A Cluster Based Self Adaptive Mapreduce Scheduling Algorithm in Hadoop

R.Thangaselvi, R.Aruna and S.AnanthaBabu

M.E :Department of Computer Science, VV College of Engineering, Tirunelveli, India
M.E :Department of Computer Science, Jayamadha College of Engineering, Kanyakumari, India
Assistant Professor, VV College of Engineering, Tirunelveli, India

Received February 2016; Accepted 18 April 2016; Available 25 April 2016

ABSTRACT

The SAMR's response time by a factor of 2. Hadoop is a free java based programming context that supports the processing of large datasets in a distributed computing environment. Mapreduce campaign is being used in hadoop for processing and generating large datasets with a parallel distributed algorithm on a cluster. A key advantage of mapreduce is that it automatically handles failures and hides the difficulty of fault tolerance from the user. Hadoop uses FIFO (FIRST IN FIRST OUT) scheduling algorithm as default in which the jobs are executed in the order they arrived. This approach suits well for homogeneous cloud and results in disadvantaged performance on heterogeneous cloud. Later the LATE (Longest Approximate Time to End) algorithm has been extended which reduces the FIFO's response time by a factor of 2. It gives better performance in heterogeneous environment. LATE algorithm is based on 3 principles i) prioritizing tasks to speculate ii) selecting fast nodes to run on iii) capping speculative tasks to stop thrashing. It takes action on appropriate slow tasks and it could not compute the remaining time for tasks appropriately and can't find the real slow tasks. Finally a SAMR(Self Adaptive MapReduce) scheduling algorithm is being used which can find slow tasks dynamically by using the historical information recorded on each node to tune parameters. In this paper we proposed a cluster based SAMR algorithm which reduces the performance of hadoop in the heterogeneous environment.

KEYWORDS: Hadoop, Mapreduce, Cloud Computing, Scheduling, SAMR, Tuning, Cluster

INTRODUCTION

Hadoop is a software library context that examine the distributed processing of large datasets across clusters of computers using simple programming method [1]. Mapreduce is the data processing framework that automatically handles failures. It deals with the implementation of processing and generating large datasets with a parallel distributed algorithm on a cluster [2]. Mapreduce is being used in cloud computing because of hiding the complexity of fault tolerance from the programmer [3]. Input data is split and fed to each node in the map phase. The results generated in this phase are shuffled and sorted then fed to the nodes in the slash phase [4]. Hadoop defaultly schedules the task using FIFO algorithm which is static [5]. Later several techniques are being developed which supports homogeneous tasks. LATE the dynamic scheduling algorithm is being introduced to schedule the jobs in heterogeneous environment [6]. Then the SAMR scheduling algorithm is being proposed which uses the historical information and find the slow nodes and launches backup tasks. The historical information is stored in name node in XML format. It adjusts time weight of each stage of map and reduce tasks according to the information in name node respectively[7]. It decreases the execution time of mapreduce job and increases the complete mapreduce efficiency in the heterogeneous environment.

In this paper we are tuning the parameters using cluster based SAMR algorithm and then allocating tasks to each node. Thus the performance of hadoop in the heterogeneous environment is increased. In Hadoop the Name node manages the entire namespace for a Hadoop cluster[9]. With HDFS federation, multiple Namenode servers maintain
namespaces and this allows for parallel scaling, performance improvement, and managing multiple namespaces. YARN, the other major advancement in Hadoop2, shows significant performance improvement for several applications, supports additional processing methods, and implements a more flexible execution engine. YARN is a resource manager that is created by separating the processing engine and resource management capabilities of MapReduce as it was executed in Hadoop 1[18]. YARN is often called the operating system of Hadoop because it is liable for managing and monitoring workloads, managing a multi-tenant environment, implementing security controls, and managing high availability features of Hadoop[10].

**Literature Survey:**

Hadoop uses FIFO algorithm which the tasks are subject to priority in the order they arrived. This algorithm takes greater response time for slower jobs when compared to faster jobs [5]. Then in round robin technique each task is given equal priority [11]. In the fair scheduling algorithm all tasks gain an adequate and equal share of resources over time[12]. Then in capacity scheduling algorithm resources are allocated in a timely approach under constraints of assigned capacities[13]. The weighted Round Robin scheduling algorithm assign weight to each queue then scheduling tasks of various sub queue according to weight[14]. Improved Weighted Round Robin scheduling uses weight update rules to reduce load and to stasis job allocation [15]. Hybrid scheduling is designed for data intensive workloads and tries to uphold data locality while execution [16]. SARS (Self-Adaptive Reduce Scheduling) can decide the run time of each reduce tasks dynamically according to each task context, includes the job completion time [17]. LATE (Longest Approximate Time to End) scheduling promotes the execution in hadoop by finding real slow jobs [6]. SAMR promotes the execution in hadoop by finding real slow jobs using the historical information [7]. The cluster based SAMR algorithm cluster the data nodes according to the historical information and the tasks are allocated to fast data nodes.

<table>
<thead>
<tr>
<th>S.NO</th>
<th>ALGORITHM</th>
<th>ADVANTAGE</th>
<th>DISADVANTAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>First In First Out(FIFO) scheduling</td>
<td>Reduces response time due to speculative execution. Works well in case of short jobs.</td>
<td>Uses fixed threshold for selecting tasks to reexecute. Can't identify which tasks to be reexecuted on fast nodes correctly.</td>
</tr>
<tr>
<td>2</td>
<td>Round Robin scheduling</td>
<td>No need to wait for the previous one to get completed</td>
<td>Largest jobs take enough time for scheduling.</td>
</tr>
<tr>
<td>3</td>
<td>Fair scheduling</td>
<td>Can work well in both small and large clusters</td>
<td>Job weight is not considered for each node.</td>
</tr>
<tr>
<td>4</td>
<td>Capacity scheduling</td>
<td>Improve the utilization of resources through dynamic adjustment of resource allocation. Improve job efficiency.</td>
<td>User needs to know system information and make queue set and queue select group for the job.</td>
</tr>
<tr>
<td>5</td>
<td>Weighted Round Robin scheduling</td>
<td>Can provide good fairness when the size of each task is same.</td>
<td>Provides unfairness for the smaller queues if the size of the task is inconsistent. Due to fixed weight, can't adjust the weight of each sub queue in real time.</td>
</tr>
<tr>
<td>6</td>
<td>Improved Weighted Round Robin scheduling</td>
<td>Easy to implement. Low cost.</td>
<td>Defects occur when consider the external influence on the time that each task took when they switched scheduling. Does not maintain stability under high concurrency, large capacity and high workload.</td>
</tr>
<tr>
<td>7</td>
<td>Hybrid scheduling</td>
<td>Fast and flexible scheduler.</td>
<td>The time taken for the creation of tasks and result retrieval is increased due to the increase in the number of tasks.</td>
</tr>
<tr>
<td>8</td>
<td>Self-adaptive Reduce scheduling(SARS)</td>
<td>Reduces completion time. Decrease the response time.</td>
<td>Only focuses on reduce process.</td>
</tr>
<tr>
<td>9</td>
<td>Longest approximate time to end(LATE) scheduling</td>
<td>Robustness to heterogeneity. Address the problem of how to robustly perform speculative execution to maximize performance.</td>
<td>Does not compute remaining time for tasks correctly and can’t find real slow tasks. Poor performance due to the static manner in computing the progress of the tasks.</td>
</tr>
<tr>
<td>10</td>
<td>Self-adaptive mapreduce(SAMR) scheduling</td>
<td>Decreases the execution time of mapreducejob. Improve the overall mapreduce performance in the heterogeneous environment.</td>
<td>Does not find the slow jobs accurately.</td>
</tr>
</tbody>
</table>

**Theoretical Foundation:**

The MapReduce framework first splits an input data file into G pieces of fixed size, typically being 16 megabytes to 64 megabytes (MB). These G pieces are then passed on to the participating machines in the cluster. Usually, 3 copies of each piece are generated for fault tolerance. It then starts up the user program on the nodes of the cluster. One of the nodes in the cluster is special the master. The rest are workers that are assigned work by the master. There are M map tasks and R reduce asks to assign. M and R is either decided by the configuration specified by the user program, or by the cluster wide default configuration. The master picks idle workers and assigns them map tasks. Once map tasks have generated intermediate outputs the master then assign reduce tasks to idle workers. Note that all map tasks have to finish before any reduce task can begin. This is because a reduce task needs to take output from every map task of the job. A worker who is assigned a map
task reads the content of the corresponding input split. It parses key/value pairs out of the input data chunk and passes each pair to an instance of the user defined map function. The intermediate key/value pairs produced by the map function are buffered in memory at the corresponding machines that are executing them. The buffered pairs are periodically written to a local disk and partitioned into R regions by the partitioning function. The framework provides a default partitioning function but the user is allowed to override this function by a custom partitioning. The locations of these buffered pairs on the local disk are passed back to the master. The master then forwards these locations to the reduce workers. When a reduce worker is notified by the master about these locations, it uses remote procedure calls to read the buffered data from the local disks of map workers. When a reduce worker has read all intermediate data, it sorts it by the intermediate key so that all occurrences of the same key are grouped together.

The sorting is needed because typically many different keys are handled by reduce task. If the amount of intermediate data is too large to fit in memory, an external sort is used. Once again, the user is allowed to override the default sorting and grouping behaviors of the framework. Next, the reduce worker iterates over the sorted intermediate data and for each unique intermediate key encountered, it passes the key and the corresponding set of intermediate values to the reduce function. The output of the reduce function is appended to a final output file for this reduce partition. When all map tasks and reduce tasks have completed, the master wakes up the user program. At this point, the MapReduce call in the user program returns back to the user code.

Fig. 1: Map Reduce framework.

Proposed Methodology:

The SAMR technique uses the historical information that is being stored in each node and using that information it finds the real slow tasks. Then it maps the slow tasks and reduces the slow tasks. In this paper we use the cluster based SAMR technique to tune the parameters in the historical information and finding the slow tasks very accurately. The proposed cluster based SAMR algorithm can solve even the most difficult clustering issues. It requires the number of clusters that we are going to use in our process. The algorithm finds k centroids, one for each cluster. Depending on the location of the centroid the result will vary. During the map phase it finds the M1 temporary value and using this value it finds in the clusters which one is closest to the M1 value. Similarly in the reduce phase it finds the R1 temporary value and using this value it finds in the clusters which one is closest to the R1 value. Based on the result the centroid location is changed and the values are recalculated again.

4b ALGORITHM:
Algorithm 1 Self-adaptive MapReduce:
1: Start procedure
2: input: Key/Value pairs
3: output: Statistical results
4: read historical information
5: tune parameters using proposed k-means clustering
6: Find slow tasks
7: Find slow task trackers
8: Launch back up tasks
9: Using the results update the historical information
10. End procedure
Algorithm 2 Cluster based SAMR:
1: Start procedure
2: Input: D-set of n datanodes, n-number of datanodes, C-set of k centroids, k-number of clusters
3: Output: A-set of k clusters
4: Compute distance between each data nodes to all centroids
5: For each Di find the closest Ci
6: Add Di to A
7: Remove Di from D
8: Repeat for all Di……DnandCi...Ck
9: End procedure

Fig. 2: MapReduce I implementation.

RESULTS AND DISCUSSION

We have proposed cluster based SAMR algorithm to improve the performance of the Self-adaptive MapReduce scheduling algorithm. The proposed cluster based SAMR algorithm works better than the SAMR algorithm. When the input file is given the job tracker manages the information and allocates tasks to its slave nodes which are also known as task trackers. The Namenode contains the metadata and it is the master of all datanodes. The jobtracker and tasktrackers communicate among themselves and the datanodes will correspond to the namenode. The proposed cluster based SAMR algorithm find the closest between each data nodes and centroids. Using this outcome it updates the historical information in the name node and find the exact slow tasks, launch backup tasks and allocate tasks to each task trackers. The proposed cluster based SAMR technique takes lesser amount of computation time than the basic SAMR technique. The table2 expresses the experimental outcome for the Self-Adaptive MapReduce and the cluster based SAMR algorithm. The cluster based SAMR algorithm process the number of files in lesser time than the existing technique. In figure 3 the comparison chart betwistthe SAMR technique and the cluster based SAMR algorithm is being sketched. The number of files are taken in X axis and the computation time in seconds is plotted in Y axis. Figure 4 shows the bar chart representation. The performance is being evaluated by calculating the recall value and the precision value. Figure5 shows the performance evaluation of the cluster based SAMR algorithm.

Table 2: Experimental results.

<table>
<thead>
<tr>
<th>No. of files</th>
<th>Computation time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SAMR</td>
</tr>
<tr>
<td>100</td>
<td>.105</td>
</tr>
<tr>
<td>200</td>
<td>.130</td>
</tr>
<tr>
<td>300</td>
<td>.145</td>
</tr>
<tr>
<td>400</td>
<td>.165</td>
</tr>
<tr>
<td>500</td>
<td>.205</td>
</tr>
</tbody>
</table>
Conclusion:

In this paper we proposed a method to improve the efficiency of the map reduce scheduling algorithms. It works more desirable than other map reduce scheduling algorithms by taking less amount of computation and gives high accuracy. We used the cluster based self-adaptive mapreduce scheduling technique together with the Self-Adaptive MapReduce (SAMR) algorithm. However this algorithm works well it can assign only one task to each data node. In the future we have decided to improve its performance by allocating more number of tasks to the data nodes.

REFERENCES

4. Yaxiong Zhao, Jie Wu and Cong Liu, “Dache: A data aware caching for bigdata applications using the MapReduce framework”, in TSINGHUA science and technology. ISSN 1007-0214 05/10, 39-50, 19-1.
5. JaideepDhok and VasudevaVarma, “Using pattern classification for task assignment in mapreduce”
14. HaiyangLi , “PWBRR algorithm of hadoop platform”.
15. Yu Liping, Yang Lishen and Zhu Liang, “Multiuser scheduling strategy research based on mapreduce cluster”
18. ArunC.Murthy, Vinod Kumar Vavilapalli, Dough Eadline, Joseph Niemiec and Jeff Markham, “Apache Hadoop Yarn: Moving beyond MapReduce and Batch Processing with Apache Hadoop, 2.
19. MapReduce, http://www.mongodb.org/display/DOCS/MapReduce
20. Junjie Wu “Advances in k-means clustering”, a data mining thinking