A Task Scheduling Algorithm for Hadoop Platform

Jilan Chen
College of Computer Science, Beijing University of Technology, Beijing, China
Email: bjut_chen@126.com

Dan Wang and Wenbing Zhao
College of Computer Science, Beijing University of Technology, Beijing, China
Email: {wangdan, zhaowb}@bjut.edu.cn

***Abstract*—MapReduce is a kind of software framework for easily writing applications which process vast amounts of data on large clusters of commodity hardware. In order to get better allocation of tasks and load balancing, the MapReduce work mode and task scheduling algorithm of Hadoop platform is analyzed in this paper. According to this situation that the number of tasks of the smaller weight job is more, while that of the larger weight job is less, this paper introduces the idea of weighted round-robin scheduling algorithm into the task scheduling of Hadoop and puts forward the weight update rules through analyzing all the situations of weight update. Experimental result indicates that it is effective in making task allocation and achieving good balance when it is applied into the Hadoop platform which uses only JobTracker scheduling.**

***Index Terms*—Cloud computing, Hadoop, MapReduce, Task scheduling**

I. INTRODUCTION

Cloud computing is a large-scale distributed computing paradigm that is driven by economy of scale, in which a pool of abstracted, virtualized, dynamical-scalable, managed computing power, storage, platforms and services are delivered on demand to external customers over the Internet [1-2]. So far, Google, Microsoft, IBM, Amazon and other IT business giants launched their cloud computing platforms and viewed the cloud computing as the future development of one of the most main strategy[3]. Therefore, the cloud computing research not only follows the trend of the development of the technology industry, also has a high application value [4-5].

At present, most cloud computing systems use Hadoop to develop and schedule programs, which is the open source programming development platform based on MapReduce model[6] and task scheduling tool. Hadoop [7] is a distributed computing framework, and it can run applications in the cluster that consists of a large number of inexpensive hardware. Hadoop can provide a stable and reliable interface for the application and build a distributed system with high reliability and good scalability for users. Hadoop also provides a reliable shared storage and analysis system. Hadoop Distribute File System (HDFS) [8] implements storage, while MapReduce [9] realizes analysis processing, the two part are the core of Hadoop.

However, Hadoop is still a new framework that needs to be improved in some aspects[10].Task Scheduling technology, one of the key technologies of Hadoop platform, mainly controls the order of task running and the allocation of computing resources, which is directly related with overall performance of the Hadoop platform and system resource utilization. Default scheduling algorithm that Hadoop platform provides is FIFO. The advantages of FIFO include simple idea and easy to be executed, light workload of job server,.etc. The disadvantages of FIFO lie in ignoring the different needs by different operations. For example, if a job analyzing massive data occupies computing resources for a long time, then subsequent interactive operations may not be processed timely. Therefore, this situation may lead to long response time and affect the users' experience.

Through analyzing the MapReduce work mode and studying the Hadoop platform architecture and existing task scheduling algorithm, this paper proposes an improved weighted round-robin task scheduling algorithm (IWRR) which is easy to be understood and implemented.

Section 2 below contains a brief review of some related research. Section 3 introduces the task scheduling of Hadoop. Section 4 describes the IWRR task scheduling algorithm including main idea and main process. Section 5 describes algorithm design and experiment result. We summarize our plan for future works in Section 6.

II. RELATED WORKS

Aiming at problems that FIFO scheduling algorithm exists, Facebook [11] and Yahoo's [12] engineers put

Manuscript received April 25, 2012; revised May 25, 2012; accepted May 15, 2012
Corresponding author:wangdan@bjut.edu.cn

doi:10.4304/jcp.8.4.929-936

forward new task scheduling algorithms respectively, which are fair scheduling [13] and capacity scheduling [14]. These two algorithms have been widely accepted by Hadoop community and been added into the new version of Hadoop. At present, a number of scholars at home also made related researches for task scheduling algorithm of the Hadoop platform. Reference [15] proposed a job scheduling algorithm based on Bayesian classification, and its most important feature was that it provided a learning stage based on Bayesian classification without presetting by learning. Reference [16] proposed a waiting scheduling based on the length of waiting-time. It made task execute more selective without strict order of queue, thereby the issues of local data was improved. Reference [17] presented a weight fair queuing scheduling algorithm based on MapReduce cluster to ensure fairness for tasks to some extent.

Since Hadoop platform uses only JobTracker scheduling, massive jobs might bring JobTracker heavy workload, thus the scheduling algorithm shouldn't be complex. The IWRR task scheduling algorithm proposed in this paper is relatively simple.

III. Task Scheduling Technology of Hadoop

Hadoop implements the Google's MapReduce programming model. MapReduce is a distributed computing model as well as the core technology in Hadoop,which has been widely used into program distributed parallel application.

Map and Reduce are two important concepts of MapReduce [18].

Map, written by the user, takes an input pair and produces a set of intermediate key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key *I* and passes them to the *Reduce* function.

The *Reduce* function, also written by the user, accepts an intermediate key *I* and a set of values for that key. It merges these values to form a possibly smaller set of values.Typically just zero or one output value is produced per Reduce invocation.The intermediate values are supplied to the user's reduce function via an iterator.This allows us to handle lists of values that are too large to fit in memory.

MapReduce is actually the process of "task decomposition and collect the results". Map decomposes the job into a number of tasks, Reduce sums up the result that multi tasks, and then gets the final results.

Besides, the following terms of Hadoop MapReduce are ofter used: (1)*job*. A job refers to every computing request for a user; (2)*Task*. A task refers to the split small parts of a job executing in a server; (3)*JobTracker*. A JobTracker refers to the Server receives users' jobs. It is also responsible for the various tasks allocation and the management of all tasks servers; (4)*TaskTracker*. A TaskTracker is responsible for executing specific tasks; (5)*Slot*. It is responsible for executing a specific task. A task server may have multiple slots.

The MapReduce computing architecture of Hadoop is shown in Figure 1.

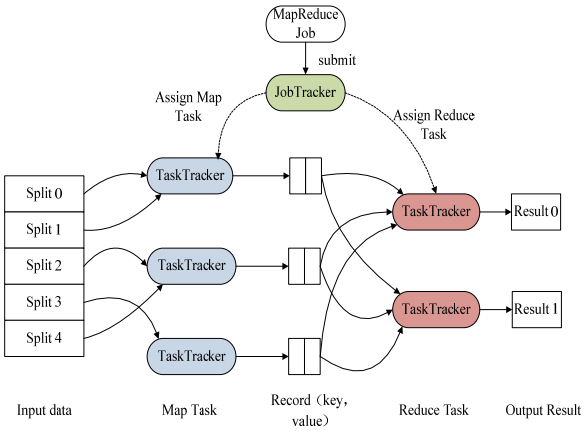

Figure 1. The MapReduce computing architecture of Hadoop

From the above Figure 1, *MapReduce* system architecture [19] is composed of *JobTracker* and *TaskTracker* service scheduling. *JobTracker* is called the job server, responsible for managing all jobs running under the framework as well as responsible for scheduling and managing *TaskTracker*. It assigns the *Map* tasks and *Reduce* tasks to idle *TaskTracker*, and makes these tasks run in parallel, and is responsible for monitoring the operational aspect of the task. *TaskTracker* is called task server, which is the core that each job assigns tasks. It is responsible for performing the task, each job is split into many tasks including *Map* tasks and *Reduce* tasks. Tasks are the basic unit of specific implementation, all of them need to be assigned to the appropriate server to perform. *TaskTracker* implements task and at the same time reports all tasks' operational aspect to *JobTracker* to help *JobTracker* understand the overall situation and distributes new tasks etc. If a *TaskTracker* fails, *JobTracker* will allocate tasks to other idle *TaskTracker* to rerun.

From the above Figure 1, calculation process of the whole MapReduce job consists of four steps including job submission, distribution & perform of Map tasks, assignment distribution and execution of Reduce tasks, and job accomplished. Execution of each task contains three steps including input prepare, algorithm execution and output. The task scheduling algorithm has a direct impact on the task execution.

Task scheduling for Hadoop platform means that allocating the appropriate tasks of appropriate job to appropriate task server. That contains two steps. The first step is to select the job, and then to select the task for this job. Poor distribution of tasks may lead to increase network traffic and unbalanced load, and so on. After Hadoop 0.19 version, the task scheduler acts as a pluggable component and becomes independent, which makes it convenient for users to provide different implementations according to their requirements.

Regarding Hadoop's MapReduce [20][21], it is *JobTracker* to submit a job by *JobClient.rubJob (job)* method. After *JobTracker* receives *JobClient's* request, it will add it into the job queue. JobTracker continuously waits for JobClient to submit jobs to it by RPC, while TaskTracker continuously sends the heartbeat signal to

JobTracker by RPC and inquiries whether there are tasks need to be executed. If there are tasks needing to be executed, *TaskTracker* will request *JobTracker* to assign it some tasks. If *JobTracker's* job queue is not empty, the *TaskTracker* will get some tasks assigned by *JobTracker*. This is an actively request process. *TaskTracker* requests initiatively JobTracker for tasks. When a *TaskTracker* receives the assigned task, it will establish corresponding task through its own scheduling in the node. Figure 2 shows the process that *MapReduce* task request scheduling.

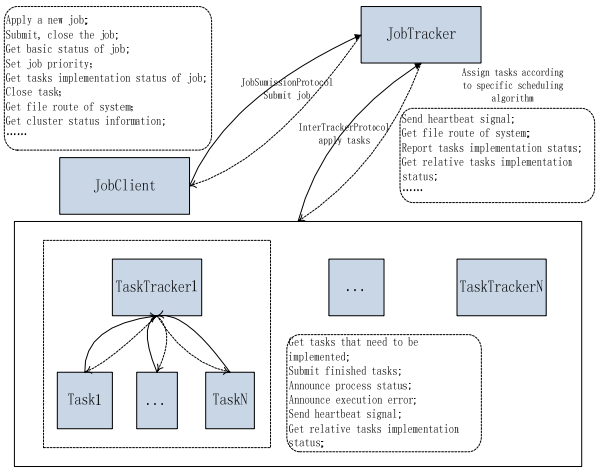

Figure 2. Process of *MapReduce* task request scheduling

From Figure 2, we can see that the Hadoop platform only uses *JobTracker* to execute schedule. Massive jobs are submitted to *JobTracker* which will bring *JobTracker* heavy workload. Therefore, the scheduling algorithm itself for the Hadoop platform needn't to be complex. This paper introduces WRR algorithm into Hadoop platform which is relatively simple and real-time.

IV. COMPARATIVE ANALYSIS OF EXISTING SCHEDULING ALGORITHM

From the second section, we know, the existing scheduling algorithm mainly includes FIFO scheduling algorithm, fair scheduling algorithm(Fair Scheduler) and computing scheduling algorithm (Capacity Scheduler), here we analyze the advantage and disadvantage of three algorithms.

A. FIFO Scheduling Algorithm

In the earliest Hadoop MapReduce computing architecture, the basic job type is large batch jobs that a single user submits, so Hadoop use FIFO (First in First out) algorithm in early scheduling algorithm[22]. The jobs of all users are referred to only one queue. According to the priority level and the time sequence when they are submitted, the entire job queue are scanned, and then a satisfactory job is selected to execute. FIFO is simple, the cost of entire cluster scheduling process is less.

Although FIFO scheduling algorithm is simple, there are a lot of limitations. It is designed only for single type of job, so when multiple users simultaneously run multiple types of jobs, performance will be relatively low. As the usage rate of Hadoop platform is increasingly high, the demand is also increased. The FIFO algorithm tends to decrease the overall performance of the platform and the utilization of system resources, and sometimes even affect the implementation of jobs. For example, if the production jobs use cluster resources for long term, it will cause the batch job of other users can't tolerant so long response time, also make interactively query of other users unscheduled for a long time. As a result, the system's interactive capabilities and the user experience is seriously affected.

B. Fair Scheduling Algorithm

Fair scheduling algorithm will make work group form the job pool according to the configured attributes (such as name, the located group), and allocate minimum shared resource number to each job pool (also can use the number of map and reduce slots to define minimum capacity of user pool, also set each pool weight). If in accordance with the minimum number of shared resources it can't allocate all resource, it will assign extra resources to each job evenly. Fair share scheduling algorithm supports job classification scheduling, so that different types of jobs obtain different resources, thereby the quality of service (QOS) is improved and paralleling number is dynamic adjusted. Therefore, it makes the job fully use system resources, and improves utilization degree of system. Meanwhile, it overcomes the defects about the FIFO algorithm such as simple, non-preemption, low resource utilization rate, without taking into account the load level of all nodes of current system and the actual state of the load, causing the actual load of the node unbalanced, thereby affecting the response time of the whole system.

C. Capacity Scheduling Algorithm

Capacity scheduling algorithm puts jobs into multiple queues in accordance with the conditions, and allocates certain system capacity for each queue. If a queue has heavy load, it seeks unallocated resources, then makes redundant resources allocated evenly to each job. Compared with the FIFO algorithm, capacity scheduling algorithm overcomes the FIFO's disadvantage such as the low utilization rate of resources. Furthermore, it supports multiple jobs to execute in parallel to improve the utilization rate of resources through the dynamic adjustment of resource allocation as well as the job efficiency. The queue set and queue selecting group of capability scheduling algorithm can not carry out automatically, the user needs to know system information and make queue set and queue select group for the job. In a large system, it will be one of big bottleneck of improving the overall performance of the system.

V. TASK SCHEDULING ALGORITHM BASED ON IMPROVED WEIGHTED ROUND-ROBIN

A. Weighted Round-Robin Algorithm

The basic idea of our weighted round-robin strategy [23] is to allocate a weight to each queue, then scheduling

tasks of different sub-queue according to weight. For the WRR scheduling algorithm, it can provide good fairness when the size of each task is same. Whereas it will bring unfairness for the smaller queues if the size of task is inconsistent. The advantage of WRR algorithm includes small overhead and easy to be implemented. However, due to the weight is fixed, WRR scheduling algorithm can't adjust the weight of each sub-queue in real time.

B. Improved WRR Algorithm for Hadoop

Based on the analysis of WRR algorithm, this paper puts forward the IWRR algorithm. The following begins to discuss the task allocation of Hadoop.

Hadoop platform puts running MapReduce job into a queue. Under the unweighted conditions, tasks of each job are submitted to TaskTracker in turn. Under the weighted conditions, multiple tasks of the larger weight job will run in a round, and the job's weight will be changed along with the increase or decrease of jobs' number. At times, if the number of tasks of the smaller weight job become more, while the number of the larger weight job is less, then the weight of the smaller weight job will be increased correspondently, so the number of tasks which are assigned to TaskTracker will be relatively increased, and the weight of the larger weight job will be appropriately decreased, the number of tasks which are assigned to TaskTracker will be relatively decreased. However, the relationship between them remains the same in order to achieve load balance.

Weight update rule we proposed is described as follows.

The Basic idea lies in allocating resources for the larger weight job and the smaller weight job on average as far as possible. Of course, the larger weight job and the smaller weight job may not absolutely share their resources evenly, we just try to make resources travel between jobs with larger weight and smaller weight. The job with smaller weight has relatively more resources so as to shorten the distance between them .

Following part analyzes how to make the larger weight job and the smaller weight job allocate resources on average as far as possible. The precondition is that the larger weight job always has more weight than the smaller weight job, and always occupies more resources.

Assuming the initial value of weight change is $\triangle w$. B represents the larger weight job, S represents the smaller weight job.

Considering the following situations：

(1) $B, S + \triangle w$

If the resource amount of the larger weight job is fixed, for the smaller weight job, we increases the amount of resource of $\triangle w$ because the premise condition is that the larger weight job always has more weight than the smaller weight job. To meet $S+\triangle w < B$, the smaller weight job relatively will have more resources. This is the first circumstance of weight update role.

(2) $B + \triangle w, S$

If we increase the larger weight job by the amount of resource of $\triangle w$, and the smaller weight job remain the same weight, the resources of the smaller weight job will become fewer and fewer for the larger weight job.

(3) $B + \triangle w, S + \triangle w$

The larger weight job and the smaller weight job are increased by the amount of resource of $\triangle w$, so the larger weight will have more resources. Although the smaller weight job's resource is also increased, they don't tend to be the average, the larger weight job will have more and more resources.

(4) $B + \triangle w, S - \triangle w$

The result will be the same as the second condition.

(5) $B, S - \triangle w$

The result will be the same as the second condition.

(6) $B - \triangle w, S$

The job with larger weight has more resources which makes the uneven distribution of resources. If the weight of job with larger weight is decreased by $\triangle w$, its relative resources will be reduced without changing the premise condition between them ($B-\triangle w > S$), so the resource gap between them relatively become less. This is the second circumstance of weight update role.

(7) $B - \triangle w, S - \triangle w$

$B-\triangle w>S-\triangle w$, the resource gap between them can't be changed in this case.

(8) $B - \triangle w, S + \triangle w$

The resource of larger weight job is decreased by $\triangle w$, the resource of smaller weight will be increased by the amount of resource of $\triangle w$, so the gap between them will be decreased. This is the third kind of circumstance of weight update role.

From the above several circumstances, three conditions of weight update rule can be summed up as:

（1）$B,S+\triangle w$

（2）$B-\triangle w,S$

（3）$B-\triangle w,S+\triangle w$

The following of the paper uses a simple example to prove that the above three ways can reduce the gap between them better.

Assuming the initial value of B is 20, the initial value of S is 11, $\triangle w$ is 4, then the changed B' and S' are like:

（1）$B'=B=20,S'=S+\triangle w=11+4=15$;

（2）$B'=B-\triangle w=20-4=16,S'=S=11$;

（3）$B'=B-\triangle w=20-4=16,S'=S+\triangle w=11+4=15$;

By the way (1) and (2), the gap between B' and S' changes from 9 to 5, the distance is narrowed obviously; In the way (3), the gap between B' and S' changes from 9 to 1. Because the larger weight job always has more weight than the smaller weight job, the relations between them can't be changed. By the way (1) and (2), $\triangle w<B-S$. When $\triangle w = 5$, by the way (3), $B'=15,S'=16$, so that $B'<S'$ is against the premise conditions. The above information can prove that this improvement is a good way to reduce the difference between the job with smaller weight and the job with larger weight to meet the condition $\triangle w <(B-S)/2$.

C. Main Process of Algorithm

Step1: When TaskTracker has leisure resources, it will initiatively submit a task allocation request to JobTracker. The request includes resource information on TaskTracker, such as network transmission, receiving

rate, free physical memory size, CPU utilization, I/O read and write speeds as well as the rest of the disk size, etc.

Step2: When JobTracker receives task allocation request from TaskTracker, it will schedule a task from the current job queues to the requested TaskTracker, and then update the remaining task information of this job, and the number of tasks is decreased by 1.

Step3: If the number of tasks of the job assigned is less than 1 after step2, the next job can be scheduled, otherwise, the current job is still scheduled waiting for the arrival of the next TaskTracker request.

Step4: When a round is over, if a job completes or a new job arrives, the program needs to update the job queue information and the weights of each job according to the weight update rule, and to recalculate assigned task number of each job, repeat steps 1-4.

VI. IMPLEMEMTATION AND EVALUATION

A. Design and Description

The scheduling process involves two queues including *jobQueue* and *taskQueue*. Elements of *jobQueue* are defined in *jobInfo*, all the elements of each *taskQueue* are correspondent with the map tasks or reduce tasks of a job. After the Hadoop adds the IWRR task scheduling strategy, the relationship among classes of the total task scheduling is shown in Figure 3.

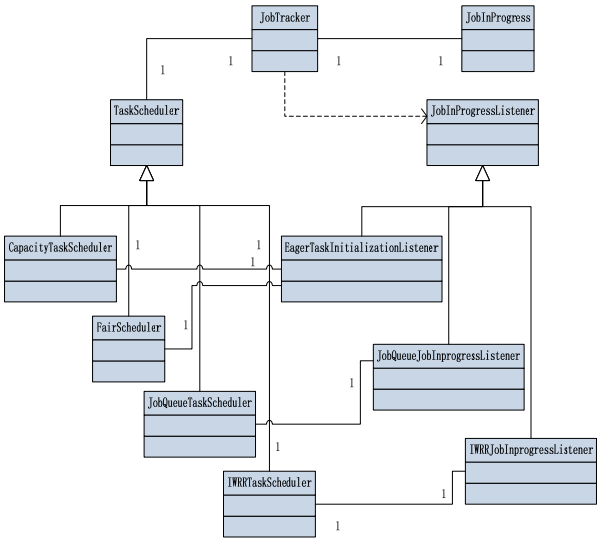

Figure 3. The class diagram of task scheduling of Hadoop

The following section introduces the specific design of *IWRRTaskScheduler*.

1) Data Structure Design

The default schedule policy of the Hadoop is defined in *JobQueueTaskScheduler*, the data structure of each element of scheduling queue is included in *JobSchedulingInfo*, which is defined as follows:

```
static class JobSchedulingInfo {
JobId  jobId;
JobPriority  priority;
long  startTime;
}
```

Our algorithm adds a few variables into *JobSchedulingInfo* class, namely:

taskNum: the task number of each job, including maptask and reducetask.

Weight: weight of job.

RoundTasks: the scheduling tasks number in each round.

2) Weight Computing Method

By default, the weight is based on the job priority. Computing method of the initial weight refers to the method that a fair scheduler provided in Reference [24], computing weight based on the size of the job. The method is described as follows:

$$\text{weight}=\begin{cases}\log_2(taskNum+1), & \text{considering job size}\\ 1, & otherwise\end{cases} \quad (1)$$

Then computing weight according to priority by formula(2):

$$\text{weight}=\text{weight} * \text{priorityFactor} \quad (2)$$

priorityFactor refers to the factor associated with job priorities, which is used to calculate the weight of the job.

The specific value is shown in Table 1.

TABLE Ⅰ.
THE SPECIFIC VALUE OF PRIORITY FACTOR

priority	priorityFactor
VERY_HIGH	4.0
HIGH	2.0
NORMAL	1.0
LOW	0.5
Default	0.25

3) Algorithm Description

The class to implement all scheduling algorithms inherit class TaskScheduler, the key part of class *TaskScheduler* is the task allocation strategy which is reflected in the *assignTasks* method.

The *assignTasks* method is described as follows.

```
Input: TaskTracker
Output: List<Task>

1   for (JobInProgress job: jobQueue) do
2    for (JobInProgress job:jobQueue) do
3     If job.status==RUNNING then
4   Compute remaining numbers of map and reduce task;
5     end if
6    end for
7   if remaining map and reduce task numbers==0 then
8    return NULL
9   else
   10        Compute the 'load factor' for maps and reduces,remaining /numTaskTrackers
11  /*In the below steps,we allocate first map tasks (if appropriate) */
                          12   for(JobInProgress job:jobQueue) do
13   if running map task numbers on TaskTracker< mapLoadFactor then
14    job.obtainNewMapTask();   /*assign map task of current jobInfo*/
15   end if
16   //Then we allocate reduce tasks(if appropriate)
17   if running reduce task numbers on TaskTracker < reduceLoadFactor then
18    job.obtainNewReduceTask();  /*assign reduce task of current jobInfo*/
19   end if
20   /*update current jobInfo information*/
21   if updatedRoundTasks<1 then
22    JobID++;
23   end if
24   if current job is the end of jobQueue then
25    Compute weights of all jobs and RoundTasks of each job;
26   end if
27  end for
```

The way to compute the weight and RoundTasks in the last step is described as follows: The value of *RoundTasks* is obtained by the method *Min (weight, taskNum)*. *RoundTasks* is the smaller value between the under integer of weight and the remaining *taskNum* of this job; The value of weight is gotten by the method updateWeight ($\triangle w$) , $\triangle w$ is the amount of weight change.

The updateWeight method is described as follows:

```
Input: △w
Output: null
1   /*factor is the key of determining weight
    change*/
2   factor=task numbers/assigned task numbers
3   if factor of larger weight job1 < factor of
    smaller weight job2
     then
4    job2 weight+=△w
5    job1 weight-=△w
6   else
7    job weight remains the same
8   end if
```

B. Experiment Environment

In order to investigate the efficiency of the proposed algorithm, experiments have been performed within local campus network. In this section, Cloud Experimental Platform of Beijing University of Technology, which is established in cooperation with IBM, is used to verify IWRR algorithm. This experiment applied for eight nodes, one node is JobTracker, other seven nodes are TaskTrackers. Each node needs to install four softwares including Redhat Linux, Hadoop-0.20.2, jdk 1.6.0 and eclipse-3.5.2.

The experimental environment is shown in Table 1.

TABLE II
EXPERIMENTAL ENVIRONMENT

hostname	IP	role
hadoop08	172.21.14.194	Namenode,master,jobTracker
hadoop01	172.21.14.196	Datanode,slave,taskTracker
hadoop02	172.21.14.166	Datanode,slave,taskTracker
hadoop03	172.21.14.110	Datanode,slave,taskTracker
hadoop04	172.21.14.100	Datanode,slave,taskTracker
hadoop05	172.21.14.139	Datanode,slave,taskTracker
hadoop06	172.21.14.130	Datanode,slave,taskTracker
hadoop07	172.21.14.101	Datanode,slave,taskTracker

C. Experiment and Evaluation

In order to make FIFO scheduling algorithm change into the IWRR scheduling algorithm, we first need to put the scheduler *IWRRTaskScheduler.jar* into

HADOOP_HOME/lib directory, then modify the configuration file *hadoop-site.conf* to join IWRR scheduling module.

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

We compare WRR algorithm with IWRR algorithm on Hadoop. Supposing there are three running jobs. In order to better verify experimental results, this paper sets the priorityFactor value of three jobs 0.25 , 2, 4 respectively.

We know that the default size of HDFS Block is 64M, in order to accelerate the process of experiment, we l modify it to be 16M.

Table III shows the information of all jobs.

TABLE III
THE INFORMATION OF ALL JOBS

	Job1	Job2	Job3
JobSize	1024M	512M	256M
Block	64	32	16
Map	64	32	16
Reduce	1	1	1
taskNum	65	33	17
priorityFactor	0.25	2	4
weight	1.5	10	16

Based on the formula (1), we can compute the Jobs weight by the taskNum and priorityFactor.

According to the task management page of MapReduce platform, we can see the job operation conditions. At time 200s, we record the result. The results about comparing algorithm WRR with IWRR is shown in Figure 4.

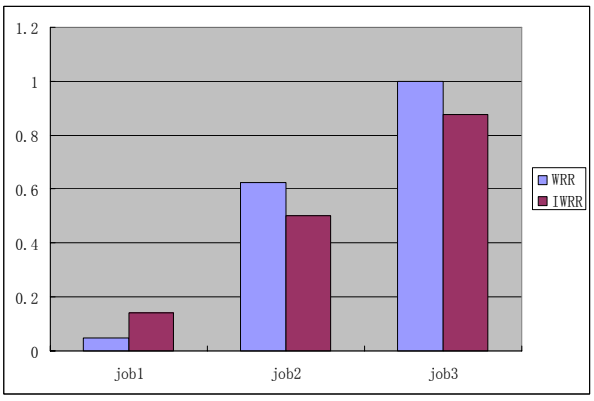

Figure 4. Comparison between WRR with IWRR

Blue bar represents WRR algorithm, red bar represents IWRR algorithm. X-coordinate represents the job1, job2 and job3, Y-coordinate represents the ratio of the number of allocated tasks for each job and the total tasks. It can be seen from the above results that the number of assigned tasks for job1 is increased relatively, job2 is reduced relatively, job3 is also reduced relatively. When the number of tasks in job1 becomes relative bigger, job3 will not remain more tasks in the queue so as to balance task allocation. Although the improvement of the WRR algorithm uses a fixed weight change($\triangle w$),researchers can predict or assess dynamically value instead of using measuring device or other complex means to determine the value.

VII. CONCLUSION

This paper analyzed and discussed the task allocation of Hadoop platform, and presented an improved weighted round-robin scheduling algorithm which used weight update rules to reduce workload and to balance tasks' allocation. The algorithm is easy to be implemented with low cost and suitable for the Hadoop platform that uses the only JobTracker to schedule. There are still shortcomings and defects including without considering external influence on the time that each task took when they switched scheduling. In addition, due to experimental limitations, we did not investigate the stability of the algorithm under the environment of high concurrency, large capacity and high workload. Further work is to improve the weight update rules considering all factors and make extensive experiments.

ACKNOWLEDGMENT

This research is supported in part by a grant from Beijing Municipal Natural Science Foundation of China (Grant No. 4122007).

REFERENCES

[1] F. Ian, Z. Yong, R. Ioan, L. Shiyong, "Cloud Computing and Grid Computing 360-Degree Compared," *Grid Computing Environments Workshop*,1-10.2008.

[2] [2] B.M Xu, N.Wang, C.Y. Li. "A Cloud Computing Infrastructure on Heterogeneous Computing Resources".*Journal of Computers*, Vol 6, No 8 (2011), l789-1796, 2011

[3] M. V. Luis, R. M. Luis, C. Juan, "A break in the Clouds: Towards a Cloud Definition," *ACM SIGCOMM Computer Communication Review*.39(1).50-55.2009.

[4] D. Hamilton, "Cloud computing. Seen as next wave for technology investors," *Financial Post 04.* http://www.financialpost.com/money/story.html?id=562877.2009.

[5] J.H. Gu, J.H.Hu, T.H. Zhao, Guofei Sun. "*A New Resource Scheduling Strategy Based on Genetic Algorithm in Cloud Computing Environment*". *Journal of Computers*, Vol 7, No.1, 42-52, Jan 2012

[6] J. Dean, S. Ghemawat, "Map/Reduce: Simplified Data Processing on Large Clusters,"*OSDI'04:Sixth Symposium on Operating System Design and Implementation*.2004.

[7] Apache Hadoop.http://hadoop.apache.org/.

[8] K. Shvachko, H. Kuang, S. Radia, "The hadoop distributed file system," *IEEE 26th Symposium on Mass Storage Systems and Technologies*, 2010.

[9] D. S. Parker, A. Dasdan, R. Hsiao, "Map-reduce-merge: simplified relational data processing on large clusters," *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*. 1029-1040.2007.

[10] L. Peng, *Cloud Computing (Second Edition)*.Publishing House of Electronics Industry.189-227.2010

[11] Facebook.http://www.facebook.com.

[12] Yahoo.http://www.yahoo.com

[13] Fair Scheduler for Hadoop. http://Hadoop.apache.org/common/docs/current/Fair_scheduler.html.2009

[14] Capacity Scheduler for Hadoop. http://Hadoop.apache.org/common/docs/current/Capacity_scheduler.html.2009

[15] X. Yi, "Research and Improvement of Job Scheduling Algorithms in Hadoop Platform," *A Dissertation Submitted for the Degree of Master*.45-51.2010

[16] W. Kai, "Research and Improvement of Job Scheduling Method for Multi-User MapReduce Clusters," *A Dissertation Submitted for the Degree of Master*..32-39.2010

[17] T. Qi, K. HuaDong, "Design and Implementation Priority Based Weighted Fair Queue of Based on MapReduce Cluster," *Computer Knowledge and Technology*.2129-2132.2011

[18] J. Dean, S. Ghemawat, "MapReduce: Simplified Data Processing on Large Cluster," OSDI'04, *Sixth Symposium on Operating System Design and Implementation*, SanFrancisco,CA,December,2004

[19] J.Dean, "Experiences with MapReduce, an abstraction for large-scale computation," *In International Conference on Parallel Architecture and Compilation Techique*,2006

[20] Apache MapReduce Architecture. http://Hadoop.apache.org/common/docs/current/mapreduce_design.html.2005

[21] J.Dean, "Experiences with MapReduce, an abstraction for large-scale computation," *In Proc. of International Conference on Parallel Architecture and Compilation Techiques*.2006

[22] M. Isard, M. Budiu, Y. Yu, "Distributed Data-Parallel Programs from Sequential Building Blocks," *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems*, ACM, 59-72. 2007.

[23] Y. Zhang, P.G. Harrison, "Performance of a priority-weighted round robin mechanism for differentiated service networks," *In the Proc. of 16th International Conference on Computer Communications and Network(*ICCCN).1198-1203.2007.

[24] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, I. Stoica, "Job scheduling for multi-user mapreduce cluster," *EECS Department, University of California, Berkeley, Tech. Rep*,2009

Jilan Chen was born in 1986. She received the M.S. degree in Computer science and technology from Beijing University of Technology in 2012. Her research interests include Cloud Computing and trustworthy software.

Dan Wang was born in 1969. She received the Ph.D degree in computer software from Northeastern University in 2002. She is a professor at Beijing University of Technology. Her research interests include trustworthy software, distributed computing, etc.

Wenbing Zhao was born in 1973. She received the Ph.D. degree in Signal and Information Process from Peking University in 2004. She is an assistant professor at Beijing University of Technology. Her research interests include data mining, trustworthy software, etc