the amount of data transferred across data centers is higher. So we draw Fig. 10 again to make the transmission time comparison.

Table 2 Response time between nodes at different locations

Node locations	Node locations (ms)	Maximum value (ms)	Average value (ms)
Same rack	0.16	0.23	0.20
Different rack	0.21	0.25	0.22
Different switches	4	14	11
Different routers	18	22	19
Different centers	24	77	55

When selecting an appropriate data center for a job, our proposed algorithm tends to choose clusters with more map-local tasks, thereby reducing the amount of data that needs to be transferred. The time taken to transfer data across clusters is usually a key factor affecting job execution time. In contrast, the MDC-Hadoop algorithm selects clusters during the shuffle phase, resulting in more data that needs to be transferred. Figure 10 shows a comparison of the transmission times between the two algorithms under the same remaining parameters.

Additionally, our proposed algorithm allows for overlapping data transmission and task execution during the map phase, which further saves job execution time. On the other hand, the MDC-Hadoop algorithm strictly follows phase execution, starting the shuffle phase only after the map phase is complete and all data has been transmitted, and then starting the reduce phase.

6 Conclusion

In this paper, we considered improving MapReduce to meet the needs of supercomputing Internet environments and minimize the overall job completion time. We developed a novel

Fig. 8 Comparison of the average execution time per job for the compared algorithms

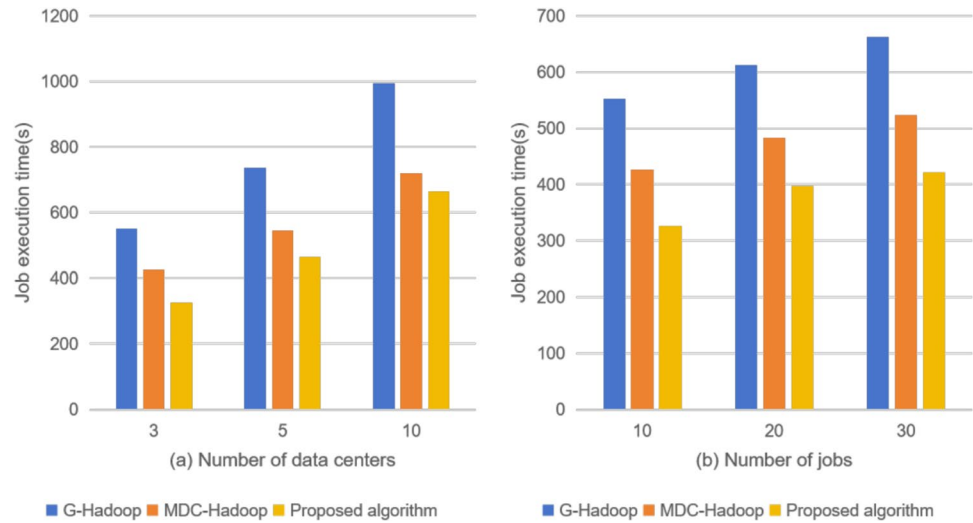


Fig. 9 Overall job average execution time comparison of the compared algorithms

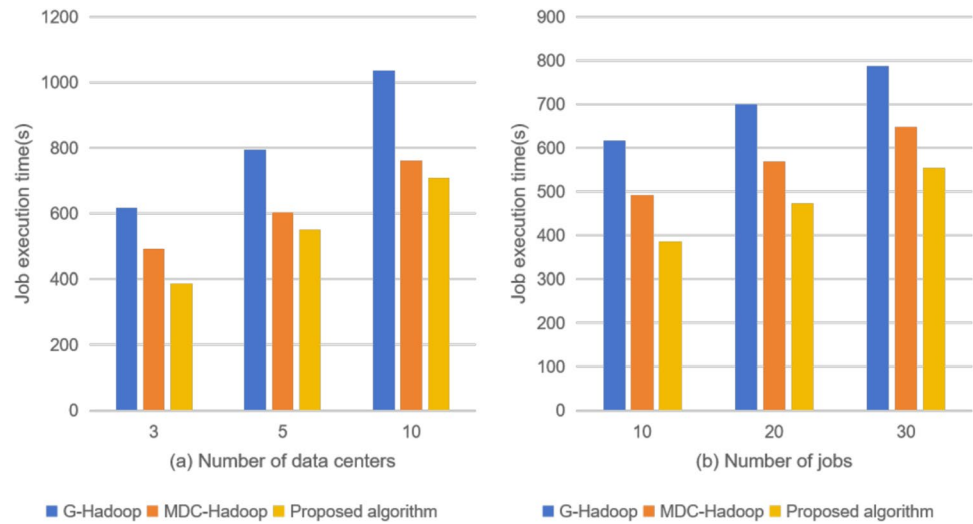
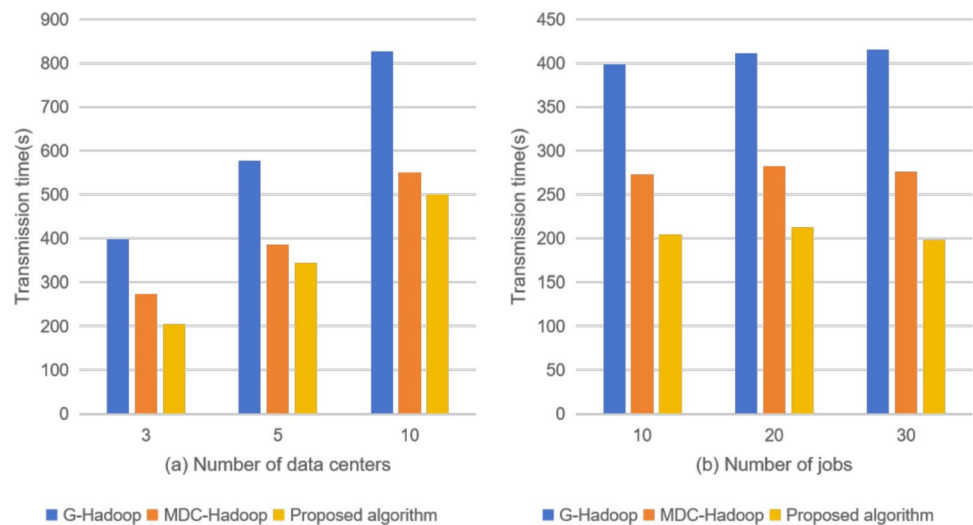


Fig. 10 Comparison of the average transmission time per job



architecture, SIE-Hadoop, to address the problem. This architecture employs a hierarchical scheduling mechanism, significantly reducing the data transfer requirements between different data centers, ensuring rational resource allocation and efficient utilization, thereby enhancing task scheduling efficiency, optimizing resource utilization, and reducing the economic cost of data transfer. Given the potential impact of frequent data transfers on job performance, the SIE-Hadoop architecture uses a data aggregation strategy during data replication to reduce transfer frequency and mitigate the negative effects of the hierarchical network architecture on data transfer. Additionally, an algorithmic framework was designed to schedule MapReduce tasks across the nodes of distributed data centers. The framework is divided into job scheduling and task scheduling steps, aiming to maximize the overlap between computation time and transfer time, thus shortening the overall execution time of jobs. Comparison with existing methods for similar problems demonstrates the effectiveness of our approach. Moreover, in the experiment, scalability tests were conducted by increasing the number of clusters from 3 to 10. The results demonstrated a linear performance improvement, highlighting the framework's strong scalability.

Acknowledgements The work is supported by the National Natural Science Foundation of China (Grant Nos. 62225205, 92055213, 62302160), the Natural Science Foundation of Hunan Province (Grant Nos. 2024JJ6154), the Science and Technology Program of Changsha (kh2301011), Shenzhen Basic Research Project (Natural Science Foundation) (JCYJ20210324140002006).

Data availability The data that support the findings of this study are available from the corresponding author upon reasonable request. Some data may not be made available due to privacy or ethical restrictions.

Declarations

Conflict of interest On behalf of all authors, the corresponding authors state that there is no Conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

Afrati, F., Dolev, S., Sharma, S., Ullman, J.D.: Meta-mapreduce: a technique for reducing communication in mapreduce computations (2015). [arXiv:1508.01171](https://arxiv.org/abs/1508.01171)

- Ahmad, F., Chakradhar, S.T., Raghunathan, A., Vijaykumar, T.: Tarazu: optimizing mapreduce on heterogeneous clusters. *ACM SIGARCH Comput. Archit. News* **40**(1), 61–74 (2012)
- Cardosa, M., Wang, C., Nangia, A., Chandra, A., Weissman, J.: Exploring mapreduce efficiency with highly-distributed data. In: *Proceedings of the Second International Workshop on MapReduce and Its Applications*, pp. 27–34 (2011)
- Chen, J.: Construction of supercomputing internet in the context of computing power networks. *Electron. Commun. Comput. Sci.* **6**(6), 1–3 (2024)
- Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. In: *osdi*, vol. 4, p. 5 (2004)
- Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. *Commun. ACM* **51**(1), 107–113 (2008)
- Duan, M., Li, K., Tang, Z., Xiao, G., Li, K.: Selection and replacement algorithms for memory performance improvement in spark. *Concurr. Comput.: Pract. Exp.* **28**(8), 2473–2486 (2016)
- Fu, Z., Tang, Z., Yang, L., Liu, C.: An optimal locality-aware task scheduling algorithm based on bipartite graph modelling for spark applications. *IEEE Trans. Parallel Distrib. Syst.* **31**(10), 2406–2420 (2020)
- Gadre, H., Roderio, I., Parashar, M.: Investigating mapreduce framework extensions for efficient processing of geographically scattered datasets. *ACM SIGMETRICS Perform. Eval. Rev.* **39**(3), 116–118 (2011)
- Guo, Z., Fox, G., Zhou, M.: Investigation of data locality in mapreduce. In: *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pp. 419–426 IEEE (2012)
- Hadoop, A.: Apache Hadoop (2024). <http://hadoop.apache.org/>
- Hammoud, M., Rehman, M.S., Sakr, M.F.: Center-of-gravity reduce task scheduling to lower mapreduce network traffic. In: *2012 IEEE Fifth International Conference on Cloud Computing*, pp. 49–58. IEEE (2012)
- Hartmann, S., Briskorn, D.: An updated survey of variants and extensions of the resource-constrained project scheduling problem. *Eur. J. Oper. Res.* **297**(1), 1–14 (2022)
- He, B., Fang, W., Luo, Q., Govindaraju, N.K., Wang, T.: Mars: a mapreduce framework on graphics processors. In: *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, pp. 260–269 (2008)
- Heintz, B., Chandra, A., Sitaraman, R.K., Weissman, J.: End-to-end optimization for geo-distributed mapreduce. *IEEE Trans. Cloud Comput.* **4**(3), 293–306 (2014)
- Huang, X., Zhang, L., Li, R., Wan, L., Li, K.: Novel heuristic speculative execution strategies in heterogeneous distributed environments. *Comput. Electr. Eng.* **50**, 166–179 (2016)
- Ibrahim, S., Jin, H., Cheng, B., Cao, H., Wu, S., Qi, L.: Cloudlet: towards mapreduce implementation on virtual machines. In: *Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing*, pp. 65–66 (2009)
- Jin, C., Buyya, R.: Mapreduce programming model for net-based cloud computing. In: *Euro-Par 2009 Parallel Processing: 15th International Euro-Par Conference, Delft, The Netherlands, August 25–28, 2009. Proceedings 15*, pp. 417–428. Springer (2009)
- Kim, S., Won, J., Han, H., Eom, H., Yeom, H.Y.: Improving Hadoop performance in intercloud environments. *ACM SIGMETRICS Perform. Eval. Rev.* **39**(3), 107–109 (2011)
- Kuhn, H.W.: The Hungarian method for the assignment problem. *Nav. Res. Logist. Q.* **2**(1–2), 83–97 (1955)
- Li, X., Chen, F., Ruiz, R., Zhu, J.: Mapreduce task scheduling in heterogeneous geo-distributed data centers. *IEEE Trans. Serv. Comput.* **15**(6), 3317–3329 (2021)
- Liu, G., Zhu, X., Wang, J., Guo, D., Bao, W., Guo, H.: Sp-partitioner: a novel partition method to handle intermediate data skew in spark streaming. *Future Gener. Comput. Syst.* **86**, 1054–1063 (2018)

- Mattess, M., Calheiros, R.N., Buyya, R.: Scaling mapreduce applications across hybrid clouds to meet soft deadlines. In: 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA), pp. 629–636. IEEE (2013)
- Naik, N.S., Negi, A., Br, T.B., Anitha, R.: A data locality based scheduler to enhance mapreduce performance in heterogeneous environments. *Future Gener. Comput. Syst.* **90**, 423–434 (2019)
- Pallickara, S., Ekanayake, J., Fox, G.: Granules: a lightweight, streaming runtime for cloud computing with support, for map-reduce. In: 2009 IEEE International Conference on Cluster Computing and Workshops, pp. 1–10. IEEE (2009)
- Pan, F., Xiong, J., Shen, Y., Wang, T., Jiang, D.: H-scheduler: storage-aware task scheduling for heterogeneous-storage spark clusters. In: 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS), pp. 1–9. IEEE (2018)
- Rabkin, A., Arye, M., Sen, S., Pai, V., Freedman, M.J.: Making every bit count in {Wide-Area} analytics. In: 14th Workshop on Hot Topics in Operating Systems (HotOS XIV) (2013)
- Ryden, M., Oh, K., Chandra, A., Weissman, J.: Nebula: distributed edge cloud for data intensive computing. In: 2014 IEEE International Conference on Cloud Engineering, pp. 57–66. IEEE (2014)
- Sciences Bulletin, C.A.: "National Supercomputer Internet" is to make the national supercomputer network? It's not that simple (2024). http://old2022.bulletin.cas.cn/zgkxyyk/ch/reader/view_news.aspx?id=20230830103959001
- Selvitopi, O., Demirci, G.V., Turk, A., Aykanat, C.: Locality-aware and load-balanced static task scheduling for mapreduce. *Future Gener. Comput. Syst.* **90**, 49–61 (2019)
- Shan, Y., Wang, B., Yan, J., Wang, Y., Xu, N., Yang, H.: Fpmr: mapreduce framework on fpga. In: Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays, pp. 93–102 (2010)
- Spark, A.: Apache spark (2024). <http://spark.apache.org/>
- Tang, Z., Ma, W., Li, K., Li, K.: A data skew oriented reduce placement algorithm based on sampling. *IEEE Trans. Cloud Comput.* **8**(4), 1149–1161 (2016)
- Tomarchio, O., Di Modica, G., Cavallo, M., Polito, C.: A hierarchical Hadoop framework to handle big data in geo-distributed computing environments. *Int. J. Inf. Technol. Syst. Approach: IJITSA* **11**(1), 16–47 (2018)
- Wang, L., Tao, J., Ranjan, R., Marten, H., Streit, A., Chen, J., Chen, D.: G-hadoop: mapreduce across distributed data centers for data-intensive computing. *Future Gener. Comput. Syst.* **29**(3), 739–750 (2013)
- Xue, R., Gao, S., Ao, L., Guan, Z.: Bolas: bipartite-graph oriented locality-aware scheduling for mapreduce tasks. In: 2015 14th International Symposium on Parallel and Distributed Computing, pp. 37–45. IEEE (2015)
- Zacheilas, N., Kalogeraki, V.: Chess: cost-effective scheduling across multiple heterogeneous mapreduce clusters. In: 2016 IEEE International Conference on Autonomic Computing (ICAC), pp. 65–74. IEEE (2016)
- Zhang, Q., Liu, L., Lee, K., Zhou, Y., Singh, A., Mandagere, N., Gopisetty, S., Alatorre, G.: Improving hadoop service provisioning in a geographically distributed cloud. In: 2014 IEEE 7th International Conference on Cloud Computing, pp. 432–439. IEEE (2014)

Zhang, J., Zhang, L., Huang, H., Jiang, Z.L., Wang, X.: Key based data analytics across data centers considering bi-level resource provision in cloud computing. *Future Gener. Comput. Syst.* **62**, 40–50 (2016)



Yalin Zhu graduated from Hunan University, Changsha, China, in 2023. Currently, she is working towards the master's degree in the College of Computer Science and Electronic Engineering, Hunan University, Changsha, China.



Youquan Chang vice president of technological innovation department in CMCC Hunan Company. He gained his master degree in Hunan University in 2020. He's known as a professional engineer in AI, machine learning and software development and has conduct several research project in CMCC Hunan Company. His recent achievement is the third level of Science and Technology Progress Award in CMCC.



Jiapeng Zhang received the B.E. degree in the Department of Electronic Engineering and Information Science from the University of Science and Technology of China, Hefei, China, in 2017 and the Ph.D. degree in Electronic Engineering from Shanghai Jiao Tong University, China, in 2022. He is currently

an assistant professor in College of Computer Science and Electronic Engineering in Hunan University. His research interests include high performance parallel algorithms, resource scheduling design and network performance analysis.



Yingjie Song received the MS degree from Hunan University, Changsha, China, in 2019. Currently, she is working towards the Ph.D. degree in the College of Computer Science and Electronic Engineering, Hunan University, Changsha, China. Her research interests include deep learning and the distributed computing.



Zhuo Tang received the Ph.D. in computer science from Huazhong University of Science and Technology, China, in 2008. He is currently a professor of the College of Computer Science and Electronic Engineering at Hunan University. He is also the chief engineer of the National Supercomputing Center in Changsha. His majors are distributed computing system, cloud computing, and parallel processing for big data,

including distributed machine learning, security model, parallel algorithms, and resources scheduling and management in these areas. He has published almost 120 journal articles and book chapters. He is a member of IEEE/ACM and CCF.