Research on Weighted Rotation Fair Scheduling Algorithm Based on Hama Parallel Computing Framework

Zhijie LIN*, Haifeng HUANG

ABSTRACT

With the development of big data technology in the industry and academia, a large number of distributed computing platforms are put into operation, of which the Apache Hadoop platform is the most widely used. Hadoop shields the underlying implementation details of a distributed system, making application developers more focused on algorithmic logic. However, Hadoop has its limitations. Its efficiency is not high when dealing with problems such as graph computing and machine learning. The emergence of the Apache Hama parallel computing framework based on the BSP (Bulk Synchronous Parallel) model makes up for this shortcoming of Hadoop. However, because the time of Hama creation is not long and the time of development is short, it is still necessary to continue to improve in many places, especially its core module—job scheduler. The Hama currently used FCFS (first come first service) job scheduling algorithm, and it is not only unable to efficiently implement multi-user shared clusters, but also has a great impact on resource utilization of clusters.

The purpose of this paper is to implement a Hama parallel computing task scheduling algorithm based on the framework of the new Hama, make up the first come first service scheduling algorithm, improve the level of utilization of the resources, and provide more flexibility for scheduling Hama, to meet the development needs of a period of time in the future. In order to achieve this goal, the main work and contributions of this article are as follows:

First of all, through studying the source code of Hama parallel computing framework and analysing the architecture of Hama, this paper studied the scheduling framework of Hama and its FCFS scheduling algorithm.

Then, based on the above research and combined with the BSP model, this paper proposes the weighted round robin fair scheduling algorithm. Under the premise of ensuring the fairness of the users and jobs, this algorithm combined fair scheduling algorithm with weighted round robin algorithm. By the way, through the fair scheduling among job pools, the weighted round robin scheduling in the job pool, the algorithm ensured the throughput of the system while reducing the load of JobTracker nodes.

*Corresponding author: bytelin@qq.com
Finally, the weighted rotation fair scheduling algorithm is tested in this paper. The experimental results show that the weighted round robin fair scheduling algorithm achieves the desired design goal successfully, and solves the shortcomings of FCFS scheduling algorithm, while improving the resource utilization ability of Hama cluster.

INTRODUCTION

In recent years, big data technology has made great progress under the joint promotion of industry and academia, and a large number of big data systems have been put into operation, and will continue to develop in the future. So far, Apache Hadoop\(^3\) platform based on Google GFS\(^1\) (Google Filesystem) and MapReduce\(^2\) has become the standard platform of big data processing in fact.

Although MapReduce has wide application scenarios, it doesn’t apply to all kinds of jobs, especially in large-scale scientific computing, such as graph computing\(^4\), machine learning, etc. In graph computation and machine learning, many algorithms, such as single source shortest path algorithm in graph computation and K-means clustering algorithm in machine learning, need lots of iterations. In MapReduce, each of these algorithms needs to generate a job to complete. Because MapReduce needs to write the final result to the file system at the end of every job. Therefore, using graph algorithm or machine learning algorithm implemented by MapReduce, a lot of time will be used to execute file I/O. So, using MapReduce to implement this kind of algorithm, its efficiency will be more efficient than single node execution, but it still can't meet the requirement\(^5\).

In order to make up for the shortage of MapReduce and improve the efficiency of graph computation, we completed the computation of webpage ranking with large amount of data (PageRank\(^6\)). Google also developed a distributed platform for processing graph dataPregel\(^5\). Pregel is based on the BSP\(^7\) (Bulk Synchronous Parallel) model. Compared to the MapReduce model, the BSP model is more suitable for the parallel computation of the iterative algorithm. After Google published the idea of Pregel system, Apache Hama\(^8\) parallel computing framework absorbed its core idea and reintegrated Hama’s system architecture. But unlike Pregel, Hama provides a universal BSP model, which can be used not only in graph computing but also for machine learning algorithms.

Hama officially became the top project of Apache open source community in 2013. The emergence of Hama is not a substitute for the existing Hadoop MapReduce platform, but a useful supplement to Hadoop MapReduce platform, making the processing and analysis tools of big data more abundant. However, because of the Hama parallel computing framework for the development of a long time, there is still a lot of deficiencies in many aspects, especially the core module of a job scheduler, just a scheduler based on a first come first serve simple, efficient resource cluster cannot be shared between multiple users, and cluster resource utilization low. In this paper, based on the in-depth study of Hama parallel computing framework, a new weighted round robin fair scheduling algorithm based on Hama parallel computing framework is proposed.

This paper is based on a deep study of the Apache Hama parallel computing framework. The weighted rotation fair scheduling algorithm based on Hama parallel computing framework is used to verify its function.

OVERVIEW OF HAMA PARALLEL COMPUTING FRAMEWORK

Hama is a distributed parallel computing framework based on BSP (Bulk Synchronous Parallel Model) model, which is mainly used for large-scale scientific
computing, including machine learning, graph theory, network and so on. The biggest advantage of the BSP model is that it can speed up the iteration. It can quickly get the feasible solution when solving a lot of iterative algorithm problems in machine learning and graph theory. Similar to Hadoop, Hama also shields the implementation details of distributed system, provides users with easy to use API, and is compatible with HDFS and HBase on storage (mainly HDFS). At present, it is convenient to deploy Hama clusters on Hadoop clusters.

At the initial release, Hama supported three computing models, the MapReduce of Hadoop, the BSP model, and the Microsoft's Dryad model. However, only the BSP model-based computing engine is retained in the subsequent release version, which is the only engine for the Hama parallel computing framework to perform the task. After the release of Google Pregel (based on the BSP model), Hama built a diagram of Google Pregel API on its BSP engine to calculate the API package.

**Hama job scheduling algorithm**

Figure1 is the job scheduling framework that Hama is currently using. The job scheduler runs on the BSP Master node. Hama realizes the plug-in operation of the job scheduler through the TaskSchedule class and the Schedulable interface. Users can customize the new job scheduler by using these two interfaces.

![Figure 1. Hama job scheduling framework.](image)

**WEIGHTED ROTATION FAIR SCHEDULING ALGORITHM**

The design idea of fair scheduling algorithm is to ensure that all users get equal resources as much as possible, so that all running jobs can share all resources equally as much as possible. But in practical applications, the effect is not very good. This paper tries to propose an improved of fair scheduling algorithm to better ensure the fairness of users and jobs, and improve the overall performance of the system.

**Algorithm thought**

The operation pool is organized by the user or the user group, and the weighted rotation method is used in the operation pool to schedule the job. When an idle TaskTracker node applies for a task, the system first selects the job pool and then selects the specific job and task.

First, select the job pool. The selection of the job pool is mainly determined by the minimum shared difference \( \text{MS}_t \) and the fair share difference \( \text{JS}_t \) of the job pool. They are related to the actual amount of resources \( \text{AR}_t \), minimum share line \( \text{MR}_t \) and fair share \( \text{JR}_t \) that the job pool actually owns.

\[
\text{MS}_t = \begin{cases} 
\text{MR}_t - \text{AR}_t, & \text{if } \text{MR}_t > \text{AR}_t \\
0, & \text{else}
\end{cases}
\]
The organization of the job pool queue, which consists of a pool of jobs that require the resource. The pool of jobs that do not need the resource is not organized in the queue. When there is a job in the queue to make a decision, the job pool with the maximum value of the job pool size is selected; otherwise, the job pool that makes the maximum value of the job pool size is selected. That is to say, priority is given to the minimum share of the pool, and then the fair share is met, and each scheduling chooses the largest pool of jobs with the largest deficit.

Second, select the job and the specific task. A new calculation cycle begins, the system will calculate the amount of computing resources allocated to each job (mainly refers to the calculation of time), specifically refers to the number of running tasks, which is contained by submission user's priority, the priority of the job, job size, operation and other factors together determine the number of tasks.

The system limits the number of jobs that can be run in a job pool within a computing cycle, and executes tasks in the computation cycle into operation queues, and other organizations that exceed the set number are organized into waiting queues. The jobs in the run queue are sorted by the weight of the job and the time order submitted by the job. The job in the run queue will get resources within this computing cycle, while the waiting queue does not get resources in this computing cycle. When a newly added job is deployed in a job pool, it is first placed in the waiting queue until the end of the calculation period. If all the tasks of a job are executed, it is removed from the run queue. At the end of the computation period, the system will update the running queue and waiting queue. When the number of jobs in the operation queue does not reach the limit value, the system will start transferring a certain number of jobs to the run queue from the queue of waiting queue.

The run queue job[i] contains the size of the job job_size[i] and the number of tasks included in the job task_count[i] information. The operation in the system is the service provided by the system. It is known. The system administrator sets the job_size[i] of each job. The size of job_size[i] is mainly determined by the type of job and the amount of data processed by the job.

task_mean_size[i] said the average size of each task assignment, running_task_count[i] said the task of each job processing in the calculation period, T[i] said each job processing in the calculation period the amount of data, T said the total data processing in the calculation period system, calculate the total amount of T in the data processing cycle to a fixed value, you can configure the file settings, the default value for the number of product BlockSize and map tasks in the cluster slot, BlockSize data block HDFS (Block) size. They have the following relationship:

\[
\text{task_mean_size[i]} = \frac{\text{job_size[i]}}{\text{task_count[i]}}
\]

\[
T[i] = \text{task_mean_size[i]} \times \text{running_task_count[i]}
\]

\[
T = \sum_{i=1}^{n} T[i] = \sum_{i=1}^{n} (\text{task_mean_size[i]} \times \text{running_task_count[i]})
\]

Weights[i] represents the weight of jobs in running queue job[i], \( U_P_i \) indicates user priority, and \( J_P_i \) indicates job priority (the priority value is higher here and the priority level is higher).

\[
\text{weights}[i] = \log_2 (U_P_i \times J_P_i + 1)
\]

In a computing cycle, the ratio of the amount of data processed by a job to the total data of the system is equal to the ratio of the weight of the operation to the sum of all the job weights processed by the system.
The number of tasks handled within the computing cycle is:

\[ \text{running_task_count}[i] = \frac{\text{weights}[i] \times \text{task_mean_size}[i] \times \text{running_task_count}[i]}{\sum_{j=1}^{n} \text{weights}[j]} \]  

(6)

The number of tasks handled within the computing cycle is:

\[ \text{running_task_count}[i] = \frac{\log_2(\text{UR}_i \times \text{TR}_i) \times T \times \text{task_count}[i]}{\sum_{j=1}^{n} \log_2(\text{UR}_j \times \text{TR}_j) \times \text{job_size}[j]} \]  

(7)

\[ \text{turn_count}[i] = \min(\text{round}(\text{running_task_count}[i]), \text{left_task_count}[i], 1) \]  

(8)

\[ \text{sum_count} = \sum_{i=1}^{n} \text{turn_count}[i] \]  

(9)

In the job scheduling, the following principles are followed:

1. the jobs in the job queue are sorted according to the priority of jobs and the time of submission. In a computing cycle, the system chooses the right queue according to the queue order.

2. After the new job enters the job pool, it will first be put into the waiting queue, and the running queue and waiting queue will be updated after the end of a computing cycle.

3. for the map task, we consider the principle of computing locally, allocate the local tasks to the map task slot as much as possible, and the local tasks are better than the frame tasks. This frame task is better than the inter frame task. The reduce task slot does not have to consider the data locality.

4. The number of actual tasks for a job to run within a computing cycle is turn_count[i].

Algorithm flow

When an idle TaskTracker node applies for a task, the system first selects the job pool according to the minimum sharing difference (\(MS_i\)) of the job pool and the fair share difference (\(JS_i\)), and then selects the task of the specific task according to the weighted rotation method. The process of assigning tasks within the job pool is shown as shown by figure2.
Performance analysis

Weighted round robin based fair scheduling algorithm assigns the number of tasks that can run for each job in a computing cycle according to the weight of jobs in the job pool, and intuitively achieves the fairness among jobs in the job pool. The original fair scheduling algorithm achieves the fairness between jobs by assigning resources for jobs that are not treated fairly in the last time period. This fairness is often lagging behind, sometimes it will cause resource grabbing among jobs. This preemption will lead to a waste of system resources, because the task that is preempted on the resource (computing node) will be allocated to other nodes to be re-executed.

Based on the fair scheduling algorithm of the weighted round robin, for each job pool, only at the beginning of the calculation cycle, need to share resources calculation for each job, the number that can run the task; in contrast, the original fair scheduling algorithm at regular intervals (default is 500ms), to calculate and update of each job fair share amount and operation deficit, the JobTracker node's resource consumption is great.

At the same time, a fair scheduling algorithm based on weighted round robin to give full consideration to the importance of local data, in the process of task selection, according to the TaskTracker node and DataNode node distance in ascending order, that is, in the frame of the local task, task frame tasks, assign tasks, the scheduling result is more optimized, so as to improve the throughput rate of Hama cluster.

EXPERIMENT AND ANALYSIS

This chapter mainly aims at the experimental verification of the improved algorithm proposed in the last chapter. The performance of the original algorithm and the improved algorithm are compared by selecting the appropriate datum program, and the experimental results are analyzed.
Experimental platform

The Hama cluster of this experiment is made up of 7 nodes. The node node0 is used to do JobTracker and NameNode other nodes to make TaskTracker and DataNode. The nodes are connected by switches, and figure3 is the network topology.

Benchmark

In order to test the performance of the algorithm accurately and accurately, it is necessary to select the appropriate benchmarks to simulate the real life. Taking into account the real-life application scenarios, this experiment takes the data deemphasis benchmark program: collecting data from all files under input path and output to output files, for repeated data appearing only once in the final output file.

Analysis

The purpose of this experiment is to compare the influence of the first come service scheduling algorithm and the weighted round robin fair scheduling algorithm on the load status of JobTracker nodes under the same scenario. Here we only compare the utilization rate of CPU. The experimental design of 2 job, job1 and job2 is a data removal program. Among them, job1 is dealing with 20 SM files with a priority of 2; job2 is to handle 10lOM files with a priority of 1. In the experiment, the job is submitted in the order of job1 and job2, and the time interval of the job submission is kept within one second. The system uses the first come first service scheduling algorithm and the weighted round robin fair scheduling algorithm to schedule the jobs respectively. From fiftieth seconds, the utilization rate of CPU of JobTracker nodes is collected every 10 seconds, and 30 times are collected.

The experimental results are shown in figure4. From the experimental results, we can see that compared with the first come first service scheduling algorithm, the CPU utilization rate of JobTracker nodes is slightly lower based on the weighted round robin fair scheduling algorithm. FCFS scheduling algorithm to select the appropriate scheduling at the same time, need to constantly update the job fair resource sharing computing capacity and operating deficits and other information; and based on fair scheduling algorithm in a weighted round robin scheduling task request need to choose the right, only need to calculate the number of each of each job in this calculation cycle can run the task before the start of calculation period. In this experiment, based on the use of CPU to reduce the JobTracker node fair scheduling algorithm under the weighted round robin rate is not very obvious, probably
because the smaller clusters tested, and the test operation instance less, resulting in JobTracker nodes do not need to consume too much CPU resources to complete the job scheduling.

![Figure 4. Experimental result diagram](image)

**CONCLUSION**

In this article, we introduced the Hadoop platform that has an important impact on Hama. Through detailed study of Hama parallel computing framework source code, the core part of in-depth study, through the study of these modules, especially on the scheduling module, and realize the subsequent design of weighted round robin fair scheduling algorithm lays a solid theoretical basis.

Based on the weighted round robin fair scheduling algorithm to guarantee the customer and fairness, fair scheduling through job pool, using weighted round robin scheduling of work in the pool, to ensure the system throughput and reduce the load of the JobTracker node.

Finally, through experiments, we can see that the weighted round robin fair scheduling algorithm achieves the expected design goal. The algorithm not only overcomes the shortcomings of the first come first service scheduling algorithm, but also provides more flexibility for Hama job scheduling.

**ACKNOWLEDGEMENT**

The article was supported by the project of "Intelligent control platform architecture and real-time transparent access technology. (No. 2017YFB0902601)" and Zhejiang Province Project" Research on the key technology and system development of automatic generation of video virtual viewpoint for security monitoring".
REFERENCES