International Journal of Grid and Distributed Computing
Vol. 9, No. 10 (2016), pp.249-262
http://dx.doi.org/10.14257/ijgdc.2016.9.10.22

A Load Balancing Algorithm Based on Fair Scheduler

Shengjun Xue, Zhengwei Chen and Jingyi Chen

School of Computer and Software, Nanjing University of Information Science&Technology, Jiangsu Engineering Center of Network Monitoring, Nanjing 210044, China
wheelschen@126.com

Abstract

With Hadoop been widely used, job scheduling technology, as a key technology in Hadoop, has been developed rapidly. However, Hadoop's default scheduling algorithm Fair Scheduler when performing the task scheduler does not consider load balancing status of each node cluster system, leading to low efficiency great job. For defect fair share scheduling algorithm, and combined LATE scheduling algorithm, load balancing scheduling algorithm based on a fair share. Experimental results show that the fair share scheduling algorithm can be improved in the scheduling task when taking into account the situation of each node load balancing, improve the efficiency of large jobs.

Keywords:*Fair Scheduler, LBFS, load balancing, Hadoop, Mapreduce*

1. Introduction

Cloud computing has gained more and more attention in the past few years, it can provide a flexible and scalable computing resources, such as computing power and data storage services to users through the organization of large-scale equipment and clusters. With the continuous development of information technology, research has focused on the Dryad [1] and MapReduce [2]. Hadoop [3], which is open source implementation of MapReduce, has been widely used in various industries, such as Facebook [4] and Yahoo [5]. It is a distributed programming model for parallel processing on TB-level data sets, including a large cluster comprised of thousands of computers. The key benefit of MapReduce is that it allows programmers to focus on the development of software, rather than deal with the underlying problem, such as parallel, scheduling, failover and input split.

MapReduce is a programming model that can deal with a large number of data sets. Users specify a map function that processes a key-value pairs. MapReduce cluster node model consists of a job scheduler (JobTracker) and multiple task schedulers (TaskTracker). Jobtracker is responsible for scheduling jobs, monitoring the TaskTrackers, and scheduling failed tasks again. TaskTracker is responsible for executing tasks. JobTracker includes a task scheduler module, responsible for the tasks assigned to TaskTracker, and periodically sends a heartbeat to Jobtracker. Hadoop scheduler checks the heartbeat and assign tasks to TaskTracker. Hadoop scheduler randomly assigns tasks to TaskTracker with the same heartbeat packet protocol.

Job scheduling is the key technology on Hadoop, it determines the order of the job scheduling. Hadoop's default scheduling policy is FIFO [6]. FIFO does not support preemptive scheduler based on priority, a job with low priority will not be scheduled until the job with a high priority is completed, that mean the job with low priority must have been in a wait state. FIFO ignores differences between users and different jobs in the absence of consideration TaskTracker load conditions. In addition to FIFO, Fair Scheduler [7-8] allows preemptive scheduling jobs [9], the goal of Facebook, which developed the Fair Scheduler, is to provide each job for fair slots, smaller jobs can get quick response, and large jobs can get performance guarantee [10].

Fair Scheduler supports multi-user multi-queue, promised fair sharing of resources within the cluster system, guaranteed minimum shared resources, and provides the function of seizing cluster resources, and to prevent insufficient disk space by limiting the number of jobs executed concurrently. Fair Scheduler has higher throughput compared with FIFO, also improve the efficiency of the user's job. The flow chart of processing jobs of Fair Scheduler is shown in Figure 1.

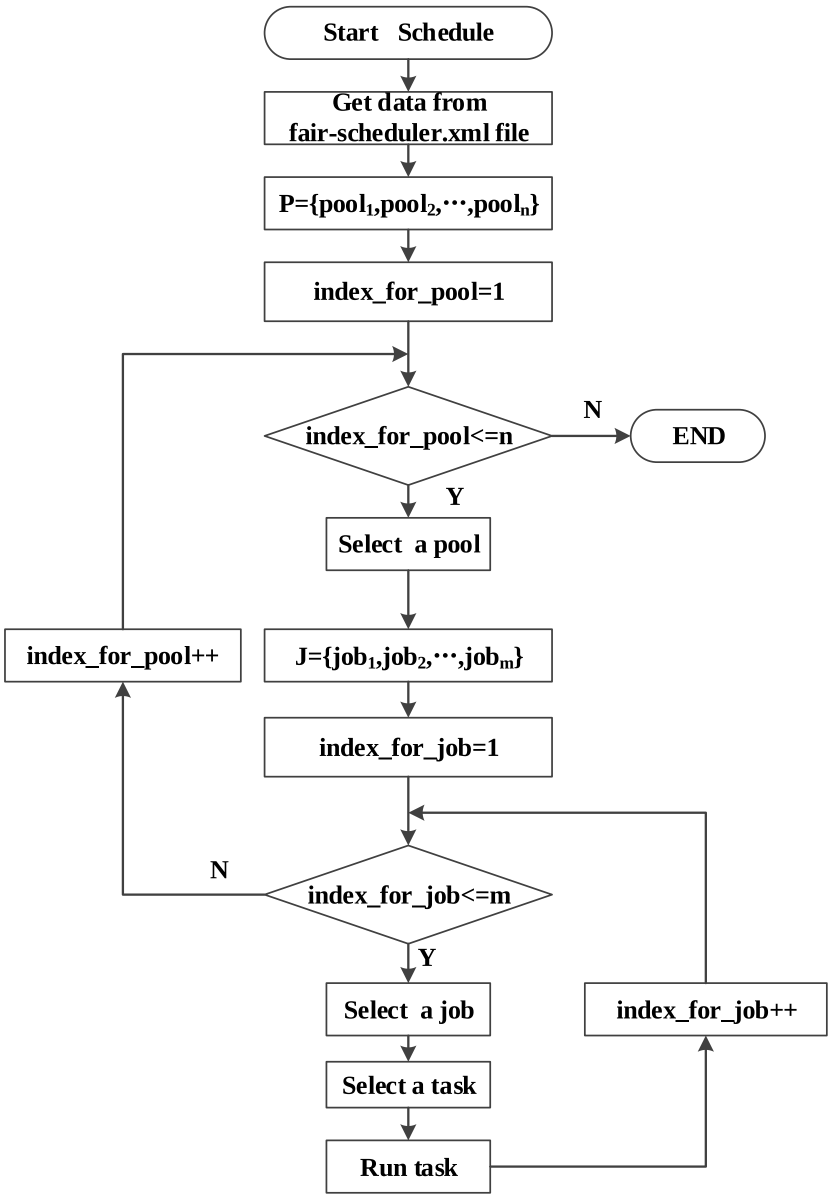

Figure 1. Fair Scheduler Flow Chart

Although Fair Scheduler can guarantee sharing resources between the jobs in the cluster, but it does not take the system load balancing into account, load tilt problems may arise. At the beginning of scheduling job, the input data is divided into the data block, then storage with various Datanode, then the job is divided into multiple tasks distributed by JobTracker onto each TaskTracker, finally, during the execution of tasks, load degree of each TaskTracker may be different. These three stages are related to the problem of load balancing. In the period of job scheduling of Fair Scheduler, the ability of each

TaskTracker to perform tasks is not the same, influenced by the data storage location, bandwidth, and other factors. Reduce phase of the mission and have to wait until after the completion of all tasks performed Map stage to perform. When the "long tail problem" appears in the Map stage to perform tasks that some inefficient TaskTracker node that slowed down the progress of the job execution, Reduce tasks must wait until the slowest node TaskTracker tasks executed on that node, Reduce phase can execute tasks. Therefore, with expect to Fair Scheduler, it may occur that each node has the phenomenon of the load unbalancing, it is necessary to conduct in-depth research, once to ensure the Fair Scheduler to schedule jobs in the process, it is possible to ensure that each node load balancing then to improve the user experience and productivity will certainly be of great significance. Therefore, a load balancing algorithm based on Fair Scheduler (LBFS) was proposed.

2. Improvement Strategy

When executing tasks, the original Fair Scheduler will record the speed rate of each node in the cluster. Traversing the TaskTracker node, if a condition consistent with Map slow TaskTracker node, then the node is added to the collection SlowMap, if a Reduce compliance with conditions a TaskTracker node slow, then the node is added SlowReduce collection. Traversing the scheduled tasks set operations, if a task is left behind to meet the conditions of Map tasks on the task of adding to the collection MapStr, if a task is left behind to meet the conditions of Reduce tasks on the task of adding to ReduceStr collection. TaskTracker when there is an empty slot, no new job request, while the number of the backup tasks does not exceed a preset limit the total number of tasks, it first determines whether the TaskTracker node is left behind, if not, on the implementation of MapStr or ReduceStr tasks. For the stragglers Map task cannot be assigned to stragglers Map node, the same reason, do not allow stragglers Reduce task assigned to stragglers Reduce nodes. Through the above description improvements that can make the improved algorithm in heterogeneous environments to achieve multi-node processing tasks faster, less slow node processing tasks, so as to achieve a balance between cluster nodes task execution speed.

2.1. Math Model

LBFS relative abbreviations are shown as table 1.

Table 1. LBFS Relative Abbreviations

Variable	Meaning	Abbreviation
JobWeight	the weight of a job	jw
JobDeficit	the deficit of a job	jd
MinSlot	the minimum promised slot of a job	ms
JobFairShare	the fair share of a job	jfs
PoolWeight	the weight of a pool	pw
TaskNum	the number of all the tasks	tn
PriorityFactor	priority factor	pf
PoolRunningJobsWeightSum	the sum of the running jobs' JobWeight	prjws
SystemJobsWeightSum	in a pool	sjws
TimeDelta	the sum of all the available jobs'	td
PoolLowJobsWeightSum	JobWeight	pljws
	interval of updating information twice	
	the sum of the low jobs' JobWeight	

In Table 1, the deficit of a job means the difference between the computation time due to ideal scheduler and the actual computation time. As for JobWeight, LBFS has three kinds of calculation methods. In the base case, not only is JobWeight based on the job priority but also based on the job size. The calculation formula is as follows:

$$jw = \begin{cases} \log_2 (tn+1), taking \ \text{int} o \ account \ the \ job \ size \ \text{int} o \ account \ the \ job \ size \\ 1 \qquad\qquad ,others \end{cases} \tag{1}$$

The number of all the tasks is defined as variable tn. Then, JobWeight is also calculated according to the priority of the job. The formula is as follows:

$$jw = jw \times pf \tag{2}$$

The priority factor of a job is defined as pf, the specific values are shown in Table 2.

Table 2. The Priority Factor of LBFS

priority	pf
VERY_HIGH	4.0
HIGH	2.0
NORMAL	1.0
LOW	0.5
DEFAULT	0.25

Finally, the formula of updating scheduled jobs' JobWeight is as follows:

$$jw = jw \times \frac{pw}{prjws} \tag{3}$$

LFBS scheduling algorithm has the variable ms, it is the short of MinSlots, viz the job can get the minimum guaranteed discharge of the Task Slots, the computing method of the variable is as following.

$$ms = \left\lfloor slotsLeft \times \frac{jw}{pljws} \right\rfloor \tag{4}$$

slotsLeft denotes the unused task slots in the system, it is the Map task slots or the sum of the Reduce task slots.

jfs variable refers to the job fair sharing capacity. In the algorithm there is a collection *jobleft*. The collection deposits the system running job. In the process of scheduling, the scheduler will traverse all the jobs in the collection *jobleft* and calculate the *jfs* value of all jobs. If a job's *ms* is greater than *jfs*, then the scheduler will assign the jfs value to the job and meanwhile removed the job from the *jobleft* collection. In the end, it will assign the rest task slot to the rest job in the *jobleft* according to the weight proportion. The compute mode of *jfs* is shown as follows.

$$jfs = slotsLeft \times \frac{jw}{sjws} \tag{5}$$

jd is the fair owe degree between two jobs. When the LBFS scheduler algorithm is assigning jobs to a TaskTracker, it only assign the task which can schedule the jobs to the current computational nodes. Meanwhile these schedulable jobs priority is sorted by the jobs *jd* values scale, viz the job which has the higher *jd* value has the higher priority. The compute mode of *jd* is shown as follows.

$$jd{=}jd + td \times \left(jfs - rt \right) \tag{6}$$

Considering the execution of job have two phases: Map and Reduce.Therefore, actual computation, the computation of *jd* will be divided in *mapdef* and *reducedef*.The computing method of the variable is as following.

$$mapDef{=}mapDef + td \times \left(mfs - rm \right) \tag{7}$$

$$reduceDef{=}reduceDef + td \times \left(rfs - rr \right) \tag{8}$$

mfs denotes the variable mapFairShare. It’s the fair share in the Map phase. *rm* denotes the runningMaps, the running task in the Map phase. *rfs* denotes the variable reduceFairShare, It’s the fair share in the Reduce phase. *rr* denotes the variable runningReduces, the running task in the Reduce phase.

LFBS scheduling algorithm combine with the conventional load balance scheduling algorithm, it need to ascertain any of the TaskTracker is fall behind, and distinguish which tasks are the dropping tasks. In the end it will establish the collection of backup tasks.

Usually we use *smtt(0<smtt<1)* to distinguish the TaskTracker of normal running Map tasks and abnormal running TaskTracker. If the system has N TaskTracker nodes, the speed of ith TaskTracker of Map task is,TT_iSpeed_m. the average speed is *AvgTTSpeed_m*. Suppose there’re *n_map* Map tasks and *n_reduce* Reduce tasks running on the *i*th TaskTracker. We can get the compute mode as following:

$$TT_iSpeed\_m{=}\sum_{j=1}^{n\_map} Speed_j \,/\, n\_map \tag{9}$$

$$TT_iSpeed\_r{=}\sum_{j=1}^{n\_reduce} Speed_j \,/\, n\_reduce \tag{10}$$

$$\mathrm{Avg}TTSpeed\_m{=}\sum_{i=1}^{N} TT_iSpeed\_m \,/\, N \tag{11}$$

$$\mathrm{Avg}TTSpeed\_r{=}\sum_{i=1}^{N} TT_iSpeed\_r \,/\, N \tag{12}$$

If a TaskTracker node satisfies the Equation (13), it indicates that the node is a Map TaskTracker node which is left behind. If a TaskTracker node satisfies the Equation (14), it indicates that the node is a Reduce TaskTracker node which is left behind.

$$TT_iSpeed\_m < \left(1 - smttt\right) \times \mathrm{Avg}TTSpeed\_m \tag{13}$$

$$TT_iSpeed\_r < \left(1 - smttt\right) \times \mathrm{Avg}TTSpeed\_r \tag{14}$$

smtt(0<smtt<1) is used to distinguish the TaskTracker of normal running Map tasks and abnormal running TaskTracker. If the *i*th Map denotes mt_i,and mt_i, satisfies the Equation (15)，then we can make sure that mt_i, is the Map task which is left behind. If the *i*th Reduce task denotes rt_i, and rt_i satisfies the Equation (16)，then we can make sure that rt_i is the Reduce task which is left behind.

$$Speed_i\_m < \left(1 - smtt\right) \times AvgSpeed\_m \tag{15}$$

$$Speed_i\_r < \left(1 - smtt\right) \times AvgSpeed\_r \tag{16}$$

Here , we use BMT to notify the backup Map tasks sets and definite the ratio of the system load and the system processing capacity as *lt(0<lt<1).tn* denotes quantity of the system processing tasks, it can also be named as the system load. We use *sa* to denote the system processing capacity, *bp(0<bp<1)* denotes the max proportion of backup task accounted for all tasks. If there is a task left behind and satisfy the Equation (19), then firstly the task which is left behind will be placed in MapStr set or ReduceStr sets. When TaskTracker have free slots, JobTracker will choose the task from the BMT sets and execute the task.

$$bp = \left(\frac{1}{lt} - 1\right) \times tn \tag{17}$$

$$lt = tn / sa \tag{18}$$

$$BackupNum < bp \times tn \tag{19}$$

2.2. Algorithm Flow Chart

LBFS scheduling algorithm use the Pool to manage jobs, every Pool has its own weight. When there are free slots on the TaskTracker, LBFS scheduling algorithm will firstly choose a Pool, and then choose a suitable job from this pool, generally it choose the job which has the highest fair deficit in the pool, and then choose the appropriate task in the job return to the TaskTracker, here we need to consider the localization feature of data. This task allocation mechanism is a loop execution. It constantly sort the Pool, and loop Pool queue to reorder the queue. The job in the poor is sorted by the fair deficit. The mechanisms circulates jobs and chooses a Task, if it choose to a Task, it will jump out of the loop of the Pool and reorder the Pools.

When the job is submitted to the cluster system, LBFS utilize Pool to organize the job. LBFS use the FIFO strategy. With the consideration of the User/Pool limit condition, LBFS selects a batch of qualified jobs as the jobs sets which are waiting for being scheduled. User limit refers to that the jobs number which belongs to the User cannot exceed the preset upper limit, the Pool limit refers to that the jobs number which belongs to the Pool also cannot exceed the preset upper limit. The algorithm sorts all jobs in the job sets based on the fair degree. The job which has a low fair degree has the higher priority to be scheduled. The fair degree refers to the difference value between the resource have been accounted by the job and the cluster resources should be obtained by the job. The fair deficit of each job depends on the difference value and the time which the job in the unfair resource allocation assignment status. At the same time, the weight of the job and the Pool weight which the job belongs to determine the computing resources which the job can obtained.

After selecting a job, the job will be divided into several tasks. When the TaskTracker requests for the task, it will judge whether the node is a slow Map node. If it is a slow Map node, then it will judge whether the node is a slow reduce node. If it is not a slow reduce node, then it will judge whether the backup task number have been reached the upper limit (BKT limit). If not, then the reduce task which is left behind will be assigned to the TaskTracker. If it is not the slow Map node, then it judge whether the backup task number have been reached the upper limit. if not, If not, then the Map task which is left behind will be assigned to the TaskTracker., and finally if the backup task numbers have been reached the upper limit, then it will adjust the upper limit of backup tasks number according to the network load conditions. LBFS task execution flow chart is shown as Figure 2.

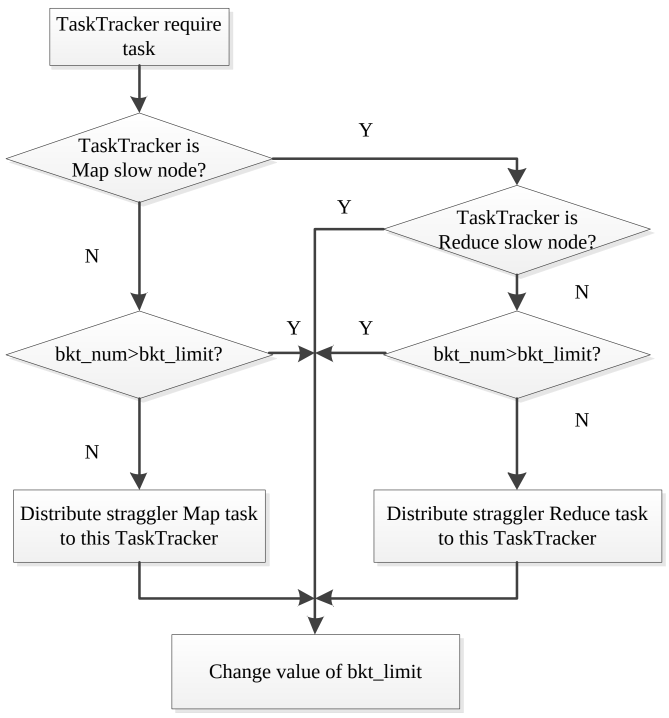
TaskTracker require task
TaskTracker is Map slow node?
Y
TaskTracker is Reduce slow node?
Y
N
N
bkt_num>bkt_limit?
Y
Y
bkt_num>bkt_limit?
N
N
Distribute straggler Map task to this TaskTracker
Distribute straggler Reduce task to this TaskTracker
Change value of bkt_limit

Figure 2. LBFS Task Execution Flow Chart

2.3. Pseudo Code of the Algorithm

Each job will be divided into multiple tasks. TaskTracker and JobTracker will communicat with each other through the "heartbeat". When TaskTracker ask JobTracker for the Task, LBFS will judge the request node, such as whether the node is slow Map node, whether the node is the slow Reduce node, and whether the number of backup Task has been the upper limit. The variables meaning of pseudo code of the algorithm are shown in Table 3.

Table 3. The Variables and the Meaning

variables	meaning
bkt_num	backup task number
SlowMap	slow Map node sets
SlowReduce	slow Reduce node sets
bkt_limit	backup task number upper limit
MapStr	The left behind Map task set
ReduceStr	The left behind Reduce task set

After the job scheduling one minute later, LBFS will count up the TaskTracker's rate according to the cluster change status, then put the Map slow node and Reduce slow node into SlowMap and SlowReduce set. The specific process is as shown in Figure 3.

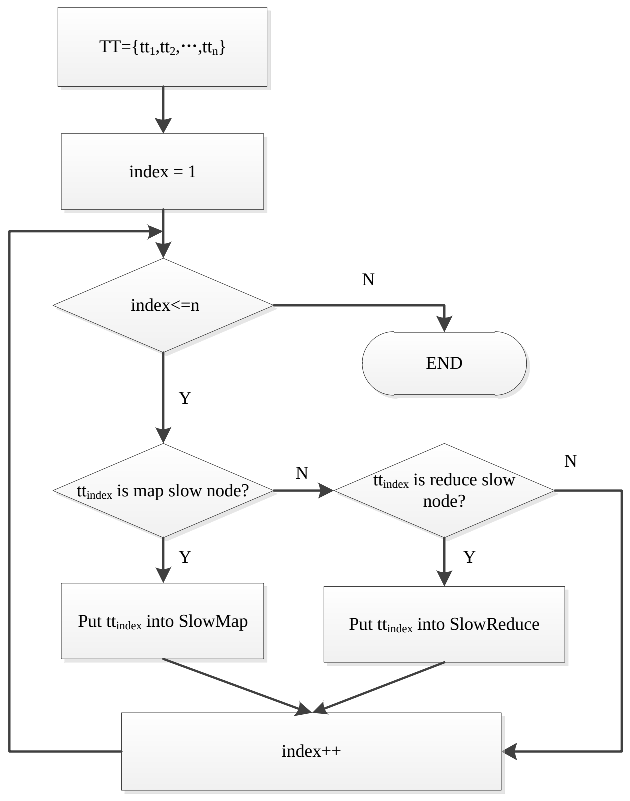
TT={tt1,tt2,⋯,ttn}
index = 1
index<=n
N
END
Y
ttindex is map slow node?
N
ttindex is reduce slow node?
N
Y
Y
Put ttindex into SlowMap
Put ttindex into SlowReduce
index++

Figure 3. Count Up the Slow Map Node and the Sloe Reduce Node

Meanwhile, LBFS also count up the map task and the reduce task which are left behind. The specific process is as shown in Figure 4.

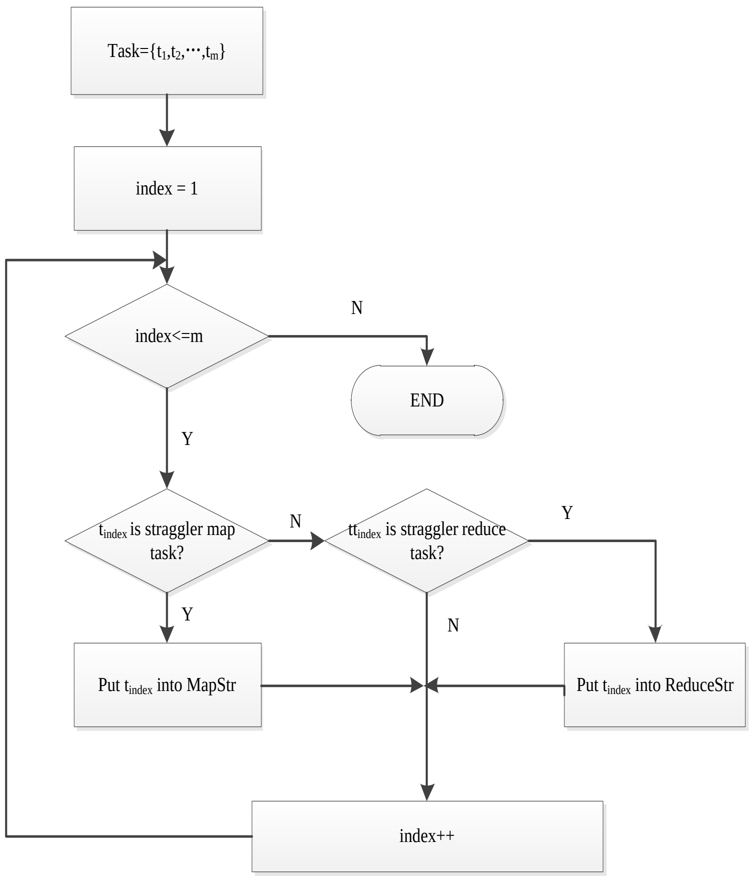
Task={t1,t2,⋯,tm}
index = 1
index<=m
N
END
Y
tindex is straggler map task?
N
ttindex is straggler reduce task?
Y
Y
N
Put tindex into MapStr
Put tindex into ReduceStr
index++

Figure 4. Count Up the Map Task and the Reduce Task which are Left Behind

The LBFS Load Balancing model algorithm pseudo code is as following

Algorithm2: LBFS(Load Balancing based on Fair Scheduler)

```
1:Define the TaskTrackers set as $TT = \{tt_1, tt_2, \ldots, tt_m\}$
2:Define the tasks set as $T = \{t_1, t_2, \ldots, t_n\}$
3:index_TT = 0
4:index_T = 0
4:while index_TT < m
5:     if $tt_{index\_TT} \in$ SlowMap then
6:          if $tt_{index\_TT} \in$ SlowReduce or bkt_num > bkt_limit
7:               change bkt_limit according to network load
8:          else
9:               distribute one Task from ReduceStr to this TaskTracker
10:         end if
11:    else if bkt_num > bkt_limit then
12:         change bkt_limit according to network load
13:    else
14:         distribute one Task from MapStr to this TaskTracker
15:    end if
16:    index_TT++
17:end while
```

LBFS defines a TaskTracker node set TT, and also defines the task set T. T traverse the TaskTracker set and judge whether the node is in SlowMap set, if in the SlowMap set, it will judge whether the TaskTracker node belongs to the SlowReduce set or the number of backup task exceed the upper limit, if it is, then adjust the bkt_limit value according to the network load, if not, then select an appropriate task to the TaskTracker from ReduceStr set. If the TaskTracker node is not in SlowMap set, then judge bkt_num is greater than the bkt_limit, if it is greater, then adjust thebkt_limit value. If the node is not a member of the SlowMap set and the bkt_num is not greater than the bkt_limit, then choose an appropriate task to the TaskTracker node from MapStr set, the index_TT progressive increase, traverse the next TaskTracker.

The job processing mechanism of the LBFS is in accordance with the Fair Scheduler. The difference is when the job is cut into many tasks, LBFS can keep each TaskTracker node load balancing. Fast nodes can process more tasks, slow nodes can process less tasks. LBFS load balance diagram is shown in Figure 5.

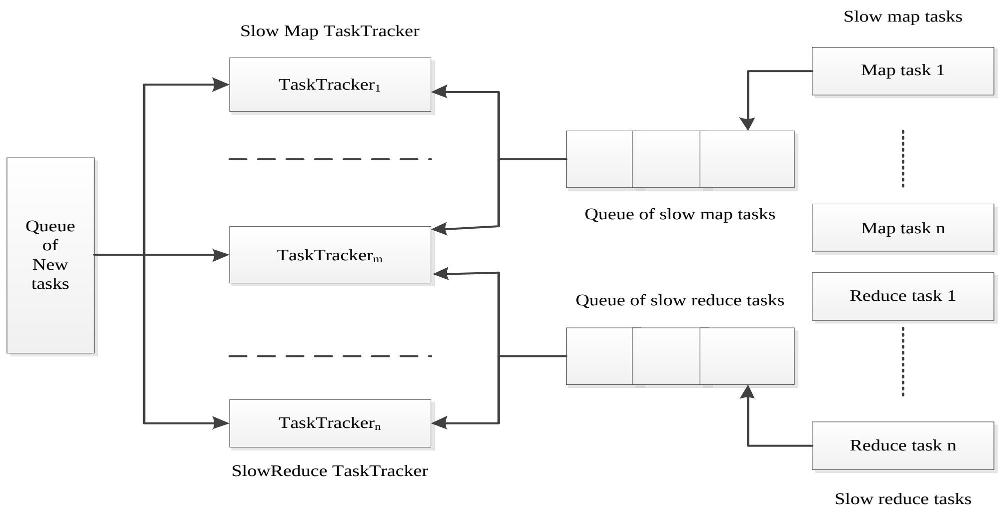

Figure 5. LBFS Load Balance Chart

3. Experiment Results and Analysis

Firstly copying the scheduler package LBFSScheduler.jar to the Hadoop root catalog lib,and modify the configuration file ~/conf/mapred-site.xml, Replacing the default scheduling algorithm, the modified file are as following:

```
<property>
<name>mapred.jobtracker.taskScheduler</name>
<value>org.apache.hadoop.mapred.LBFSScheduler</value>
</property>
```

In order to verify the improved LBFS scheduling algorithm, we select 20G text files, using WordCount count up the words frequency. This experiment compare the LBFS scheduling algorithm and Fair Scheduler (FS), LBFS and FS process the jobs on the Hadoop cluster system 10 times and record the time after completing scheduling the job. The related configuration of the Hadoop cluster system is the same as the configuration of validation PCSP scheduling algorithm, such as the value of the Block Size is 64M, dfs.replication value is 3. After running WordCount 10 times, FS and LBFS running time comparison is shown in Figure 6.

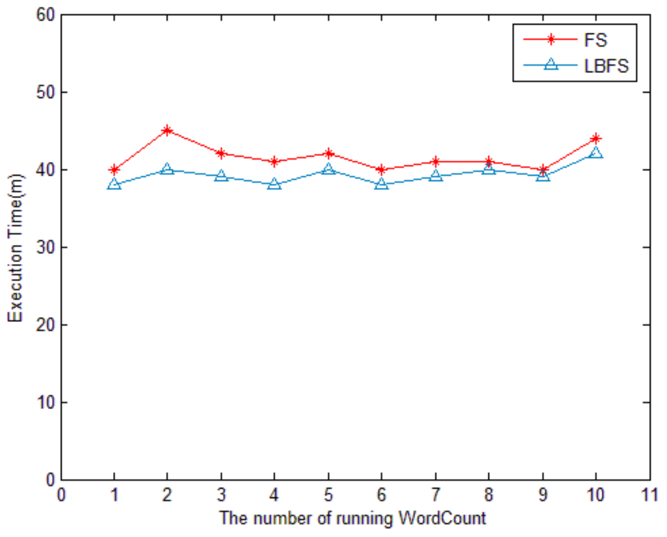

Figure 6. The Time Comparison between FS and LBFS

From the scheduling results, we can be find that LBFS time overhead is always lower than the FS. For 20 g jobs, FS spending at least 40 minutes, while the highest time overhead for LBFS is only 38 minutes. And with the increase of experiments number, LBFS scheduling time overhead tends to be stable, basically in 37 minutes floating up and down. Therefore, LBFS scheduling algorithm has better convergence compared with the FS, and more stable on performance.

For 20G text files, as the PCSP scheduling algorithm, we divided the 20G file into the 4G, 8G, 12G, 16G, 20G, the five jobs, respectively using the LBFS and FS scheduling these five different sizes jobs, and two algorithms have the same experimental conditions and configuration. The experimental results are shown in Figure 7.

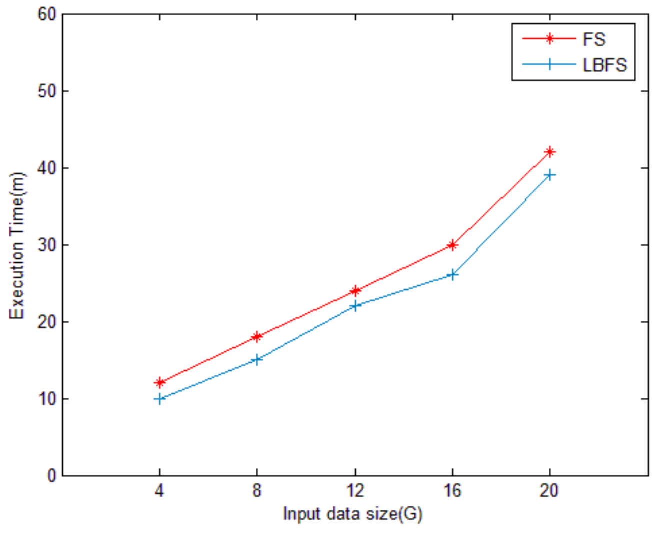

Figure 7. FS and LBFS Performance Comparison

From Figure 7, we can find that after LBFS and FS respectively scheduling different scale jobs, LBFS time overhead is significantly below the FS. It is visible that LBFS load balancing algorithm can make full use of the resources in the cluster system. The optimization LBFS algorithm is more conform to the needs of the production jobs.

In Figure 8, we use 1 GB WordCount job. Each job configure with different algorithms, namely, FIFO, CS and LBFS scheduling algorithm, processing WordCount jobs, each scheduling algorithm run the job 10 times and record the running time. Three kinds of scheduling algorithm have the same environment configuration in order to avoid different configuration to influence the result of the experiment.

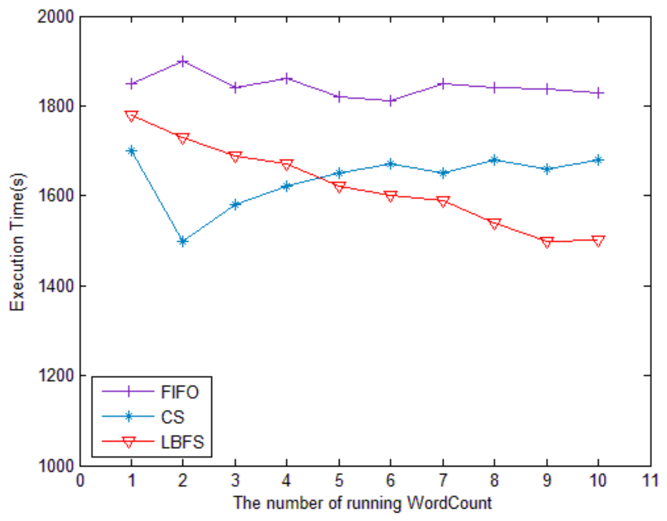

Figure 8. FIFO, CS and LBFS Performance Comparison

As we can see from the Figure 8, overall, FIFO running time is higher than LBFS and CS. However CS and FIFO algorithm performance are volatile. With the learning time increasing, the LBFS scheduling algorithm running time is basically tend to 24 minutes.

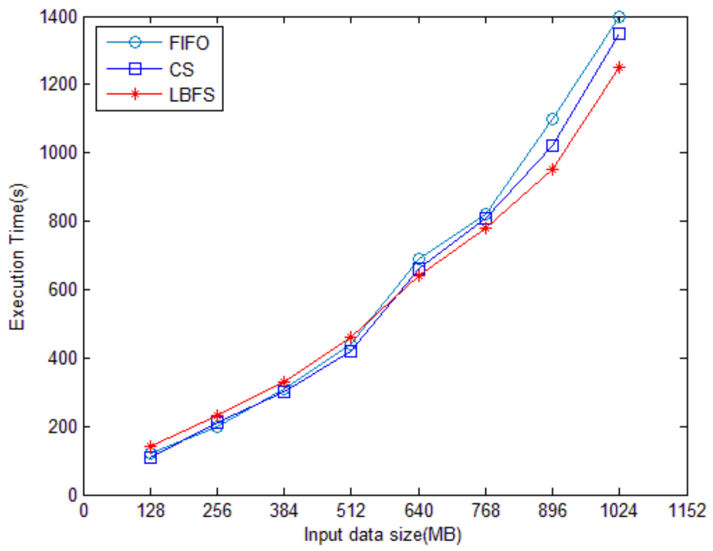

Figure 9. FIFO, CS and LBFS Performance Comparison

To the 1GB WordCount job, we divided the 1G file into the 128M, 256M, 384M, 512M, 640M, 768M, 896M, 1024M, the eight jobs. The BlockSize is 16. From Figure 9, with the data size increasing, the overhead raising speed of FIFO and CS is higher than LBFS.

4. Conclusion

The experimental results show that the LBFS optimize the original scheduling algorithm. According to the questions, putting forward the load balancing model based on fair share scheduling, and successfully deployed on the Hadoop. The experimental results show that: (1) the LBFS time overhead is better than fair share scheduling algorithm. (2) LBFS make up the limitations of the fair share scheduling algorithm. In the future study, we will consider the influence of the MapReduce data distribution in the heterogeneous system.

References

[1] M. Isard, M. Budiu and Y. Yu, Dryad:, "distributed data-parallel programs from sequential building blocks", ACM SIGOPS Operating Systems Review. ACM, **(2007)**, pp. 59-72.

[2] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters", Communications of the ACM, **(2008)**, pp. 107-113.

[3] L. Wang, J. Tao and R. Ranjan, "G-Hadoop: MapReduce across distributed data centers for data-intensive computing", Future Generation Computer Systems, **(2013)**, pp. 739-750.

[4] S. Ibrahim, H. Jin, L. Lu, "Maestro: Replica-aware map scheduling for mapreduce", Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on. IEEE, **(2012)**, pp. 435-442.

[5] D. Yoo and K. M. Sim, "A comparative review of job scheduling for MapReduce", Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on. IEEE, **(2011)**, pp. 353-358.

[6] B. Aksanli, J. Venkatesh and L. Zhang, "Utilizing green energy prediction to schedule mixed batch and service jobs in data centers", ACM SIGOPS Operating Systems Review, **(2012)**, pp. 53-57.

[7] Z. Dadan, W. Xieqin and J. Ningkang, "Distributed Scheduling Extension on Hadoop", Cloud Computing. Springer Berlin Heidelberg, **(2009)**, pp. 687-693.

[8] D. Yoo and K. M. Sim, "A comparative review of job scheduling for MapReduce", Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on. IEEE, **(2011)**, pp. 353-358.

[9] L. Liu, Y. Zhou and M. Liu, "Preemptive hadoop jobs scheduling under a deadline", Semantics, Knowledge and Grids (SKG), 2012 Eighth International Conference on. IEEE, **(2012)**, pp. 72-79.

[10] A. Hadoop, "Capacity Scheduler Guide", http://aechive,clouder, com/cdh/3/hadoop/apacity_ scheduler, html, **(2012)** April 25.