Received 30 June 2023, accepted 18 July 2023, date of publication 24 July 2023, date of current version 2 August 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3298049

RESEARCH ARTICLE

A Task-Oriented General-Purpose Distributed Computing System Based on CLTS Scheduling Algorithm

MIN HUANG[1,2], YUNXIANG ZHAO[2], AND YAZHOU CHEN[1], (Member, IEEE)
[1]Army Engineering University, Shijiazhuang Campus, Shijiazhuang 050003, China
[2]Hebei University of Science and Technology, Shijiazhuang 050018, China

Corresponding author: Yazhou Chen (chen_yazhou@sina.com)

This work was supported by the Defense Industrial Technology Development Program under Grant JCKYS2020DC1.

ABSTRACT Most current research on commonly used distributed computing frameworks neglects the difficulty of learning these frameworks. These commonly used distributed computing frameworks are designed for professionals, and users must understand them well to use them for calculation tasks. In order to solve the challenging balance problem between versatility and user learning difficulty, a new distributed computing system is proposed. The design of this system is based on the idea of divide and conquer in distributed computing. The paper uses Netty, Zookeeper, Hadoop Distributed File System (HDFS) and other tools to implement the system functions. Firstly, the paper analyzes the research status of distributed computing frameworks at home and abroad; Then, according to the idea of divide and conquer, the paper uses Netty to implement the basic architecture of the system, uses Zookeeper and Netty to implement relevant distributed mechanisms, uses HDFS as the distributed storage, designs and implements a Task-Oriented General-Purpose (TOGP) distributed computing system, TOGP; Finally, the paper intensely studies the task scheduling problem of the system, proposes a scheduling algorithm, Node Processing Capacity and Distributed Lock Task Scheduling (CLTS) algorithm. The system scalability test proves that the TOGP has good horizontal scalability. In the comparative experiment, the processing capacity of the TOGP using the round robin is about 16% higher than that of Hadoop, which proves that the TOGP is available and efficient. A series of experiments prove that the CLTS scheduling algorithm can effectively adapt to the heterogeneous cluster environment and efficiently schedule tasks.

INDEX TERMS Distributed computing, load balance task-oriented, task scheduling.

I. INTRODUCTION

With the rapid development of information technology, massive data will be generated daily, and the vast amount of data has far beyond the range that a single computer can handle. Currently, there are two methods to deal with such an enormous amount of data: centralized computing [1] and distributed computing [2]. Centralized computing refers to the use of parallel computing technologies such as OpenMP to make full use of the computing resources of one computer, which almost wholly depends on the processing capacity of one computer, to improve computing efficiency. Distributed computing makes full use of the computing capacity of multiple computers by dividing the calculation task into several subtasks and assigning these subtasks to multiple computers for processing in turn, thereby saving computing time and improving computing efficiency. The programming of centralized computing is simple, and the project is easy to maintain. However, centralized computing does not have scalability at the computer level. The computing capacity of a centralized computing system depends entirely on the performance of one computer. If one computer cannot provide sufficient computing capacity, it can only upgrade hardware equipment or use another higher-performance computer.

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

Compared with centralized computing, distributed computing has a broader application prospect: distributed computing frameworks have scalability, which can expand more machines in the computer cluster to improve the computing capacity of the entire computing system. At the fault tolerance level, the computing information can be stored in other computers in the cluster through redundant backup and other ways so that the collapse of a few computers will not affect the regular operation of the entire cluster. In addition, using centralized computing to process big data has high requirements for computer configuration. In contrast, distributed computing has a higher cost-performance ratio. Users only need to use several ordinary computers to process calculation tasks that only high-performance computers can centrally process. Nowadays, distributed computing technology has been widely used to process data. Up to now, there are many commonly used distributed computing frameworks, such as Hadoop and Spark. These distributed computing frameworks can be applied to many fields, as shown in Figure 1.

These distributed computing frameworks have been widely used. With the continuous iterative updating of versions, the structure of these distributed computing frameworks has become increasingly complex. Around these distributed computing frameworks and related tools, one relatively large ecosystem [3], [4] and a research system have been formed. Many papers have conducted in-depth research on the parameter configuration, job scheduling and analysis, data processing flow and other aspects of these frameworks. These studies cannot change the fact that these commonly used distributed computing frameworks are designed for professionals. Users need to have a deep understanding of the framework structure to use the framework to execute distributed calculation tasks efficiently. Nevertheless, even these common distributed computing frameworks cannot solve all distributed computing problems.

Taking Hadoop as an example, under the MapReduce programming model, users only need to implement the mapper, reducer, and driver classes to deal with data. However, there are still problems with Hadoop itself: MapReduce is unsuitable for processing all batch tasks. When users want to process some tasks inappropriate for disassembling into Map and Reduce, they must consider transforming the business logic into MapReduce, dramatically increasing the users' workload. In addition, common distributed computing frameworks such as Hadoop and Spark are challenging to meet some special distributed computing requirements. For example, distributed computing technology is difficult to apply to large-scale simulation calculation. The model data in the field of simulation calculation is stored in a particular format and can only be read with simulation calculation software. The particular format of data and the particular processing method of these data make it difficult for distributed computing frameworks such as Hadoop or Spark to give full play to their advantages. Suppose users want to force distributed computing frameworks such as Hadoop or Spark to handle these special requirements. In that case, the original structure of the frameworks needs to be significantly modified, and the cost is too high.

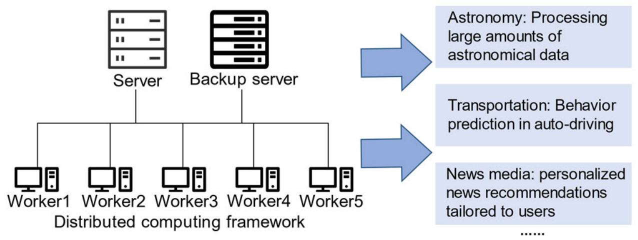

Server
Backup server
Worker1
Worker2
Worker3
Worker4
Worker5
Distributed computing framework
Astronomy: Processing large amounts of astronomical data
Transportation: Behavior prediction in auto-driving
News media: personalized news recommendations tailored to users
......

FIGURE 1. Application fields of distributed computing frameworks.

Many special distributed computing requirements have emerged with the wide application of distributed computing technology. Designing a corresponding distributed computing system for each requirement is impossible from the cost perspective. If a distributed computing system can simultaneously have relatively low cost and the versatility to handle particular distributed calculation tasks, it must have the following characteristics:

1) The system needs to maintain a lightweight structure. The lightweight structure can reduce the system's learning difficulty and deployment cost. Users can deploy and use the distributed computing system to complete calculation tasks at a lower cost.

2) The system needs to provide users with some Application Programming Interfaces (API) to modify the system according to their needs and then handle some particular distributed calculation tasks to achieve its versatility.

3) The system needs to utilize the resources in the cluster fully. Different calculation tasks rely on different parts of the computers in the cluster, such as computationally intensive tasks that rely more on CPUs. Therefore, to adapt to the needs of various types of distributed calculation tasks, the system must be able to utilize all resources in the cluster.

Therefore, it is necessary to design a more general-purpose distributed computing system. In order to reduce users' learning difficulty and maintain versatility, the system needs to encapsulate the details of distributed computing and provide users with some corresponding APIs to allow users to implement relevant business logic according to actual needs. This paper proposes a task-oriented general-purpose distributed computing system, TOGP. The implementation of TOGP uses Netty, an event-driven network application development framework, Zookeeper, a distributed application coordination service software, and HDFS. According to the current needs and application scenarios, TOGP focuses more on batch processing.

The TOGP adopts the task-oriented design concept to achieve versatility. By dividing the calculation task into data and code and managing them separately, users can call data and code like calling components or modify the way the system reads data according to the actual needs of the calculation task. The TOGP implements the task type mechanism, allowing users to customize new task types under the system structure so that even if the system itself cannot fully meet the

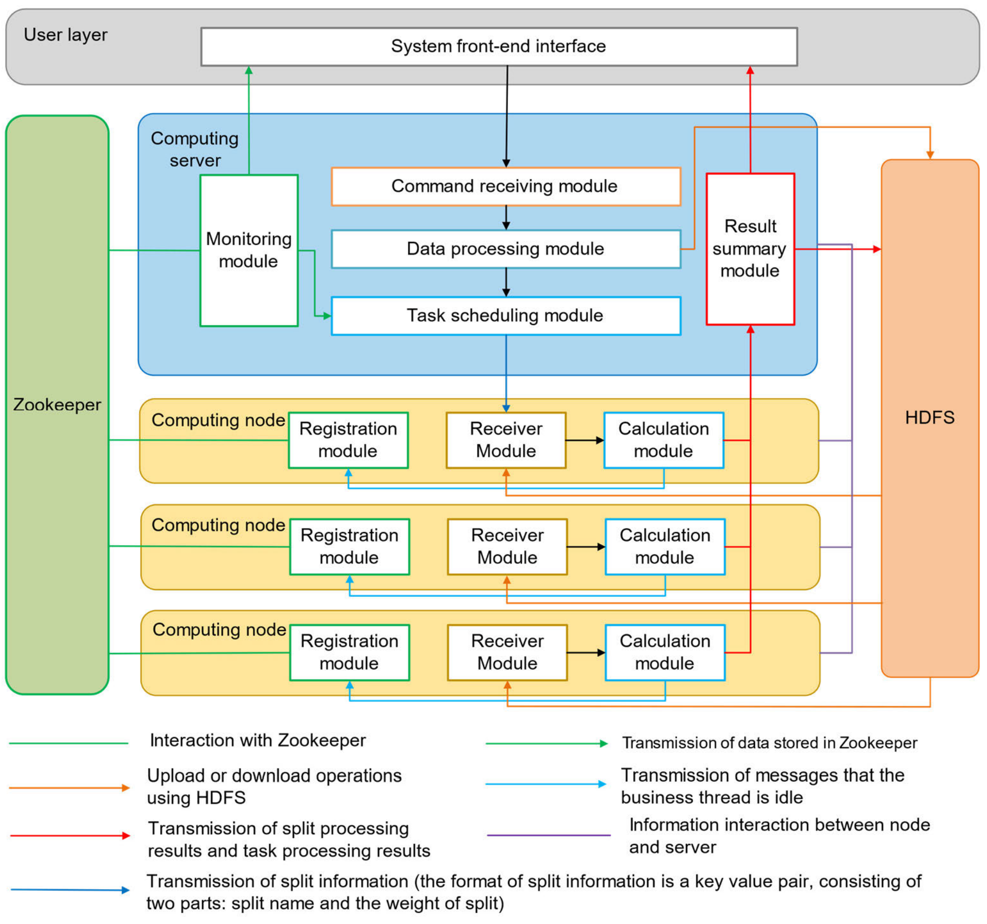

User layer
System front-end interface
Computing server
Monitoring module
Command receiving module
Data processing module
Task scheduling module
Result summary module
Zookeeper
HDFS
Computing node
Registration module
Receiver Module
Calculation module
Interaction with Zookeeper
Transmission of data stored in Zookeeper
Upload or download operations using HDFS
Transmission of messages that the business thread is idle
Transmission of split processing results and task processing results
Information interaction between node and server
Transmission of split information (the format of split information is a key value pair, consisting of two parts: split name and the weight of split)

FIGURE 2. Organization chart of the TOGP.

requirements of particular calculation tasks, users can solve this problem by customizing task types and corresponding business logic. Based on the TOGP, this paper profoundly studies the task scheduling problem of the system. As the research result, the paper proposes the CLTS scheduling algorithm, which can effectively improve the efficiency of task scheduling in a heterogeneous cluster environment.

To sum up, the major contributions of this paper are as follows:

1) This paper proposes a distributed computing system, TOGP. The TOGP is designed and implemented using Netty, Zookeeper, and HDFS.

2) Using task-oriented design concepts to treat data and code independently. Users can customize the reading method of data.

3) This paper proposes the CLTS scheduling algorithm. The CLTS scheduling algorithm can adapt to the heterogeneous cluster environment and efficiently schedule tasks.

The remainder of the paper is organized as follows. Section II discusses related work. Section III describes the design and implementation of the TOGP and the CLTS scheduling algorithm. Section IV shows the experimental results and analysis. Section V concludes the paper.

II. RELATED WORK

Google put forward the distributed computing model MapReduce in the paper [5] published in 2004. This computing model abstracts the business into two parts: Map and Reduce. Users only need to write their Map and Reduce programs according to their business needs and then write a driver to realize distributed computing. There is still much research [6], [7], [8] and applications [9], [10] on the MapReduce model.

In 2009, the AMP laboratory of Berkeley University proposed the distributed computing framework Spark and published a related paper [11] in 2010. Unlike Hadoop, which has a distributed storage system, HDFS, and a resource management tool, Yarn, Spark is more like a tool for processing big data. It does not provide a file management system and will not store data. Spark's distributed memory computing model is more efficient and inherits MapReduce's horizontal scalability. Spark uses Directed Acyclic Graph (DAG) to describe the calculation logic, which solves the limitations of the MapReduce model.

In 2013, Zaharia et al. improved Spark and proposed Spark Streaming [12], a distributed computing framework for stream processing using micro-batch processing.

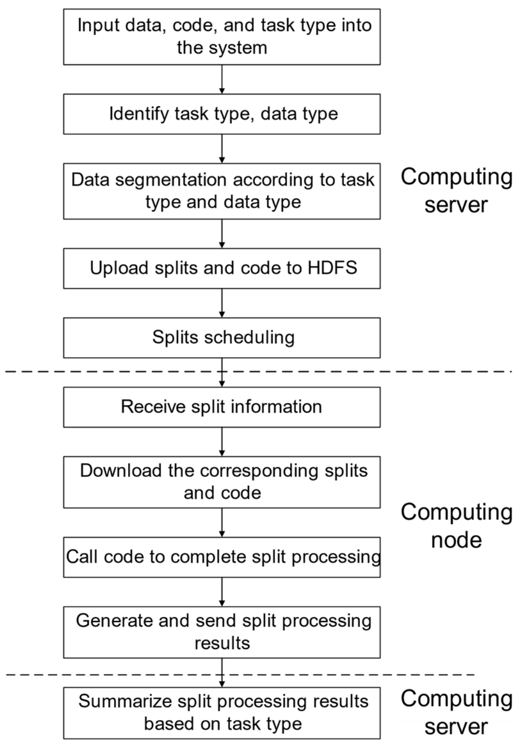

Input data, code, and task type into the system
Identify task type, data type
Data segmentation according to task type and data type
Upload splits and code to HDFS
Splits scheduling
Computing server
Receive split information
Download the corresponding splits and code
Call code to complete split processing
Generate and send split processing results
Computing node
Summarize split processing results based on task type
Computing server

FIGURE 3. **The data processing flow of the TOGP.**

In 2015, Carbon et al. proposed the distributed computing framework Flink [13] based on the data flow model. Flink supports batch processing and stream processing. In the same year, Kulkarni et al. proposed the distributed computing framework Heron [14]. It solves the problems of the distributed stream processing system Strom in the execution model, resource allocation and task scheduling.

In 2017, Wei et al. designed and implemented a distributed computing framework, OpenCluster [15], to process massive data generated by modern astronomical telescopes. This framework can be used to rapidly develop high-performance data processing systems for astronomical telescopes and significantly reduce software development costs. In the same year, to solve the problem that the existing distributed computing frameworks do not support the Reinforcement Learning (RL) algorithm, UC Berkeley RISE-Lab proposed the distributed computing framework Ray [16], which uses the dynamic task graph computing model. In the same year, Venkataraman et al. improved Spark Streaming and proposed the distributed computing framework Drizzle [17], which reduced the processing delay to 100ms while retaining the low recovery delay of Spark Streaming. In 2017, Wu et al. designed and implemented the distributed parallel computing framework SummaryMR [18] using C++ to solve the problem of complex configuration steps and the lack of scalability of the existing framework.

In 2019, Saha et al. proposed the Mesos framework for container based Message Passing Interface (MPI) jobs, Scylla [19]. This framework uses docker swarm to realize the communication between the containerized tasks (MPI processes) and Mesos and act as resource managers, thus realizing the distributed computing of MPI tasks.

In 2020, Prakash et al. improved the shortcomings of traditional general distributed computing frameworks such as MapReduce in graph computing. Using the improved MapReduce model, they solved the Erdos-Renyi (ER) graph problem [20]. In the same year, Xu et al. designed and implemented a distributed computing framework [21] capable of predicting wind speed big data based on Spark.

In 2023, Sun et al. analyzed the MapReduce programming model, proposed a random sample partition data model for the problem of the MapReduce programming model in processing big data, and designed a non-MapReduce distributed computing framework for this data model [22].

According to the above analysis of the current research situation, scholars at home and abroad study distributed computing frameworks in two ways. The first is to conduct more in-depth research based on a commonly used distributed computing framework to make up for the original shortcomings of the distributed computing framework, while the second is to design a new distributed computing framework. The designed distributed computing framework will often process data in a particular format. So far, mainstream distributed computing frameworks have formed an enormous ecosystem after years of development. Further optimization based on the mainstream distributed computing frameworks often does not reduce the complexity and difficulty of these frameworks, so the current situation of mainstream distributed computing frameworks for professionals has stayed the same. Most of the distributed computing frameworks designed by scholars are designed to handle some particular type of data, and these frameworks are often not versatile enough.

III. DESIGN AND IMPLEMENTATION OF TOGP

The TOGP adopts the Client/Server structure. The Client/Server structure is a software system architecture that can fully utilize the advantages of the hardware environment on both sides, allocate tasks reasonably to the client and server sides, and reduce communication overhead. In the TOGP, the computing server and computing node are implemented by Netty. The computing nodes deployed on different computers can communicate with the computing server through the network to complete the calculation task together. The design idea of the TOGP is: that the computing server receives the data, code and task type uploaded by users and divides the data into data splits according to the task type and data type; The computing node can dynamically join the cluster. The computing server will dynamically allocate the split information to the computing nodes according to the status of the current online nodes. The computing node will call the code to process the splits and upload the split processing results to the computing server. Finally, the computing server will summarize and count according to the task type to obtain the running results of the calculation task. Figure 2 shows the

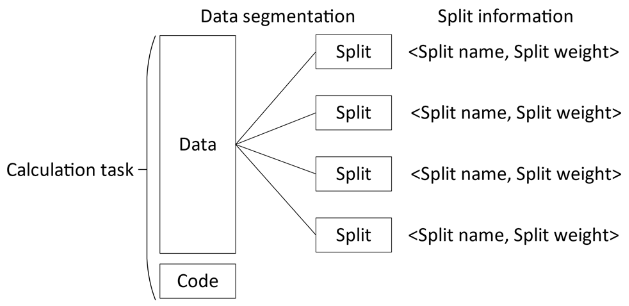


FIGURE 4. How the TOGP processes calculation tasks.

organizational structure of the TOGP. Figure 3 shows the data processing flow of the system.

Figure 4 shows how the TOGP processes calculation tasks. The TOGP divides the calculation task into data and code and manages them separately. Through data segmentation, TOGP divides data into splits. Splits' name combines the data name and serial number, and TOGP does not allow the same data name to appear. So, each split has a unique split name. TOGP will collect the split names of these splits and calculate the weight of each split, ultimately generating split information in key-value pair format.

The specific flow of system data processing can be obtained by combining Figure 2, Figure 3 and Figure 4. When the user submits data, code and task type to the TOGP, the command receiving module receives the information submitted by the user and checks whether the corresponding file exists. If there is a corresponding file, the data processing module will find the corresponding segmentation method according to the task type and data type, segment the data into splits, and upload the splits and code to HDFS. The task scheduling module will schedule according to the computing nodes' status and split information. Finally, based on the scheduling results, the computing server will send split information to the current online computing nodes.

After receiving the split information, the computing node's receiver module will download the code and corresponding splits from the HDFS. And then, the calculation module will call the code to process the splits and send one message and split processing result to the registration and result summary modules. The message sent to the registration module will be fed back to the monitoring module of the computing server. It will eventually be used as a part of the computing node status as a reference for task scheduling. The result summary module of the computing server receives the split processing results uploaded by each node, calls the corresponding summary method according to the task type, and finally obtains the running results of the calculation task. The running results will be backed up to HDFS and submitted to users.

The TOGP requires at least one computing server and one computing node. Theoretically, the system can allow hundreds of nodes to participate in distributed computing. The actual maximum number of nodes that the system can run depends on the performance of the computing server, network communication rate, queue length of the task scheduling

TABLE 1. Custom communication protocol.

Protocol type	Type of data to be transmitted	Protocol function
0x00	Boolean	Delete redundant node addresses
0x01	String	Send the information entered by the user to the data processing module
0x02	Map	Send the storage address of the splits on the HDFS
0x03	String	Send split processing results
0x04	String	Send IP: port format node address
0x05	String	Send information related to split processing

TABLE 2. Task types supported by the TOGP.

Task type	Description
DistributedCalculationTask	It is the most fundamental task type in the system and the parent of other task types.
MapReduceTask	For MapReduce-type tasks, the split processing result is a key-value pair.
IterativeTask	In iterative tasks, the split itself will go through multiple rounds of processing.
Other	It only uses distributed computing technology to process data in a way that does not need to summarize the results of split processing or only needs to merge the results.

module and other indicators. In the TOGP, the computing server and computing node are usually deployed separately in the cluster computers. One single computer can deploy one computing server and multiple computing nodes at the same time. Deploying multiple nodes on one computer may cause different nodes to interact with each other, so this is not recommended.

The system uploads data splits and code to HDFS mainly for fault tolerance. HDFS will store the data of each file in blocks, and each data block will keep multiple copies. These copies of data blocks are distributed on different nodes in the HDFS cluster, ensuring that even if there are problems with a few computing nodes, they will not affect the relevant data of the calculation task.

Some communications in the TOGP use the customized communication protocol based on MessagePack and Netty. Table 1 shows the specific contents of the communication protocol.

In distributed computing, the processes of different types of calculation tasks vary greatly. Hence, most distributed computing frameworks provide multiple operators or multiple computing models to achieve versatility. The TOGP uses task-oriented design, classifies the calculation tasks according to the summary logic of the splits computing results, summarizes three common task types under the most basic task types, and designs the corresponding data segmentation method and processing results summary methods according to the three task types. Table 2 lists the task types supported by the TOGP.

Firstly, the TOGP defines the most basic task type, distributed calculation task, which does not have the corresponding data segmentation and result summary methods. However, other task types can inherit from this type. Developers can inherit new task types by inheriting this type.

Under the most basic task type, the TOGP sets three task types by default. Each task type has its corresponding data segmentation and result summary method. If developers implement new task types, they must also implement corresponding data segmentation and result summary logic. The "other" type is mainly considered for special-needs calculation tasks. For example, when using simulation software to calculate models, if the angle is used as the basis for partitioning, different computers in the cluster can conduct simulation experiments according to different angles, and the final experimental results only need to be merged.

By following various measures, TOGP is easy for users to learn and use:

1) There is a deployable front-end interface program for TOGP. The front-end interface program can be connected to the computing server, allowing users to submit or check computing tasks in the web page, and monitor the cluster status. The front-end interface does not involve distributed mechanisms or scheduling algorithm mechanisms. TOGP can be run directly without the front-end interface.

2) The TOGP provides users with some APIs to modify the TOGP according to their needs and then handle some particular calculation tasks. TOGP's lightweight structure can reduce the system's learning difficulty and deployment cost.

3) The processing of data by TOGP relies on the code uploaded by users. Users do not need to transform the business logic into a certain computing model.

A. COMPUTING SERVER

The computing server is the core of TOGP. Its main jobs include:

1) Receiving the data, code and task type uploaded by users.

2) Segmenting the data according to the task type and data type.

3) Uploading the splits and code to HDFS.

4) Monitoring the status of the computing nodes.

5) Scheduling the splits according to the status of the computing nodes and the split information.

6) Monitoring the running status of the calculation tasks.

7) Receiving the split processing results.

8) Summarizing and statistics the split processing results.

The monitoring mechanism on the computing node is based on Zookeeper. When the computing node goes online, an ephemeral node will be registered in the designated directory of the Zookeeper cluster. The naming format of the ephemeral node is IP: port. The session between the computing node and the computing server ends when the computing node is offline. Because the life cycle of the Zookeeper ephemeral node depends on the session, the ephemeral node will be deleted. As long as the computing server uses Zookeeper's watch mechanism to monitor the specified directory, it can sense all computing nodes currently online on time. The computing node will send heartbeat packets to the server at regular intervals. If the computing node cannot connect to the computing server, it will try to reconnect.

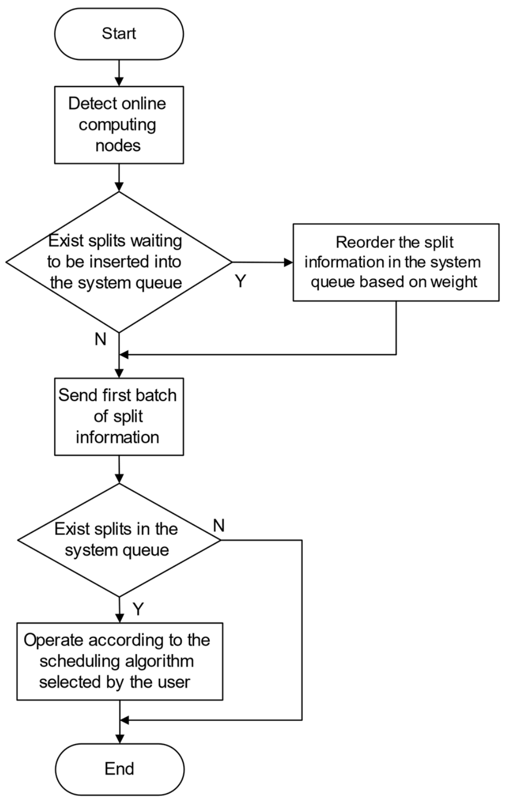

Start
Detect online computing nodes
Exist splits waiting to be inserted into the system queue
Y
Reorder the split information in the system queue based on weight
N
Send first batch of split information
Exist splits in the system queue
N
Y
Operate according to the scheduling algorithm selected by the user
End

FIGURE 5. The flow chart of the task scheduling module.

The core of the computing server is the task scheduling module. The task scheduling algorithm of the system is built into the task scheduling module. Before sending split information to computing nodes every time, the task scheduling module will check the current online nodes. The design of the task scheduling module allows the node to join or exit the operation at any time. Suppose a computing node is offline and cannot automatically reconnect to the server. In that case, the computing server will recycle unfinished splits allocated to this node by collecting information on these splits. The information on unfinished splits will rejoin the system queue. The task scheduling module maintains the system queue, and the system queue is composed of three queues. These three queues represent low, medium and high task priorities. The task scheduling module will extract the split information from the queue like a weighted round robin and send it to the computing node.

Figure 5 shows the flow chart of the task scheduling module.

In the task scheduling module, there are two scheduling algorithms. One of the algorithms is round robin. When the round robin is used as the scheduling algorithm, the system will directly send splits evenly to each computing node without considering the computing capacity of nodes, which means that the round robin cannot schedule splits in real time according to the processing capacity of nodes Given the shortcomings of the round robin, this paper introduces the distributed service registration and discovery mechanism and the dynamic feedback mechanism of node processing capacity into the task scheduling problem of the TOGP and proposed the CLTS scheduling algorithm. The CLTS scheduling algorithm is based on node processing capacity and distributed lock. It is designed to adapt to heterogeneous cluster environments and schedule tasks according to the actual processing capacity of nodes.

B. COMPUTING NODE

The main jobs of the computing node include:

1) Receiving the split information sent by the computing server.

2) Downloading the code and corresponding splits from HDFS according to the name of splits.

3) Calling the code to process the splits.

4) Sending the split processing results to the computing server.

5) Monitoring the local status and collecting status information, such as CPU utilization, memory utilization, hard disk utilization, etc.

6) Sending the heartbeat packet to the computing server.

7) Updating its state information to the Zookeeper cluster.

The computing node regularly monitors the local computer state through Operating System and Hardware Information (OSHI). It puts the local state information into the heartbeat packet sent to the computing server. Because there is already a distributed service registration and discovery mechanism implemented by the Zookeeper ephemeral node feature to determine whether the node is online, in the TOGP, the heartbeat packet sent by the computing node is mainly used to send the local state information to the server.

Because of the requirement of task scheduling and distributed service registration and discovery mechanism, the computing node needs to connect with the Zookeeper cluster. Through the Zookeeper cluster, the status of computing resources and the processing status of splits are fed back to the computing server as a reference for the server to schedule tasks.

The computing node's calculation module maintains a business thread pool to implement parallel processing within the node. The capacity of the business thread pool depends on the number of CPU cores of the computer where the node is located. The capacity of the business thread pool will also be sent to the server as part of the node status information.

C. CLTS SCHEDULING ALGORITHM

The scheduling algorithm used in distributed computing frameworks determines the utilization rate of computing resources to a large extent. The TOGP uses round robin as the scheduling algorithm in the system implementation phase. As the default scheduling algorithm for distributed computing frameworks such as Flink and Storm, the round robin has the advantages of fewer parameters, simple operation and easy replication. However, in practical applications, the round robin performs poorly in the heterogeneous cluster. The heterogeneous cluster considered in this paper refers to a computer cluster with different computing capacities due to significant differences in disk capacity, CPU cores and speed, and memory capacity among multiple internal nodes. In the situation of using a distributed computing system that can run multiple calculation tasks simultaneously, along with the start or end of other calculation tasks, the computing capacity that can be provided for the newly added calculation task on the same computer will also change.

Since the development of distributed computing technology, many task scheduling algorithms have been formed. These task scheduling algorithms can be divided into classical and heuristic task scheduling algorithms. Classic scheduling algorithms include round robin, first come first service algorithm, fair scheduling algorithm, etc. The heuristic task scheduling algorithm comprises a genetic algorithm, ant colony optimization, simulated annealing algorithm, particle swarm optimization algorithm, etc.

This paper studies task scheduling algorithms to solve the problem of system load balancing. It is found that although some of the current classical scheduling algorithms have few parameters and simple operations and are easy to replicate, these classical scheduling algorithms do not consider the performance differences between the nodes in the heterogeneous cluster and are prone to load imbalance. While some current heuristic task scheduling algorithms can adapt to complex scenarios and rapidly expand the global search, they have problems such as high randomness, easy fall into local optimal solutions, and challenging control parameters. Therefore, this paper studies the shortcomings of round robin in calculation task processing and proposes the CLTS scheduling algorithm.

CLTS scheduling algorithm is designed based on the premise that the number of data splits is far more than the sum of each node business thread pool. Most distributed calculation tasks meet this premise considering the large amount of data in distributed computing. CLTS scheduling algorithm is further designed based on the round robin implemented in the TOGP. Based on the round robin, the CLTS scheduling algorithm has three problems to be solved.

The first problem is the control of split scheduling. In the round robin, because the task scheduling module only needs to know whether the computing nodes are online, it directly gives the scheduling results of all splits according to the nodes' online status. Suppose a computing node is offline and cannot reconnect to the server. In that case, the task scheduling module will schedule splits sent to this node, and the computing server did not receive processing results again, according to the round robin. According to the parallel mechanism of computing nodes, the capacity of the business thread pool on computing nodes is limited. If the split information that exceeds the capacity of the thread pool is received, the

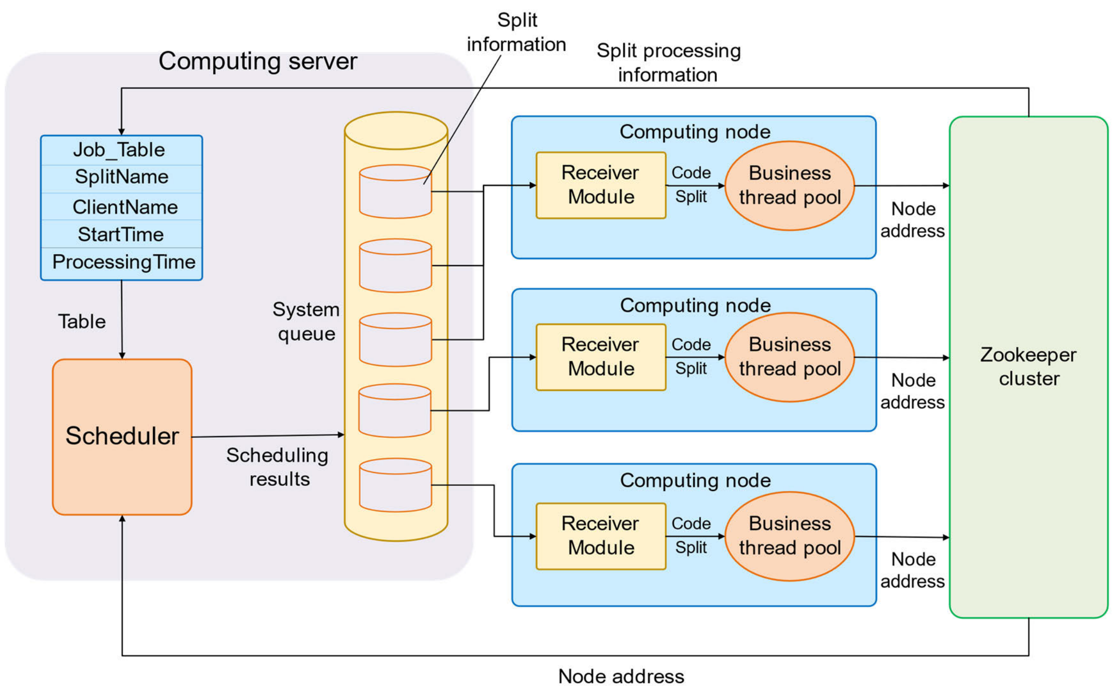


FIGURE 6. **The structure of the CLTS scheduling algorithm.**

redundant split information can only wait in the thread pool's queue. If the task scheduling module sends too much split information to one computing node at one time, and then the node suddenly drops and cannot reconnect, it can only start the fault-tolerant mechanism of the system. The system needs to find a large number of split information sent to this node and continue to send this split information again, which is costly. Therefore, whether from the perspective of computing efficiency or fault tolerance, it is necessary to control the scheduling of splits.

The second problem is the perception of node computing resources. In the TOGP, computing resources refer to the number of idle threads in the business thread pool. The second problem is closely related to the first problem. Under the condition that the server can sense the nodes' computing resources, the task scheduling module can send the appropriate amount of split information to computing nodes with idle threads according to the number of idle threads of each node to avoid accumulating split information in the queue.

The third problem is the perception of nodes' processing capacity. After solving the first two problems, the task scheduling module can only perceive which node has idle business threads but does not perceive the computing capacity of nodes. Therefore, when the number of remaining splits is small, it is possible that the task scheduling module erroneously sends splits to the node with extremely weak computing capacity. So, it is necessary to design a perception mechanism for the processing capacity of the nodes. Under the perception mechanism, computing nodes will actively provide the computing server with information such as the processing time of splits, which will be continuously updated and used as the basis for task scheduling. Figure 6 shows the structure of the CLTS scheduling algorithm.

The CLTS scheduling algorithm has made certain modifications to the original functions of the computing server and computing node of the TOGP. It allows the task scheduling module to make decisions based on the progress of split processing.

In the CLTS scheduling algorithm, the computing server sends the first batch of split information based on the connection status of each computing node. The computing server can accurately perceive all computing nodes with idle business threads simply by continuously monitoring the changes in the node.

The pseudo-code for the processing flow of the computing server after the user uploads calculation tasks in the CLTS scheduling algorithm is shown in Algorithm 1.

When the CLTS scheduling algorithm is first started, there is no historical data. The computing server has to use the round robin as the scheduling algorithm to send the first batch of splits. Once a computing node has processed a split, it will register the computing node's address in the Zookeeper cluster. The scheduler of the computing server monitors the changes in the specified directory in the Zookeeper cluster and can get the node's address with idle computing resources. When the computing node finishes processing a split, it will automatically record the processing time of the split and update the processing time to the Zookeeper cluster. The relevant information uploaded to the Zookeeper cluster will be obtained by the server and used as an essential basis for judging the computing capacity of nodes. After getting the node address and the split processing time, the scheduler

Algorithm 1 The Pseudo-Code For the Processing Flow of the Computing Server After the User Uploads Calculation Tasks in the CLTS Scheduling Algorithm

```
1.Receive the data, code and task type uploaded by the user
2.Divide the data into splits according to the task type and
  data type
3.Upload splits and code to HDFS
4.Check online computing nodes and send split
  information to these computing nodes
5.loop:
6.   Detect a change in the data of the node in the
     Zookeeper cluster
7.   if The computing server has received all split
     processing results of one calculation task then
8.      Call the corresponding result summary method
        according to the task type
9.   end if
10. if System queue is empty then
11.    end loop
12. else
13.    if The proportion of unprocessed splits is less than
       10% then
14.       Search the table for T and Tmin. (T is the time
          spent processing other splits of this calculation
          task by the computing node with idle threads. Tmin
          is the shortest time to process this calculation
          task's splits)
15.       if $T > N \times Tmin$ (N is the parameter set by the
          user, with a default value of 4) then
16.          Update the data of the node in the Zookeeper
             cluster to the address of the computing node
             corresponding to Tmin
17.       end if
18.    end if
19.    Get split information from the system queue
20.    Send split information according to the data of the
       node in the Zookeeper cluster
21.    goto loop
22. end if
```

determines whether to refer to the node address with idle computing resources and the split processing progress of the current calculation task and obtains the final scheduling result. Then the scheduler executes the final scheduling result and sends the split information to the specified computing node.

During the operation of the CLTS scheduling algorithm, if there are newly added computing nodes, they will register their addresses in the Zookeeper cluster like other computing nodes that have processed splits.

In the practical application of the CLTS scheduling algorithm, the computing server will maintain a table for each calculation task, as shown in Table 3.

During the execution of the calculation task, the computing node needs to collect split processing information. This part of the business is handled by an independent thread pool, which will not affect the processing of calculation tasks. The computing server will obtain split processing information through the Zookeeper watch and dynamically update the table. The system takes the time spent by computing nodes to process a split of the calculation task as the basis for judging the computing capacity of nodes so that the computing capacity of each node can be compared. Through this comparison, the server can have a relative understanding of the computing capacity of each node. When the TOGP detects that the number of remaining splits is small, it combines the table data with the nodes with an idle business thread as the basis for task scheduling.

When the number of unprocessed splits is large, the TOGP will fully use the computing resources of all nodes in the cluster and send split information to nodes with idle computing resources. When the number of unprocessed splits is small, the task scheduling module will actively retrieve the information in the corresponding table when it perceives idle computing resources. Finally, the task scheduling module makes a choice by comparing the computing capacity of different nodes. While using computers with weak computing capacity in the cluster, this mechanism can avoid unnecessarily slowing down the processing of calculation tasks due to sending split information to nodes with inadequate computing capacity at the later stage of task processing.

TABLE 3. Table maintained by the computing server.

Node name	Split name	Start time	Processing time
192.168.0.13: 44658	WordCount233_part3	2023-03-11 21:11:17	2096ms
192.168.0.14: 49154	WordCount233_part5	2023-03-11 21:11:16	3354ms
192.168.0.16: 58078	WordCount233_part6	2023-03-11 21:11:15	2347ms

Algorithm 2 The Pseudo-Code For the Processing Flow of the Computing Node After the Computing Server Sends Split Information in the CLTS Scheduling Algorithm

```
1.Receive split information from the computing server
2.Find the corresponding splits and code from HDFS based on the
split information
3.Download corresponding splits and code from HDFS
4.Call the code to process splits
5.Collect split processing information and send the split processing
results to the computing server
6.Register a Zookeeper ephemeral sequential node under the
node in the Zookeeper cluster
7. if The serial number of this Zookeeper ephemeral sequential
node is not the smallest then
8.   Register a Zookeeper watch for watching the previous
     Zookeeper ephemeral sequential node
9.   Receive notifications from Zookeeper watch
10. end if
11.Update the data of the node in the Zookeeper cluster.
   The updated data is the address of this computing
   node and split processing information. (The format of
   updated data is: node address; node name; split name;
   start time; processing time)
```

TABLE 4. Cluster configuration.

Computer name	Internal storage	Disk	IP address	Function
Server	8GB	150GB	192.168.0.15	Computing server
Worker1	4GB	100GB	192.168.0.13	Computing node
Worker2	4GB	100GB	192.168.0.14	Computing node
Worker3	4GB	100GB	192.168.0.16	Computing node
Worker4	4GB	100GB	192.168.0.17	Computing node
Worker5	4GB	100GB	192.168.0.18	Computing node
Worker6	4GB	100GB	192.168.0.19	Computing node

Algorithm 2 shows the pseudo-code for the processing flow of the computing node after the computing server sends split information in the CLTS scheduling algorithm.

After receiving the split information sent by the computing server, the computing node will download the corresponding splits and code from HDFS and call the code to process the splits. When the node's business thread finishes processing the current split, it will try to update the node's data in the Zookeeper cluster. The updated data is the address of this computing node. Under the influence of distributed lock, only one computing node can change the data of the node at the same time.

The computing server can accurately perceive all computing nodes with idle business threads simply by continuously monitoring the changes in the node in the Zookeeper cluster. Therefore, the computing server can only send split information to computing nodes with idle threads. The distributed service registration and discovery mechanism is implemented.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

A. EXPERIMENTAL ENVIRONMENT

Figure 7 shows the network environment for TOGP deployment.

The TOGP deploys in a cluster of multiple computers. The operating system for these computers is Ubuntu 20.04 LTS. For efficiency reasons, computers in the cluster have deployed the Zookeeper and HDFS. Table 4 shows the configuration of these computers.

Due to the many responsibilities of the primary computer, such as being responsible for data segmentation and uploading splits to HDFS and interacting with other computers, it is necessary to ensure the stable operation of the entire system. The primary computer needs to have a higher configuration. According to the TOGP's design, the computing server and computing node can be deployed on different computers.

Because the TOGP needs to use Zookeeper for distributed coordination services, computers in the cluster should deploy Zookeeper. Table 5 shows the planning of the Zookeeper cluster.

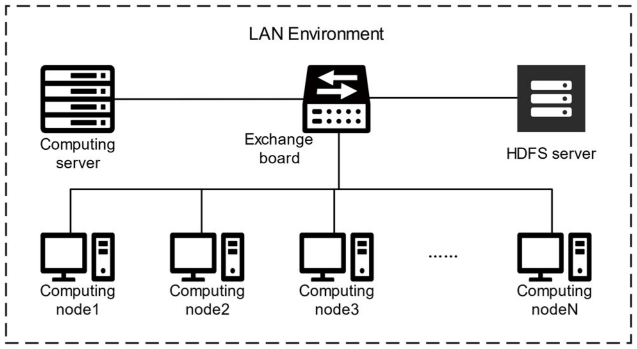


FIGURE 7. The network environment for TOGP deployment.

TABLE 5. Planning for zookeeper cluster.

IP address	Computer name	Myid	Role
192.168.0.13	Worker1	1	Follower
192.168.0.14	Worker2	2	Follower
192.168.0.15	Server	3	Leader
192.168.0.16	Worker3	4	Follower
192.168.0.17	Worker4	5	Follower
192.168.0.18	Worker5	6	Follower
192.168.0.19	Worker6	7	Follower

B. SYSTEM SCALABILITY TEST

The core idea of distributed systes is to allow multiple computers to work together to complete some tasks that are difficult to be handled by one single computer, such as tasks with high concurrency or large amounts of data. Therefore, distributed systes have to use a network to connect multiple computers. This structure means that distributed systems must have scalability. As the most significant feature of distributed systems, scalability allows distributed systems to adapt to changes in requirements.

Scalability is divided into horizontal scalability and vertical scalability. Horizontal scalability means that the cluster's overall performance can be improved by increasing the number of computers in the cluster. In contrast, vertical scalability means that the overall performance of the cluster can be improved by improving the performance of each computer in the cluster.

The performance of one single computer cannot be improved without limitation, but it is easy to add a large number of computers to the cluster. Therefore, many distributed computing frameworks tend to emphasize horizontal scalability, that is, to improve the efficiency of distributed computing by increasing the number of computers in the cluster. Therefore, it is necessary to test the horizontal scalability of the TOGP.

The horizontal scalability test of the system involves deploying the TOGP in the cluster and running a K-means algorithm to process the same iris dataset with different numbers of computing nodes enabled. This dataset is generated by Python following the Iris dataset, and its size is 2.1GB. Figure 8 shows the results of the horizontal scalability test.

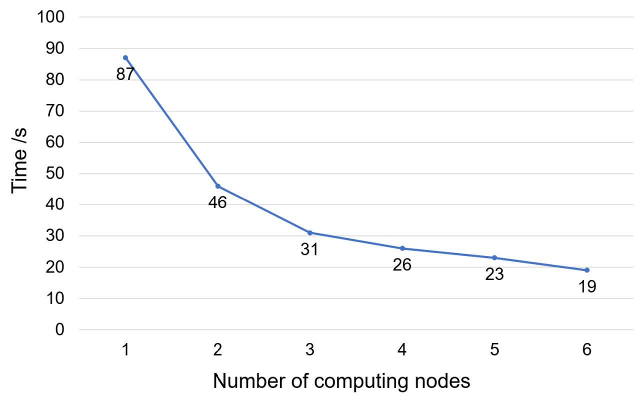


FIGURE 8. Horizontal scalability test.

From the test results of horizontal scalability, deploying more nodes can effectively speed up the processing efficiency of calculation tasks. The TOGP can fully use the newly added computing resources in the cluster to complete calculation tasks and has horizontal scalability.

C. SYSTEM PERFORMANCE COMPARATIVE EXPERIMENT

For distributed computing frameworks, performance is a significant factor. The performance of distributed computing frameworks determines whether frameworks can efficiently complete the calculation task. In the system performance comparative experiment, this paper selects Hadoop to compare with the TOGP using the round robin. The calculation example used in the experiment is word frequency statistic, and the version of Hadoop used is 2.10.1.

This paper chooses Hadoop as a comparison object for the following reasons:

1) Many distributed computing frameworks mentioned above are not open source, and some are designed to handle data in special formats, which need to be more versatile.

2) The TOGP is similar to Hadoop in structure. Both use HDFS as distributed storage and are implemented by Java.

3) Hadoop is a classic distributed computing framework widely used and is still being updated. Therefore, it is more valuable to compare with Hadoop.

The example used in the experiment is word frequency statistic. Its content is to count the number of occurrences of each word in single or multiple text files and output each word and its number of occurrences in the form of key-value pairs. The dataset of word frequency statistic used in this comparative experiment is generated by Python. The words are separated by spaces. The size of the dataset is 3.15GB.

The comparative experiment is conducted in a cluster environment built by four computers, server, worker 1, worker 2 and worker 3. In order to reduce the impact of network speed fluctuations, it is necessary to use the same dataset to conduct multiple experiments. In the comparative experiment, Hadoop uses its default word frequency statistic program. Because Hadoop's word frequency statistic program requires the dataset to be uploaded to HDFS ahead of time, the TOGP reads the data directly from the local computer and uploads splits to HDFS after data segmentation. For fairness, the time Hadoop performs word frequency statistics is the time the dataset is uploaded and the time Hadoop's word frequency statistic program performs data processing. The time for the TOGP to perform word frequency statistics includes the time for data segmentation, the time for uploading splits to HDFS, and the time for the TOGP to process the splits. Figure 9 shows the experimental results.

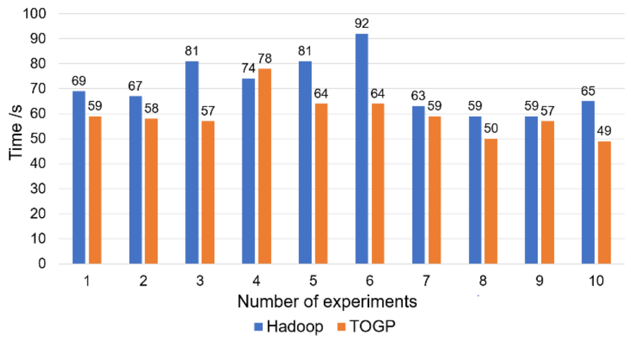


FIGURE 9. System performance comparative experiment.

As seen from Figure 9, in most cases, the computing efficiency of the TOGP is higher than that of Hadoop, and only in a few instances the computing efficiency of the two is about the same. According to the average of ten experiments, the performance of TOGP using the round robin is about 16% higher than that of Hadoop. The performance of TOGP exceeds that of Hadoop for the following reasons:

1) The TOGP can fully use computing resources to process the calculation tasks efficiently.

2) Hadoop is designed to handle very large-scale data, but due to computer hardware limitations, the size of the dataset is only at the GB level.

3) Compared with Hadoop's computing process, the computing process of TOGP reduces the number of times to write data to disk.

4) The configuration of each worker computer is the same, and there is no significant difference in processing capacity. In this case, the round robin is more efficient than Hadoop's task scheduling algorithm.

Therefore, it can be proved that the performance of TOGP is higher than that of Hadoop. By combining the performance comparative experiment with the horizontal scalability test, different types of calculation tasks have been processed on the TOGP. It can be considered that the TOGP has availability.

D. EVALUATION OF THE CLTS SCHEDULING ALGORITHM

The comparative experiment part of the CLTS scheduling algorithm needs to evaluate the computing capacity of computers, and different types of calculation tasks have different requirements on the computer configuration.

Therefore, the whole comparative experiment part needs to use the same type of calculation tasks. The calculation task used in the experiment is word frequency statistic, which best reflects the idea of distributed computing. The datasets used in the word frequency statistic experiment are generated by Python, with spaces interval between words, and the sizes of the datasets are at GB level. The formats are identical between different datasets, differing only in content and size.

In the experiment to verify the algorithm's effectiveness, the experimental evaluation index is the running time of the calculation task. Because the CLTS scheduling algorithm is designed to adapt better to the heterogeneous cluster, it is necessary to simulate the heterogeneous cluster environment before conducting comparative experiments. The heterogeneous cluster considered in this paper refers to a cluster with different computing capacities due to significant differences in disk capacity, CPU cores, speed and memory capacity among multiple nodes of the same cluster.

Four computers were used in the scheduling algorithm comparative experiment: server, worker 1, worker 2 and worker 3. According to the cluster configuration, the hardware configurations of each computing node are identical. Therefore, the computing capacity of each computing node is not significantly different. So, we need to use other methods to simulate the heterogeneous cluster.

Before verifying whether the CLTS scheduling algorithm can adapt to the heterogeneous cluster environment, the effectiveness of the CLTS scheduling algorithm should be evaluated. Three datasets are used for word frequency statistics to evaluate the efficiency of the CLTS scheduling algorithm fully. The hardware configuration of each computing node is identical, so there is little difference in its computing capacity. In this case, the simple-structure round robin has become the optimal scheduling algorithm under this condition because the scheduling results fully match the scheduling based on the computing capacity of the nodes. Therefore, we only need to compare the CLTS scheduling algorithm with a theoretical optimal algorithm, the round robin.

All three datasets are generated by Python, and the sizes of dataset 1, dataset 2, and dataset 3 are 4.09GB, 6.06GB, and 11.12GB, respectively. The TOGP uses round robin and the CLTS scheduling algorithm as scheduling algorithms to run word frequency statistic tasks. Figure 10 shows the experimental results.

According to the experimental results, it can be found that the effect of the CLTS scheduling algorithm is not different from that of the round robin, which is caused by the following reasons:

1) The configuration of each worker machine is identical, so the computing capacity gap of each node in the cluster is tiny. The scheduling results given according to the nodes' computing capacity are consistent with the round robin's scheduling results, which does not reflect the advantages of the CLTS scheduling algorithm.

2) CLTS scheduling algorithm has distributed service registration and discovery mechanism and dynamic feedback mechanism of node processing capacity implemented by Zookeeper. These mechanisms will also occupy a certain amount of resources and slightly affect the efficiency of the CLTS scheduling algorithm. In contrast, the structure of the round robin is relatively simple, so that the round robin will save some time.

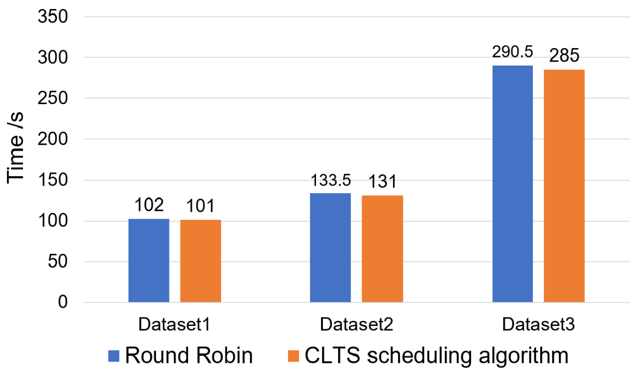


FIGURE 10. The comparative experiment of scheduling algorithms.

Because the computing capacity of each node in the cluster is not significantly different, the above experiment is not enough to prove that the CLTS scheduling algorithm can better adapt to an environment of the heterogeneous cluster. Therefore, further experimentation is needed to simulate an environment of the heterogeneous cluster.

This paper initially adopted the approach of running other programs to occupy the computing resources, thus expanding the computing capacity gap of nodes in the cluster. However, the effect of this approach is not apparent, and the impact on computing capacity cannot be quantified. Therefore, this paper can only simulate the decline of one node's computing capacity by directly adding a time delay to the calculation module in worker 1. Under the time delay mechanism, worker 1 will delay when it receives the splits, so the node must spend more time processing the same number of splits. The specific delay time is set manually, accurate at the millisecond level, which is easy to quantify and control. By adding a delay to the node's calculation module, the computing capacity of the node can be significantly reduced, and the situation of the heterogeneous cluster can be simulated.

Therefore, the content of the comparative experiment is to add a delay to the computing node deployed on worker 1 and use the round robin, the smooth weighted round robin based on the node processing capacity, and the CLTS scheduling algorithm to process the same dataset under different lengths of time delay. The smooth weighted round robin based on node processing capacity is a smooth weighted round robin implemented in the system based on the dynamic feedback mechanism of node processing capacity in the CLTS scheduling algorithm. Figure 11 shows the experimental results.

As shown in Figure 11, as the delay of the computing node deployed on worker 1 computer increases, the computing capacity of the node decreases, and the advantages of the CLTS scheduling algorithm become more and more evident. To summarize the advantages of the CLTS scheduling algorithm, the experimental results in Figure 11 can be further analyzed.

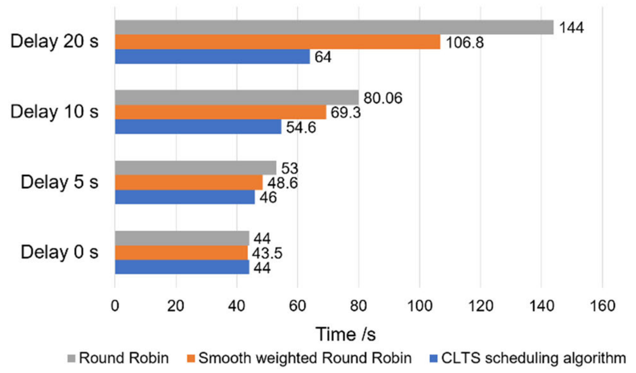


FIGURE 11. The comparative experiment of scheduling algorithms with time delay.

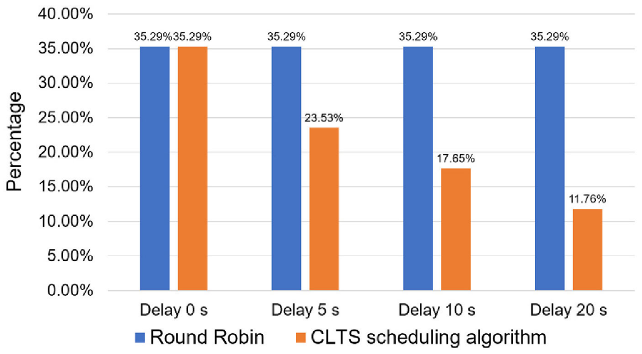


FIGURE 12. Statistics on the proportion of splits.

Figure 12 shows the percentage of node processing splits under round robin and CLTS scheduling algorithm with a time delay effect in comparative experiments.

Under the mechanism of the CLTS scheduling algorithm, the worse the computing capacity of the computing node, the more inclined the server is to send split information to other nodes. Finally, the node with the worse computing capacity will be assigned less work.

With the increase in time delay, the computing node's efficiency of processing splits affected by latency will worsen. Although the first round of three scheduling algorithms is round robin, over time, the CLTS scheduling algorithm can enable the computing server to promptly monitor the operation of business threads in the cluster and send split information to nodes with idle business threads. The mechanism of the CLTS scheduling algorithm enables it to make full use of computing resources in the cluster. At the same time, it can also adapt to the situation where there are significant differences between multiple nodes in the same cluster regarding disk capacity, CPU cores and speed, and memory capacity, which leads to significant differences in the computing capacity of nodes.

According to the experimental results shown in Figure 11 and Figure 12, the CLTS scheduling algorithm can better adapt to the heterogeneous cluster environment. Compared with the smooth weighted round robin based on node processing capacity, the CLTS scheduling algorithm is more effective.

V. CONCLUSION

In order to solve the problem that the current commonly used distributed computing frameworks are challenging to balance between versatility and learning difficulty, this paper proposes a distributed computing system, TOGP. The TOGP adopts the task-oriented design concept, dividing the calculation tasks into multiple tasks and implementing the corresponding processing methods according to different task types. Based on the TOGP, this paper makes more profound research on the task scheduling problem of the system. By introducing distributed service registration and discovery mechanism and mechanism of node computing capacity detection into the task scheduling problem of the system, the paper proposed the CLTS scheduling algorithm.

The system scalability test proves that the TOGP has good horizontal scalability. In the comparative experiment, the computing performance of the TOGP using the round robin is about 16% higher than that of Hadoop, which proves that the TOGP has a high processing capacity. A series of experiments prove that the CLTS scheduling algorithm can effectively adapt to the heterogeneous cluster environment and efficiently schedule splits.

In future work, on the one hand, we will continue to expand the task types supported by the TOGP, further optimize the data segmentation method, and further study the computational model of stream processing to achieve stream processing in the system; On the other hand, container technology and support for resource management platforms such as Yarn will be introduced to achieve accurate control of computing resources and maximize the utilization efficiency of resources.

REFERENCES

[1] V. A. Shchapov, G. F. Masich, and A. G. Masich, "The technology of processing intensive structured dataflow on a supercomputer," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, vol. 2, Helsinki, Finland, Aug. 2015, pp. 235–240, doi: 10.1109/Trustcom.2015.589.

[2] H. Tian, A. W. Liew, and H. Shen, "Advances in parallel and distributed computing and its applications," *Concurrency Comput., Pract. Exper.*, vol. 34, no. 2, p. e6667, Oct. 2021, doi: 10.1002/cpe.6667.

[3] S. Tang, B. He, C. Yu, Y. Li, and K. Li, "A survey on spark ecosystem: Big data processing infrastructure, machine learning, and applications," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 1, pp. 71–91, Jan. 2022, doi: 10.1109/TKDE.2020.2975652.

[4] S. Wilson and R. Sivakumar, "Twitter data analysis using Hadoop ecosystems and Apache Zeppelin," *Indonesian J. Electr. Eng. Comput. Sci.*, vol. 16, pp. 1490–1498, Dec. 2019, doi: 10.11591/ijeecs.v16.i3.pp1490-1498.

[5] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008, doi: 10.1145/1327452.1327492.

[6] Y. Wang and Y. Wu, "Universal coded distributed computing for MapReduce frameworks," 2022, *arXiv:2201.06300*.

[7] J. Li, Y. Liu, J. Pan, P. Zhang, W. Chen, and L. Wang, "Map-Balance-Reduce: An improved parallel programming model for load balancing of MapReduce," *Future Gener. Comput. Syst.*, vol. 105, pp. 993–1001, Apr. 2020, doi: 10.1016/j.future.2017.03.013.

[8] E. Gavagsaz, A. Rezaee, and H. H. S. Javadi, "Load balancing in join algorithms for skewed data in MapReduce systems," *J. Supercomput.*, vol. 75, no. 1, pp. 228–254, Jan. 2019, doi: 10.1007/s11227-018-2578-0.

[9] W. Shi, D. B. Tang, and P. Zou, "Multi-objective automated guided vehicle scheduling based on MapReduce framework," *Adv. Prod. Eng. Manag.*, vol. 16, pp. 37–46, Mar. 2021, doi: 10.14743/apem2021.1.383.

[10] T. Gao, Y. Guo, B. Zhang, P. Cicotti, Y. Lu, P. Balaji, and M. Taufer, ''Memory-efficient and skew-tolerant MapReduce over MPI for supercomputing systems,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 12, pp. 2734–2748, Dec. 2020, doi: 10.1109/TPDS.2019.2932066.

[11] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, ''Spark: Cluster computing with working sets,'' in *Proc. 2nd USENIX Conf. Hot Topics Cloud Comput.*, Boston, MA, USA, Jun. 2010, pp. 1–7, doi: 10.5555/1863103.1863113.

[12] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, ''Discretized streams: Fault-tolerant streaming computation at scale,'' in *Proc. 24th ACM Symp. Operating Syst. Princ.*, New York, NY, USA, Nov. 2013, pp. 423–438, doi: 10.1145/2517349.2522737.

[13] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, ''Apache Flink: Stream and batch processing in a single engine,'' *IEEE Data Eng. Bull.*, vol. 38, no. 4, pp. 28–38, Dec. 2015.

[14] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja, ''Twitter Heron: Stream processing at scale,'' in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, New York, NY, USA, May 2015, pp. 239–250, doi: 10.1145/2723372.2742788.

[15] S. Wei, F. Wang, H. Deng, C. Liu, W. Dai, B. Liang, Y. Mei, C. Shi, Y. Liu, and J. Wu, ''OpenCluster: A flexible distributed computing framework for astronomical data processing,'' *Publications Astronomical Soc. Pacific*, vol. 129, no. 972, Feb. 2017, Art. no. 024001, doi: 10.1088/1538-3873/129/972/024001.

[16] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Pauk, M. I. Jordan, and I. Stoica, ''Ray: A distributed framework for emerging AI applications,'' in *Proc. 13th USENIX Conf. Operating Syst. Design Implement.*, Carlsbad, CA, USA, Oct. 2018, pp. 561–577, doi: 10.5555/3291168.3291210.

[17] S. Venkataraman, A. Panda, K. Ousterhout, M. Armbrust, A. Ghodsi, M. J. Franklin, B. Recht, and I. Stoica, ''Drizzle: Fast and adaptable stream processing at scale,'' in *Proc. 26th Symp. Operating Syst. Princ.*, New York, NY, USA, Oct. 2017, pp. 374–389, doi: 10.1145/3132747.3132750.

[18] R. Wu, L. Huang, P. Yu, and H. Zhou, ''SunwayMR: A distributed parallel computing framework with convenient data-intensive applications programming,'' *Future Gener. Comput. Syst.*, vol. 71, pp. 43–56, Jun. 2017, doi: 10.1016/j.future.2017.01.018.

[19] P. Saha, A. Beltre, and M. Govindaraju, ''Scylla: A Mesos framework for container based MPI jobs,'' 2019, *arXiv:1905.08386*.

[20] S. Prakash, A. Reisizadeh, R. Pedarsani, and A. S. Avestimehr, ''Coded computing for distributed graph analytics,'' *IEEE Trans. Inform. Theory*, vol. 66, no. 10, pp. 6534–6554, Oct. 2020, doi: 10.1109/TIT.2020.2999675.

[21] Y. Xu, H. Liu, and Z. Long, ''A distributed computing framework for wind speed big data forecasting on Apache Spark,'' *Sustain. Energy Techn.*, vol. 37, Feb. 2020, Art. no. 100582, doi: 10.1016/j.seta.2019.100582.

[22] X. Sun, Y. He, D. Wu, and J. Z. Huang, ''Survey of distributed computing frameworks for supporting big data analysis,'' *Big Data Mining Anal.*, vol. 6, no. 2, pp. 154–169, Jun. 2023, doi: 10.26599/BDMA.2022.9020014.

[23] G. Chen, M. Rui, and K. Lu, ''A parallel computing framework for big data,'' *Front. Comput. Sci.*, vol. 11, no. 4, pp. 608–621, Aug. 2017, doi: 10.1007/s11704-016-5003-y.

[24] Y. Zhang, Q. Gao, L. Gao, and C. Wang, ''iMapReduce: A distributed computing framework for iterative computation,'' *J. Grid Comput.*, vol. 10, no. 1, pp. 47–68, Mar. 2012, doi: 10.1007/s10723-012-9204-9.

[25] M. T. Islam, S. Karunasekera, and R. Buyya, ''Performance and cost-efficient spark job scheduling based on deep reinforcement learning in cloud computing environments,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 7, pp. 1695–1710, Jul. 2022, doi: 10.1109/TPDS.2021.3124670.

[26] F. Hu, C. Yang, J. L. Schnase, D. Q. Duffy, M. Xu, M. K. Bowen, T. Lee, and W. Song, ''ClimateSpark: An in-memory distributed computing framework for big climate data analytics,'' *Comput. Geosci.*, vol. 115, pp. 154–166, Jun. 2018, doi: 10.1016/j.cageo.2018.03.011.

[27] F. Li, J. Chen, and Z. Wang, ''Wireless MapReduce distributed computing,'' *IEEE Trans. Inf. Theory*, vol. 65, no. 10, pp. 6101–6114, Oct. 2019, doi: 10.1109/TIT.2019.2924621.

[28] Y.-C. Zhang, X.-Y. Wang, C. Wang, Y. Xu, J.-W. Zhang, X.-D. Lin, G.-Y. Sun, G.-L. Zheng, S.-H. Yin, X.-J. Ye, L. Li, Z. Song, and D.-D. Miao, ''Bigflow: A general optimization layer for distributed computing frameworks,'' *J. Comput. Sci. Technol.*, vol. 35, no. 2, pp. 453–467, Mar. 2020, doi: 10.1007/s11390-020-9702-3.

[29] N. Shakya, F. Li, and J. Chen, ''On distributed computing with heterogeneous communication constraints,'' *IEEE/ACM Trans. Netw.*, vol. 30, no. 6, pp. 2776–2787, Dec. 2022, doi: 10.1109/TNET.2022.3181234.

[30] B. L. Morris and A. Sjjellum, ''MPIgnite: An MPI-like language and prototype implementation for Apache Spark,'' 2017, *arXiv:1707.04788*.

[31] Z. Wang, S. Zhang, B. He, and W. Zhang, ''Melia: A MapReduce framework on OpenCL-based FPGAs,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 12, pp. 3547–3560, Dec. 2016, doi: 10.1109/TPDS.2016.2537805.

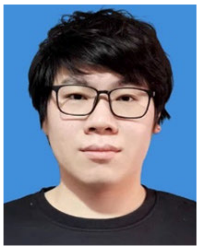
[32] R. Kurte, Z. Salcic, and K. I. Wang, ''A distributed service framework for the Internet of Things,'' *IEEE Trans. Ind. Informat.*, vol. 16, no. 6, pp. 4166–4176, Jun. 2020, doi: 10.1109/TII.2019.2948046.

[33] Z. Huang, F. Liu, M. Tang, J. Qiu, and Y. Peng, ''A distributed computing framework based on lightweight variance reduction method to accelerate machine learning training on blockchain,'' *China Commun.*, vol. 17, no. 9, pp. 77–89, Sep. 2020, doi: 10.23919/JCC.2020.09.007.

[34] R. Mahmud and A. N. Toosi, ''Con-Pi: A distributed container-based edge and fog computing framework,'' *IEEE Internet Things J.*, vol. 9, no. 6, pp. 4125–4138, Mar. 2022, doi: 10.1109/JIOT.2021.3103053.

[35] Z. He, G. Liu, X. Ma, and Q. Chen, ''GeoBeam: A distributed computing framework for spatial data,'' *Comput. Geosci.*, vol. 131, pp. 15–22, Oct. 2019, doi: 10.1016/j.cageo.2019.06.003.

MIN HUANG received the M.Eng. degree in software engineering from the Beijing Institute of Technology, Beijing, China, in 2004. He is currently pursuing the Ph.D. degree in electrical engineering with the National Key Laboratory on Electromagnetic Environment Effects, Shijiazhuang, Hebei, China. He is currently an Associate Professor with the Hebei University of Science and Technology, Shijiazhuang. His research interests include machine learning, natural language processing, and artificial intelligence.



YUNXIANG ZHAO is currently pursuing the master's degree with the Hebei University of Science and Technology. His research interests include distributed computing and big data processing

YAZHOU CHEN (Member, IEEE) received the B.Eng., M.Eng., and Ph.D. degrees from the Shijiazhuang Mechanical Engineering College, Shijiazhuang, China, in 1996, 1999, and 2002, respectively. He is currently a Professor and the Director of the Army Engineering University, Shijiazhuang Campus, China. His research areas of interest are EMC and lightning protection.