An Improvement of Scheduling Algorithm For Heterogeneous Hadoop Cluster Using Congestion Control Concept

Apiluck Songwattanasakul Author, Assc.Prof Natawut Nupairoj Advisor
Chulalongkorn University Bangkok, Thailand.

ABSTRACT: Hadoop is the open source software based on MapReduce Algorithm. It is a programming model implementation support big data processing. Currently Hadoop is implemented base on Homogeneous Cluster so the scalability becomes too serious problem. Although many people offer the solution of the Heterogeneous Cluster scheduler for task allocation with the most resource efficiency, the approach consider about I/O Monitoring and Network Utilization is important factor to consider about job scheduler because according to Amdahl’s law “The speedup of a program using multiple processors in parallel computing is limited by the time needed for the sequential fraction of the program”. Hadoop Default Scheduler has big problem about the job allocation because it is scheduler base on configurable and all node in the cluster will assign the equal job size. It works fine if the cluster is Homogeneous Cluster. It still have one of the feature call Back up task sometimes is not identify speculative task correctly. Researcher purpose Heterogeneous Hadoop Cluster Algorithm. Scheduler can self-scheduling rely on Network Utilization, Processing Time and I/O Utilization using the congestion control concept in TCP/IP Protocol after that we purpose the heuristic equal to identify appropriate block size assign for each node in each job. Scheduler can be adjust the job size dynamically so Algorithm can separate into 2 phase. First Part it use for initial cluster parameter to calculate job allocation. We use the congestion control concept to implementation after we finish we can identify how many size we can assign for each node between the processing. Second part when task is submitted to node. We implement the monitoring thread in the period of time for update the parameter cluster and we use update parameter to consider the target task need to have the backup task. The purpose of this algorithm reduces the processing time and resource efficiency in term of CPU processing and reduces useless back up task. We evaluation based on Word Count, Grep and Sorting. Compare with Default Scheduler Algorithm.

Keywords—Hadoop, Parallel Computing, Scheduler Algorithm,

I. INTRODUCTION

Today in the large scale computing, cloud computing and internet service are growing so fast. According to Google search engine service for MapReduce framework. It has 20 petabytes data per day need to processing [1]. Nowadays many kind of the processing use MapReduce Process for example MapReduce for Information Storage and Retrieval [2] [3], Web Search Engine [4] and Image Processing [5], First, It is initiated by Google using GFS [6] and BigTable [7] after that Facebook bring algorithm to develop in term of real time processing [8]. Recently, MapReduce was applied in Yahoo, Twitter [9] etc.

The open-source project Hadoop is the most popular MapReduce framework. Hadoop is the framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability and fault-tolerant manner, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers [10]. Hadoop MapReduce is developed base on the assumption of process on Homogeneous Cluster (Assume all machine equal in term of performance). In the real situation, Homogeneous Cluster occurred a bit difficult in term of business because it grows rapidly so scalability is become important factor. The most of cluster in the world is Heterogeneous Cluster when hadoop runs on the Heterogeneous Cluster therefore it is inefficiency.

As reference from the part, the extension of hadoop is implemented by LATE Scheduling Algorithm. It tried to improve efficiency of hadoop scheduler using the concept of attempting to find real slow tasks by calculating Time to End (TTE) and launch speculative task (Backup Task) for the longest TTE [11]. However speculative task launch by LATE is based on assumption of data locality so network and I/O utilization do not include the algorithm therefore it is not valid scenario when the cluster has many machine.

Shortly, Researcher presented another algorithm extension from LATE Scheduling Algorithm called SAMR: a Self-Adaptive MapReduce scheduling algorithm. SAMR is inspired by facts that slow tasks prolong the execution time of the whole job so algorithm kept history information separately for each node in the cluster [12] therefore it is more relevant than LATE Scheduling Algorithm but this algorithm also not include network and I/O utilization. It still getting the same issue when the scenario as mentioned.

To the end, we would like to share the information about solution to improve hadoop scheduler. According to problem scenario when the cluster has many machine network and I/O utilization will become important factor also therefore we would like to propose the algorithm by solve the problem as
Below:
- Using congestion control concept (TCP/IP Network) use for testing I/O utilization and exactly know how big of the HDFS File need to assign to TaskTracker(TT).
- We propose the equation to calculate speculative task using network and I/O factor so it can launch speculative more relevant help the overall cluster resource. It can improve cluster utilization when we process more than one job in one time.

II. RELATED WORK

For more understanding, this session will explain about how the mapreduce work and brief of the knowledge consist of basic of the mapreduce algorithm, related work in this field and introduction lead down to the research

A. Basic Concept of Hadoop

This paper information contains some parameter define in the paper as table I

<table>
<thead>
<tr>
<th>Name</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NameNode</td>
<td>NN</td>
<td>Records data location and scheduler task in the cluster.</td>
</tr>
<tr>
<td>DataNode</td>
<td>DN</td>
<td>Stored and processing the data</td>
</tr>
<tr>
<td>JobTracker</td>
<td>JT</td>
<td>Manage MapReduce job located in NameNode (NN)</td>
</tr>
<tr>
<td>TaskTracker</td>
<td>TT</td>
<td>Manage Task assign from JT</td>
</tr>
<tr>
<td>Machine</td>
<td>( M_{c,n} )</td>
<td>Machine in the cluster (1…n)</td>
</tr>
<tr>
<td>Map Tasks</td>
<td>( M_{t,n} )</td>
<td>Map Task (1…n)</td>
</tr>
<tr>
<td>Reduce Tasks</td>
<td>( R_{t,n} )</td>
<td>Reduce Task (1…n)</td>
</tr>
<tr>
<td>Task Size</td>
<td>( T_{s,n} )</td>
<td>Task size for each TT (1…n)</td>
</tr>
<tr>
<td>Progress Rate</td>
<td>( PR_{c,n} )</td>
<td>Progress Rate (1…n)</td>
</tr>
<tr>
<td>Response Time</td>
<td>( List &lt; B,T &gt; )</td>
<td>Response time data mapping between Block per Second</td>
</tr>
<tr>
<td>Network Utilization</td>
<td>( List &lt; MC_{c,n} BW &gt; )</td>
<td>Network Utilization Data</td>
</tr>
<tr>
<td>Remaining Data Size</td>
<td>RDP</td>
<td>Remaining data size for Job In Progress</td>
</tr>
</tbody>
</table>

MapReduce is one of the programming technic using divide (Map Phase) and conquer (Reduce Phase) support big data processing in term of the parallel processing it will make it fast and reliable so running the data on Mapreduce need to submit the job into the cluster in this research. It called “Job”. Job can divide the data related by that Job. Job Split Size and data location is considered by Machine call “NameNode (NN)”. The result after split by NameNode (NN) it called “Task” after that it will submit task into “DataNode (DN)” for processing “Map Phase” and “Reduce Phase” processing by sequentially.

In the cluster, it contains NameNode (NN) and DataNode (DN). NameNode (NN) is responsible for scheduling the task and conducts the way to split the job inside the cluster including store the location of HDFS file. There is tracker inside the NameNode (NN) called “JobTracker (JT)” to communicate between NameNode (NN) and DataNode (DN) when the task has been assigned to DataNode (DN). It is responsible for processing the task assign by NameNode (NN). There are one of the module called “TaskTracker (TT)” to monitor the progress of task and report back to JobTracker (JT) for control the scheduling correctly, adjust the appropriate parameter and make cluster into the most efficiency.

First, User has been submitted Job into Hadoop Cluster. Job will go to NameNode (NN) for scheduling and dispense the task to DataNode (DN) follow by the algorithm has been implemented inside the cluster. Second, DataNode (DN) retrieves the data split and processing Map Function. The output of the Map Function is the key/pair values. The results are written in the local disk. Third, Data on the local disk do the Partitioning Function to group the data prepared for the Reduce Phase, Fourth after Partitioning is finished will return the result update the location of the HDFS file to NameNode (NN). It will inform the TaskTracker (TT) again to do the Reduce Phase by getting the location from NameNode (NN). DataNode (DN) will retrieve the data using remote procedure call to each other and processing Reduce Function. The output of the reduce phase is appended to final output file for the reduce partition and acknowledge NameNode (NN). MapReduce has been finished and perform back to user about the result.

B. Hadoop Default Scheduler Algorithm

Job has been submitted into the cluster. Scheduler will calculate size of the task assign it to TaskTracker (TT) in
cluster base on parameter \textit{mapred.max.split.size} and split the task size as a configuration after that submit task running for each DataNode between the processing of the data. TaskTracker (TT) monitors the task progress calculated parameter called \textit{Progress Score (PS)}. The number is between 0 and 1.

For the Map Phase, progress Score (PS) is calculated by one of the parameter called Input Data Read

For the Reduce Phase, progress Score (PS) divided into 3 phase Copy Phase, Sort Phase and Reduce Phase so for each stage can be calculate the progress score (PS) follow by this parameter below

- \textit{Copy Phase} when the task is copying outputs of all maps. In this phase, the score is the percent of maps that output has been copied from.
- \textit{Sort Phase} when map outputs are sorted by key. Here the score is the percent of data merged.
- \textit{Reduce Phase} when a user-defined function is applied to the map outputs. Here the score is the percent of data passed through the reduce function.

In the Hadoop Scheduler, They have the Progress Score (PS) separately for each TaskTracker (TT). Equation 1 is how to calculate the progress score

\[
PS = \begin{cases} 
\frac{M}{N} & \text{For MT} \\
\frac{1}{3}(K + \frac{M}{N}) & \text{For RT} 
\end{cases}
\]  

(1)

Overall of the cluster call Average Progress Score (APS). The parameter can calculate by Equation 2

\[
\text{AverageProgressScore(APS)} = \frac{\sum_{\text{MC}} PS_{\text{MC}}}{MC_{\text{MC}}}
\]  

(2)

Hadoop mark the TaskTracker (TT) as struggler machine if Equation 3 is true

\[
PS_{\text{1..MC}_{\text{MC}}} < \text{APS} - 0.2
\]  

(3)

Meanwhile, if cluster have TaskTracker (TT) become to ideal node. Hadoop launch the backup task (Speculative Task) for struggler machine.

Although the metrics like progress score (PS), it can make sense to identify the weak point of the machine in the cluster. It works properly in the homogeneous cluster because the machine performance is equal or approximate with in the cluster.

Finally, to support multiple job running in the same time so hadoop implement job queue use the FIFO algorithm to schedule the job into the cluster. Recently, the open source developed the job queue priority.

\section{C. LATE MapReduce Scheduling Algorithm}

LATE Scheduler algorithm has been extended the default hadoop scheduler. It finds the remaining time to finish the task. Suppose Task size is T and already processing TR. LATE Algorithm estimate process rate (PR) as Equation 4

\[
PR_{1..n} = \frac{PS_{1..n}}{TR_{1..n}}
\]  

(4)

After that Algorithm using process rate (PR) to calculate Time to End parameter (TTE) as Equation 5

\[
TTE_{1..n} = \frac{1 - PS_{1..n}}{PR_{1..n}}
\]  

(5)

Algorithm calculated TTE for each TaskTracker. Session B Default scheduler algorithm will make struggler machine when Equation 3 is true. LATE consider the most TTE has been launch speculative task first when TaskTracker in the cluster is idle under this rule below

- LATE does not launch backup task to TaskTracker lower than threshold called \textit{SlowNodeThershold}.
- LATE does not launch backup task if backup task is launched more than threshold called \textit{SpeculativeCap}

LATE Algorithm assume major of map nodes are data-locality so network and I/O utilization is not considered in the LATE Algorithm so they expected the data locality when Algorithm launch speculative task so that is not valid scenario when the cluster has many machine and machine is not in the same rack will impact this assumption [11].

\section{D. Problem for Hadoop Default Scheduler}

According to Hadoop Default Scheduler algorithm, it can conclude the situation break with the assumption when we run processing data base on heterogeneous cluster environment.

1. In Hadoop Default Scheduler can calculate the progress score as Equation 1. In the reduce phase it can separate in 3 phase. Refer to the equation algorithm; it gives the weight equal for all phase so the copy phase is very fast. The most important part for reduce phase it should be reduce phase properly so we need to give not equal weight for the progress score. If copy phase is not fine but reduce phase is fast. This scenario can happen when the cluster has many machine and machine is not in the same rack will impact the assumption.

2. Hadoop default scheduler mark the struggler machine sometimes wrongly because the start processing is different time because the data is not locality need to remote procedure call and progress rate is not equal because of heterogeneous cluster, for example Machine A has progress score (PS) 0.7 need 100 seconds to finish processing. Machine B has progress score (PS) 0.5 need 50 seconds to finish so default scheduler will mark
machine B as struggler machine, Unless the real situation machine A is the weak point for this processing.

3. Hadoop default scheduler split the job using parameter call mapred.max.split.size. Now hadoop limit with this split size and the job split is equal in all node using the concept of data locality but heterogeneous cluster the job split size should not be equal for all node because the performance for each machine is not equal. Some machine has process rate so high. Scheduler need to assign the bigger task than the machine lower performance.

4. Speculative laugh is not calculate the time to move the data across machine from TaskTracker to backup TaskTracker so sometime if it is included the network and I/O constrain inside it is not worth to launch backup task.

For the problem number 2, LATE algorithm propose the algorithm extension from the hadoop default scheduler using the concept as always launch the slowest task in the cluster by implement the parameter called Time to End (TTE) to predict the finish processing time. Recently, SAMR propose the algorithm extension on the LATE algorithm by implement the historical information tuning the parameter inside the cluster. The parameter become to the part of the scheduler. It can identify slow TaskTracker and launch the speculative task relevant and more efficiency.

For the problem number 3 and 4, we propose the solution in our research to improve the scheduler make it more efficiency and reduce the waste resource for the speculative task. We use the congestion control concept of the TCP/IP to identify network utilization and I/O. Not only, can make the decision more accurate about the size to assign for each TaskTracker in heterogeneous cluster but also can consider speculative task launch with appropriately.

III.RESEARCH IMPLEMENTATION

Our implementation has inspired from LATE [11] and SAMR [12] but we try to close weakness as mention in session 2. We use concept of TCP/IP Congestion Control (Slow Start and Congestion Avoidance Technic) finding the information about Network and I/O Utilization. Research proposes additional parameter to make the scheduler more accurate when JobTracker (JT) split job using the concept as high performance machine need to assign the large size of task but we still use data locality concept as Hadoop propose in the default scheduler algorithm. In this session we explain separated by phase propose in our implementation. We have the prototype from another research [13]. It consists of Data Initialization Session A describe about how we get the Network and I/O Utilization information when we start up the cluster. Data Distribution Session B describe about how we assign task to TaskTracker (TT) and Backup Task Launch Consideration Session C describe about when we launch the backup task.

A. Data Initialization

Data initialization phase is purposed for getting the information when we initialized cluster. The information is contained Network Utilization, I/O Utilization and processing rate. Data Initialization phase is implemented base on as mention

Initialization cluster started up scheduler submit test data job base on Congestion Control concept. It starts from slow start phase (data block size increase as the exponential distribution) after that response time increase until it reach the threshold (TS). It changed from slow start to congestion avoidance phase until it reaches the fuzzy threshold (FTS). According to the parameter can get in this step will explain below

- I/O Utilization \( (IOR_{1,n}) \) Algorithm use the benchmark TESTDFSIO as provide in the Hadoop build in functionality but the way scheduler assign the size will use the concept of the congestion control after reach the fuzzy threshold (FTS) point will know the parameter for each machine inside cluster.

- Processing Rate \( (PR_{1,n}) \) between the progress of startup cluster. Scheduler assigns the task to TaskTracker (TT). It reported the parameter back to Scheduler.

- Network Utilization \( (List <MC,BW>_{1,n}) \) machine inside the cluster send the ICMP Package to another machine using the concept of token assign (NameNode put message for each node) as shown in Figure 2.

![Fig. 2. Network Utilization implementation](image)

B. Data Distribution

Algorithm got all information about mentioned in session A. Scheduler calculated task size assign for each TaskTracker (TT) base on their parameters. Algorithm separate the logic into 2 types of Job. Firstly I/O Bound Job is the kind of job required a
lot of I/O but required few processing logic for example Word Count and Grep Word. Secondly Processing Bound Job is the type of job required high performance of processing so algorithm has one parameter define in the configuration file. Different job use different heuristic to calculate how big assign for each TaskTracker (TT).

According to Session A, each TaskTracker (TT) has the parameter list contained $IORate_{1\ldots n}, PR_{1\ldots n}$, so algorithm use all these parameter to calculate value called performance ratio ($PerfR_{1\ldots n}$). Performance ratio can calculate using Equation 6 if it is I/O Bound Job.

$$PerfR_{1\ldots n} = 2IORate_{1\ldots n} + PR_{1\ldots n}$$  \hspace{1cm} (6)

If it is Processing Bound Job Performance ratio can calculate using Equation 7

$$PerfR_{1\ldots n} = IORate_{1\ldots n} + 2PR_{1\ldots n}$$  \hspace{1cm} (7)

Define Job contained data $N$ block number and number of map task ($Num_m$) so number of block assign for each task tracker ($Num_B$) will be calculate as Equation 8

$$Num_B = \frac{PerfR_i}{\sum_{i=1}^{Num_m} PerfR_i} \times N$$  \hspace{1cm} (8)

$$N = \sum_{i=1}^{Num_m} Num_B$$  \hspace{1cm} (9)

$$Num_B = \frac{PerfR_i}{\sum_{i=1}^{Num_m} PerfR_i} \times N$$  \hspace{1cm} (10)

$$N = \sum_{i=1}^{Num_m} Num_B$$  \hspace{1cm} (11)

Map phase use equation 8 and 9. Reduce phase use equation 10 and 11. To make it more understand, we explain by the example after explain in Session IV will show the value from the experiment.

For example cluster will explain for the I/O Bound Job so we will use equation 6 to calculate $PerfR_{1\ldots n}$. If data size is 200 Blocks

$$PerfR_A = 2(23) + 3 = 49, PerfR_B = 2(44) + 7 = 95$$

$$PerfR_C = 2(10) + 8 = 28, PerfR_D = 2(20) + 10 = 50$$

After calculation it will find summation of $PerfR$ from this example value is 222 so it can find the result of block size assign for each TaskTracker using equation 8.

$$\frac{49}{222} \times 200 = 44 \text{ Blocks} \times \frac{95}{222} \times 200 = 86 \text{ Blocks}$$

$$\frac{28}{222} \times 200 = 26 \text{ Blocks} \times \frac{50}{222} \times 200 = 45 \text{ Blocks}$$

Algorithm will give the most important to data locality so task assignment will assign to Machine B first and assign to the lower as sequence until the rest of the data in the machine is not locality anymore after that Algorithm will calculate for target node need to remote procedure call move data but we need to calculate the adjustment block size base on equation 12 so we need to define Processing node (PCN), target data node (TDN), Block Size need to remote call (BD) and Network Utilization between PCN and TDN (NWA)

$$Num_{PCN}(1 + \frac{PR_{TDN}}{PR_{PCN}} + \frac{PR_{TDN}}{NWA}) = BD$$  \hspace{1cm} (12)

According to Equation 12, Algorithm will improve the performance if the number of the replication data is configure suitable with the heterogeneous cluster because the remote procedure call under the assumption if more replicate data can be process data faster because the opportunity to hit the data locality is improved as mention in the Hadoop purpose of MapReduce.

After adjustment of block size, Algorithm distribute data to TaskTracker (TT) and it will do processing but we still got problem about parameter is not up to date after we got the information from the Data Initialization Phase (Session A) so we implement the update thread as mention in Session C.

C. Network and I/O Information Update

Additional module implemented for monitoring after cluster started. We need to update the information up to date to make it more efficiency. Module characteristic is working on period of time when it reach time configure inside the configuration. It will wake up to update the information. JobTracker (JT) assign token message to get all information trigger for each TaskTracker (TT). The list of parameter need to update is the same as mention in Session A but we have additional parameter called Remaining data size (RDS) remaining data for each TaskTracker in that time to predict the speculative task in the
D.Struggler Machine and Speculative Task

Hadoop Default Scheduler has concept of the speculative task but process to detect the weak point of the cluster. (Struggler Machine) Default Scheduler use Average Progress Score to detect the Struggler Machine but we use Progress Rate (PR) in each machine to predict the Struggle Machine. Our algorithm use the logic as mention

- Each machine in cluster, we have separate Progress Rate (PR) when the PR has been tendency dropped from the average progress rate we save separately for each cluster when the progress rate lower than average progress rate minus threshold configuration (TC) as mention in equation 13. Scheduler mark that TaskTracker (TT) as Struggler Machine

\[ PR_{1...n} < \text{Avg}_{PR_{1...n}} - TC \]

- Scheduler waited until in the cluster has node available will launch backup task base on equation 14 criteria. Scheduler makes a decision for this task need to launch the speculative task or not.

\[ \frac{RDS}{PR_{PCN}} > \frac{TS}{NWA} + \frac{TS}{PR_{TDN}} \]

IV. EVALUATION

In term of Evaluation, we verify the effectiveness of algorithm we proposed by an experiment. In particular, we try to answer three questions below

- Is the parameter we add in (I/O Rate and Process Rate) recorded correctly trend in the same direction as cluster environment?
- What is the performance of the algorithm compare with default scheduler in heterogeneous cluster environment?
- CPU and Resource Utilization compare between algorithm and default

A.Experimental Environment

We establish the experimental environment by using the virtual machine on the physical server. Physical server split it to 6 virtual machines. Spec show in table II

<table>
<thead>
<tr>
<th>Machine Name</th>
<th>Core No.</th>
<th>Memory</th>
<th>OS</th>
<th>Java JDK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Namenode</td>
<td>1 Core</td>
<td>2 GB</td>
<td>Ubuntu 12.04</td>
<td>1.7_0.51</td>
</tr>
<tr>
<td>Datanode – 1</td>
<td>2 Core</td>
<td>4 GB</td>
<td>Ubuntu 12.04</td>
<td>1.7_0.51</td>
</tr>
<tr>
<td>Datanode – 2</td>
<td>2 Core</td>
<td>2 GB</td>
<td>Ubuntu 12.04</td>
<td>1.7_0.51</td>
</tr>
<tr>
<td>Datanode – 3</td>
<td>1 Core</td>
<td>6 GB</td>
<td>Ubuntu 12.04</td>
<td>1.7_0.51</td>
</tr>
<tr>
<td>Datanode – 4</td>
<td>1 Core</td>
<td>2 GB</td>
<td>Ubuntu 12.04</td>
<td>1.7_0.51</td>
</tr>
<tr>
<td>Datanode – 5</td>
<td>1 Core</td>
<td>4 GB</td>
<td>Ubuntu 12.04</td>
<td>1.7_0.51</td>
</tr>
</tbody>
</table>

Our implementation based on Hadoop 1.2.1 because this is the latest version we can get from Apache Trunk [14]. We plug in the additional implementation as mentioned in Session III. The benchmarks used in the experiments are examples in Hadoop “Sort” and “WordCount”, because the two benchmarks use in the standard evaluation [11] [12].

B.Monitoring Parameter I/O Rate and Progress Rate

Before evaluating of the performance compare with default scheduler algorithm. The implementation of additional parameter called I/O Rate and Process Rate. We use the congestion control concept as mentioned in Session III so we submitted the Job processing until I/O congestion. We keep the statistic in the file using JSON format as show in Figure 4

Fig. 4. JSON format for monitoring parameter

After implementation, we run “WordCount” to test the monitoring parameter is related to the size we split assign to TaskTracker (TT) so the checking we use the 2 GB data size. The result of the split size is showed in Table III.

<table>
<thead>
<tr>
<th>Machine</th>
<th>Task Allocation Length(MB)</th>
<th>Total Task (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Namenode</td>
<td>49.896089, 49.896089, 49.896089</td>
<td>149.688267</td>
</tr>
<tr>
<td>d1c-3</td>
<td>94.746774, 94.746774, 94.746774</td>
<td>248.240322</td>
</tr>
<tr>
<td>d1c-2</td>
<td>87.721575, 87.721575, 87.721575</td>
<td>263.164725</td>
</tr>
<tr>
<td>d1c-1</td>
<td>88.120293, 88.120293, 88.120293</td>
<td>264.360879</td>
</tr>
<tr>
<td>d2c-2</td>
<td>169.097527, 169.097527, 169.097527</td>
<td>507.292581</td>
</tr>
<tr>
<td>d2c-1</td>
<td>177.084412, 177.084412, 177.084412</td>
<td>531.253236</td>
</tr>
</tbody>
</table>
According to Table III, the tendency of size is related to machine spec as mentioned in Table II in the same session. The machine has multi-core processing getting the task size more than the single-core processing. Follow by the equation 10 and 11. The parameter is dynamically so it is not required to change when we changed the environment.

C. Algorithm Performance Evaluation

We perform the experiment using benchmarks “WordCount” and “Sort”. We are submitted the job characteristics as random word generator for Wordcount and characteristics as random key and value generator (Sequence file generator) for Sorting. Those two benchmarks, we are submitted base on the difference of job size (2, 4, 6 and 8 GB). Table IV is the result of the experiment.

Table IV
Experiment Result for Wordcount (MS)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Benchmark</th>
<th>N</th>
<th>2 GB</th>
<th>4 GB</th>
<th>6 GB</th>
<th>8 GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop Default</td>
<td>Word Count</td>
<td>1</td>
<td>282</td>
<td>506</td>
<td>674</td>
<td>912</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>273</td>
<td>503</td>
<td>682</td>
<td>890</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>277</td>
<td>494</td>
<td>686</td>
<td>902</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>280</td>
<td>495</td>
<td>676</td>
<td>905</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>274</td>
<td>496</td>
<td>681</td>
<td>900</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td>277.2</td>
<td>498.8</td>
<td>679.8</td>
<td>901.8</td>
</tr>
<tr>
<td>Hadoop (with congestion control)</td>
<td>Word Count</td>
<td>1</td>
<td>252</td>
<td>413</td>
<td>620</td>
<td>823</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>256</td>
<td>412</td>
<td>641</td>
<td>810</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>244</td>
<td>401</td>
<td>629</td>
<td>806</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>245</td>
<td>421</td>
<td>638</td>
<td>815</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>257</td>
<td>414</td>
<td>640</td>
<td>803</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td>250.8</td>
<td>412.2</td>
<td>633.6</td>
<td>811.4</td>
</tr>
<tr>
<td>Delta</td>
<td></td>
<td></td>
<td>26.4</td>
<td>86.6</td>
<td>46.2</td>
<td>90.4</td>
</tr>
<tr>
<td>Delta (%)</td>
<td></td>
<td></td>
<td>9.523</td>
<td>17.36</td>
<td>6.796</td>
<td>10.02</td>
</tr>
<tr>
<td>AVG Delta (%)</td>
<td></td>
<td></td>
<td>10.9269742</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table V
Experiment Result for Sort (MS)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Benchmark</th>
<th>N</th>
<th>2 GB</th>
<th>4 GB</th>
<th>6 GB</th>
<th>8 GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop Default</td>
<td>Sort</td>
<td>1</td>
<td>177</td>
<td>330</td>
<td>454</td>
<td>653</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>155</td>
<td>304</td>
<td>470</td>
<td>632</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>164</td>
<td>316</td>
<td>465</td>
<td>650</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>174</td>
<td>325</td>
<td>463</td>
<td>643</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>168</td>
<td>321</td>
<td>459</td>
<td>642</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td>181.6</td>
<td>347.8</td>
<td>415.8</td>
<td>641.2</td>
</tr>
<tr>
<td>Delta</td>
<td></td>
<td></td>
<td>14.2</td>
<td>28.6</td>
<td>53.8</td>
<td>97.2</td>
</tr>
<tr>
<td>Delta (%)</td>
<td></td>
<td></td>
<td>7.709</td>
<td>8.223</td>
<td>10.39</td>
<td>13.11</td>
</tr>
<tr>
<td>AVG Delta (%)</td>
<td></td>
<td></td>
<td>10.39298145</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

D. CPU and Resource Utilization Comparable

We did performance evaluation between clusters running we were monitoring Resource for each machine. Graph Figure 5 show the resource CPU and Memory graph compare between default and our algorithm.

Table IV and Table V shows the efficiency of our algorithm we propose the dynamic task size for each TaskTracker (TT) when running WordCount and Sort benchmark. If we compare with Hadoop default scheduler as assign the task to TaskTracker (TT) as equal. For each Job perspective it decreases the execution time for mapreduce job as mentioned in the table 10.92% for WordCount and 9.85% for Sort. That is because we can predict the suitable size of the task from monitoring we are adding to the implementation so it can reduce the request task time for TaskTracker (TT) and assign the appropriate size for TaskTracker (TT).
but if we compare with our algorithm, it is different between timing behaviors of CPU smooth processing. Memory size is very fast increase when map phase has been finish and going to be reduce phase.

V. FUTURE WORK

Algorithm stills have additional logic to plugin as we mention in the Session III.

- Partitioning session process transformation from map phase to reduce phase need to adjust size appropriate with cluster performance. In this point, our algorithm does not care about we just only assign it to the reduce task so somehow it is not suitable for the node in the cluster when we do the reduce phase.

- Speculative Task backup currently is required to backup launch for all task again. To make it more efficiency we can run only the remaining data size but we need to relocation the data pointer in JobTracker. (JT) that is the next challenge point.

REFERENCES


Mr. Apiluck Songwattanasakul came from Samutprakran, Thailand; I have born in Bangkok, Thailand. In 2009, I graduated Bachelor Degree of computer engineering from King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand. Currently I study Master Degree of computer science at Chulalongkorn University.