Improving Job Scheduling in Hadoop
MapReduce

Himangi G. Patel, Richard Sonaliya
Computer Engineering, Silver Oak College of Engineering and Technology, Ahmedabad, Gujarat, India.

Abstract—Hadoop is a framework for processing large amount of data in parallel with the help of Hadoop Distributed File System (HDFS) and MapReduce framework. Job scheduling is an important process in Hadoop MapReduce. MapReduce scheduler does not scale well in heterogeneous environment. As an extension of Hadoop scheduler, LATE MapReduce scheduling algorithm takes heterogeneous environment into consideration. However, its performance is much poor due to the static manner in which it computes progress of tasks. So, neither Hadoop nor LATE schedulers are desirable in heterogeneous environment. To address this issue, M_LATE (Modified Late) scheduler has introduced, which calculates progress of tasks dynamically and adapts to the continuously varying environment automatically. When a job is committed, M_LATE splits the job into lots of fine-grained map and reduce tasks, then assigns them to a series of nodes. Meanwhile, it reads historical information which stored on every node and updated after every execution. Then, M_LATE adjusts time weight of each stage of map and reduce tasks according to the historical information respectively. Thus, it gets the progress of each task accurately and finds which tasks need backup tasks. What’s more, it identifies slow nodes and classifies them into the sets of slow nodes dynamically. According to the information of these slow nodes, M_LATE will not launch backup tasks on them, ensuring the backup tasks will not be slow tasks any more. It gets the final results of the fine-grained tasks when either slow tasks or backup tasks finish first. The proposed algorithm is evaluated by extensive experiments over various heterogeneous environment. Experimental results show that M_LATE significantly decreases the time of execution up to 25% compared with Hadoop’s scheduler and up to 14% compared with LATE scheduler.

Index Terms - MapReduce, Scheduling algorithm, Heterogeneous environment, Modified LATE

I. INTRODUCTION

Apache Hadoop is a software framework for processing BigData such as data in the range of petabytes [5]. The framework was originally developed by Doug Cutting, the creator of Apache Lucene, as part of his Web Search Engine Apache Nutch. Hadoop leverages the power of distributed computing with the help of Google Map Reduce Framework and Hadoop Distributed File System (HDFS).

MapReduce is the framework for processing large volume of datasets as key value pairs[4]. MapReduce divides each job in to two types of functions, map and reduce. Both Map and Reduce functions take input as key value pairs and emits the result as another set of key value pairs. Each job is divided in to number of map tasks and Reduce tasks. The input is initially processed in distributed map tasks and aggregate the result with the help of reduce tasks.

A key benefit of MapReduce is that it automatically handles failures, hiding the complexity of fault-tolerance from the programmer. If a node crashes, MapReduce reruns its tasks on a different machine. Equally importantly, if a node is available but is performing poorly, a condition that we call a straggler, MapReduce runs a speculative copy of its task (also called a “backup task”) on another machine to finish the computation faster. Without this mechanism of speculative execution1, a job would be as slow as the misbehaving task. Stragglers can arise for many reasons, including faulty hardware and misconfiguration. Google has noted that speculative execution can improve job response times by 44% [1].

As an important extension of Hadoop, LATE scheduling algorithm tries to improve Hadoop by attempting to find real slow tasks by computing remaining time of all the tasks. It selects a set of tasks which have longer remaining time than those of all other nodes, and considers the set of tasks are slow tasks. However it does not compute the remaining time for tasks correctly, and can not find real slow tasks in the end.

In this work, we address the problem of how to robustly perform speculative execution to maximize performance. Hadoop’s scheduler starts speculative tasks based on a simple heuristic comparing each task’s progress to the average progress. Although this heuristic works well in homogeneous environments where stragglers are obvious, we show that it can lead to severe performance degradation when its underlying assumptions are broken. We design an improved scheduling algorithm that reduces Hadoop’s response time by a factor of 2.
To this end, we share the similar idea to LATE scheduling algorithm in this paper and propose M\_LATE: Modified LATE algorithm. M\_LATE significantly improves MapReduce in terms of saving time of execution as well as system resources. M\_LATE is inspired by facts that slow tasks prolong the execution time of the whole job and nodes requires various time in accomplishing the same tasks due to their differences, such as capacities of computation and communication, architectures, memorizes and power. Although Hadoop and LATE also launch backup tasks for slow tasks, they cannot find the appropriate tasks which are really prolong the execute time of the whole job because the two scheduler always use a static way to find slow tasks.[6]

On the contrary, M\_LATE incorporates historical information recorded on each node to tune parameters and find slow tasks dynamically. M\_LATE can find slow tasks which need backup task really. In order to save system resources, M\_LATE classifies slow nodes into map slow nodes and reduce slow nodes further. M\_LATE defines fast nodes and slow nodes to be nodes which can finish a task in a shorter time and longer time than most other nodes. Map/reduce slow nodes means nodes which execute map/reduce tasks using a longer time than most other nodes. In this way, M\_LATE launches backup map tasks on nodes which are fast nodes or reduce slow nodes.

The most important concept of this paper are as follows:

- It uses historical information to tune weight of each stage dynamically.
- It takes the two stages characteristic of map tasks into consideration for the first time.
- It classifies slow nodes into map slow nodes and reduce slow nodes further.

The rest of the paper is organized as follows. Section II describes the background. Section III introduces the M\_LATE and reports the implementation details. Section IV describes the experimental results. Section V draws the conclusion with pointing out our future work.

**II. BACKGROUND**

In this section, we provide background information about MapReduce and scheduling in Hadoop.

A. MapReduce

MapReduce is straightforward, consisting of two phases: map phase and reduce phase in sequence, and reduce-side produces the final output. As shown in Figure 1. A large number of data analytical jobs can be expressed as a set of MapReduce jobs. MapReduce jobs are automatically parallelized and executed on a cluster of commodity machines, a MapReduce job consists of two phases, map phase and reduce phase. The map phase processes input key/value pairs, and produces a new list of key/value pairs for each pair. The reduce phase performs group-by according to the intermediate key, and produces a final key/value pair per group. MapReduce performs the following transformation to their input [3]:

\[
\text{map}(K_1, V_1) \rightarrow \text{list}(K_2, V_2) \\
\text{reduce}(K_3, \text{list}(V_2)) \rightarrow \text{list}(K_3, V_3)
\]

The runtime system automatically splits the input dataset into even-sized data blocks and dynamically schedules the data blocks to the available compute nodes for processing [8], each node in the cluster has a number of slots to host map and reduce tasks. MapReduce is proven to be highly scalable in real systems.

B. Scheduling in Hadoop

In this section, we describe the mechanism used by Hadoop to distribute work across a cluster. We identify assumptions made by the scheduler that hurt its performance. These motivate our LATE scheduler, which can outperform Hadoop’s by a factor of 2.

The goal of speculative execution is to minimize a job’s response time. Response time is most important for short jobs where a user wants an answer quickly, such as queries on log data for debugging, monitoring and business intelligence. Short jobs are a major use case for MapReduce. For example, the average MapReduce job at Google in September 2007 took 395 seconds [1].

Hadoop monitors the progress of tasks using “Progress Score(PS)” (range from 0 to 1). The average progress score is $PS_{avg}$. The Progress Score of the $ith$ task is $PS[i]$. Suppose the number of tasks which are being executed
is $T$, the number of key/value pairs need to be processed in a task is $N$, the number of key/value pairs have been processed in the task is $M$, and the task has finished $K$ stages (only for reduce task. There are three stages in a reduce task: copy data phase, sort phase and reduce phase).[4] Hadoop gets $PS$ according to the Eqs. 1 and 2 and launches backup tasks according to the Eq. 3.

$$PS = \begin{cases} 
\frac{M}{N} & \text{For MapTask} \\
\frac{1}{3} \times (K + M / N) & \text{For ReduceTask}
\end{cases} \quad (1)$$

$$PS_{\text{avg}} = \frac{\sum \text{PS}[i]}{T} \quad (2)$$

$$For \ task \ T_i: PS[i] < PS_{\text{avg}} - 20\% \quad (3)$$

If the Eq. 3 is fulfilled, $T_i$ needs a backup task.

C. LATE MapReduce scheduling algorithm

LATE MapReduce scheduling algorithm always launches backup tasks for those tasks which have more remaining time than other tasks. Suppose a task $T$ has run $Tr$ seconds. Let $PR$ denotes the progress rate of $T$, and $TTE$ denotes how long time remaining until $T$ was finished. LATE MapReduce scheduling algorithm computes $PR$ and $TTE$ according to the Eqs. 4 and 5. $PS$ in the Eq. 4 is computed according to the Eq. 1.

$$PR = \frac{PS}{T_r} \quad (4)$$

$$TTE = \frac{(1.0 - PS)}{PR} \quad (5)$$

Although LATE uses an efficient strategy to launch backup tasks, it often launches backup tasks for inappropriate tasks. This is because LATE cannot find $TTE$ for all the running tasks correctly.

One shortcoming of LATE, which is same to Hadoop, is the values of $R1$, $R2$, $R3$, $M1$, $M2$ are 0.33, 0.33, 0.34, 1 and 0 respectively. This setting may lead to the wrong $TTE$. Suppose $R1$, $R2$ and $R3$ are 0.6, 0.2 and 0.2 respectively in a real system. When the first stage finishes in $Tr$ seconds, the reduce task still needs $(1 - 0.6) \times (Tr / 0.6) = 0.67Tr$ seconds to finish the whole task. However, the $TTE$ computed in LATE scheduling algorithm is $(1 - 0.33) \times (Tr / 0.33) = 2.6Tr$ seconds. Another shortcoming of LATE MapReduce scheduling algorithm is it does not distinguish map slow nodes and reduce slow nodes. One node may executes $MT$ quickly, but executes $RT$ slower than most of other nodes. LATE just considers one node either fast node or slow node, does not classify slow nodes further.

### III. M_LATE: MODIFIED LATE SCHEDULING ALGORITHM

M_LATE is developed with a similar idea to LATE MapReduce scheduling algorithm. However, M_LATE gets more accurate $PSs$ of all the tasks by using historical information. By using accurate $PSs$, M_LATE finds real slow tasks and decreases more execute time compared with Hadoop and LATE. Algorithm illustrate the process of M_LATE.

**Algorithm: M_LATE algorithm**

**INPUT:** Key/Value Pairs

1) Read historical information and tuning of parameters using historical information.
2) Find Slow Task
3) Find Slow Task Tracker
4) Launch Backup Task
5) Collect Results and update information.

**OUTPUT:** Statistical results

A. Reading historical information and tuning parameters using historical information

TTs read historical information ($R1$, $R2$, $R3$, $M1$, $M2$) that recorded on the disks from nodes where the TTs are running. The historical information is stored on every node in xml format. The values of $R1$, $R2$, $R3$, $M1$, $M2$ are 0.33, 0.33, 0.34, 1 and 0 by default. Algorithm 1 updates the values after every execution (line 5 in Algorithm).

B. Finding slow tasks

$SLOW \ TASK \ CAP(StaC)$(range from 0 to 1) is used to classify tasks into fast and slow tasks. If the progress rate of task $Ti$, $PRi$ and the average progress rate of all the running tasks, $APR$ fulfill the Eq. 6, $Ti$ is judged to be slow task. Suppose the number of task running now is $N$, $APR$ is computed according to the Eq. 7.

$$PR_i < (1.0 - StaC) \times APR \quad (6)$$

$$APR = \sum_{j=1}^{N} PR_j / N \quad (7)$$

According to the Eq. 6, if $StaC$ is too small (close to 0), $M_LATE$ will classify some fast tasks into slow tasks. If $StaC$
is too large (close to 1), M\textsubscript{LATE} will classify slow tasks into fast tasks.

C. Finding slow TaskTrackers

\textit{SLOW TRACKER CAP}(\textit{ST}\textsubscript{rC})(\textit{range from 0 to 1}) is used to classify \textit{TaskTrackers} into fast \textit{TTs} and slow \textit{TTs}. One \textit{TT} only runs on one node, so slow nodes are the same as slow \textit{TTs}.

If \textit{ST}\textsubscript{rC} is too small (close to 0), \textit{SAMR} will classify some fast \textit{TTs} into slow \textit{TTs}. If \textit{ST}\textsubscript{rC} is too large (close to 1), \textit{SAMR} will classify some slow \textit{TTs} into fast \textit{TTs}.

\textit{SLOW TRACKER PRO}(\textit{ST}\textsubscript{rP})(\textit{range from 0 to 1}) is used to define the maximum proportion of slow \textit{TTs} in all the \textit{TTs}. Suppose the number of slow \textit{TTs} is \textit{SlowTrackerNum}, the number of all the \textit{TTs} is \textit{TrackerNum}. The Eq. 8 must be fulfilled in the system.

\begin{equation}
\textit{SlowTrackerNum} < \textit{ST}\textsubscript{rP} * \textit{TrackerNum} \quad (8)
\end{equation}

D. Launching backup tasks

If there are slow tasks, and Eq. 15 is fulfilled, a backup task can be launched when some of \textit{TTs} are free. \textit{BACKUP PRO}\textsubscript{BP}(\textit{range from 0 to 1}) is used to define the maximum proportion of backup tasks in all the tasks. Suppose the number of backup tasks is \textit{BackupNum}, the number of all the running tasks is \textit{TaskNum}. The Eq. 9 must be fulfilled in the system.

\begin{equation}
\textit{BackupNum} < \textit{BP} * \textit{TaskNum} \quad (9)
\end{equation}

IV. EVALUATION

In order to verify the effectiveness of \textit{M\textsubscript{LATE}}, we carry a series of experiments. In particular, we try to answer 3 questions below:

- What’s the best parameters for \textit{M\textsubscript{LATE}}?
- Is the historical information recorded correct in \textit{M\textsubscript{LATE}}?
- What is the performance of \textit{M\textsubscript{LATE}} in heterogeneous environment?

A. Experimental environment

We establish experimental environment by using virtual machines on three personal computers. All the virtual machines use Ubuntu operating system. The version of JDK is 1.8.0.45, and the version of Hadoop is 2.6.0. The \textit{M\textsubscript{LATE}} is implemented based on Hadoop 2.6.0. Because we cannot get the primary version of LATE MapReduce scheduling algorithm, so we implement a new one, according to the method above. I have also implemented this algorithm on multinode live Hadoop on amazon EC2.
V. CONCLUSION

In this paper, we have proposed M_LATE: Modified LATE algorithm, which uses historical information and classifies slow nodes into map slow nodes and reduce slow nodes. Experimental results have shown the effectiveness of the self-adaptive MapReduce scheduling algorithm. It runs slow tasks and launch backup tasks while correctly classifying the nodes and saving a lots of system resources. It decreases the time of execution about 17%, in heterogeneous environment compared to LATE.

REFERENCES
