Distributed and Parallel Databases (2022) 40:135–163
https://doi.org/10.1007/s10619-021-07375-6

HTD: heterogeneous throughput-driven task scheduling algorithm in MapReduce

Xite Wang[1] · Chaojin Wang[1] · Mei Bai[1] · Qian Ma[1] · Guanyu Li[1]

Accepted: 1 October 2021 / Published online: 28 October 2021

Abstract
As one of the most popular parallel data processing models, data analysis system MapReduce has been widely used in many fields. Task scheduling is the core module in MapReduce system, and the quality of the scheduling algorithm directly affects the processing capacity of the system. Since new nodes need to be continuously added in the cluster to improve the processing capacity of the cluster, objectively, the heterogeneity of the cluster is caused. Heterogeneous environment is common in practical application scenarios, but there has been little research on task scheduling in heterogeneous environment. For this reason, this paper presents an in-depth study of task scheduling in heterogeneous environment and proposes a new task scheduling algorithm HTD. First, we give a formal definition of the throughput-driven task scheduling problem in a heterogeneous environment. Second, we design the scheduling algorithm HTD, which quickly obtains the completion sequence of a jobs set and optimizes the task scheduling details in heterogeneous environment. Finally, a series of experiments show the efficiency and effectiveness of the algorithm.

Keywords MapReduce · Scheduling · Heterogeneous · Throughput

1 Introduction

Increasingly a huge amount of data has been produced by using the internet. How to effectively store and process massive data has become a hot issue in the field of data management. MapReduce [1] has become one of the most effective tools for big data analysis due to its strong availability, high scalability and many other advantages. MapReduce framework uses a master-slave architecture. The master node is responsible for cluster management and task scheduling. For any given MapReduce job, the master node splits it into multiple map tasks and reduce

✉ Xite Wang
xite-skywalker@163.com

[1] Information Science and Technology College, Dalian Maritime University, Dalian, China

tasks, and distributes them to idle slave nodes for processing. And the master node monitors the progress of the task. Each slave node has a certain number of map and reduce task slots for processing the map and reduce tasks assigned by the master node. The slave node also periodically uses the heartbeat mechanism to report the execution status of the task to the master node.

Task scheduling is the core module of MapReduce systems, and the quality of the scheduling algorithm directly affects the processing capacity of the system. Some studies [2–4] have been done on MapReduce task scheduling, but most of the research results are still in their infancy and do not provide in-depth analysis of specific application scenarios, thus failing to meet the actual needs of users. Additionally, many IT companies have built their own MapReduce clusters to handle their internal data analysis operations. In 2005, the Apache Foundation announced the Hadoop system [5], an open source implementation of the MapReduce parallel processing framework, which laid the foundation for later work. At present, many institutions have launched derivative products based on MapReduce, which greatly extend the functions of the MapReduce system, such as the SQL query processing framework Hive based on MapReduce [6], Hadoop-EDF, a distributed signal processing tool to load European Data Format(EDF) data based on Hadoop MapReduce [7]. With the company's own growth and rising business volume, the existing clusters are limited in size and their performance is no longer sufficient to meet the business needs. As a result, new nodes need to be added constantly to increase the processing power of the cluster. The hardware configuration of the new nodes is often superior to the old nodes, which makes the coexistence of multiple batches of nodes in the cluster. And this objectively results in the heterogeneity of the cluster.

In recent years Tiwari et al. [8], Benifa et al. [9] and Jiang et al. [10] gave studies on MapReduce systems and their scheduling algorithms. These studies gave scheduling algorithms for MapReduce for different problems and discussed their drawbacks. In addition, the study of task scheduling in heterogeneous environments is also a focal point of current research. For the situation that each node has different computing capabilities, how to determine the order of jobs execution and how to efficiently allocate jobs to each node are the key issues of research. Chen et al. [3] studied scheduling tasks with deadline constraints in heterogeneous environments by converting the MapReduce scheduling problem into a graph problem and then solving it using heuristic algorithm Ant. Ahmad et al. [11] and Hsieh et al. [12] improved the matching of resources and jobs by improving the MapReduce performance. Cheng et al. [13] proposed an adaptive task tuning method to improve the performance of MapReduce clusters in heterogeneous environments. Rasooli et al. [14] proposed a new scheduling method to improve the average completion time of jobs. Although there have been some studies on scheduling tasks in heterogeneous environment, they do not delve into the characteristics of MapReduce systems. Our paper addresses the scheduling characteristics of MapReduce and proposes the HTD algorithm.

In this paper, primarily for heterogeneous MapReduce clustered environment, we study the problem of task scheduling with throughput as the target. The contributions of this paper are as follows:

(1) A formal definition of the throughput-driven scheduling problem is given in a heterogeneous MapReduce environment.
(2) We propose a novel strategy, Heterogeneous Throughput-Driven task scheduling algorithm in MapReduce (HTD). The algorithm includes three main parts. First, for a single job, a task assignment scheme is designed in a heterogeneous environment, and the job parameters are estimated. Secondly, for a job set, the parameters estimate of each job are used to generate the optimal execution sequence of the jobs. Finally, the sequence is brought into a heterogeneous environment to further optimize the specific details of task assignment and obtain the final task execution scheme.
(3) We prove the efficiency and effectiveness of the HTD through a series of experiments.

Section 2 introduces an overview of MapReduce and related research of scheduling algorithm. Section 3 proposes a formal definition of the throughput-driven scheduling problem in a heterogeneous MapReduce environment. Section 4 describes the method for estimating the job parameters and the details of HTD in Sect. 4. Finally, Sect. 5 contains the experimental result of HTD and the summarizes the content of the paper.

2 Background knowledge

We mainly introduce the framework of MapReduce and the related work of task scheduling in MapReduce.

2.1 Framework of MapReduce

The job that MapReduce can handle is composed of map phase and reduce phase. In the map phase, the huge input file (usually stored in the distributed file system HDFS) is first parsed into a set of key-value pairs. The required data (key-value pairs) is extracted from them to form intermediate results. The reduce phase must be executed after the map phase is completed. The goal is to integrate the intermediate results obtained in the map phase and output the final result in the form of key-value pairs in reduce phase.

Figure 1 describes the general flow of a MapReduce job execution. When the user wants to carry out a data analysis work, he first needs to write the corresponding map and reduce functions according to his own requirements to form a job j.

After j is submitted to the system, the master node will divide it into m map tasks and r reduce tasks, and assign them to slave nodes for processing. Usually m is the number of fragments contained in the HDFS of the input file of j, and r is specified by the user or is the default value. In the map phase, a fragment corresponds to a map task. The data in the fragment will be assigned to a specific map task slot (such as M_1) and parsed into a series of key-value pairs in (k_1, v_1) format. Then, each map task slot calls the map function submitted by the user, and outputs a series of

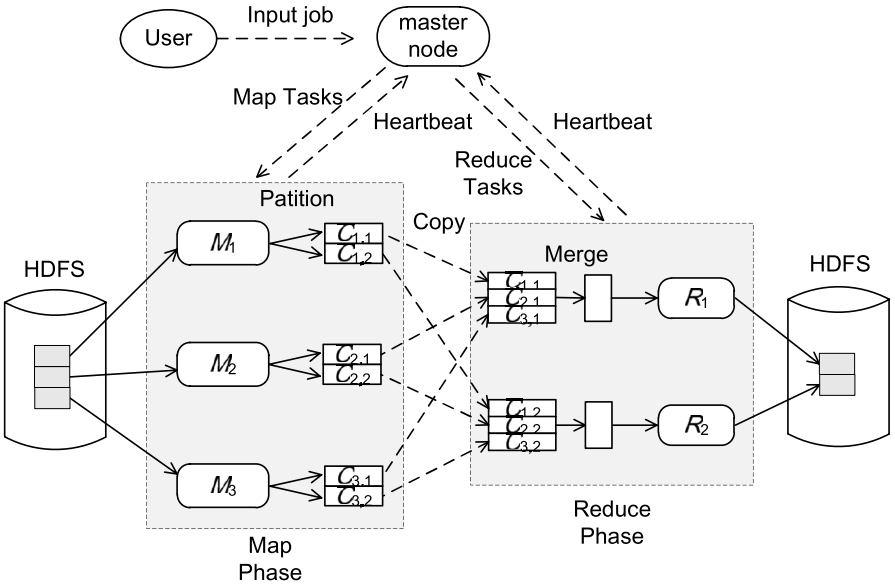


Fig. 1 MapReduce framework

key-value pair in (k_2, v_2) format as intermediate results. The intermediate results produced by each map task will be divided into r blocks and sent to the corresponding reduce task slots. The partition method usually uses a hash function to ensure that key-value pairs with the same key are divided into the same block.

As shown in Fig. 1, the job contains two reduce tasks, so the intermediate result produced by M_1 is divided into two blocks, where $C_{1,1}$ is sent to R_1, and $C_{1,2}$ is sent to R_2. Similarly, the intermediate result $C_{2,1}$ produced by M_2 is sent to R_1, $C_{2,2}$ is sent to R_2, and so on. In the reduce phase, first, each reduce task slot (such as R_1) merges the received blocks together to form a series of key-value pairs in $(k_2, list(v_2))$ format.

In order to achieve high availability in a large-scale cluster, MapReduce adopts a dynamic task assignment model. For a new job, the scheduler first appends its map/reduce tasks to the map/reduce task sequences. Instead of assigning all the tasks to work nodes in advance, the master node waits the work node to apply. Each time a work node becomes idle, it will apply a new task. At this time, the master chooses a task from the sequence and allocate it to the work node.

2.2 Related work

In the past few years, a lot of research work has been done to solve scheduling problems in different situations. Aiming at the heterogeneity of cluster nodes in processing power and storage capacity, Bellatreche et al. [15] proposed the Fragmentation and Allocation algorithm, a cost model that considers both fragmentation and allocation parameters for performing fragmentation and allocation phase, respectively. At the fragmentation phase, the allocation phase/decision task is completed

accordung to the Fragmentation and Allocation algorithm. Over heterogeneous database clusters, the main Parallel Rational Data Warehouses (PRDW) design problems are solved by Hill Climbing and Genetic Algorithm. At the allocation phase, an innovative matrix-based formalism is introduced, which can capture the interaction between fragments, input queries, and cluster node characteristics, thereby driving the corresponding data allocation tasks. Buffer management is often ignored, in query optimization techniques such as materialized views. In order to avoid its influence, Kerkad et al. [16] combined the query scheduling problem with the buffer management problem, and proposed an inspiring beehive algorithm. Considering the impreciseness of query execution time, a cost-aware scheduling algorithm Shepherd [17] is proposed. It uses a histogram with query execution time distribution as a means to query whether the scheduler input is valid. At the same time, job scheduling is also an important technology to improve the overall performance of distributed systems. In cloud computing, the Cost-based job scheduling (CJS) algorithm [18] considers two types of jobs at the same time, takes the characteristics of the job as the main basis for scheduling decisions, and considers data and computing requirements at the same time. For large-scale computing systems, a duplication-based compile-time task scheduling heuristic algorithms are usually used, but relying on duplication will consume additional energy [19]. Therefore, a green mechanism applied to any scheduling is proposed to reduce energy consumption while maintaining the same application completion time.

Task scheduling for MapReduce has also received extensive attention, and various scheduling methods have been proposed for different situations. The default first in first out (FIFO) strategy of the Hadoop system is the most basic and effective scheduling strategy. FIFO strategy organizes all the jobs received by the system into a queue. The jobs in the queue are sorted according to priority, and then sorted according to the order of submission. Late submissions are ranked behind. During the execution of the MapReduce, if a task is executed too slowly, the system will judge it as a straggler, and the scheduler will re-select another node to execute the task, that is, the backup operation. Zaharia et al. [20] pointed out that the default backup operation of MapReduce often fails to improve the efficiency of calculation in a heterogeneous environment, and therefore proposed the longest approximate time to end (LATE) scheduler. The algorithm first improved the method of judging straggler in MapReduce. After that, the performance of each child node and the processing speed of each task are estimated, and the slower tasks are backed up on nodes with better performance, which significantly improves the performance of the heterogeneous MapReduce cluster.

For task scheduling in a heterogeneous MapReduce cluster environment, many scholars have conducted research and proposed different scheduling strategies. Kwon et al. [21, 22] analyzed the skew problem [23–25] in the task execution process for the heterogeneous MapReduce cluster environment, and proposed the Skewtune algorithm. Skew means that the processing time of multiple map (or reduce) tasks of a job is very different, usually caused by heterogeneous clusters and uneven data distribution. Analysis of queries often ignores buffer management and assumes a fixed order and are known in advance. However, these assumptions are too strong. In order to overcome this problem, the basic idea is to perform a split operation at

an suitable point in time, and redistribute part of the data in the long task to the node where the short task is located for processing. Pericini et al. [26] proposed the Bron-Kerbosch-Clique algorithm in combination with Simulated Annealing, Local Beam Search and Stochastic Beam Search. Wang et al. [27] analyzed the reasons for poor computing capability in a heterogeneous MapReduce cluster environment, and proposed the ActCap algorithm to dynamically determine the computing capability of cluster nodes and allocate data to reduce network overhead. Wang et al. [28] improved the random task sequence into a sorted task sequence, that is, using improved heartbeats and task deadlines to improve the efficiency of job execution between geographically distant data centers. Hsieh et al. [12] proposed a job scheduler called Job Allocation Scheduler (JAS), which aims to balance resource utilization. For various task workloads, JAS classifies the tasks and then assigns the tasks to the CPU-bound queue or I/O-bound queue. For heterogeneous environments, the use of nodes is improved and a dynamic job allocation scheduler with locality is created. Wang et al. [29] designed a predictive model to predict the end time of the task, which is used to allocate the corresponding data to the corresponding nodes in advance to reduce time consumption. On the basis of this prediction model, a scheduling algorithm based on task matching is proposed. The tasks in the queue are scheduled by considering the situation of each node in the cluster. Chen et al. [30] use rack as the basic unit of resources to design multi-job pre-mapping algorithm to optimize the execution order of jobs.

In summary, the existing work still cannot fully solve the throughput-driven scheduling problem in a heterogeneous MapReduce environment, so we propose an effective algorithm to solve this problem.

3 Preliminary

3.1 Problem definition

In this paper, we focus on the problem of task scheduling with throughput as the target for heterogeneous MapReduce cluster environments. First, for ease of description, Table 1 describes the notation used in this chapter.

We are given a MapReduce cluster $N = \{N_1, N_2, \ldots, N_{|N|}\}$ for a heterogeneous environment, which includes $|N|$ nodes. For each node N_i, we denote N_i. cost as the computing capability of N_i, $ration_{i,j}$ as ratio of time spent by nodes N_i and N_j on the same task. In this paper, we select standardized nodes N_1, so the computing capability of $N_1.cost = 1$. Simply, if a map task for j_i needs s seconds on N_1 and ratio of time spent by nodes N_i and N_1 is $ration_{i,1}$, the time for N_i to process this task is $ration_{i,1} \times s$ s. The parameter estimation method is given is Sect. 3.2.

The set of jobs in a cluster is denoted by $J = \{j_1, j_2, \ldots, j_{|J|}\}$ which includes $|J|$ stand-alone jobs. For each j_i, $j_i.M_{num}$ is the number of map tasks for j_i and $j_i.R_{num}$ is the number of reduce tasks for j_i. The time spent on a map task for j_i at N_m is defined as $j_i.T^m_{map}$ and the time spent on a reduce task for j_i at N_m is defined as $j_i.T^m_{reduce}$.

Table 1 Summary of notations

Symbol	Description
N	The heterogeneous MapReduce cluster
N_i	The i^{th} node in a cluster
$N_i.cost$	Computing capability of N_i
$ration_{i,j}$	Ratio of time spent by nodes N_i and N_j on the same task
J	A job set
j_i	The i^{th} job
$j_i.M_{num}$	Number of map tasks for j_i
$j_i.R_{num}$	Number of reduce tasks for j_i
$j_i.T^m_{map}$	Time spent on a map task for j_i at N_m
$j_i.T^m_{reduce}$	Time spent on a reduce task for j_i at N_m

Given a heterogeneous cluster $N = \{N_1, N_2, \dots, N_{|N|}\}$ and a set of jobs $J = \{j_1, j_2, \dots, j_{|J|}\}$, we aim to complete all jobs for J in the shortest possible time by setting up a reasonable scheduling algorithm.

Intuitively, in order to complete all jobs for J as quickly as possible and increase the throughput of the system, the scheduling algorithm should make full use of the computation resources in the system and reduce resource idleness. In the following text, we use an example to elaborate on the problem. As shown in Fig. 2, given a cluster $N = \{N_1, N_2, \dots, N_{|N|}\}$, we assume that the computing capability is known, $N_1.cost = 1$, $N_2.cost = 1.3$, $N_3.cost = 0.8$, $N_4.cost = 1.5$. For each job, the following parameters are known. (1) The number of map and reduce tasks. (2) Time spent on a map and reduce task at N_1. If a simple FIFO scheduling strategy is applied, all jobs are executed in the original order. Job j_1 is processed first. After about 42 s, the map phase ends.The numbers of the map tasks executed on node N_1, N_2, N_3, N_4 are 14, 10, 17, 9. Then, j_1 goes to the reduce phase and the map phase of j_2 begins. In this order, it is expected that all jobs can be completed in 140.2 s at the earliest. At this point, there are substantial resource constraints and the processing capacity of the system is not fully utilized. Conversely, if the order of job execution is re-rationalized and the characteristics of heterogeneous clusters are fully considered, idle resource can be greatly reduced and processing efficiency improved. As shown in Fig. 2, if the scheduling method in this paper is followed, we change the job sequence to j_3, j_4, j_2, j_1. The tasks can be reasonably assigned based on the computing capability of each node. With the adjustments, all jobs can be completed in 107.2 s. It is easy to observe that HTD reduces the idle time of the system resources and increases the throughput.

3.2 Parameter estimation

Based on the problem description above, it is clear that two parameters need to be estimated. (1) the computing capability of each node in the cluster, (2) the

$J = \{j_1, j_2, j_3, j_4\}$

$N = \{N_1, N_2, N_3, N_4\}$

$N_1.cost = 1$

$N_2.cost = 1.3$

$N_3.cost = 0.8$

$N_4.cost = 1.5$

	$j_i.M_{num}$	$j_i.R_{num}$	$j_i.T^1_{map}$	$j_i.T^1_{reduce}$
j_1	50	40	3s	1s
j_2	30	20	4s	2s
j_3	20	30	2s	3s
j_4	10	40	4s	2s

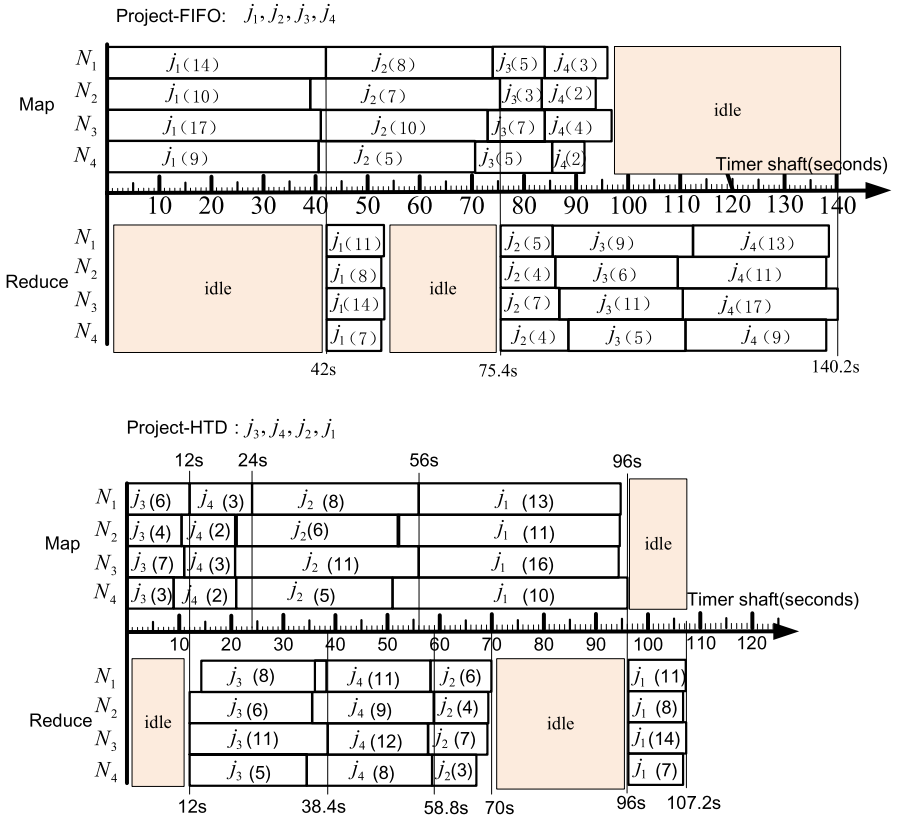


Fig. 2 Example of MapReduce throughout-driven scheduling on heterogeneous environment

processing time of a map/reduce task for each job. The basic estimation method is described as follows.

We randomly select a set *ST* of tasks to test the computing capability of nodes. For each task *t*, we record the corresponding execution time on node *N* as et^t_N. The average execution time for node *N*, $aet(N) = \frac{\sum_{t \in ST} et^t_N}{|ST|}$. Then, For node N_i, its computing capability with respect to the standardized node N_1 is $N_i.cost = aet(N_i)/aet(N_1)$. For example, assume that these is a cluster with three nodes $N = \{N_1, N_2, N_3\}$, and several test tasks are randomly selected. We run these tasks on N_1, N_2, N_3 and compute the related average execution time 2.3 s, 2.7 s, 1.8 s separately. Then the computing capability is $N_1.cost = 1$, $N_2.cost = 2.7/2.3 = 1.174$, $N_3.cost = 0.8$.

For a MapReduce job j waiting to be executed, we use history reocords and sampling method to estimate the processing time. First, we check the system processing history and look for the jobs with the same type of j. Using these similar jobs, we compute the average processing time of map (reduce) tasks on node N_1 as $j.T\_his^1_{map}$ ($j.T\_his^1_{reduce}$). Second, we randomly extract a small fraction of input data to form a simplified version of j (consisting of only one map and one reduce task). We run the simplified job on node N_1. By keeping track of the processing time for the map and reduce phases, we can compute the average processing time $j.T\_samp^1_{map}$, $j.T\_samp^1_{reduce}$. Then, Combining the history value and the sampling value, we estimate $j_i.T^1_{map} = \alpha \cdot j.T\_his^1_{map} + (1-\alpha) \cdot j.T\_samp^1_{map}$, $j_i.T^1_{reduce} = \alpha \cdot j.T\_his^1_{reduce} + (1-\alpha) \cdot j.T\_samp^1_{reduce}$.

Based on the above parameters, according to Eq. 1, we obtain the processing time for map and reduce tasks on node N_m for job j_i.

$$\begin{cases} j_i.T^m_{map} = N_m.cost \times j_i.T^1_{map} \\ j_i.T^m_{reduce} = N_m.cost \times j_i.T^1_{reduce} \end{cases} \tag{1}$$

Note that we only use these estimated values to generate the initial scheduling plan. As the processing continues, the parameter can be re-computed and the scheduling plan can be adjusted.

4 HTD algorithm description

In this section, we describe the detail of HTD. First, for a single job, we designed a task scheduling scheme for a heterogeneous environment, and the parameters of the jobs are estimated. Second, for a set of jobs, we generate an optimal execution sequence of the jobs using the parameter estimates of the jobs. Finally, the sequence is brought into the heterogeneous environment to further optimize the details of the task assignment and obtain the final task execution scheme.

We are given a MapReduce cluster $N = \{N_1, N_2, \ldots, N_{|N|}\}$ for a heterogeneous environment and a set of jobs $J = \{j_1, j_2, \ldots, j_{|J|}\}$. The goal of this section is to design a scheduling scheme that allows all jobs to be completed in the shortest possible time. First, we design a scheduling scheme for a single job, which ensures that a single job can be completed in the shortest possible time. Then, by setting up a reasonable scheduling order, all the jobs in the job set can be completed in the shortest possible time.

4.1 Scheduling scheme for a single job

Given a MapReduce job j, it consists of multiple map and reduce tasks. To complete j quickly, the map and reduce tasks for job j need to be completed in the shortest possible time. In the methods in this section, the map and reduce tasks have the same processing strategy, so only the map task is presented as an example. A formal definition of the problem is given below first.

We are given a MapReduce cluster $N = \{N_1, N_2, \ldots, N_{|N|}\}$ for a heterogeneous environment, the computing capability on any node and the number of maps $j.M_{num}$ for job j. We assume that Num_i map tasks are assigned for job j on each node N_i, $\sum_{i=1}^{|N|} Num_i = j.M_{num}$. All map tasks for j are cost time $obj(N,j) = \max_{i\in[1,|N|]}\{N_i.cost \times Num_i \times j.T^1_{map}\}$. The goal of this section is to design the allocation method in such a way that the value $obj(N, j)$ is minimized. In other words, all nodes in the cluster complete their tasks in the same time as possible.

In a homogeneous environment, it is only necessary to evenly distribute the map tasks to each node to ensure that the tasks on all nodes are completed in approximately the same time. But in a heterogeneous environment, this even distribution scheme is obviously unreasonable. Therefore, the following theorems are proposed.

Theorem 1 *Given two heterogeneous node N_1 and N_2, the number of map tasks for job j assigned on N_i is Num_i and the number of map tasks for job j assigned on N_j is Num_j. If $Num_i/Num_j = N_j.cost/N_i.cost$, the node N_i and N_j can both accomplish their tasks.*

Proof The map task for job j cost time $N_i.cost \times Num_i \times j.T^1_{map}$ on node N_i and the map task for job j cost time $N_i.cost \times Num_i \times j.T^1_{map}$ on node N_j. Based on $Num_i/Num_j = N_j.cost/N_i.cost$, the time cost on N_i and N_j is the same. □

According to Theorem 1, all map tasks for job j can be distributed among the nodes of a cluster in such a way that all nodes accomplish their respective map tasks as simultaneously as possible.

Suppose that the number of map task for job j is Num_1 on node N_1, then by Theorem 1 the number of map tasks assigned to N_i is $Num_1/N_1.cost$. Due to $\sum_{i=1}^{|N|} Num_i = j.M_{num}$, we obtain $\sum_{i=1}^{|N|} 1/N_i.cost \times Num_1 = j.M_{num}$. We can calculate the number of map tasks $Num_1 = \lfloor j.M_{num}/\sum_{i=1}^{|N|} 1/N_i.cost \rfloor$. For example, on the example in Fig. 2, the capabilities of the four nodes are $N_1.cost = 1$, $N_2.cost = 1.3$, $N_3.cost = 0.8$ and $N_4.cost = 1.5$. The number of map tasks is $j_1.M_{num} = 50$ for job j_1. Based on the Equation above, we obtain $Num_1 = \lfloor 50/(1 + 1/1.3 + 1/0.8 + 1/1.5) \rfloor = 13$, $Num_2 = \lfloor 13/1.3 \rfloor = 10$, $Num_3 = 16$ and $Num_4 = 8$.

When the number of assignments for a job is not exactly proportional to the number of nodes, since the above calculations take lower bounds, the sum of the number of assignments on each node can only be less than or equal to the number of assignments for the job.

For the remaining unassigned tasks, we select the node that minimizes the objective function $obj(N, j)$ and assign it. Continuing with the above example, $Num_1 = 13$, $Num_2 = 10$, $Num_3 = 16$, $Num_4 = 8$ and $j_1.M_{num} = 50$, there are three map tasks that have not been assigned for job j_1. Based on the assigned strategy, the cost time on N_1 is $j.T^1_{map} \times N_i.cost \times 13 = 39s$ and the cost time on N_2, N_3 and N_4 is 39 s, 28.4 s and 36 s. The current objective function is

$\max\{39, 39, 38.4, 36\} = 39s$. At this point, the three remaining tasks have not been assigned. The first remaining task is processed and the new objective function is found to be the smallest 40.5 s after assigning it to node N_4. Then the remaining two tasks are processed sequentially and assigned to node N_3 and N_1. Finally, we can calculate that $Num_1 = 14$, $Num_2 = 10$, $Num_3 = 17$ and $Num_4 = 9$. The objective function is 42 s. Arranging the above methods, we can get Algorithm 1. In Algorithm 1, first, we initially assign a certain number of tasks to each node (lines 1–5). Since the initialization phase is a lower bound value for each acquisition, the number of tasks assigned to each node during the initialization phase must be less than or equal to the number of tasks on the node. It is therefore necessary to further allocate the remaining unassigned tasks (lines 6–18), and to ensure that the allocated objective function $obj(N, j)$ is minimized so that all tasks of the job j are completed in the shortest possible time. We will see that the number of unassigned tasks remains $O(|N|)$ after initialization, and a maximum of $|N|$ attempts are required for each task. We obtain that the time complexity required for this step is $O(|N|^2)$.

Algorithm 1 Single Job Processing

Input: a heterogeneous cluster $N = \{N_1, N_2, \ldots, N_{|N|}\}$, the computing capability $N_i.cost$ on each node, the job j, and the number of map tasks $j.M_{num}$ for j
Output: the number of map tasks Num_i for the job j on each node N_i

1: $Num_1 = \lfloor j.M_{num} / \sum_{i=1}^{|N|} 1/N_i.cost \rfloor$
2: $obj(N, j) \leftarrow 0$
3: **for** each node N_i in the cluster do **do**
4: $\quad Num_i = \lfloor Num_1 / N_i.cost \rfloor$
5: **end for**
6: remaining number of unassigned jobs $RNum = j.M_{num} - \sum_{i=1}^{|N|} Num_i$
7: **for** each remaining unassigned task T_j do **do**
8: $\quad Time \leftarrow \infty$, $TNode \leftarrow \emptyset$
9: $\quad$ **for** each node N_i in cluster N do **do**
10: $\quad\quad$ assigning a task to N_i
11: $\quad\quad obj(N, j) = \max_{i \in [1, |N|]} \{N_i.cost \times Num_i \times j.T_{map}^1\}$
12: $\quad\quad$ **if** $obj(N, j) < Time$ **then**
13: $\quad\quad\quad TNode = N_i$, $Time = obj(N, j)$
14: $\quad\quad$ **end if**
15: $\quad$ **end for**
16: $\quad$ assign this remaining job T_j to node $TNode$
17: $\quad$ the number of assignment job to TNode $Num_{Tnode} += 1$
18: **end for**

Algorithm 1 can be used to estimate the minimum processing time $j.TC_{map}$ (or $j.TC_{reduce}$) for the map (or reduce) phase of an arbitrary job j in a heterogeneous cluster. In the example shown in Fig. 1, the number of map tasks for job j_1 is $j_1.M_{num} = 50$. The number of map tasks for j_1 that should be assigned on the four nodes are 14, 10, 17, and 9 respectively. The objective function is 42 s, which means that the map phase for j_1 costs $j_1.TC_{map} = 42s$. The number of reduce tasks for j_1 is $j_1.M_{num} = 40$. The number of reduce tasks for j_1 that should be assigned on the four nodes are 11, 8, 14, and 7 respectively. The objective function is 11.2

s, which means that the map phase for j_1 costs $j_1.TC_{reduce} = 11.2s$. Then, the total time for j_1 cost $42 + 11.2 = 52.3s$.

4.2 Scheduling scheme for multi-jobs

Based on the results of the analysis in the previous section, you can find the shortest possible time spent on a heterogeneous cluster for the map and reduce phases of each job. In this section, for a set of jobs in a cluster, we use the estimated time to overall scheduling. Table 2 shows the time cost of the map and reduce phases for each job in Fig. 1.

In general, multiple jobs executing on a cluster can be divided into two scheduling methods. One is shared scheduling, where multiple jobs are processed simultaneously in the cluster (such as Fair Scheduler). And the other is a map (or reduce) task only processes one job at the same time in the cluster (such as FIFO Scheduler), which is called exclusive scheduling in this section. In this section, we adopt the exclusive scheduling approach. The following provides the necessary definitions and then describes the advantages of exclusive scheduling over shared scheduling.

Definition 1 (Sequence) For a set of job J (the number of jobs is recorded as $|J|$), the sequence Seq is an ordered list of the jobs in J, which restricts the scheduling order of all the jobs in the set.

Definition 2 (Optimal sequence) The sequence that can complete all the jobs in J the fastest is called the optimal sequence of all sequence in the job set J.

Theorem 2 *Given two jobs j_m and j_n, note that the time t is consumed to complete all tasks for jobs j_m and j_n using the exclusive scheduling method and according to the optimal sequence. Then the time t must be less than or equal to the time required to complete and under any shared scheduling method.*

Proof Suppose the optimal sequence j_m, j_n. 1) If $j_m.TC_{map} \geq j_n.TC_{reduce}$, the time cost to complete these two jobs using the exclusive scheduling method $t = j_n.TC_{reduce} + j_m.TC_{map} + j_m.TC_{reduce}$. It is easy to find that if a shared scheduling method is used, the time cost cannot be less than this value. 2) If $j_m.TC_{map} < j_n.TC_{reduce}$, the time cost using the exclusive scheduling method

Table 2 The estimated completion time of jobs using Algorithm 1

jobs	j_1	j_2	j_3	j_4	Jobs	j_1	j_2	j_3	j_4
$j_1.TC_{map}$	42 s	35.2 s	12 s	12 s	$j_1.TC_{reduce}$	11.2 s	12 s	26.4 s	22.4 s
Num_1^{map}	14	8	6	3	Num_1^{reduce}	11	6	8	11
Num_2^{map}	10	6	4	2	Num_2^{reduce}	8	4	6	8
Num_3^{map}	17	11	7	3	Num_3^{reduce}	14	7	11	14
Num_4^{map}	9	5	3	2	Num_4^{reduce}	7	3	5	7

$t = j_n.TC_{map} + j_n.TC_{reduce} + j_m.TC_{reduce}$. If the shared scheduling strategy is used, the time to complete the map phase must be greater than $j_n.TC_{map}$. Therefore, the overall time cost must be greater than the time cost to complete the map phase. □

Corollary 1 *Given a set of jobs $J = \{j_1, j_2, \ldots, j_{|J|}\}$, note that the time t is consumed to complete all tasks in J using the exclusive scheduling strategy and according to the optimal sequence. Then the time t must be less than or equal to the time required to complete and under any shared scheduling strategy.*

Based on Corollary 1, in the system with throughput as the goal, the exclusive scheduling strategy is better than the shared scheduling, and the job set can be completed in a shorter time. In addition, it is easy to find that the completion time of the job set varies when using different sequences. Here is how to calculate the completion time of a job set when using different sequences.

Given a sequence $Seq_1 = \{j_1, j_2, \ldots, j_{|J|}\}$ of a job set, we define job j_i at position i in the sequence. All map tasks for j_i can be completed at time point $\sum_{m-1}^{i} .TC_{map}$. Then the reduce task for j_i can be only start after the time point $\sum_{m-1}^{i} .TC_{map}$. If it is known that reduce task completes at time point $j_{i-1}.ComReduce$ for j_{i-1}, the reduce task for j_i can be start execution at time point $j_i.ComReduce = \max\{\sum_{m-1}^{i} j_m.TC_{map}, j_{i-1}.ComReduce\}$. Then the reduce task for j_i can be completed at time point $j_i.ComReduce = j_i.startReduce + j_i.TC_{reduce}$. According to the above introduction, it is possible to quickly get the completion time of each job's reduce task when a set of execution sequences is identified. And the last job's reduce completion time is the final completion time of all jobs in the set. The following Eq. 2 can be used to find the completion time of the reduce task for each job in the sequence $Seq = \{j_1, j_2, \ldots, j_{|J|}\}$.

$$\begin{cases} j_1.ComReduce = j_1.TC_{map} + j_1.TC_{reduce} \\ j_2.ComReduce = \max\{\sum_{m=1}^{i} j_m.TC_{map}, j_1.ComReduce\} + j_2.TC_{reduce} \\ \cdots \\ j_i.ComReduce = \max\{\sum_{m=1}^{i} j_m.TC_{map}, j_{i-1}.ComReduce\} + j_i.TC_{reduce} \\ \cdots \\ j_{|J|}.ComReduce = \max\{\sum_{m=1}^{|J|} j_m.TC_{map}, j_{|J|-1}.ComReduce\} + j_{|J|}.TC_{reduce} \end{cases} \tag{2}$$

Figure 3 depicts the execution of the jobs in Table 2 when using different sequences. For example, the completion sequence $Seq = \{j_1, j_2, \ldots, j_{|J|}\}$ of four jobs specified in Scheme 1. All map tasks of j_1 are finished in 42 s and starts to enter the reduce phase with the reduce completion time of 53.2 s. However, when the map phase of j_4 completes at 101.2 s, the reduce task is still incomplete. Therefore, the reduce phase of j_4 needs to wait until the reduce phase of j_3 completes at 115.6 s. Finally, the completion time of j_4 is 138 s. By comparing the two different scenarios in Fig. 2, it can be seen that the final completion time of the map task for all four jobs is 101.2 s regardless of the scheduling order set, but the final completion time for reduce is different.

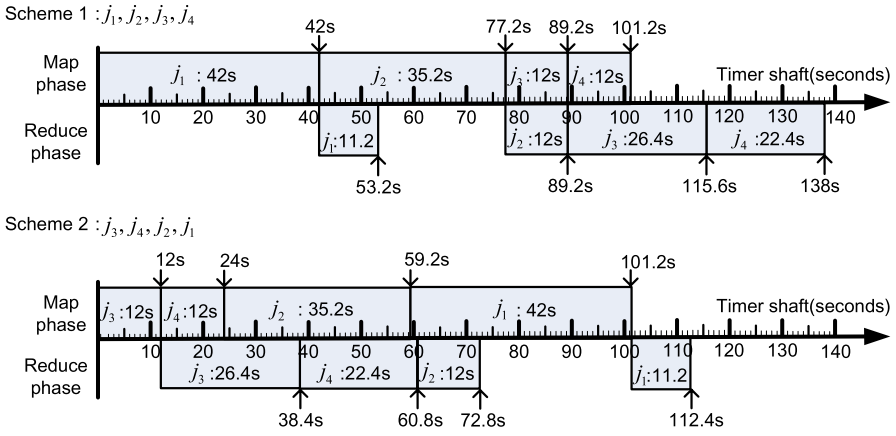


Fig. 3 The completion time of jobs for different scheduling orders

Using Eq. 2, the final completion time of a given sequence can be quickly obtained. However, given a set of $|J|$ jobs, the set has a total of $|J|! = \pi_{i=1}^{|J|} i$ sequences. Obviously, it is unreasonable to select the optimal sequence by enumerating all the sequences. Several effective methods for obtaining optimal sequences quickly are described below.

Given a set of jobs $J = \{j_1, j_2, \ldots, j_{|J|}\}$, we need to get the following statistics: the total time $J.MapCost = \sum_{i=1}^{|J|} j_i.TC_{map}$ in the map phase and the total time $J.MapCost = \sum_{i=1}^{|J|} j_i.TC_{map}$ in the reduce phase.

Theorem 3 *Given a set of jobs* $J = \{j_1, j_2, \ldots, j_{|J|}\}$ *and a sequence* $Seq = \{j'_1, j'_2, \ldots, j'_{|J|}\}$, *the sequence is optimal if the sequence ensures that the time spent on the map phase of the jobs is in ascending order, and the time spent on the reduce phase is exactly in descending order.*

Proof Supposing that after the last job's map tasks are completed and the reduce tasks of all previous jobs are completed, then the time to complete all jobs is $J.MapCost + j'_{|J|}.TC_{reduce}$ in the sequence. The sequence *Seq* is optimal because the reduce time of $j'_{|J|}$ is the shortest. Conversely, after the last job's map task is completed, there are still the reduce tasks of the previous jobs that have not been completed, there will be no free time from the start of the reduce task, since the map phase is in ascending order, and the reduce phase is in descending order. Therefor the time to complete all jobs in the sequence is $J.ReduceCost + j'_1.TC_{map}$. Since the map phase of j'_1 takes the shortest possible time, the sequence *Seq* must be the optimal sequence. This theorem is proven. □

From Theorem 3, then we can directly determine the optimal sequence when the map task time is in ascending order and the reduce task time is exactly in descending order. However, when the jobs do not satisfy this feature, the following conclusion

can be drawn from observing Fig. 2. The task completion time of the map phase is the same regardless of the job sequence used. The completion time of the sequence depends on the idle time of the reduce phase. Therefore, this section goes through the following three steps to find the optimal sequence for any set.

Step 1 Simplify the original set of jobs. We use the merge strategy to merge the corresponding jobs, thereby reducing the number of jobs to be calculated and obtaining a simplified job set.

Step 2 The optimal sequence of the set of simplified jobs is obtained and the sequence is taken as a qualifying sequence with the completion time of the sequence as a bound value.

Step 3 Using the pruning scheme, we verify all sequences whose completion time may be less than the bound value until the final optimal sequence is found.

(1) Simplify the original set of jobs.

This step reduces the number of jobs to be calculated by merging the appropriate jobs to obtain a simplified job set. Given two jobs j_m and j_n, the following conditions need to be met before they can be combined. (1) If $j_m.TC_{map} < j_n.TC_{map}$ and $j_m.TC_{reduce} > j_n.TC_{map}$, we can merge the two jobs in order j_m,j_n and get a new job $j_{m,n}$. The map and reduce completion times for the new job are $j_{m,n}.TC_{map} = j_m.TC_{map}$ and $j_{m,n}.TC_{reduce} = j_m.TC_{reduce} + j_n.TC_{reduce} - j_n.TC_{map}$. (2) If $j_m.TC_{map} < j_n.TC_{map}$, $j_m.TC_{reduce} < j_n.TC_{map}$ and $j_n.TC_{reduce} < j_m.TC_{map}$ and the completion time of two jobs j_n,j_m in the sequence is less than the completion time of j_m,j_n, we merge the two jobs in the sequence j_n,j_m and get the job $j_{n,m}$. We have $j_{n,m}.TC_{map} = j_n.TC_{map}$ and $j_{n,m}.TC_{reduce} = j_n.TC_{reduce} + j_m.TC_{reduce} - j_m.TC_{map}$. These two merge conditions guarantee that the merged jobs are not idle at all when it is executed in the reduce phase and it is executed in the local optimal sequence.

The simplified process of the original jobs set can be summarized as sorting all jobs by map phase in descending order of time cost, starting with the first job and working backwards, and then merging if the merge condition 1 is met. Repeat this process until we find the last job. Then we search forward from the second job, if the merge condition 2 is met, we merge. Repeat the process until the last

	$j_i.TC_{map}$	$j_i.TC_{reduce}$
j_1	15s	10s
j_2	20s	35s
j_3	30s	40s
j_4	45s	30s
j_5	60s	50s
j_6	70s	40s

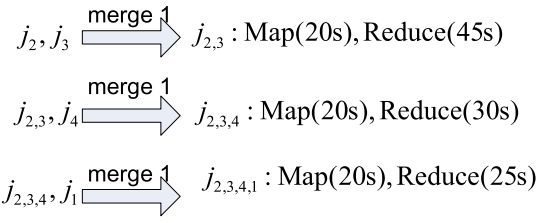


Fig. 4 Jobs merge strategy

job is executed and the merge process is complete. As shown in Fig. 4, when the jobs are listed in ascending order of map time, the jobs j_2,j_3 are merged into $j_{2,3}$ according to merged condition 1. Similarly, $j_{2,3}$,j_4 is merged into $j_{2,3,4}$ according to merger condition 1. Then, we execute a reverse find and $j_{2,3,4}$,j_1 is merged into $j_{2,3,4,1}$ according to merge condition 2. After merging is complete, we obtain a simplified set of jobs $\{j_{2,3,4,1},j_5,j_6\}$.

(2) Simplifying the optimal sequence of sets

The simplified set of jobs resulting is $J_{short} = \{j_1,j_2,\ldots,j_{|J_{short}|}\}$ from the merger. The set has the following characteristics. When all jobs are listed in ascending order of their map task times, the reduce times of all jobs are less than the map times of the jobs that come after them. The sum of the map task completion times for all jobs $J_{short}.MapCost = \sum_{i=1}^{|J_{short}|} J_i.TC_{map}$ and the sum of the reduce task completion times for all jobs $J_{short}.ReduceCost = \sum_{i=1}^{|J_{short}|} J_i.TC_{reduce}$ are recorded here.

Since the jobs in the simplified job set J_{short} are sorted in descending order of time spent on the map phase, a base sequence $seq_{basic} = \{j_1,j_2,\ldots,j_{|J_{short}|}\}$ can be determined in that order. Using Eq. 2, we know that its completion time is $seq_{basic}.cost = J_{short}.MapCost + j_{|J_{short}|}.TC_{reduce}$. Using the completion time of this basic sequence as a limit, the optimal sequence of J_{short} can be derived by the following pruning method. Based on the given sequence $seq_{basic} = \{j_1,j_2,\ldots,j_{|J_{short}|}\}$, this paper determines a sequence seq from backward and forward and makes the completion time of the sequence seq less than the completion time of the sequence $seq_{basic} = \{j_1,j_2,\ldots,j_{|J_{short}|}\}$. The steps are as follows.

First, we identify all the jobs that can be placed last in *seq*, and require the reduce completion time of this job be less than $j_{|J_{short}|}.TC_{reduce}$. A candidate set $|J_{short}|$.candi can be formed for all jobs that meet this condition. If a certain job j'_{-1} in $|J_{short}|.candi$ is selected, the compensation factor $j'_{-1}.factor = j_{|J_{short}|}.TC_{reduce} - j'_{-1}.TC_{reduce}$ of the corresponding job j'_{-1} can be recorded.

Second, with the 1st from the end job j'_{-1} known, we need to find the appropriate job that can be ranked in the second from the end. If the compensation factor of the 2nd from the end job is j'_{-1}.factor, the 2nd from the end job j'_{-2} must satisfy condition that the reduce completion time of job j'_{-2} is less than $j'_{-1}.factor + j'_{-1}.TC_{map}$. We record that the compensation factor of the 2nd from the end job j'_{-2} is $j'_{-2}.factor = j'_{-1}.factor + j'_{-1}.TC_{map} - j'_{-2}.TC_{reduce}$.

After it has been determined that the *i*th from the end job of the sequence is j'_{-i}, the corresponding compensation factor is j'_{-i}. The $(i+1)^{th}$ from the end job need to meet the conditions that the reduce completion time of job $j'_{-(i+1)}$ is less than $j'_{-i}.factor + j'_{-i}.TC_{map}$. We record that the compensation factor of the job $j'_{-(i+1)}$ is $j'_{-(i+1)}.factor = j'_{-i}.factor + j'_{-i}.TC_{map} - j'_{-(i+1)}.TC_{reduce}$. When it is determined that the compensation factor to a certain point is 0, the algorithm can be terminated.

After the iterative function algorithm $Selectjob(seq_{-i}, -(i+1), factor)$ determines the $(i+1)^{th}$ from the end job, the candidate set for the $(i+1)^{th}$ from the end job can be determined using step 3 above.

Algorithm 2 $Selectjob(seq_{-i}, -(i+1), factor)$

the ith
if $i+1 = |J_{short}$— **then**
  the job $j = J_{short} - seq_{-i}$ is not selected by seq_{-i}
  if $j.TC_{reduce} - j_{-i}.TC_{map} < factor$ **then**
    put j in the first place of seq_{-i} **return** seq_{-i}
  elsereturn null
  end if
end if
all jobs that are not selected by seq_{-i} form a set of *Remain*
for each job $j^{'}$ in *Remain* **do**
  if $j^{'}.TC_{reduce} - j_{-i}.TC_{map} < factor$ and $j^{'}.TC_{reduce} \leq j_{-i}.TC_{map}$ **then**
    add $j^{'}$ to the candidate set $pos - Candi_{-(i+1)}$
  else if $j^{'}.TC_{reduce} - j_{-i}.TC_{map} < factor$ and $j^{'}.TC_{reduce} > j_{-i}.TC_{map}$ **then**
    add $j^{'}$ to the candidate set $neg - Candi_{-(i+1)}$
  end if
end for
if $pos - Candi_{-(i+1)}$ and $neg - Candi_{-(i+1)}$ are null **then return** null
end if
sort the jobs in the $pos - Candi_{-(i+1)}$ in descending order of completion time for the reduce task
sort the jobs in the $neg - Candi_{-(i+1)}$ in descending order of completion time for the reduce task
$Candi_{-(i+1)}$ is the jobs in $pos - Candi_{-(i+1)}$ and the jobs in $neg - Candi_{-(i+1)}$
for each job $j^{'}$ in $Candi_{-(i+1)}$ **do**
  put j at the (i+1)th from the end job position of seq_{-i} to form a new sequence $seq_{-(i+1)}$
  $seq_{-(i+1)}.factor = factor + j_{-i}.TC_{map} - j^{'}.TC_{reduce}$
  sequence $temp = Selectjob(seq_{-(i+1)}, -(i+2), seq_{-(i+1)}.factor)$
  if *temp* is not null **then return** *temp*
  end if
end for
return null

Algorithm 3 The Optimal Sequence Generation

Input: simplification set $seq_{basic} = \{j_1, j_2, \ldots, j_{|J_{short}|}\}$
Output: baseline sequence $seq_{basic} = \{j_1, j_2, \ldots, j_{|J_{short}|}\}$

```
repeat
    all jobs with Reduce duration less than seq_basic.cost − J_short.MapCost form the
    candidate set |J_short|.candi
    for each job j in |J_short|.candi do do
        make j as the first from end job in seq_{-1}
        seq_{-1}.factor = seq_basic.cost − J_short.MapCost − j.TC_reduce
        seq = Selectjob(seq_{-1}, −2, seq_{-1}.factor)
        if seq is not null then
            make seq as new baseline sequence seq_basic
        end if
    end for
until seq is null
return the optimal sequence seq_basic
```

Using an iterative function (Algorithm 2), Algorithm 3 shows how to determine the optimal sequence for finding the simplified job set.

Through the following example, how to find the optimal sequence on the simplified set is explained in detail. The three jobs shown in Fig. 5 are the simplified job set of Fig. 4. First, a qualifying sequence $seq_{basic} = \{j_{2,3,4,1}, j_5, j_6\}$ can be determined. When determining the third job (the first from end job), the reduce completion time of the third job is required to be less than 40 s, and only the job $j_{2,3,4,1}$ meets this condition. When the third job is identified, the second job (the second from end job) needs to satisfy the condition that the reduce task is less than $15 + 20 = 30$ s. The rest of the jobs j_5,j_6 do not satisfy this condition, so it can be determined that the sequence $seq_{basic} = \{j_{2,3,4,1}, j_5, j_6\}$ is the optimal sequence for the set of simplifications.

(3) Optimal sequence of the original set

The final sequence of the original set can be obtained using the optimal sequence of the simplified set that was previously obtained. We substitute this optimal sequence into the unmerged original jobs to form a sequence of a sequence of

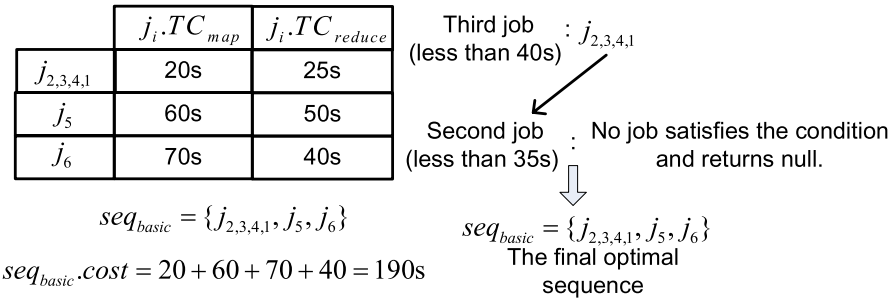


Fig. 5 Optimal sequence of reduction set

original job set, and record the completion time of the sequence as a bound value. In the examples shown in Figs. 4 and 6, the optimal sequence of the simplified set is $\{j_{2,3,4,1}, j_5, j_6\}$ and the current qualifying sequence $\{j_2, j_3, j_4, j_1, j_5, j_6\}$ can be obtained when it is brought into the original job set J. According to Eq. 2, it corresponds to a completion time of 280 s. The statistical value of the job set is known $J.MapCost = 240$ s, $J.ReduceCost = 205$ s.

The determination of the optimal sequence of the original job set is similar to the procedure for determining the optimal sequence of the simplified job set. When the statistical value of the original job is $J.MapCost \geq J.ReduceCost$, the original sequence can be used to determine the optimal sequence using Algorithm 3. As shown in Fig. 6, using the optimal sequence of simplified sequences, the base sequence $\{j_2, j_3, j_4, j_1, j_5, j_6\}$ of the original set can be obtained which corresponds to a completion time $seq_{basic}.cost = 240$ s. Algorithm 3 is then used to recursively find the optimal sequence and determine the jobs in the new sequence in reverse order. First, we select the job j_1 as the 5th job and get the corresponding compensation factor of 30. Then it is determined that the 5th job is j_4 with the corresponding compensation factor of 15. And so on, until the 1st job is found to be j_2 and the compensation factor is greater than 0. A new base sequence $seq'_{basic} = j_2, j_3, j_4, j_1, j_5, j_6$ can be obtained. Bringing the new base sequence into Algorithm 3, the 6th job can only take j_1, while the 5th job is required to have a reduce task completion time of less than 15 + 15 = 30 s. It can be seen in the figure that there is no job that meets the conditions, so the algorithm ends. The final optimal sequence obtained is $seq_{opt} = \{j_2, j_3, j_4, j_1, j_5, j_6\}$.

When $J.MapCost < J.ReduceCost$, using Algorithm 3 to determine the order of job execution in reverse order would result in the existence of a large set of

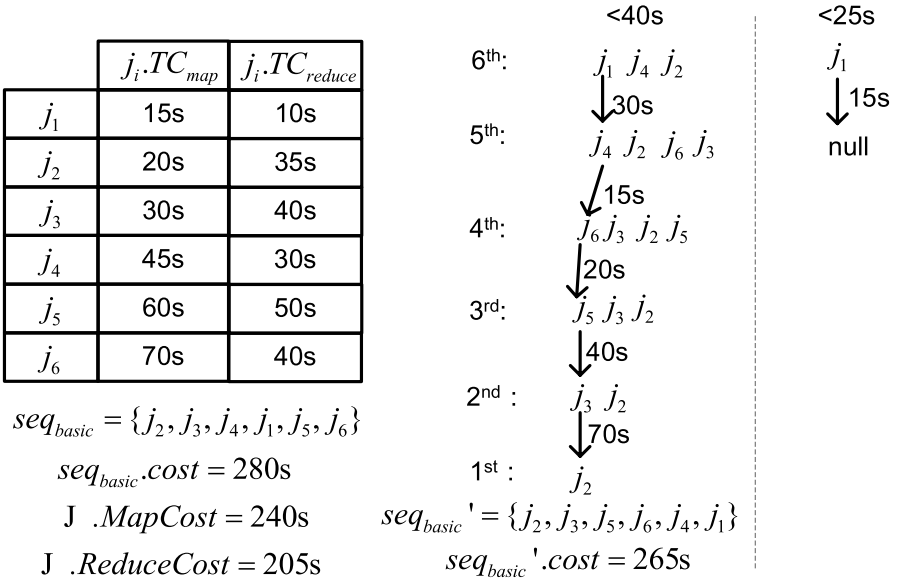


Fig. 6 Optimal sequence of original set based on reverse order

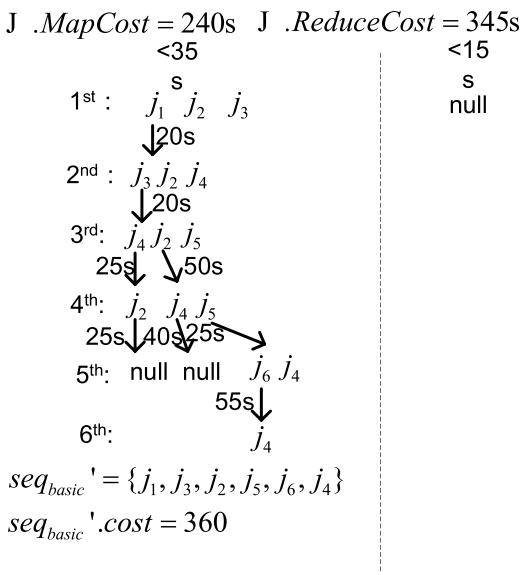
	$j_i.TC_{map}$	$j_i.TC_{reduce}$
j_1	15s	30s
j_2	20s	35s
j_3	30s	50s
j_4	45s	20s
j_5	60s	100s
j_6	70s	110s

$j_{1,2,3,4}$: Map(15s), Reduce(40s)

$j_{5,6}$: Map(60s), Reduce(140s)

$seq_{basic} = \{j_1, j_2, j_3, j_4, j_5, j_6\}$

$seq_{basic}.cost = 380$



$seq_{basic}' = \{j_1, j_3, j_2, j_5, j_6, j_4\}$

$seq_{basic}'.cost = 360$

Fig. 7 Optimal sequence of original set based on positive sequence

candidates for each step, without utilizing the filtering of the pruning strategy. Therefore, after adjusting Algorithm 3, the positive order can be used to determine the order of operation and achieve better pruning results. The sequence determination method of the positive sequence is basically the same as the reverse sequence method. The specific steps are shown below.

First, we identify all the jobs in seq that can be place and require that the map completion time for these jobs is less than $seq_{basic}.cost - J.ReduceCost$. When a job j_1' is selected that meets the condition, it corresponds to a compensation factor $j_1'.factor = seq_{basic}.cost - J.ReduceCost - j_1'.TC_{map}$.

Second, with the 1st job known to be j_1', we search further for jobs that can be ranked 2nd. The compensation factor for j_1' is j_1'.factor. Then the 2nd job needs to meet the condition that the map completion time is less than $j_1'.factor + j_1'.TC_{reduce}$. The compensation factor for j_2' is $j_2'.factor = j_1'.factor + j_1'.TC_{reduce} - j_2'.TC_{map}$.

When it has been determined that the ith job is j_i' and its corresponding compensation factor is j_i'.factor, the $(i+1)^{th}$ job j_{i+1}' needs to meet the condition that the map completion time is less than $j_i'.factor + j_i'.TC_{reduce}$. The compensation factor for j_{i+1}' is $j_{i+1}'.factor = j_{i+1}'.factor + j_i'.TC_{reduce} - j_{i+1}'.TC_{map}$.

As shown in Fig. 7, for a given set of jobs, it can be found that the completion time of its underlying sequence is 380 s, $J.MapCost = 240$ s, $J.Reduce = 345$ s. If the optimal sequence is still determined in reverse order, it can be found that all jobs can be placed in the end with a large compensation factor. This creates a situation where almost all sequences need to be enumerated in order to find the final optimal sequence. In order to enhance the filtering capacity of the compensation factor, a positive sequence is used to determine the order of operation. As shown in

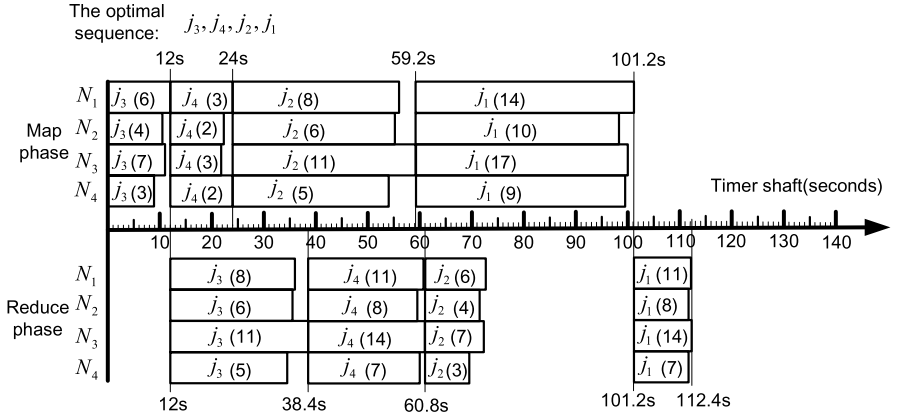


Fig. 8 The estimated execution strategy of jobs in heterogeneous cluster

Fig. 7, after determining the 1st job j_1, we can determine that the set of candidates for the 2nd job are j_2,j_3,j_4 and select the first element as the 2nd job. And so on, a new basic sequence can be identified as $seq'_{basic} = \{j_1,j_3,j_2,j_5,j_6,j_4\}$. Using the new basic sequence as a qualifying sequence, it can be found that the map completion time without jobs is less than 15s, so the algorithm ends. Finally, we can obtain the optimal sequence is $seq_{basic} = \{j_1,j_3,j_2,j_5,j_6,j_4\}$.

4.3 Adjustments to the multi-operational scheduling solution

Given a set of jobs J, Algorithm 3 can be used to find the optimal sequence of all jobs executed in J so that all jobs in J can be completed in the shortest possible time. Given the optimal sequence $seq_{opt} = \{j_1, j_2, \dots, j_{|J|}\}$ for J, we only need process the jobs on the order of execution of the optimal sequence. Algorithm 1 gives the number of map and reduce tasks that each node should execute on a heterogeneous cluster for each job. However, the execution scheme for each job given in Algorithm 1 is based on the assumption that the entire heterogeneous cluster starts executing the job at the same time. In the actual execution, the completion time of job j_1 at each node of the heterogeneous cluster during the map phase or reduce phase is likely to be different, so job j_2 starts at each node at a different time. As in Fig. 8, the optimal sequence is known to be j_3,j_4,j_2,j_1 and the completion time at each node is different when job j_3 is executed during the map phase. It can be found that after node N_2,N_3,N_4 has finished executing, N_1 continues to execute. To further increase the throughput capacity of the system, the scheduling scheme should be further adjusted to reduce node idleness. Figure 9 shows the performance of the adjusted operations.

The adjustment method of the optimal sequence is based on an improvement of Algorithm 1. The specific adjustment process is as follows.

(1) For the first job in the sequence, we distribute the tasks of this job to each node according to Algorithm 1. After assignment, we can record the number of assignments on each node and the completion time of the 1st job on each node.

(2) On node N_i, it can record its compensation time $N_i.suptime = N_1.comptime_1 - N_i.comptime_1$ compared to N_1. According to Algorithm 3, it is possible to initialize to obtain $Num_1 = \lfloor j_m.M_{num} / \sum_{i=1}^{|N|} 1/N_i.cost \rfloor$ when processing the m^{th} job j_m in the optimal sequence. On node N_i, its initialization number is $Num_i = \lfloor Num_1/N_i.cost + N_i.suptime/j_m.T^i_{map} \rfloor$. For unassigned tasks, lines 7–15 in Algorithm 1 are followed.

As shown in Fig. 9, the optimal sequence of the entire job set is $seq_{opt} = \{j_3, j_4, j_2, j_1\}$. First, job j_3 is processed according to Algorithm 1. It can be seen that the number of tasks assigned to node N_1,N_2,N_3,N_4 in the map phase of j_3 are 6,4,7,3 respectively. We record the completion time corresponding to each node, $N_1.comptime_1 = 12s$, $N_2.comptime_1 = 10.4s$, $N_3.comptime_1 = 11.4s$, $N_4.comptime_1 = 9s$. Then we process the map task assignment of the job j_4, and calculate according to the initialization, $Num_1 = 2$, $Num_2 = \lfloor Num_1/1.3 + 1.6/(4 \times 1.3) \rfloor = 1$, $Num_3 = 2$, $Num_4 = 1$. We count the completion time on each node as $N_1.comptime_2 = 20s$, $N_2.comptime_2 = 15.6s$, $N_3.comptime_2 = 17.6s$, $N_4.comptime_2 = 15s$. At this point, j_4 still has four map tasks that are not allocated. According to the greedy allocation strategy in Algorithm 3, the remaining four jobs are assigned to nodes N_2, N_3, N_4 and N_1. Through the adjustment of the above method, the final job distribution can be obtained, as shown in Fig. 9.

When the final execution scheme is determined, the master node assigns the corresponding number of tasks to each node. When a node's task slot finishes executing the current task, the next task is executed as planned.

During execution, it is necessary to count the values, record the average processing time of its map task and reduce tasks on each node for each executed job. Using the above statistics value information, the true computing capacity of each node on a heterogeneous cluster can be calculated. If the actual computational capacity differs from the previous estimates by more than 10%, the HTD algorithm needs to be re-executed to generate the latest optimal sequence and gives the latest execution

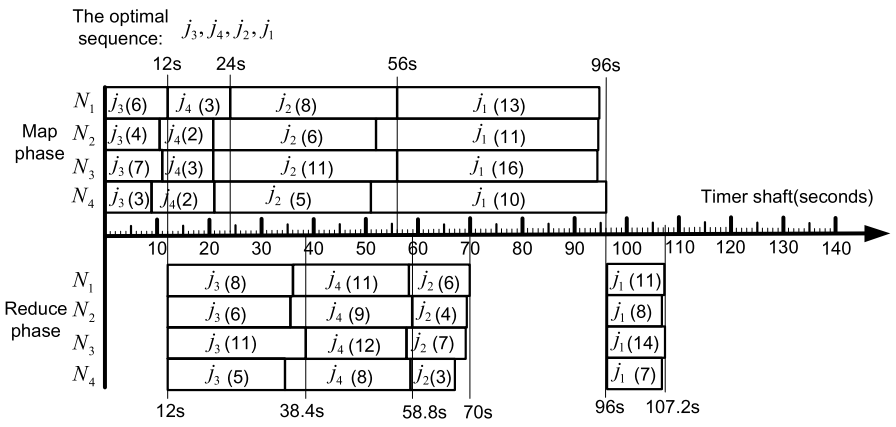


Fig. 9 The execution strategy of jobs in heterogeneous cluster after adjustment

Table 3 Parameter settings

Parmeter	Default value	Range of values
Number of map tasks (job size)	100	50, 100, 150, 200
Number of jobs	20	10, 20, 30, 40, 50
Cluster size	25	10, 15, 20, 25, 30

strategy. In addition, the HTD algorithm is restarted periodically to avoid large gaps in estimates that can affect the performance of the system.

5 Experimental evaluation

In this section, the performance of the proposed HTD algorithm is experimentally verified. The experimental MapReduce cluster is built using Hadoop (version 1.0.4) and consists of one master node and 25 slave nodes. Each slave node consists of an Intel Core i3 2100 CPU, 8 GB of memory, and 500 GB of hard drive. In order to build a heterogeneous environment and make the computing capacity of each node different, in the experiment, we control the computing capacity by locking the CPU frequency. Specifically, 25 slave nodes are equally divided into 5 groups, and each group includes 5 nodes. Then we lock the CPU frequency of these 5 groups of nodes at 3.1 GHz, 2.8 GHz, 2.3 GHz, 2.0 GHz, and 1.6 GHz. Each slave node contains a map task slot and a reduce task slot. The input file fragment size is 64 MB. The experiment uses five common MapReduce jobs to build a job set, including word count, inverted index, distributed grep, join and distributed sorting.

We test the time cost of generating the HTD scheduling scheme in Sect. 5.1. We then verify the throughput capability of the system when using HDT for task scheduling. The main measure is the final completion time of the job set. The scheduling strategies for the experimental comparison include FIFO, LATE, and Skewtune.

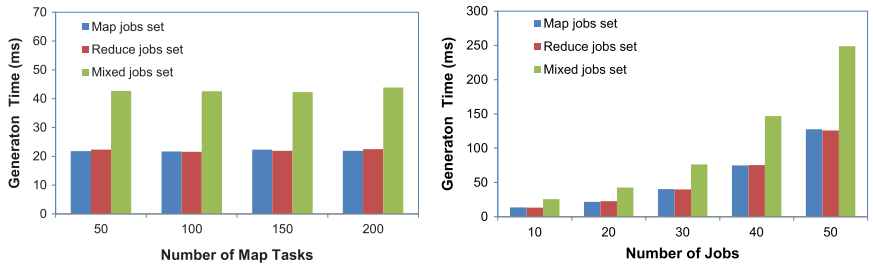
5.1 Time cost of generating the HTD

In order to adequately simulate various application scenarios, three sets of jobs are constructed in this experiment. The first group of job sets include word count, inverted index and distributed grep. Their common feature is that the completion time of the map task is longer than the completion time of the reduce task, which is recorded as the map jobs set. The second group of job sets include join and distributed sorting. Their common feature is that the completion time of the reduce task is longer than the completion time of the map task, which is recorded as the reduce jobs set. The third group of job sets is a mixed jobs set, with half of the above two jobs.

In our experiences, we test the effect of job size, number of jobs and cluster size on generation time. The default values and ranges of variation for each parameter are shown in Table 3.

Figure 10a depicts the effect of jobs size on generation time of HTD. When the number of map tasks for a job becomes larger, it means that the job is processing a larger amount of data and consuming more time. As can be seen in Fig. 10a, the generation time of HTD remains essentially unchanged as the job increases. In the map jobs set, the map task completion time of all jobs is greater than the reduce task completion time, and the reverse order filtering method can quickly obtain the optimal sequence. In the reduce job set, the reduce task completion time of all jobs is greater than the map task completion time, and the positive order filtering method can quickly obtain the optimal sequence. In the mixed jobs set, the filtering effects of the reverse order filtering method and the positive order filtering method are both worse than those of the previous two job sets. Therefore, under the same conditions, the generation of the HTD of the mixed jobs set is the slowest. However, it can be seen from the figure that no matter how the job set changes and how the job size is set, the generation time of the HTD is at the millisecond level, which hardly has any impact on the processing of the job.

Figure 10b depicts the effect of jobs number on the generation time of the HTD. As the number of jobs increases, the generation time of the HTD also increases significantly. As the number of jobs increases, the number of sequences that can be formed will increase sharply. But the filtering method can filter out most sequences,



(a) Effect of Jobs Size on Generation of HTD (b) Effect of Jobs Number on Generation of HTD

(c) Effect of Cluster Size on Generation of HTD

Fig. 10 Time cost on generating of HTD

so the generation time of HTD increases linearly with the increase of the number of jobs. Even when the number of jobs reaches 50, the generation time of the HTD will still not exceed 1 s. Therefore, even in the face of a large number of jobs, the generation time of the HTD is still within an acceptable range. In addition, under the same job parameters, the generation time of the mixed jobs set is still longer than the map jobs set and the reduce jobs set.

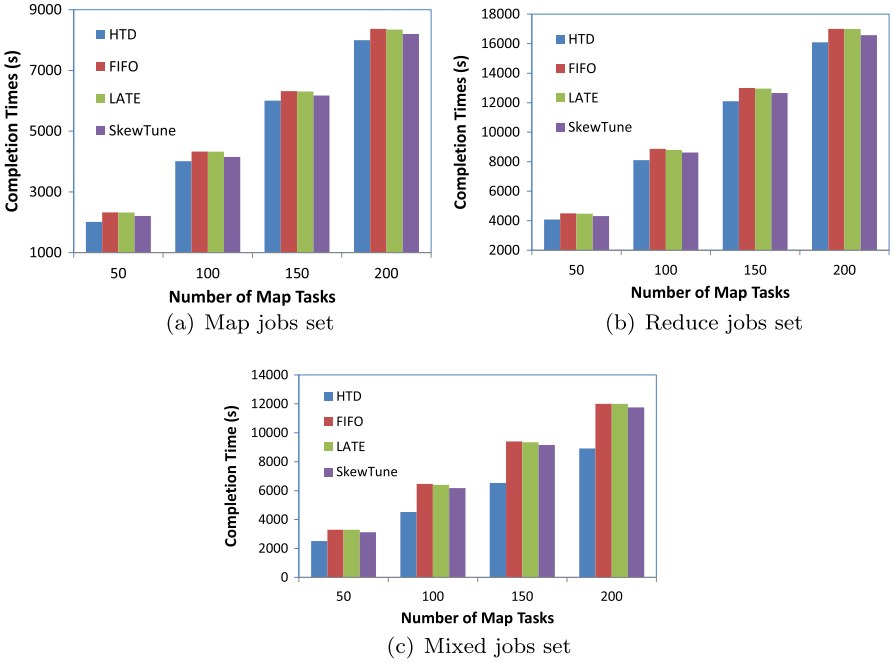
Figure 10c depicts the effect of cluster size on the generation time of the HTD. As the cluster size increases, the number of nodes participating in the calculation increases. The generation time of the HTD also increases slightly. Since the generation time for the first and third steps of the HTD generation step increases linearly as the cluster size increases. However, the generation time of HTD depends mainly on the second step, and the completion time of this step is independent of the cluster size. Therefore, the final generation time of the HTD increases only slightly, and the increase is negligible in the figure.

From the above experimental results, it can be found that no matter how the job parameters and clusters change, the generation of the HTD scheduling scheme is extremely fast. Although compared with other scheduling algorithms, HTD requires preprocessing to arrange all the job execution order. Even in the face of a large-scale job set, the preprocessing time consumption is completely within the acceptable range and will not affect the execution of the job. Next, we further verify the throughput capability of the system using the HTD.

5.2 Performance evaluation of HTD

This section verifies the throughput capacity of the system when using different algorithms. In the experiment, the throughput is measured by the final completion time of the job set. The comparison algorithm is FIFO, LATE and Skewtune.

Figure 11 depicts the effect of jobs size on completion time. As the jobs size increases, the completion time of the four scheduling algorithms increases. In the map/reduce jobs set, the completion time of all jobs mainly depends on the completion time of the map/reduce phase. In this single type of set, the HTD is only slightly better than the other three. In details, Skewtune performs a little better than FIFO and LATE because of the task reallocation technique in the last wave of jobs. But, the the system resource utilization using these three schedulers is good enough, so the improvement that HTD can bring is limited. However, in the mixed jobs set, the total processing time of map tasks of all jobs is not much different from the total processing time of reduce tasks. The HTD can make full use of system resources, so that the idle time of the map and reduce task slots in the system is very small, and the execution is completed as soon as possible. In the FIFO and LATE, when the first half of the execution comes from the job in the map jobs set, the reduce task slot has a lot of idle time, and when the second half of the execution comes from the job in the reduce jobs set, the map task slot has a lot of idle time. Considering the skew situation in MapReduce, Skewtune can split a slow task into fragments and reallocate them to the workers with good-performance. the scheduler can mitigate heterogeneous problems, but still cannot solve the problem of idle resources.



(a) Map jobs set (b) Reduce jobs set

(c) Mixed jobs set

Fig. 11 Effect of jobs size on completion time

Compared with the three above, HTD has achieved a very obvious advantage in the mixed jobs set. No matter how the job size changes, the throughput capacity of HTD is better than the others. Especially in the mixed jobs set, this advantage is very obvious.

Figure 12 depicts the effect of jobs number on completion time. As the number of jobs increases, the completion time of various job sets increases linearly. Since the completion time of each job in the reduce jobs set is higher than the completion time of each job in the map jobs set, the completion time in the reduce jobs set is the slowest. Similarly, no matter how the number of jobs changes, the completion effect of the HTD is always better than that of the other three scheduling algorithms. This advantage is particularly prominent in the mixed job set.

Figure 13 depicts the effect of cluster size on completion time. We found that as the size of the cluster increases, the number of nodes participating in computing increases, and the computing capability of the cluster per unit time increases. Therefore, the completion time of the jobs set decreases linearly. Regardless of the cluster size, the HTD in this paper shows excellent performance, which is better than the existing ones.

To sum up, in a heterogeneous MapReduce environment, the HTD scheduling algorithm proposed in this chapter arranges a reasonable job execution sequence so that the job set can be completed in the shortest possible time. Compared with the existing scheduling algorithm, HTD can bring higher throughput to the MapReduce system.

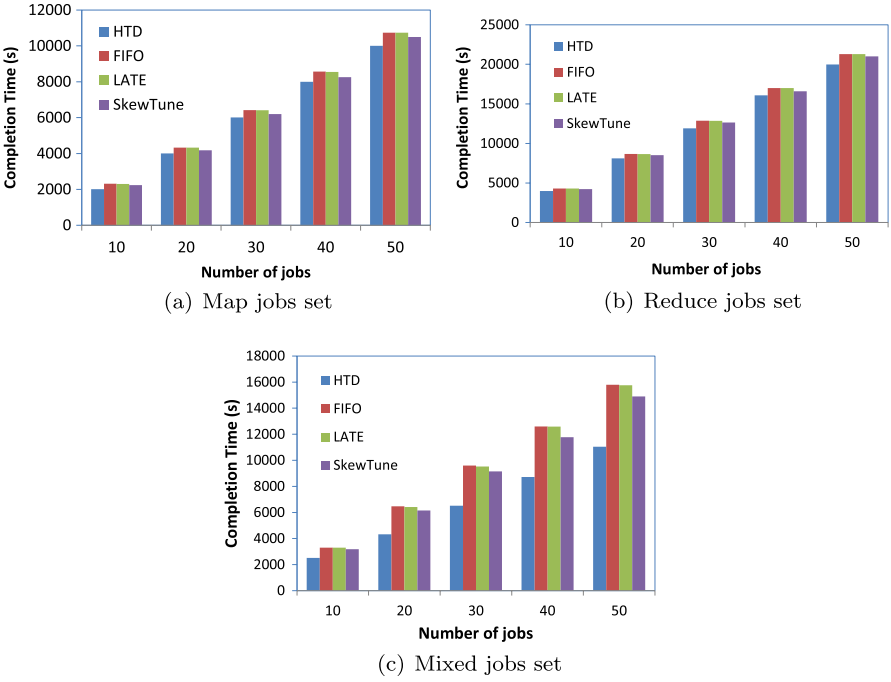


Fig. 12 Effect of jobs number on completion time

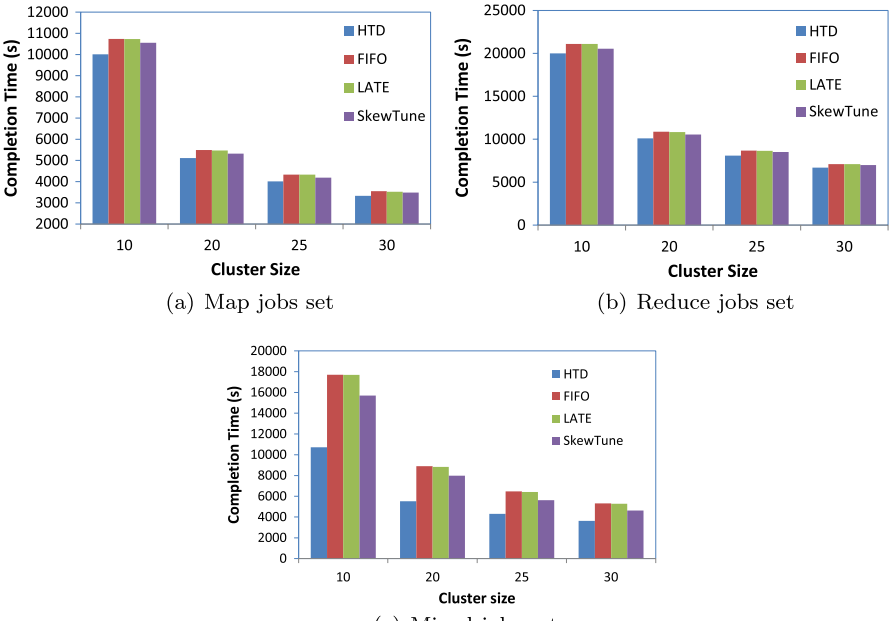


Fig. 13 Effect of cluster size on completion time

6 Conclusion

We mainly study the batch-job processing problem in a heterogeneous MapReduce environment, and design an effective scheduling algorithm HTD for improving the system throughput. The algorithm adopts a reasonable job execution sequence to ensure that all jobs are completed in short possible time. The HTD mainly includes 3 steps. First, we design the assignment scheme for a single job in a heterogeneous environment, and estimate the parameters for all the jobs in the job set. Second, we generate the optimal execution order for the MapReduce environment. Finally, we apply this optimal execution order into the heterogeneous environment and adjust the number of assignments of each job on the nodes. At last, we verify the the effectiveness and efficiency of HTD through experiments. As the results show, our method can reduce the processing time for batch jobs in MapReduce clusters, and shows significant advantages in mixed-job situations. In the following research, we will further study the scheduling strategy under the network constraints, and improve the system throughput in complex network environments.

Acknowledgements This work is supported by the National Natural Science Foundation of China (Grant Nos. 61602076, 61702072, 62002039, 61976032), the China Postdoctoral Science Foundation funded projects (Grant Nos. 2017M611211, 2017M6211, 2019M661077), the Natural Science Foundation of Liaoning Province (Grant No. 20180540003), CERNET Innovation Project (Grant No. NGII20190902).

References

1. Maleki, N., Faragardi, H.R., Rahmani, A.M., Conti, M., Lofstead, J.F.: TMaR: A two-stage MapReduce scheduler for heterogeneous environments. Hum. Centric Comput. Inf. Sci **10**, 42 (2020)
2. Mitsuzuka, K., Hayashi, A., Koibuchi, M., Amano, H., Matsutani, H.: In-switch approximate processing: Delayed tasks management for MapReduce applications, 2017 27th International Conference on Field Programmable Logic and Applications (FPL), pp. 1–4 (2017)
3. Chen, C., Lin, J., Kuo, S.: MapReduce scheduling for deadline-constrained jobs in heterogeneous cloud computing systems. IEEE Trans. Cloud Comput. **6**(1), 127–140 (2018)
4. Shen, H., Sarker, A., Yu, L., Deng, F.: Probabilistic network-aware task placement for MapReduce scheduling. In: 2016 IEEE International Conference on Cluster Computing (CLUSTER), pp. 241–250 (2016)
5. http://hadoop.apache.org
6. Camacho-Rodríguez, J., Chauhan, A., Gates, A., et al.: Apache hive: From MapReduce to enterprise-grade big data warehousing. In: Proceedings of the 2019 International Conference on Management of Data, pp. 1773–1786 (2019)
7. Wu, Y., Li, X., Liu, J., Cui, L.: Hadoop-EDF: Large-scale distributed processing of electrophysiological signal data in hadoop MapReduce. In: 2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), pp. 2265–2271 (2019)
8. Tiwari, N., Sarkar, S., Bellur, U., Indrawan, M.: Classification framework of MapReduce scheduling algorithms. ACM Comput. Surv. **47**, 49:1-49:38 (2015)
9. Bibal Benifa, J.V.: Dejey, performance improvement of MapReduce for heterogeneous clusters based on efficient locality and replica aware scheduling (ELRAS) strategy. Wirel. Pers. Commun. **95**, 2709–2733 (2017)
10. Jiang, Y., Zhu, Y., Weili, W., Li, D.: Makespan minimization for MapReduce systems with different servers. Fut. Gener. Comput. Syst. **67**, 13–21 (2017)
11. Ahmad, F., Chakradhar, S.T., Raghunathan, A., Vijaykumar, T.N.: Tarazu: Optimizing MapReduce on heterogeneous clusters. ASPLOS **40**, 61–74 (2012)
12. Hsieh, S., Chen, C., Chen, C., Yen, T., Hsiao, H., Buyya, R.: Novel scheduling algorithms for efficient deployment of MapReduce applications in heterogeneous computing environments. IEEE Trans. Cloud Comput. **6**(4), 1080–1095 (2018)
13. Cheng, D., Rao, J., Guo, Y., Jiang, C., Zhou, X.: Improving performance of heterogeneous MapReduce clusters with adaptive task tuning. IEEE Trans. Parallel Distrib. Syst. **28**(3), 774–786 (2017)

14. Rasooli, A., Down, D.G.: COSHH: A classification and optimization based scheduler for heterogeneous Hadoop systems. Future Gener Comput Syst **36**, 1–15 (2014)
15. Bellatreche, L., Cuzzocrea, A., Benkrid, S.: Effectively and efficiently designing and querying parallel relational data warehouses on heterogeneous database clusters: The F&A approach. J. Database Manag. **23**(4), 17–51 (2012)
16. Kerkad, A., Bellatreche, L., Richard, P., Ordonez, C., Geniet, D.: A query beehive algorithm for data warehouse buffer management and query scheduling. Int. J. Data Warehousing Mining (IJDWM) **10**(3), 34–58 (2014)
17. Chi, Y., Hacigümüs, H., Hsiung, W.-P., Jeffrey, F.: Naughton: Distribution-based query scheduling. Proc. VLDB Endow. **6**(9), 673–684 (2013)
18. Mansouri, N.: Cost-based job scheduling strategy in cloud computing environments. Distrib. Parallel Databases **38**(2), 365–400 (2020)
19. Hagras, T., Atef, A., Mahdy, Y.B.: Greening duplication-based dependent-tasks scheduling on heterogeneous large-scale computing platforms. J. Grid Comput. **19**(1), 13 (2021)
20. Zaharia, M., Konwinski, A., Joseph, A.D., Katz, R., Stoica, I.: Improving MapReduce performance in heterogeneous environments. OSDI **8**, 29–42 (2008)
21. Kwon, Y., Balazinska, M., Howe, B., et al.: SkewTune: Mitigating skew in MapReduce applications. ACM SIGMOD Int. Conf. Manag. Data **2012**, 25–36 (2012)
22. Kwon, Y., Balazinska, M., Howe, B., et al.: SkewTune in action: Mitigating skew in MapReduce applications. Proc. VLDB Endow. 2012 **5**(12), 1934–1937 (2012)
23. Hammoud, M., Rehman, S., Sakr, M.: A data locality and skew aware task scheduler for MapReduce in cloud computing. Bloomsbury Qatar Found. J. **2011**, 1 (2011)
24. Yu, X., Kostamaa, P.: Efficient outer join data skew handling in parallel DBMS. Proc. VLDB Endow. **2**(2), 1390–1396 (2009)
25. Kwon, Y.C., Balazinska, M., Howe, B., Rolia, J.A.: Skew-resistant parallel processing of feature-extracting scientific user-defined functions. SoCC **2010**, 75–86 (2010)
26. Pericini, M.H., Leite, L.G., Carvalho-Junior, D., Francisco, H., Machado, J.C., Rezende, C.A.: MAP-Skew metaheuristic approaches for partitioning skew in MapReduce. Algorithms **12**(1), 5 (2019)
27. Wang, B., Jiang, J., Yang, G.: ActCap: Accelerating MapReduce on heterogeneous clusters with capability-aware data placement. INFOCOM **2015**, 1328–1336 (2015)
28. Wang, J., Li, X.: Task scheduling for MapReduce in heterogeneous networks. Clust. Comput. **19**(1), 197–210 (2016)
29. Wang, M., Wu, C.Q., Cao, H., Liu, Y., Wang, Y., Hou, A.: On MapReduce scheduling in hadoop yarn on heterogeneous clusters. TrustCom/BigDataSE **2018**, 1747–1754 (2018)
30. Chen, L., Liu, Z.-H.: Energy- and locality-efficient multi-job scheduling based on MapReduce for heterogeneous datacenter. Serv. Orient. Comput. Appl. **13**(4), 297–308 (2019)