Cost-Efficient Scheduling of Streaming Applications in Apache Flink on Cloud

Hongjian Li, Jianglin Xia, Wei Luo, and Hai Fang

Abstract—Stream processing has been gaining extensive attention in the past few years. Apache Flink is a new generation of distributed stream processing engines that can process a great deal of data in real-time with low latency. But the default scheduler of Flink adopts a random task scheduling strategy, which does not consider the cost and load balancing in the cloud environment. In this article, a cost-efficient task scheduling algorithm (CETSA) and a cost-efficient load balancing algorithm (LBA-CE) for Flink are proposed to reduce the job execution cost while optimizing load balancing. First, a cost-efficient model and a load balancing model based on Flink are constructed. Then, the core mechanism of Flink task scheduling is improved based on the cost-efficient model and the improved task scheduler is implemented. In addition, the concept of node adaptation is introduced into cost-efficient scheduling according to the load balancing model, ensuring that the cluster load is balanced as much as possible while reducing the cost in a heterogeneous cluster. Extensive experiments have been performed with Hibench's Wordcount and Fixwindow workloads in the cloud environment. The experimental results indicate that compared to the baseline scheduling algorithm, the proposed algorithms reduce the cost by about 37.9% and 20.2% on average, and the load deviation of the cluster is reduced by about 23.1% and 24.6% on average, respectively. In summary, the proposed algorithms in this paper can significantly reduce the cost of executing jobs and optimize the load balancing of the cluster in Flink.

Index Terms—Cost-efficient, flink, stream processing, load balancing, task scheduling

1 Introduction

With the rapid development of information technology such as big data, artificial intelligence, and the Internet of Things globally, various data in many fields are constantly growing and accumulating down [1]. Therefore, the immediate processing of data is becoming a hot topic in the field of big data [2]. There are three main big data frameworks: Apache Spark [3], Apache Storm [4] and Apache Flink [5]. It is becoming popular to deploy big data applications on the cloud, because cloud platforms can provide a large number of scalable resources for big data processing as needed [6]. However, there exist some problems with deploying big data processing frameworks on the cloud, mainly related to the task scheduling policies of big data frameworks and the configuration of nodes in clusters [7]. For cloud platforms, the types of compute instances provided may be different. For example, Amazon Cloud provides several different types of virtual machine instances such as memory-optimized and storage-optimized. It is difficult to implement cost-efficient scheduling procedures on the cloud platform because of various types of VM instances. However, the type of virtual machine can significantly affect the performance of the job. Therefore, it is important to study how to reduce the cost of stream processing while ensuring the system performance on the cloud. When Flink clusters are deployed on the cloud, the resource requirements of different topological jobs (Flink Jobs) and subtasks divided by jobs may be different [8]. When a Flink job is submitted to the cluster, the default scheduler of Flink is sequential for task selection, i.e., First Come First Serve (FCFS). Because Flink selects nodes randomly, it does not consider the relationship between the resource demand of the task and the resources owned by the nodes. The former may bring some problems [9], [10], [11], such as too much dispersal of tasks at the node level, resource waste and load imbalance.

Recently, a lot of studies related to task-based scheduling have been widely studied [12], [13], [14], [15], [16], [17], [18]. In [12], by analyzing and studying the topology of jobs and communication traffic between nodes, the authors proposed a task scheduling framework based on job topology and traffic which significantly improved the throughput and saved resources. To save the problem of Flink performance degradation and resource overuse under the fluctuating load of stream computing, a Flink-based elastic scheduling method was proposed in [13] to improve the cluster throughput. Focusing on the problem of implementing runtime task-aware scheduling in stream computing, a lightweight elastic stream computing scheduler was designed in [14], which showed more significant results in reducing response time and improving throughput. To address common challenges in the task scheduling process and to improve the overall performance of the cluster, a Storm-based online task scheduling method was proposed in [15] to reduce the completion time by 20% and increase the throughput by 0.5 times. The delay-aware and resource-aware scheduling of big data stream computing system was studied in [16], and a resource-aware scheduling framework (LR-Stream) was proposed in Storm

• *The authors are with the Department of Computer Science and Technology, Chongqing University of Posts and Telecommunications, Chongqing 400065, China. E-mail: lihj@cqupt.edu.cn, s210201111@stu.cqupt.edu.cn, {1216770825, 1214316932}@qq.com.*

Manuscript received 17 July 2022; revised 18 November 2022; accepted 26 December 2022. Date of publication 30 December 2022; date of current version 11 July 2023.
(Corresponding author: Hongjian Li.)
Recommended for acceptance by X. Li.
Digital Object Identifier no. 10.1109/TBDATA.2022.3233031

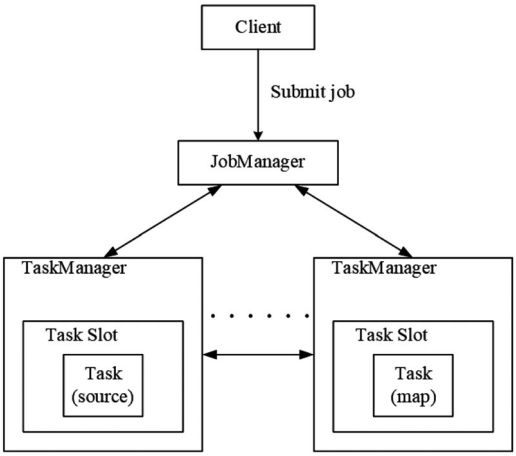


Fig. 1. Flink architecture diagram.

to reduce the delay of stream computing systems and improve throughput. An efficient job scheduler for big data processing systems was proposed in [17], which can effectively simulate the shortest job-first scheduling strategy without knowing the job size in advance and reduce the job response time while achieving better fairness. A heuristic scheduling algorithm was proposed in [18] to improve Storm's throughput by reliably and efficiently discovering highly communicated tasks by utilizing graph partitioning algorithms and mathematical optimization software packages. However, these studies mainly focused on performance-oriented scheduling algorithms for streaming applications, and cost-efficient scheduling of streaming applications from a cloud perspective is not considered.

Based on the scenario of deploying big data applications on the cloud, we studied the task scheduling strategy on Flink and proposed an improved cost-efficient task scheduler. Moreover, the Flink-based load balancing algorithm was proposed to reduce the cost while optimizing the load balancing of the cluster on the cloud. The main contributions of this paper are as follows:

- An integrated cost-efficient model and a cost-efficient load balancing model are constructed. The composition of the cost-efficient model mainly includes resource cost, communication cost, and scheduling cost.
- Based on the cost-efficient model, a cost-efficient task scheduling algorithm (CETSA) is proposed for Flink to reduce the cost of heterogeneous clusters in cloud environments.
- A cost-efficient load balancing algorithm (LBA-CE) is proposed to optimize the load balancing while reducing the cost of the cluster.

The rest of the paper is organized as follows: In Section 2, we introduce the background and related work. Sections 3 and 4 present the proposed model and algorithm. Section 5 evaluates and analyzes the experiments. Finally, conclusions and future work are presented in Section 6.

2 Background and Related Works

2.1 Background

Generally, the architecture of a Flink cluster based on Standalone is shown in Fig. 1. The Flink infrastructure still adopts the design pattern of the master-slave which consists of two important roles, namely the JobManager (JM) serving as the master node and the TaskManager (TM) serving as the slave node [19]. Among them, the JobManager is mainly used in the Flink cluster as the master node to process the submitted applications, which are also called topology jobs. The TaskManager is the slave node in the Flink cluster. A Flink cluster usually consists of a JM and many TMs, which communicate through the Actor System. It should be noted that there can only be one JM. The number of tasks that can be executed is determined by the number of task slots of the TM. There is at least one task slot in each TM.

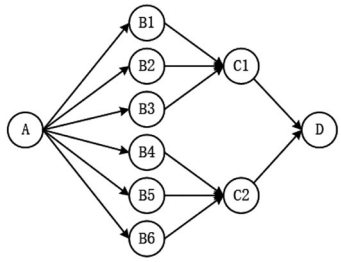


Fig. 2. Topology of the operation.

Flink provides a default task scheduling strategy, and the specific process of the default scheduling strategy is described below. When the client receives a job submitted by Flink UI, it will parse the corresponding program and finally generate JobGraph. The Client will submit the JobGraph to JobManager to generate the execution graph. Then, the master node will start the task scheduling process, which usually includes two phases: slot selection and task deployment.

The slot selection is the so-called task assignment. Before the assignment starts, the execution graph in the form of topology is parsed and each vertex in the topology, i.e., subtask, is obtained and put into the task list. In the assignment, each subtask is assigned a slot by iterating through the task list. In the selection of slots, the list of free slots is obtained first, then the list of slots is obtained according to the resource requirement parameters. And then the slot of a node is randomly selected from the slot list. Finally, after determining the mapping between all the tasks and the corresponding slots, tasks are deployed to the corresponding nodes. When scheduling tasks on the cloud, it causes higher execution cost and does not consider the load balancing problem because of the default scheduler of Flink selecting nodes randomly. Suppose there are 4 task managers $TMs = \{TM_1, TM_2, TM_3, TM_4\}$ in the cluster and each TM has 4 task slots. A topological job is submitted to the Flink cluster, assuming the dependencies among the operators in the job as shown in Fig. 2. There are four operators A, B, C and D with parallelism degrees of 1, 6, 2 and 1. After the default task scheduling, the final matching relationship between the task and the slots in the node may be as shown in Fig. 3.

2.2 Related Works

Currently, deploying big data clusters on the cloud is becoming more and more popular. Many scholars have conducted in-depth discussions from the perspectives of cost-efficient of task scheduling and load balancing of clusters [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], respectively.

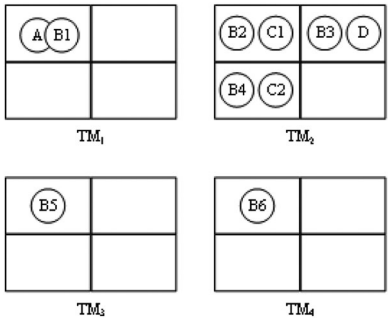


Fig. 3. Mapping of tasks to slots in nodes.

Two job scheduling algorithms based on cost-efficient for Spark clusters were proposed in [20], which allocated tasks according to node resource availability and reduced at least 34% in terms of resource cost. A real-time workflow scheduling method was presented in [21], which is expected to provide users with satisfactory service quality while processing jobs in an energy-efficient and economical manner. Focusing on the problem that SLA (Service Level Agreement) based scheduling strategy can adversely affect the Qos and resource utilization, the study [22] presented a load balancing scheduler in heterogeneous cloud environments. It effectively optimized the resource utilization, service completion time and execution cost. To reduce the total cost of a Storm cluster in the cloud environment, a cost-efficient model and a cost-efficient scheduling algorithm for Storm were proposed in [23]. To optimize the virtual machine usage cost for both local and cloud resources and maximize the job deadline met percentage, an efficient scheduling algorithms is proposed in [24] that adopts different virtual machine instance pricing in a hybrid cloud deployed cluster.

Focusing on the network structure model, load balancing algorithms and heterogeneous environments, the study [25] combined distributed scheduling to design a series of algorithms to find the best balance between resource occupancy, deadline constraints and load balancing. It reduced the load deviation by an average of 2%. An online scheduler based on load-awareness was presented in [26], providing a user-friendly interface to effectively solve the problem of cluster load balancing and latency. The communication between virtual machines is also one of the main problems causing load imbalance in the cluster. Therefore, a dynamic load balancing task scheduling strategy was proposed in [27] to make the cluster load balancing in a heterogeneous scenario while reducing the communication overhead between VMs. The study [28] proposed a scheduling strategy that can dynamically map tasks to nodes by predicting the load state of nodes after they were assigned to a task. It provided significant improvements in optimizing the reliability and load balancing level of the system. To balance the load of the nodes, the study [29] presented a task scheduler that considered both the current load of the nodes and the task deadlines to allocate tasks. The summary of the above studies is shown in Table 1.

Most of the current studies are focused on performance optimization of other big data frameworks such as Storm and spark. However, there is no study that considers the cost of Flink running streaming applications on the cloud which is considered in our work. In this paper, a cost-efficient scheduling algorithm for Flink is proposed to minimize the execution cost and optimize the cluster load when the capacity of cluster resources meets the job resource requirements.

3 System Model

In this section, we present a cost-efficient (including resource cost, communication cost, scheduling cost) model based on Flink and a cost-efficient load balancing optimization model to save the problem of high job execution cost and load imbalance caused by the default of Flink scheduling strategy. Furthermore, the abbreviations involved in this section is shown in Table 2.

3.1 Cost-Efficient Model

A cost-efficient model based on streaming data processing framework is constructed. It mainly considers the resource rental cost caused by running topology jobs, the communication cost caused by data exchange between nodes and the scheduling cost caused by different task scheduling strategies.

The execution graph generated by the JobManager in Flink describes the dependencies between subtasks and is usually represented as a DAG (Directed Acyclic Graph) in the form of topology. For each scheduling of a task, it is actually finding a suitable node to place the subtask. The subtasks in the topology job are defined as a collection $T = \{T_1, T_2, \ldots, T_n\}$, and for each machine node that constructs a cluster defined as a collection $N = \{N_1, N_2, \ldots, N_n\}$.

TABLE 1
Summary of the Studies

Scheduling Strategy	Platform	Optimization goal	Cloud environment	Dynamic
[20] 2020	Spark	Resource usage cost	✓	✓
[21] 2019	Simulation	QoS Metrics, Energy efficiency	✓	✓
[22] 2019	Simulation	Resource utilization, Cost-efficient	✓	✓
[23] 2021	Storm	Cost-efficient	✓	✓
[24] 2021	Spark	Cost, Sla	✓	✓
[25] 2019	Simulation	Load balancing	✓	✓
[26] 2020	Heron	Inter-node traffic, CPU utilization	×	✓
[27] 2021	CloudSim	Load balancing	✓	✓
[28] 2020	Simulation	Load balancing	✓	✓
[29] 2021	Storm	Load balancing	×	✓

TABLE 2
Abbreviations Used in the Proposed Models

Notation	Definition
T	Set of tasks in the topology job
T_j	The jth task of the set T
N	Set of nodes in the cluster
N_i	The ith node of the set N
$RD^c_{T_j}$	The resource requirement of the task T_j for CPU
$RD^m_{T_j}$	The resource requirement of the task T_j for the memory
$RA^c_{Slot_i}$	The current CPU resource availability of the node slot
$RA^m_{Slot_i}$	The current memory resource availability of the node slot
$RV^c_{N_i}$	The CPU resource capacity of the node N_i
$RV^m_{N_i}$	The memory resource capacity of the node N_i
RC	The resource rental cost of completing the job
$RC(k)$	The resource rental cost for a k-type cloud host to perform a certain topology job
P_k	The charging standard of the k-type cloud host
ET	The completion time of executing the topology job submitted by the user
CC	The sum of communication costs between nodes
P_f	The price of communication between nodes
$D(T_i,\ T_j)$	The network distance between the task T_i and T_j
$f(T_i,\ T_j)$	The task T_i and T_j resulting inter-node traffic
$flow(T_i,\ T_j)$	The communication traffic between the task T_i and T_j
$SN(T_i,\ T_j)$	Judging whether the task T_i and T_j are assigned to the same cloud host node
XOR	Exclusive OR
SC	The scheduling cost generated when Flink adopts different scheduling strategies
ST	The time spent by Flink to allocate tasks using different task scheduling algorithms
$Time_{\ Dep}$	The time when the states of all subtasks in the topology job are transferred to the deploying state
$Time_{\ Cre}$	The time when the JobManager completes the creation of all subtask instances according to the execution graph of the job
WEC	The execution cost of the job
RTH	The threshold of a certain resource in the node
RV	The capacity of a certain resource in the node
$RTH^c_{N_i}$	The CPU resource threshold of the node N_i
$RTH^m_{N_i}$	The memory resource threshold of the node N_i
$Data_{in}$	The size of the total data stream transmitted to T_j by the upstream task
$\rho_c^{T_j}$	The CPU resources consumed when a record in the data stream is processed
$\rho_m^{T_j}$	The memory resources consumed when a record in the data stream is processed
CE_{N_i}	The cost evaluation value when the node N_i processes a certain amount of data
$Cost_{N_i}$	The cost overhead when the node N_i processes a certain amount of data
$Tuple_{N_i}$	The amount of data processed by the node N_i in a period of time
Avg^{DR}_{cpu}	The average CPU utilization in a period of time (DR)
B	The predicted value of the busyness
X	The number of time periods
EP_x	The predicted value of the busyness in the xth period
F_{N_i}	The fitness value of the node N_i
B_{N_i}	The predicted value of the busyness of the node N_i
AF	The average fitness of all nodes in the cluster
U_{cpu}	The CPU utilization of the node
U_{mem}	The memory utilization of the node
$Time_{free}$	The CPU free time
$Time_{total}$	The CPU total usage time
$Load_{N_i}$	The load of the node N_i
$\overline{Load}$	The average of all nodes load in the cluster
S	The cluster load standard deviation

In Flink, a slot is a basic unit for dividing node resources and a container for executing tasks. Therefore, assigning a subtask to the slot of a node, it should be considered whether the resources currently owned by the node slot can meet the basic requirements for different types of resources to execute the task. When the subtask's demand for a certain type of resource exceeds the capacity of the resource in the node slot, the subtask cannot be assigned to the slot. So the constraint relationship between the resource requirements of the subtasks currently in the allocation stage and the available resource capacity of the optional node slot is shown in Eqs. (1) and (2).

$$RD^c_{T_j} \ \leq \ RA^c_{Slot_i} \tag{1}$$

$$RD^m_{T_j} \ \leq \ RA^m_{Slot_i} \tag{2}$$

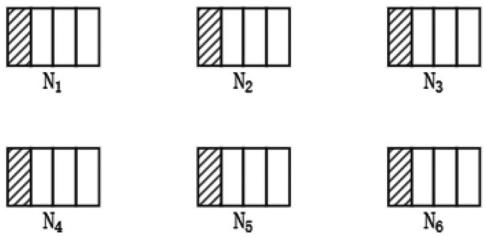


Fig. 4. Flink subtasks distributed on cloud hosts.

$RD^c_{T_j}$ represents the resource requirement of the task T_j for the CPU and $RD^m_{T_j}$ represents the resource requirement of the task T_j for the memory. $RA^c_{Slot_i}$ and $RA^m_{Slot_i}$ represent the current CPU and memory resource availability of the node slot, respectively.

Usually, the resources owned by each node are limited, so the resources owned by the entire Flink cluster are also limited. According to the classification of Amazon Cloud, cloud host nodes include general-purpose and computing-optimized, and so on. Different tasks generally have different requirements for CPU resources and memory resources. Some tasks are CPU-intensive and some tasks are memory-intensive. Therefore, after a topology job is scheduled, for multiple subtasks allocated to a node N_i, the sum of the resource requirements of these subtasks cannot exceed the total resource ability of the node, as shown in Eqs. (3) and (4).

$$\sum_{j=1}^{n} RD^c_{T_j}\ \zeta(T_j, k,\ N_i)\ \leq\ RV^c_{N_i} \tag{3}$$

$$\sum_{j=1}^{n} RD^m_{T_j}\ \zeta(T_j, k,\ N_i) \leq\ RV^m_{N_i} \tag{4}$$

T_j represents a task in the topology job. $RV^c_{N_i}$ represents the CPU resource capacity of the node N_i, and $RV^m_{N_i}$ represents the memory resource capacity of the node. The function of $\zeta(T_j, k,\ N_i)$ is to judge the physical location of the task, indicating whether the task T_j is assigned to the node N_i of type k, as shown in Eq. (5).

$$\zeta(T_j, k,\ N_i) = \begin{cases} 1, & if\ task\ T_j\ is\ allocated\ to\ node\ N_i; \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

3.1.1 Resource Cost

Due to the different types of cloud hosts, the price is different. Usually, among the same type of cloud hosts, the cloud hosts with larger resource capacity are priced higher [30]. Suppose a Flink-based high-availability cluster is deployed on the cloud platform, and the notation m for the type of cloud host nodes. At this time, a topology job is submitted to the cluster and recorded as a job. After the task is scheduled for the job, it occupies some cloud hosts. However, a small number of tasks being placed on each cloud host, not only the resources of each machine cannot be fully used, but also the cost of resource rental is too high, as shown in Fig. 4.

In the above cluster, the resource rental cost for job execution needs to be calculated by the sum of the rental costs of different types of cloud hosts, as shown in Eqs. (6), (7), (8), and (9).

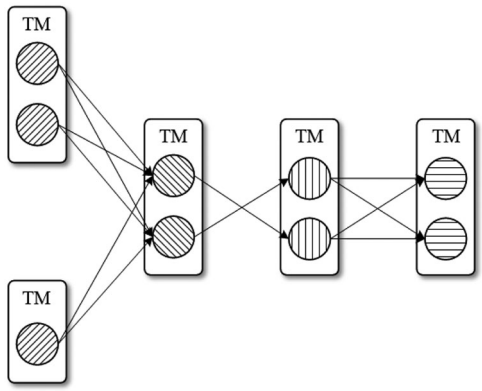


Fig. 5. Example of communication between nodes.

$$RC = \sum_{k=1}^{m}\sum_{i=1}^{n} RC(k) * u(N_i) \tag{6}$$

$$RC(k) = P_k * ET_{max} \tag{7}$$

$$ET_{max} = Max(ET(N_i, Job)) \tag{8}$$

$$u(N_i) = \begin{cases} 1, & if\ VM\ \ N_i\ is\ used; \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

As shown in Eq. (6), RC represents the resource cost of completing the job which is the sum of the rental overhead of all different types of cloud hosts when executing jobs. Among them, m represents the number of types of cloud hosts, and n represents the number of cloud hosts. $RC(k)$ represents the resource cost that a k-type cloud host performs a certain topology job. As shown in Eq. (7), it is calculated by multiplying the charging standard of the k-type cloud host by the time it takes for the cluster to complete the topology job. ET_{max} is the completion time of executing the topology job submitted by the user, which is expressed as the maximum time spent by all nodes in the cluster to execute the assigned subtasks, as shown in Eq. (8). As shown in Eq. (9), the function $u(N_i)$ indicates whether the cloud host of the type is used in this job execution. If the node is used, the result is 1 and the cost of the node needs to be calculated. If the node is not used, the result is 0 and the cost of the node does not need to be calculated.

3.1.2 Communication Cost

For a topology job with a definite structure in a Flink cluster, no matter what method is used to schedule the subtasks, the sum of data transmitted between subtasks remains unchanged. Therefore, if subtasks with dependencies between each other are always assigned to different nodes, the communication cost will increase with the increase of communication volume. The communication between nodes is shown in Fig. 5.

In the cloud platform, the communication between nodes is usually regarded as an integral part of the computing cost. Because if there is too much network transmission between nodes, the resulting transmission delay will cause the job running time to be longer, and will also cause communication overhead. Therefore, cloud service providers generally charge additional fees that is communication cost, because data transmission actions generated on communication links between nodes. The calculation method of communication cost is shown in Eq. (10).

$$CC = \sum_{i,j\in\{1,\ldots,n\}} P_f * D(T_i,\ T_j) * f(T_i,\ T_j) \tag{10}$$

CC represents the sum of communication costs between nodes generated by subtasks in the process of job execution, and P_f represents the price of communication between nodes. T_i and T_j represent the ith and jth subtasks, respectively. $D(T_i,\ T_j)$ represents the network distance between the task T_i and T_j respectively assigned cloud host nodes. $f(T_i,\ T_j)$ represents the task T_i and T_j resulting inter-node traffic, as shown in Eq. (11).

$$f(T_i,\ T_j) = flow(T_i,\ T_j) *\ SN(T_i,\ T_j) \tag{11}$$

$flow(T_i,\ T_j)$ represents the communication traffic between the task T_i and T_j, and it is still uncertain whether it is the communication traffic between nodes at this time. The meaning of the function $SN(T_i,\ T_j)$ is to judge whether the task T_i and T_j are assigned to the same cloud host node, and the return result is 1 or 0. If they are not allocated to the same cloud host, the communication traffic $flow(T_i,\ T_j)$ between them is regarded as inter-node communication, otherwise, it is regarded as intra-node communication.

Here is a brief description of the judgment method used in this paper to determine whether two tasks are assigned to the same node. For example, for a node N_l, T_i and T_j are respectively passed into the Eq. (5) as the first parameter of the function to obtain two functional expressions, and then judge whether the two subtasks are assigned to the same cloud host through the XOR result of the two functional expressions, as shown in Eq. (12).

$$SN(T_i,\ T_j) =\ \zeta(T_i, k,\ N_l)\ \oplus\ \zeta(T_j,\ k, N_l) \tag{12}$$

3.1.3 Scheduling Cost

When different scheduling strategies assign tasks to subtask sets in the same topology job, the time cost is different. Because the different algorithm complexity and the task scheduling occurs before the task manager executes the task. The scheduling overhead before the task starts executing is defined as the scheduling cost.

Since subtasks are created in the Created state and change to the Deploying state when they are scheduled to complete. Therefore, this paper takes the time when the subtask state changes to Created state as the starting point and the time when the subtask completes scheduling flow to Deploying state as the ending point, and takes the time interval between them as the scheduling time of the policy. The scheduling cost is calculated as shown in Eqs. (13) and (14).

$$SC = \sum_{k=1}^{m}\sum_{i=1}^{n} P_k\ * u(k,\ N_i) *\ ST \tag{13}$$

$$ST = Time_{\ Dep} - Time_{\ Cre} \tag{14}$$

SC represents the scheduling cost generated when Flink adopts different scheduling strategies, and P_k represents the charging standard for k-type cloud hosts given by the cloud service provider to the user. $u(k,\ N_i)$ is shown in Eq. (9). ST represents the time spent by Flink to allocate tasks using different task scheduling algorithms, that is the scheduling time of different strategies, as shown in Eq. (14). $Time_{\ Dep}$ represents the time when the states of all subtasks in the topology job are transferred to the deploying state. $Time_{\ Cre}$ represents the time when the job manager completes the creation of all subtask instances according to the execution graph of the job.

3.1.4 Execution Cost

The cost of executing the job is the execution cost in this paper, including three types: resource cost, communication cost and scheduling cost. To better evaluate the execution cost of the job, the weighted sum is defined as the execution cost of the job. Therefore, an integrated execution cost model is constructed in this section by setting different weights for the three costs, as shown in Eqs. (15) and (16).

$$WEC = \alpha_1 * RC + \alpha_2 * CC + \alpha_3 * SC \tag{15}$$

$$\alpha_1 + \alpha_2 + \alpha_3 = 1 \tag{16}$$

WEC is the weighted sum of the three costs (the execution cost of the job), which is used as a comprehensive evaluation index of the cost in this paper. RC, CC and SC represent the resource rental cost, the communication cost between nodes and the scheduling cost generated by the task scheduling strategy when the job is completed, respectively. α_1, α_2 and α_3 represent the weight of the above three costs, respectively.

3.2 Cost-Efficient Load Balancing Model

The purpose of Flink scheduling of tasks is to divide the CPU, memory, and other resources of the nodes [31]. These tasks running based on the node's computing resources are the current workload of the node. If a load of a node is higher, the higher the ratio of resources used by tasks in the node to the total resource capacity of the node.

Constructing a load balancing model, it is necessary to consider the resource constraints of each node on its tasks when the Flink cluster performs task scheduling. For a node N_i and multiple tasks $T = \{T_1, T_2, \ldots, T_n\}$ running on it, the total CPU resource requirements of all tasks in the set T should be less than or equal to the node's CPU resource threshold according to CPU resources. According to memory resources, the total memory resource requirements of all tasks should be less than or equal to the node's memory threshold. The definition of specific resource constraints is shown in Eqs. (17), (18), and (19).

$$RTH = RV *\ \delta \tag{17}$$

$$\sum_{j=1}^{n} RD^c_{T_j}\ \zeta(T_i, k,\ N_i) \le RTH^c_{N_i} \tag{18}$$

$$\sum_{j=1}^{n} RD^m_{T_j}\ \zeta(T_j, k,\ N_i) \le\ RTH^m_{N_i} \tag{19}$$

RTH is the threshold of a certain resource in the node, and if this threshold is exceeded, it will affect the performance of the node. RV is the capacity of a certain resource in the node, referring to Eqs. (3) and (4). δ is a number between 0 and 1, representing the percentage of resource

threshold to node resource capacity. $RTH^{c}_{N_i}$ is the CPU resource threshold of the node N_i and $RTH^{m}_{N_i}$ is the memory resource threshold of the node N_i. For other elements in the formula, please refer to the description in Table 2.

For the resource requirements of tasks, Buyya et al., [31] proposed a model based on data flow size and resource consumption to calculate the resource requirements of tasks, as shown in Eqs. (20) and (21).

$$RD^{c}_{T_j} = Data_{in} * \rho_c^{T_j} \tag{20}$$

$$RD^{m}_{T_j} = Data_{in} * \rho_m^{T_j} \tag{21}$$

$Data_{in}$ represents the size of the total data stream transmitted to T_j by the upstream task. $\rho_c^{T_j}$ and $\rho_m^{T_j}$ represent the CPU and memory resources consumed when a record in the data stream is processed, respectively.

To ensure that the cluster can execute jobs at a relatively low cost, here is a simplified definition of the cost-efficient of each slave node in the cluster executing tasks. It is necessary to obtain the amount of data processed by each task manager within a certain period of time and the corresponding cost of processing these data. Then, the cost of the node processing unit data is calculated. Finally, the cost evaluation value of the node is obtained, as shown in Eq. (22).

$$CE_{N_i} = \frac{Cost_{N_i}}{Tuple_{N_i}} \tag{22}$$

CE_{N_i} and $Cost_{N_i}$ represent the cost evaluation value and the cost overhead when the node N_i processes a certain amount of data, respectively. $Tuple_{N_i}$ is the amount of data processed by the node N_i in a period of time, that is the sum of data processed by all tasks in the node during this period of time.

In addition, it is necessary to predict the busyness of nodes [32]. If a node has a high CPU utilization for a long time, it means that the node is busy. Therefore, the average CPU utilization in a period of time (DR) is defined as Avg^{DR}_{cpu}. The proportion EP of the time when the CPU utilization exceeds the Avg^{DR}_{cpu} in the time period DR is used as a means of predicting the busyness of a node. Finally, the fitness of each node is defined by combining the cost evaluation and busyness of the node. Then the average fitness value AF of the cluster is obtained through the fitness of each node, as shown in Eqs. (23)–(25).

$$B = \frac{\sum_x^X EP_x}{X} \tag{23}$$

$$F_{N_i} = \frac{1}{CE_{N_i} * B_{N_i}} \tag{24}$$

$$AF = \frac{\sum_{i=0}^{n} F_{N_i}}{n} \tag{25}$$

B represents the predicted value of the busyness which is the average value of EP in multiple time periods. X represents the number of time periods, and EP_x represents the predicted value of the busyness in the xth period. F_{N_i} and B_{N_i} represent the fitness value and the predicted value of the busyness of the node N_i, respectively. AF is the average fitness of all nodes in the cluster.

As for the load of the node, since the resource requirements of the task are mainly concentrated on the CPU and memory resources of the node, the load of the node is defined by the CPU utilization and memory utilization of the node. For different types of tasks, the weight of CPU and memory utilization need to be calculated separately, as shown in Eqs. (26)–(28).

$$U_{cpu} = 1 - \frac{Time_{free}}{Time_{total}} \tag{26}$$

$$U_{mem} = 1 - \frac{Mem_{free}}{Mem_{total}} \tag{27}$$

$$Load_{N_i} = \alpha * U_{cpu} + \beta * U_{mem} \tag{28}$$

U_{cpu} and U_{mem} represent the CPU utilization and memory utilization of the node, respectively. $Time_{free}$ and $Time_{total}$ represent the CPU idle time and total usage time, respectively. Mem_{free} and Mem_{total} represent the remaining capacity and total memory capacity, respectively. $Load_{N_i}$ represents the load of the node N_i. α and β are defined as the weight of CPU and memory utilization, respectively, as shown in Eq. (29).

$$\alpha + \beta = 1 \tag{29}$$

For the evaluation of cluster load balancing, there are usually two definitions. One is that all nodes in the cluster perform the same number of tasks or process roughly the same amount of data. The other one is that all nodes in the cluster have roughly the same resource utilization [33]. However, in a heterogeneous big data cluster, the first definition ignores the node differences in the cluster. Therefore, in order to better evaluate the overall load situation of the cluster, the second definition is used here. The standard deviation of the load of all slave nodes in the cluster is calculated by Eq. (28) and defined as S, regarded as the load balance degree of the cluster. These can be expressed as Eqs. (30) and (31).

$$\bar{Load} = \frac{1}{n}\sum_{i=1}^{n} Load_{N_i} \tag{30}$$

$$S = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(Load_{N_i} - \bar{Load}\right)^2} \tag{31}$$

$\bar{Load}$ is the average of all nodes load in the cluster, and n is the number of slave nodes. S is the cluster load standard deviation calculated by Eqs. (26), (27), and (28). The smaller the standard deviation of the node is, the more balanced the load of each node is.

4 Algorithm Design and Implementation

4.1 System Architecture

By modifying the default Flink task scheduling module and introducing some new functional components, an extended Flink scheduling system is implemented in this paper. The system includes an improved Flink task scheduler, a resource monitoring module, a scheduling monitoring module, a data storage module and a data analysis and a processing module. Among them, the improved task scheduler is implemented by modifying the code at the core scheduling module (Flink Runtime) of the job manager in the Flink cluster. The resource monitoring module is responsible for

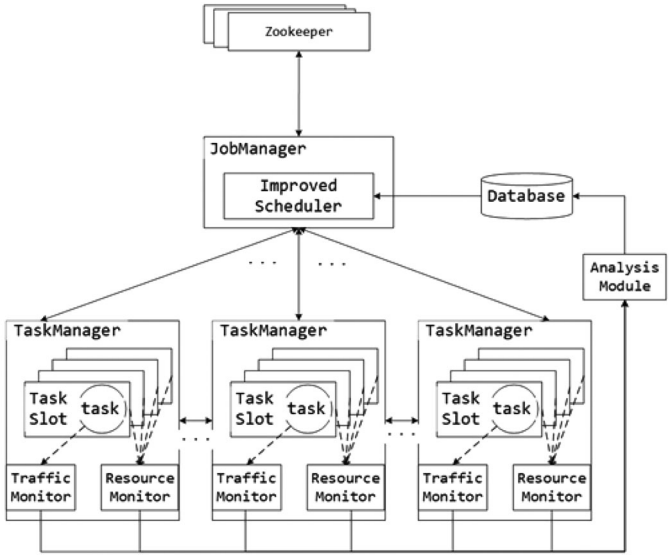


Fig. 6. Improved Flink architecture diagram.

obtaining the resource change information when the cluster TM nodes are running. The communication monitoring module is used to obtain the inter-node communication between each TM and the amount of data processed by each subtask per unit time when the cluster executes jobs. The function of the analysis and the processing module is to read the runtime data saved by the monitoring module, and then perform analysis and processing to obtain the corresponding information such as resources and costs. The improved Flink architecture is shown in Fig. 6.

4.2 Cost-Efficient Task Scheduling Algorithm

According to the cost-efficient model in Section 3.1, a task scheduling algorithm for Flink is proposed. First, the algorithm considers the heterogeneity of the cluster environment. There may be multiple types of nodes in the cluster and each type of nodes has different computing resources. So the nodes have different processing capabilities and rates. Second, the number of cloud hosts required to execute jobs is minimized by selecting nodes that are as cost-efficient as possible for task execution to ensure that the computational resources of the nodes are fully utilized. Then, tasks with dependencies are assigned to minimize inter-node communication. The flowchart of cost-efficient task scheduling algorithm is shown in Fig. 7.

For the proposed scheduling algorithm in Flink, the pseudo-code of cost-efficient task scheduling algorithm is shown in Algorithm 1, where lines 3-5 are the preparation work, including obtaining the cost priority list, etc. Lines 6-19 are the specific scheduling process. Lines 22-24 indicate collecting, processing and saving the runtime information of the cluster. Moreover, there are some definitions about various terms and abbreviations used in the Algorithm 1. JRI is the cluster runtime information corresponding to the Job. NPL is the node priority list obtained by sorting the node cost information and STL is a scheduling task list. NCIL is a list of cost information for each node. FreeSlotList and freeSlots represent the list of free slots and the free slots that meet the resource demand, respectively.

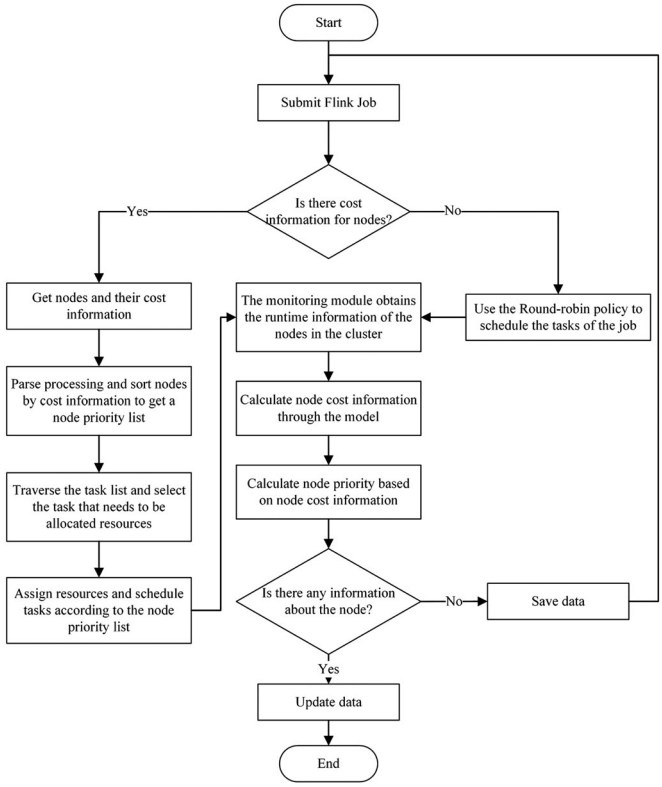


Fig. 7. The flowchart of cost-efficient task scheduling algorithm.

Algorithm 1. Cost-Efficient Task Scheduling Algorithm

Input: Flink job
Output: Task scheduling results for the current job
1: Get cluster runtime information JRI
2: if JRI != null then
3: parse JRI and get cost information NCIL of each node
4: sort the nodes by cost information to get the node priority list NPL
5: for subtask in STL do ▷ Traversing the Task List STL
6: initially get the freeSlotList
7: get the slots that meet the requirements into freeSlots
8: get the node with the highest priority from the NPL
9: if there are no free slots in the node then
10: reacquire the node with higher priority from the NPL
11: end if
12: for slot in freeSlots do ▷ Iterate through the list of free slots
13: if slot belongs to node then
14: the number of available slots for the node minus 1
15: return slot
16: end if
17: end for
18: end for
19: else
20: scheduling Jobs with Round-Robin scheduling strategy
21: end if
22: get cluster runtime information
23: get cost information
24: update or save information

The number of nodes in the cluster is n and the total number of free slots is f. The number of tasks is m. Since the nodes are prioritized, the time complexity here is $O(nlogn)$. Second, the nested for loops are performed for both the

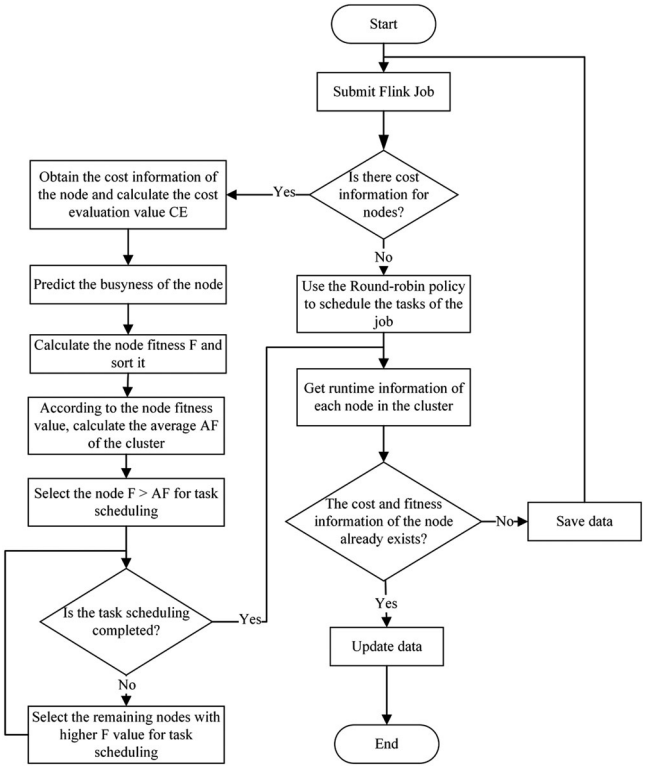


Fig. 8. The flowchart of cost-efficient load balancing algorithm.

subtask list and the free slot list, and the cluster can work properly only when the number of tasks is satisfied, so the time complexity here is $O(m^2)$. In summary, the time complexity of this algorithm is $O(nlogn + m^2)$ during task scheduling.

4.3 Cost-Efficient Load Balancing Algorithm

Based on the cost-efficient load balancing model described in Section 3.2, a cost-efficient load balancing algorithm (LBA-CE) is proposed. The algorithm is based on the cost of cluster execution jobs, considering the heterogeneity of the cluster. It predicts the busyness of nodes to calculate the cost-based load adaptation of each node, and then finds the appropriate node for the task based on the adaptation. The flowchart of cost-efficient load balancing algorithm is shown in Fig. 8.

The pseudo-code of the load balancing optimization algorithm proposed in this section is shown in Algorithm 2. Some definitions about various terms and abbreviations used in the Algorithm 2 are as follows. CE is the cost evaluation value and AF is the average fitness of nodes in the cluster. CEL is a list of the cost evaluation value of nodes. BEL is a list of the predicted value of node busyness and FL is a list of fitness values. HFL and LFL represent the list with higher fitness values and the list of with lower fitness values, respectively. Among them, lines 3-7 are the preparation work, such as obtaining the historical runtime information of tasks, calculating the cost evaluation value CE of nodes, the predicted value of node busyness, etc., and finally obtaining the fitness of nodes and the average value AF of the fitness of nodes in the cluster. Lines 8-13 are the AF as the dividing point to put nodes into different node lists. Lines 14-32 are the task scheduling process, where nodes are selected based on their fitness. When assigning tasks to nodes, it is also necessary to pay attention to whether the total resource demand of tasks on the node exceeds the resource threshold of the node. If assigning tasks to a node will cause the resource occupation to exceed the threshold, the node will not continue to participate in task scheduling. Lines 35-37 indicate collecting, processing and saving the runtime information of the cluster.

Algorithm 2. Cost-Efficient Load Balancing Algorithm

Input: Flink job
Output: Task scheduling results for the current job
1: Get the cluster runtime information JRI
2: if JRI != null
3: calculate the cost evaluation value of the node and save it to the list CEL
4: calculate the busy evaluation value of the node and save it in the list BEL
5: calculate the node fitness F and save it to the node list FL
6: sorting FLs in descending order according to fitness
7: calculate the average of the fitness of all nodes in the cluster AF
8: for node in FL do ▷ Iterate the node list
9: If the F value of node $\geq$ AF then
10: nodes are added to the HFL list
11: Else nodes are added to the LFL list
12: end if
13: end for
14: for subtask in STL do ▷ Iterate the task list STL
15: initially get freeSlotList
16: get the slots that meet the requirements into freeSlots
17: for node in HFL do ▷ Iterate through the list with higher adaptation
18: if HFL list has no node then
19: add the node in LFL to HFL
20: end if
21: if node has no free slot then
22: continue
23: end if
24: for slot in freeSlots do ▷ Iterate through the list of free slots
25: if slot belongs to node then
26: update the number of available slots in the node
27: return slot
28: end if
29: end for
30: end for
31: end for
32: else
33: use Round-Robin scheduling strategy
34: end if
35: get cluster runtime information
36: get cost information
37: update or save data

Suppose that the number of nodes in the cluster is n and the number of slots owned by the cluster is f. Suppose that the number of tasks in the topology job is m. In addition, n, f, and m are all positive integers. The time complexity of the algorithm is $O(nlogn + mnf)$ during task scheduling.

TABLE 3
The Priority of Nodes

Node IP	Node Name	Number of Slots	Node Priority
172.23.67.13	m3	4	1
172.23.67.12	m2	4	2
172.23.67.14	m4	4	3
172.23.67.15	l1	4	4
172.23.67.17	l3	4	5
172.23.67.18	l4	4	6
172.23.67.19	xl1	4	7
172.23.67.16	l2	4	8
172.23.67.22	xl4	4	9
172.23.67.21	xl3	4	10
172.23.67.20	xl2	4	11

4.4 Complexity Comparison of State-of-the-Art Models

There exist some related state-of-the-art models need to be discussed. The greedy algorithm adapted from the Best-Fit-Decreasing (GS-BFD) is proposed in [20] which can improve cost-efficiency of a cloud deployed Spark cluster. The time complexity of GS-BFD is $O((n^2 logn)(p+r))$, where p and r is the total number of deadline constrained and regular jobs, respectively that need to be scheduled. And the total number of VMs in the cluster is n. In addition, the First Fit (FF) and Greedy Iterative Optimization (GIO) are proposed in [24] which can optimize the virtual machine usage cost for both local and cloud resources and maximize the job deadline met percentage. The time complexity of FF is $O(ne)$, where the current job's total number of executor requirements is e. Compared with CETSA proposed in this paper, FF has a lower time complexity because it does not sort the list of Virtual Machines(VMs) according to the cost of VMs resulting in a waste of costs. Moreover, GIO is suitable for the hybrid cloud and the time complexity of GIO is $O(n + nlogn + ne)$.

4.5 Scientific Workflow Test

We choose Hibench's Wordcount workload to test the scientific workflow of the proposed algorithm at a rate of 20,000 records/second. First, the runtime information of the cluster is obtained through the monitoring module of the cluster. And then, the runtime information is parsed by the data analysis and processing module, to calculate the cost of each node in the cluster at the current moment according to the proposed cost-efficient model. Based on the cost of each node, the priority of the node is obtained. The lower the cost of the node is, the higher the priority of the node is. When the number of records processed by the cluster reaches tens of millions, the priority of each node in the cluster at the current moment is calculated as show in Table 3.

In addition, we can get the information of nodes and tasks assignment at present by the UI interface of Flink. To visualize the assignment between tasks and nodes, the assignment between tasks and nodes at present is shown in Fig. 9. Since the task parallelism is 20 and each node has 4 slots, tasks will be assigned to run on the 5 nodes with the lowest cost. As shown in Fig. 9, assigning tasks to nodes, the node with higher priority is always selected first to minimize the job execution cost of the cluster. Therefore, it can be concluded that our proposed workflow is scientifically efficient.

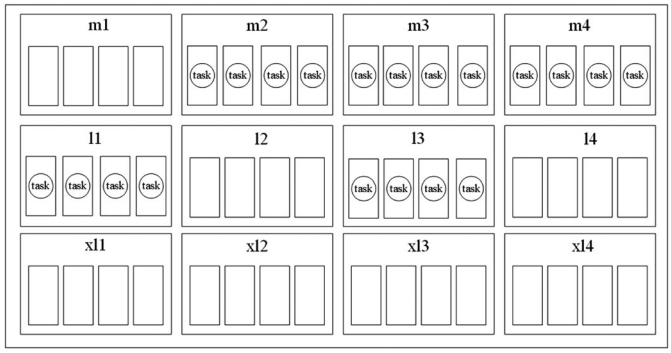


Fig. 9. The assignment between tasks and nodes.

5 Evaluation and Analysis

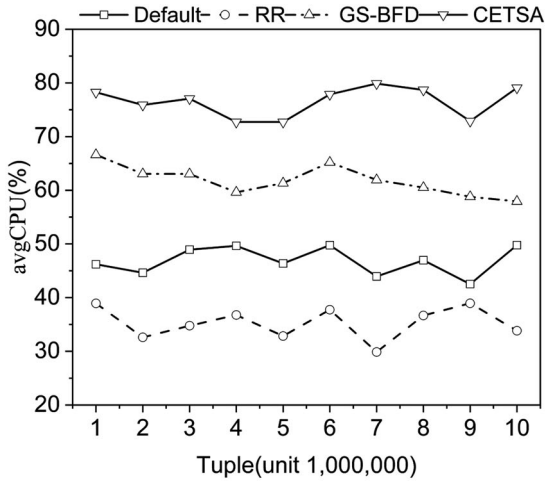
5.1 Experimental Setup

The experimental environment is built on ZTE Cloud Platform. The infrastructure consists of 12 virtual machines created on different hosts of the cloud platform, with a total cluster of 96 CPU cores and 144 GB of memory capacity. The specific configurations of the nodes are shown in Table 4. On each VM node, Zookeeper, Hadoop, Flink and Kafka are installed and deployed as a cluster. For the resource constraint between tasks and nodes, it is usually set to 0.8 in big data clusters, which means that the resource threshold of a node is 80% of the total resources. When the total resource occupation of tasks on a node reaches this value it is considered that the node's resources are almost consumed, and if this value is exceeded, there will be an impact on the performance of the node [34]. In this paper, when calculating the node load, the weighted sum of CPU and memory utilization is taken as 0.8 and 0.2, respectively.

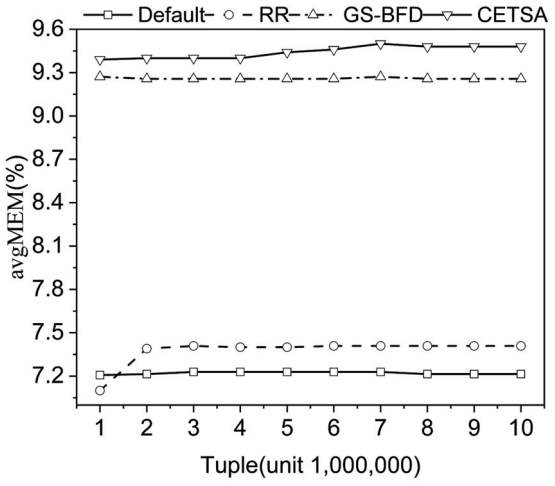
To test the performance of the cost-efficient task scheduling algorithm (CETSA) and the cost-efficient load balancing algorithm(LBA-CE) proposed in this paper, the default scheduling algorithm of Flink (Default), the Round-Robin (RR) and the greedy scheduling algorithm based on Best-Fit-Decreasing (GS-BFD) [20] are selected as the comparison algorithms in this experiment. Hibench is deployed on the master node to test the performance of the proposed scheduling algorithms under Wordcount and Fixwindow. HiBench [35] is a big data benchmark tool developed by Intel. In addition, the default data generation rate of Hibench is set to 60,000 records/s. Meanwhile, Wordcount and Fixwindow in Hibench are selected to test different scheduling algorithms.

TABLE 4
Experimental Cluster Details

VM type	CPU (cores)	Memory (GB)	Pricing ($/s)	Role	Quantity	Flink version
Small	4	8	2.417E-3	1JM, 3TM	4	
Medium	8	12	4.861E-3	TM	4	Flink-1.11.1
Large	12	16	7.778E-3	TM	4	



(a) Average CPU utilization



(b) Average memory utilization

Fig. 10. Comparison of average resource utilization under wordcount.

5.2 Result Analysis and Discussion

5.2.1 Wordcount

The Wordcount workload in Hibench is mainly used to count words. The source operator reads the data stream from Kafka. The data transformation operator splits the long string into multiple strings using a number of spaces as separators, and then counts the words represented by the strings and saves the intermediate results.

Fig. 10 shows the variation of the average resource utilization at a data generation rate of 60,000 records/s under Wordcount. The CETSA improves the average CPU utilization compared to the three policies mentioned above (Default, RR and GS-BFD) by at least 36%, 106% and 23%, at least 14%, 20%, and 2% improvement in average memory utilization, respectively.

Fig. 11 shows that LBA-CE policy proposed in this paper reduces about 37.9%, 36.4% and 3.3% of the execution cost on average compared to the three task scheduling policies such as the Default, RR and GS-BFD, respectively. Moreover, CETSA proposed in this paper reduces about 37.3%, 58.9% and 11.8% of the execution cost compared with the Default, RR and GS-BFD, respectively.

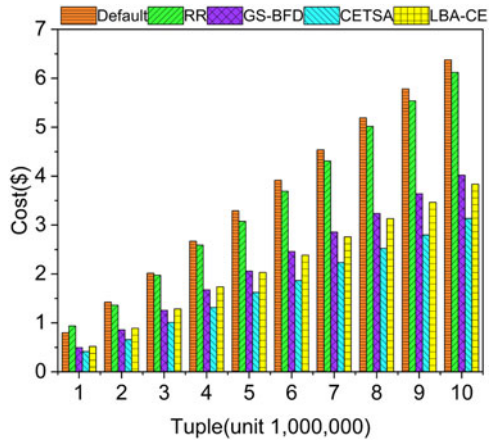


Fig. 11. Comparison of the total execution cost at a data generation rate of 60,000 records/s.

Table 5 shows the comparison of execution time for Wordcount at a data generation rate of 60,000 records/s. It can be seen that the execution time of CETSA will be slightly longer than RR and LBA-CE, and shorter than the default policy and GS-BFD. This is because CETSA assigns tasks too densely compared to the first two, which may cause data buildup at some nodes, making the processing delay increase and thus prolonging the execution time. The LBA-CE has the shortest execution time in comparison to the three policies, such as Default, GS-BFD and CETSA, with an average reduction of about 11.9%, 9.2% and 7.1%, respectively.

Fig. 12 shows the comparison of degree of load balancing for Wordcount. It can be seen that the LBA-CE proposed in this paper has a more significant improvement in the overall load balancing of the cluster. Compared with the Default, the BFD and the CETSA proposed in this paper, the LBA-CE reduces volatility of node resource utilization by about 23.1%, 12.5%, and 23.9% on average, respectively. Compared with RR, most of the time the LBA-CE and CETSA proposed in this paper perform better in terms of the degree of load balancing. In conclusion, the LBA-CE performs better than the RR policy and the volatility is reduced by about 4.5% on average.

Fig. 13 shows the comparison of Inter-node communication costs for Wordcount at a data generation rate of 60,000 records/s. It can be found that the LBA-CE generates lower inter-node communication costs than the Default and RR, with an average reduction of about 18.5% and 11.5%, respectively. In contrast, the communication cost generated

TABLE 5
Comparison of Execution Time Under Wordcount

Tuple (unit 1,000,000)	Default (s)	RR (s)	GS-BFD (s)	CETSA (s)	LBA-CE (s)
1	27	29	26	27	25
2	48	42	45	43	43
3	68	61	66	66	62
4	90	80	88	86	79
5	111	95	108	106	98
6	132	114	129	122	115
7	153	133	150	146	133
8	175	155	170	165	151
9	195	171	191	183	167
10	215	189	211	205	185

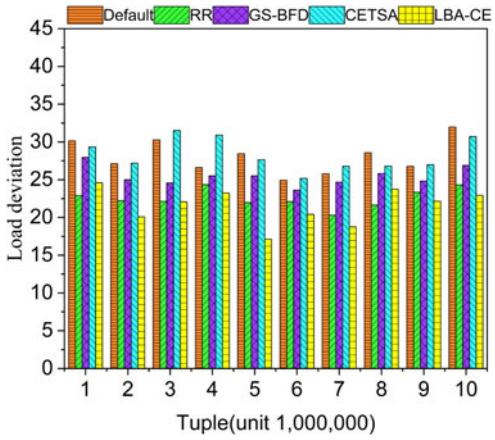


Fig. 12. Comparison of degree of load balancing under Wordcount.

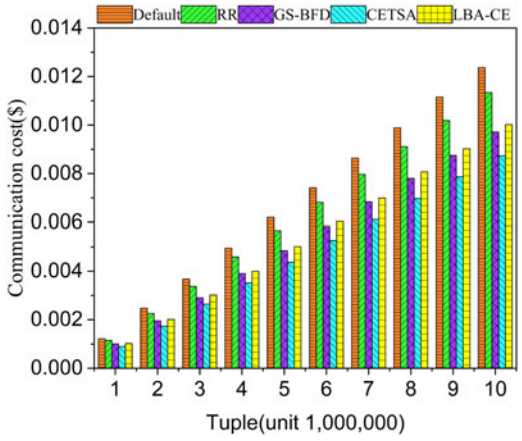


Fig. 13. Comparison of inter-node communication costs under Wordcount.

by the LBA-CE proposed in this paper is relatively slightly higher compared to both the GS-BFD and the CETSA. Moreover, CETSA proposed in this paper reduces the communication cost by about 22%, 24.8% and 8.7% respectively compared to the other three strategies.

Fig. 14 shows the comparison of throughput of clusters for Wordcount at a data generation rate of 60,000 records/s. When running the Wordcount workload, the LBA-CE improves the overall cluster throughput by about 17.8%, 8.2%, 17.3%, and 13.2% compared to the Default, RR, GS-BFD and CETSA, respectively.

Fig. 15 shows the comparison of SLA violation rates for Wordcount at a data generation rate of 60,000 records/s. It can be seen that the default scheduling policy, GS-BFD and CETSA have higher SLA violation rates compared to the RR policy and LBA-CE.

Fig. 16 shows the execution cost comparison of different scheduling strategies when executing Wordcount load processing 10 million records at different rates. Compared with the other three strategies, CETSA saves about 37.3%, 58.9% and 11.8% of the execution cost, respectively.

5.2.2 Fixwindow

One of the core functions of streaming processing framework is to perform window calculations on unbounded

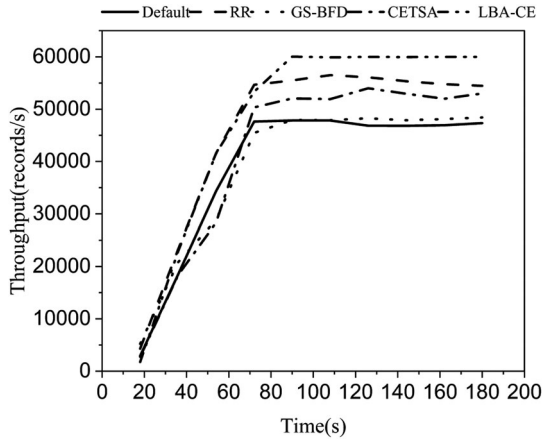


Fig. 14. Comparison of throughput under Wordcount.

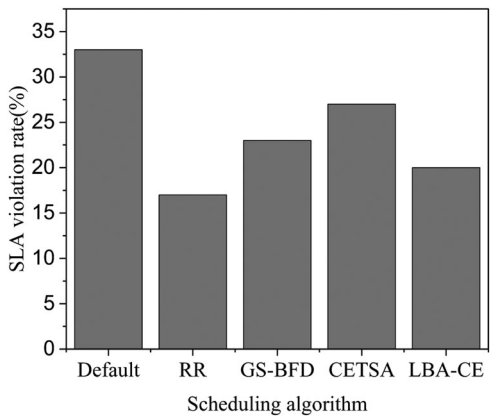


Fig. 15. Comparison of SLA violation rate for different scheduling algorithms under Wordcount.

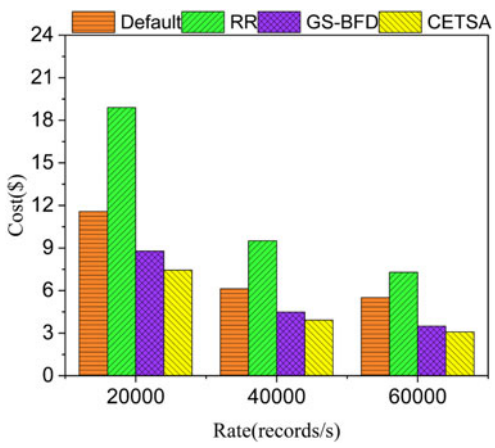
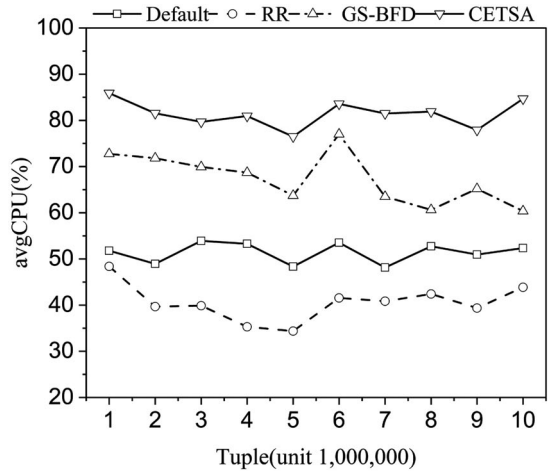


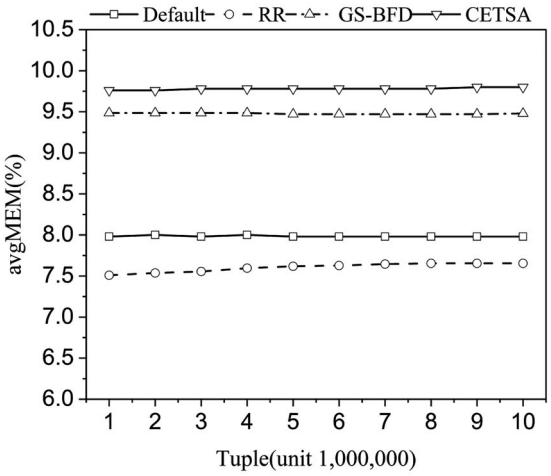
Fig. 16. Comparison of the total execution cost at different data generation rates under Wordcount.

data streams. The Fixwindow is to perform window-based aggregation.

The runtime resource utilization at a data generation rate of 60,000 records/s under Fixwindow is shown in Fig. 17. The CETSA proposed in this paper achieves a fuller utilization of node resources in the cluster compared to Default, RR and GS-BFD. For the average CPU utilization of the cluster, it improves about 58.3%, 100.6% and 20.8%, respectively. For



(a) Average CPU utilization



(b) Average memory utilization

Fig. 17. Comparison of average resource utilization under Fixwindow.

the average memory utilization of the cluster, it is improved by about 11.8%, 28.6%, and 2.8%, respectively.

Fig. 18 shows the comparison of inter-node communication cost at a data generation rate of 60,000 records/s when running Fixwindow load under different scheduling policies. It can be seen that LBA-CE reduces the communication cost

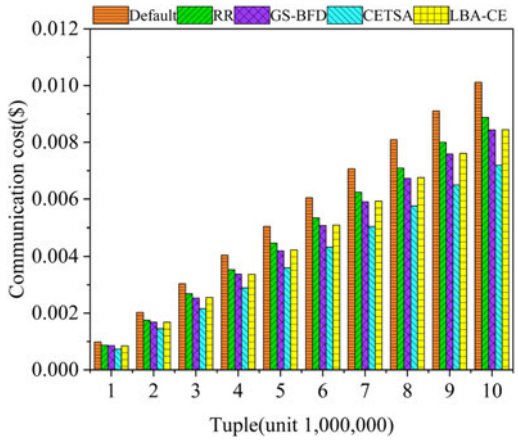


Fig. 18. Comparison of inter-node communication costs under Fixwindow.

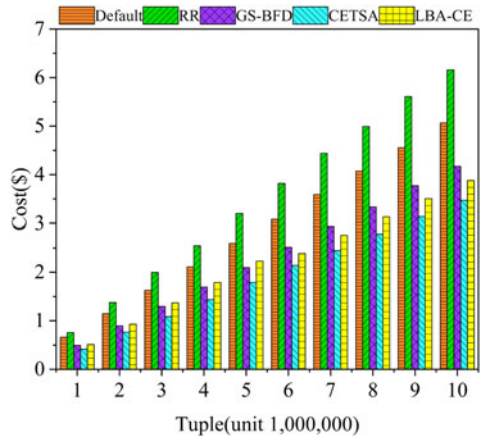


Fig. 19. Comparison of execution cost under Fixwindow.

by 16.2% and 4.6% compared to the Default and RR, respectively, and generates the same inter-node communication cost compared to the GS-BFD policy. Compared with the CETSA, the trend is the same as under Wordcount. Additionally, the CETSA proposed in this paper reduces the communication cost by at least about 5%, 25.1% and 7.3% compared with the three strategies of Default, RR and GS-BFD.

As shown in Fig. 19, the execution cost of the LBA-CE is reduced by about 20.2%, 34.3%, and 6.2% compared to the Default, RR, and GS-BFD, respectively. CETSA is a little lower than LBA-CE in terms of cluster execution cost. Because CETSA schedules the tasks in the topology jobs according to the cost information of the nodes executing the tasks, it pursues to complete the jobs submitted to the Flink cluster with the lowest possible execution cost.

Fig. 20 shows the comparison of degree of load balancing under Fixwindow. The LBA-CE reduces the load deviation about 24.6%, 4.5%, 25% and 24.3% on average compared to the Default, RR, GS-BFD and CETSA, respectively.

Table 6 shows the execution time spent by the Flink cluster running the Fixwindow application to process 1 million to 10 million records under different scheduling policies. It can be seen that the execution time of the LBA-CE is slightly longer than that of the RR, but there is not much difference compared to RR. Moreover, The LBA-CE reduces the execution time by about 7.5%, 13.9% and 17.8% compared to Default, GS-BFD and CETSA, respectively.

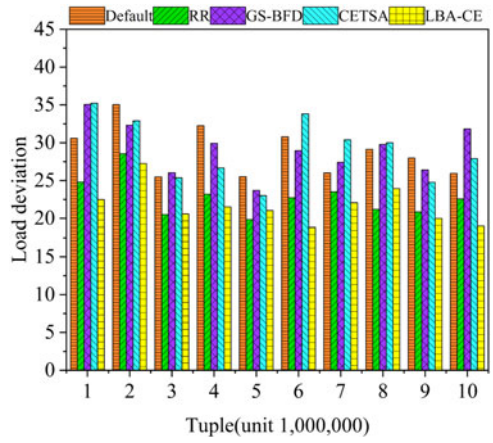


Fig. 20. Comparison of degree of load balancing under Fixwindow.

TABLE 6
Comparison of Execution Time Under Fixwindow

Tuple (unit 1,000,000)	Default (s)	RR (s)	GS-BFD (s)	CETSA (s)	LBA-CE (s)
1	26	22	26	27	22
2	45	40	47	50	40
3	64	58	68	71	59
4	83	74	89	94	77
5	102	93	110	117	96
6	121	111	132	140	115
7	141	129	154	160	133
8	160	145	175	182	151
9	179	163	198	205	169
10	199	179	219	227	187

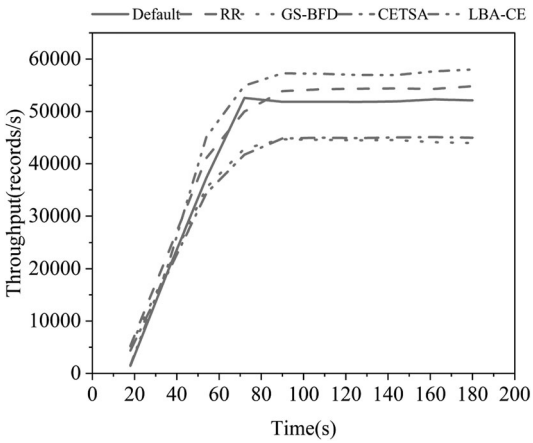


Fig. 21. Comparison of throughput under Fixwindow.

Fig. 21 shows the variation of cluster throughput when using different scheduling policies for job scheduling. It can be seen that the overall throughput of the cluster with different scheduling policies basically stabilizes between 70 and 90 seconds after the start of the job, and tends to increase until this time. The LBA-CE improves cluster throughput about 10.3%, 5.7%, 29.1% and 27.5%, respectively, compared to the other policies.

The SLA violation rate is shown in Fig. 22. It can be seen that the Default, GS-BFD and CETSA have higher SLA violation rates compared to the RR and LBA-CE. This is because the first three policies do not consider too much the relationship between the size of the data volume processed by the task and the processing capacity of the nodes.

Under the three data generation rates, when different scheduling strategies are used to execute the Fixwindow, the final execution cost comparison is shown in Fig. 23. Compared with the GS-BFD, CETSA reduces the execution cost by about 21.1%, 14.7% and 9.1% respectively. The reason is that CETSA considers not only the interdependence between upstream and downstream tasks but also the cost of executing tasks on different nodes.

5.2.3 Discussion

Recent studies [12], [13], [14], [15], [19], [20], [21] in task scheduling algorithms for big data processing mainly focus on improving performance, such as improving throughput

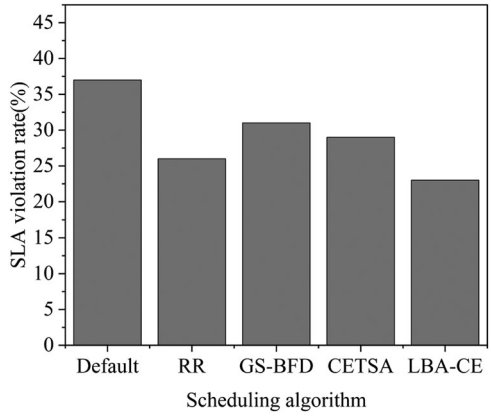


Fig. 22. Comparison of SLA violation rate for different scheduling algorithms under Fixwindow.

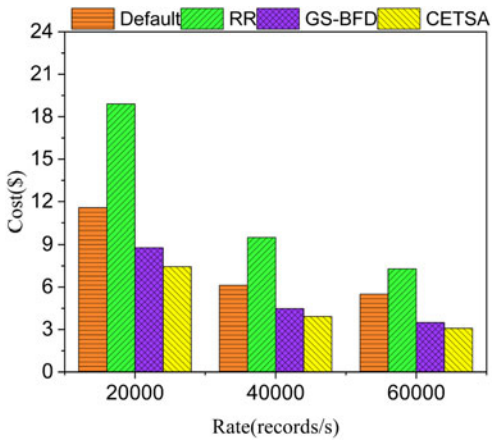


Fig. 23. Comparison of the total execution cost at different data generation rates under Fixwindow.

and resource utilization and reducing latency. Muhammad et al., [12] proposed an improved task scheduling strategy by monitoring the resource utilization of nodes in real-time while executing tasks, which increased the throughput of the cluster by about 30%. Li et al., [13] proposed a Flink-based elastic scheduling method to improve the cluster throughput in order to solve the problem of performance degradation and resource overuse of Flink under a fluctuating load of stream computing. Compared with these studies, the algorithms proposed in this paper not only focus the cost on the cloud, but also considers the cluster performance.

From the experimental results, it improved that the cost-efficient task scheduling algorithm (CETSA) proposed in this paper can significantly reduce the total cost consumption in Flink cluster on the cloud compared with the traditional scheduling algorithms. To reduce the total cost consumption while ensuring load balancing in Flink cluster on the cloud, a cost-efficient load balancing algorithm (LBA-CE) is proposed in this paper. The experiment indicates that LBA-CE can effectively reduce the total cost while ensuring load balancing of the Flink cluster on the cloud.

6 Conclusion

Flink is becoming more and more popular in the big data age because its functional features such as low processing

latency, relatively high throughput and checkpoints for fault recovery. In this paper, we proposed a cost-efficient scheduling algorithm and a cost-efficient load balancing algorithm to optimize some problems of the default task scheduling algorithm of Flink. This paper makes the following contributions. To solve the problem that the default scheduling algorithm of Flink may cause excessive job costs, a cost-efficient model based on the big data processing framework and a cost-efficient scheduling algorithm are proposed for Flink. This algorithm can effectively reduce the total cost consumption, but it may cause load imbalance in Flink cluster. Therefore, a cost-efficient load balancing model and a cost-efficient load balancing algorithm are proposed to trade off load balancing and the cost. The experiment of Flink cluster using Hibench workloads on the cloud shows that the proposed algorithms can significantly reduce the cost and optimize the cluster load compared with the traditional scheduling algorithms.

The future work will be focusing on the following aspects. The relationship between job execution time and cluster execution cost will be considered, and the trade-off between the two will be made according to the business realities. We will focus on deep reinforcement learning and try to combine it with the task scheduling process of the big data framework.

REFERENCES

[1] Y. Pu and J. Yu, "Energy-efficient strategy for data migration and merging in storm," *J. Commun.*, vol. 40, no. 12, 2019, Art. no. 68.

[2] B. Gupta and D. P. Agrawal, *Handbook of Research on Cloud Computing and Big Data Applications in IoT*. Hershey, PA, USA: IGI Global, 2019.

[3] S. Ramírez-Gallego et al., "An information theory-based feature selection framework for big data under apache spark," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 48, no. 9, pp. 1441–1453, Sep. 2018.

[4] M. Farrokh, H. Hadian, M. Sharifi, and A. Jafari, "SP-ant: An ant colony optimization based operator scheduler for high performance distributed stream processing on heterogeneous clusters," *Expert Syst. Appl.*, vol. 191, 2022, Art. no. 116322.

[5] P. Carbone, S. Ewen, G. Fóra, S. Haridi, S. Richter, and K. Tzoumas, "State management in apache flink®: Consistent stateful distributed stream processing," *Proc. VLDB Endowment*, vol. 10, no. 12, pp. 1718–1729, 2017.

[6] W. Dai, L. Qiu, A. Wu, and M. Qiu, "Cloud infrastructure resource allocation for big data applications," *IEEE Trans. Big Data*, vol. 4, no. 3, pp. 313–324, Sep. 2018.

[7] W. Zheng, Y. Qin, E. Bugingo, D. Zhang, and J. Chen, "Cost optimization for deadline-aware scheduling of big-data processing jobs on clouds," *Future Gener. Comput. Syst.*, vol. 82, pp. 244–255, 2018.

[8] M. R. HoseinyFarahabady, A. Jannesari, J. Taheri, W. Bao, A. Y. Zomaya, and Z. Tari, "Q-Flink: A QoS-aware controller for apache flink," in *Proc. IEEE/ACM 20th Int. Symp. Cluster Cloud Internet Comput.*, 2020, pp. 629–638.

[9] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, and R. Katz, "Multi-task learning for straggler avoiding predictive job scheduling," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 3692–3728, 2016.

[10] M. Ibrahim, S. Nabi, A. Baz, N. Naveed, and H. Alhakami, "Towards a task and resource aware task scheduling in cloud computing: An experimental comparative evaluation," *Int. J. Netw. Distrib. Comput.*, vol. 8, no. 3, pp. 131–138, 2020.

[11] S. K. Garg, A. N. Toosi, S. K. Gopalaiyengar, and R. Buyya, "SLA-based virtual machine management for heterogeneous workloads in a cloud datacenter," *J. Netw. Comput. Appl.*, vol. 45, pp. 108–120, 2014.

[12] A. Muhammad and M. Aleem, "A3-storm: Topology-, traffic-, and resource-aware storm scheduler for heterogeneous clusters," *J. Supercomput.*, vol. 77, no. 2, pp. 1059–1093, 2021.

[13] Z. Li et al., "Flink-ER: An elastic resource-scheduling strategy for processing fluctuating mobile stream data on flink," *Mobile Inf. Syst.*, vol. 2020, pp. 1–17, 2020.

[14] D. Sun, S. Gao, X. Liu, F. Li, X. Zheng, and R. Buyya, "State and runtime-aware scheduling in elastic stream computing systems," *Future Gener. Comput. Syst.*, vol. 97, pp. 194–209, 2019.

[15] S. Liu, J. Weng, J. H. Wang, C. An, Y. Zhou, and J. Wang, "An adaptive online scheme for scheduling and resource enforcement in storm," *IEEE/ACM Trans. Netw.*, vol. 27, no. 4, pp. 1373–1386, Aug. 2019.

[16] D. Sun, H. He, H. Yan, S. Gao, X. Liu, and X. Zheng, "LR-stream: Using latency and resource aware scheduling to improve latency and throughput for streaming applications," *Future Gener. Comput. Syst.*, vol. 114, pp. 243–258, 2021.

[17] Z. Hu, B. Li, Z. Qin, and R. S. M. Goh, "Low latency big data processing without prior information," *IEEE Trans. Cloud Comput.*, vol. 9, no. 4, pp. 1521–1534, Fourth Quarter 2021.

[18] L. Eskandari, J. Mair, Z. Huang, and D. Eyers, "I-scheduler: Iterative scheduling for distributed stream processing systems," *Future Gener. Comput. Syst.*, vol. 117, pp. 219–233, 2021.

[19] T. Toliopoulos and A. Gounaris, "Adaptive distributed partitioning in apache flink," in *Proc. IEEE 36th Int. Conf. Data Eng. Workshops*, 2020, pp. 127–132.

[20] M. T. Islam, S. N. Srirama, S. Karunasekera, and R. Buyya, "Cost-efficient dynamic scheduling of big data applications in apache spark on cloud," *J. Syst. Softw.*, vol. 162, 2020, Art. no. 110515.

[21] G. L. Stavrinides and H. D. Karatza, "An energy-efficient, QoS-aware and cost-effective scheduling approach for real-time workflow applications in cloud computing systems utilizing DVFS and approximate computations," *Future Gener. Comput. Syst.*, vol. 96, pp. 216–226, 2019.

[22] A. Hussain, M. Aleem, M. A. Iqbal, and M. A. Islam, "SLA-RALBA: Cost-efficient and resource-aware load balancing algorithm for cloud computing," *J. Supercomput.*, vol. 75, no. 10, pp. 6777–6803, 2019.

[23] H. Li et al., "A cost-efficient scheduling algorithm for streaming processing applications on cloud," *Cluster Comput.*, vol. 25, no. 2, pp. 781–803, 2022.

[24] M. T. Islam, H. Wu, S. Karunasekera, and R. Buyya, "SLA-based scheduling of spark jobs in hybrid cloud computing environments," *IEEE Trans. Comput.*, vol. 71, no. 5, pp. 1117–1132, May 2022.

[25] Y. Ping, "Load balancing algorithms for big data flow classification based on heterogeneous computing in software definition networks," *J. Grid Comput.*, vol. 18, no. 2, pp. 275–291, 2020.

[26] Y. Zhang, J. Yu, L. Lu, Z. Li, and Z. Meng, "L-Heron: An open-source load-aware online scheduler for Apache Heron," *J. Syst. Archit.*, vol. 106, 2020, Art. no. 101727.

[27] F. Ebadifard and S. M. Babamir, "Autonomic task scheduling algorithm for dynamic workloads through a load balancing technique for the cloud-computing environment," *Cluster Comput.*, vol. 24, no. 2, pp. 1075–1101, 2021.

[28] F. Ebadifard, S. M. Babamir, and S. Barani, "A dynamic task scheduling algorithm improved by load balancing in cloud computing," in *Proc. IEEE 6th Int. Conf. Web Res.*, 2020, pp. 177–183.

[29] D. Choi, H. Jeon, J. Lim, K. Bok, and J. Yoo, "Dynamic task scheduling scheme for processing real-time stream data in storm environments," *Appl. Sci.*, vol. 11, no. 17, 2021, Art. no. 7942.

[30] P. Han, C. Du, J. Chen, F. Ling, and X. Du, "Cost and makespan scheduling of workflows in clouds using list multiobjective optimization technique," *J. Syst. Archit.*, vol. 112, 2021, Art. no. 101837.

[31] X. Liu and R. Buyya, "D-Storm: Dynamic resource-efficient scheduling of stream processing applications," in *Proc. IEEE 23rd Int. Conf. Parallel Distrib. Syst.*, 2017, pp. 485–492.

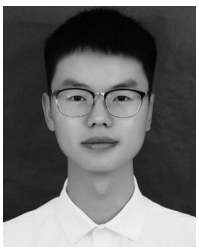
[32] Y. Xin, Z.-Q. Xie, and J. Yang, "A load balance oriented cost efficient scheduling method for parallel tasks," *J. Netw. Comput. Appl.*, vol. 81, pp. 37–46, 2017.

[33] X. Cao, S. Gao, and L. Chen, "Gossip-based load balance strategy in big data systems with hierarchical processors," *Wireless Pers. Commun.*, vol. 98, no. 1, pp. 157–172, 2018.

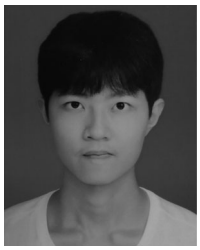
[34] F. Lombardi, L. Aniello, S. Bonomi, and L. Querzoni, "Elastic symbiotic scaling of operators and resources in stream processing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 3, pp. 572–585, Mar. 2018.

[35] Y. Samadi, M. Zbakh, and C. Tadonki, "Performance comparison between Hadoop and spark frameworks using HiBench benchmarks," *Concurrency Comput.: Pract. Experience*, vol. 30, no. 12, 2018, Art. no. e4367.

Hongjian Li received the PhD degree in computer software and theory from the University of Electronic Science and Technology of China, in 2012. He is an associate professor of the Chongqing University of Posts and Telecommunications, China. His research interests include cloud computing, big data analysis, and parallel computing.



Jianglin Xia is currently working toward the master degree with the School of Computer Science, Chongqing University of Posts and Telecommunications, China, under the supervision of Dr. Li. His research interests include in big data analysis.



Wei Luo is currently working toward the graduate degree in computer science and technology with the Chongqing University of Posts and Telecommunications, China, Under the guidance of Dr. Hongjian Li. His research interests include cost effective optimization of big data platforms and data analytics.

Hai Fang is currently working toward the master degree with the School of Computer Science, Chongqing University of Posts and Telecommunications, China, under the supervision of Dr. Li. His research interests include energy efficiency in big data analysis.