Survey on MapReduce Scheduling Algorithms

Liya Thomas,
Mtech Student,
Department of CSE,
SCTCE,TVM

Syama R,
Assistant Professor
Department of CSE,
SCTCE,TVM

ABSTRACT
MapReduce is a programming model used by Google to process large amount of data in a distributed computing environment. It is usually used to perform distributed computing on clusters of computers. Computational processing of data stored on either a file system or a database usually occurs. MapReduce takes the advantage of locality of data, processing data on or near the storage areas, thereby avoiding unnecessary data transmission. The simplicity of the programming model and the automatic handling of node failures hiding the complexity of fault tolerance make MapReduce to be used for both commercial and scientific applications. As MapReduce clusters have become popular these days, their scheduling is one of the important factor which is to be considered. In order to achieve good performance a MapReduce scheduler must avoid unnecessary data transmission. Hence different scheduling algorithms for MapReduce are necessary to provide good performance. This paper provides an overview of four different scheduling algorithms for MapReduce namely; Scheduling algorithm in Hadoop, Longest Approximate Time to End (LATE) MapReduce scheduling algorithm, Self-Adaptive MapReduce(SAMR) scheduling algorithm and Enhanced Self-Adaptive MapReduce scheduling algorithm(ESAMR). An overview of these techniques is provided through this paper. Advantages and disadvantages of these algorithms are identified.

Keywords
MapReduce; Programming model; Scheduling algorithms;

1. INTRODUCTION
Nowadays large amounts of data are generated from different sources like scientific instruments, digital media, web authoring etc. The effective storage, querying and analyzing of these data has become a grand challenge to the computing industry. Inorder to meet the high storage and processing of these huge amounts of data the solution is to build a data-centre scale computer system. Such a system consists of hundreds, thousands or even millions of computers which are connected through a local area network organized in a data centre. Parallel processing of these huge amounts of data is necessary in order to process the data in a timely manner.

One of the most popular programming paradigms that enables to process huge amount of data in parallel is the MapReduce programming model. MapReduce is a programming model and an associated implementation for processing and generating large data sets [1]. Under this model, each application is implemented as a sequence of MapReduce operations consisting of a map stage and a reduce stage that process a large number of independent data items. The map stage processes a key/value pair to generate a set of intermediate key/value pairs and the reduce stage merges all the intermediate values associated with the same intermediate key.

One of the important features of MapReduce is that it automatically handles node failures, hiding the complexity of fault tolerance from the programmers. MapReduce allows the map and reduction operations to be done in parallel. Even though this process appears to be inefficient when compared to different sequential algorithms, MapReduce allows the processing of large datasets that can be handled by commodity machines. The parallel processing also provides recovery from partial failure of servers during the operations. As these MapReduce clusters have become popular these days, their scheduling is one of the important factors to be considered.

This paper provides an overview of different scheduling algorithms used in MapReduce in order to improve its performance. The advantages and disadvantages of these algorithms are also described. A description of the MapReduce programming model is also given.

2. BACKGROUND
In this section, the MapReduce programming model and the scheduling in MapReduce are described.

2.1 Programming Model
MapReduce [1] is a programming model enabling a large amount of nodes to handle huge data by cooperation. MapReduce hides the details of parallel execution and makes the programmers focus on data processing. The main functions in MapReduce are Map and Reduce. Map function which is written by user takes as input a key/value pair and gives as output an intermediate key/value pair. Now all the intermediate key/value pair with same intermediate key is grouped together and is given to the Reduce function. The Reduce function which is also written by the user, accepts the entire intermediate key and the set of values associated with it. For each intermediate key the Reduce function applies on all the set of values and returns a zero or more aggregated results [2].

The map and reduce functions used by the user have the following associated types:

Map (key1, value1) ->list (key2, value2)

Reduce (key2, list (value2)) ->list (value2)

2.2 MapReduce Scheduling System
Execution of MapReduce scheduling system involves six steps as illustrated in the Fig 2.1 [1].

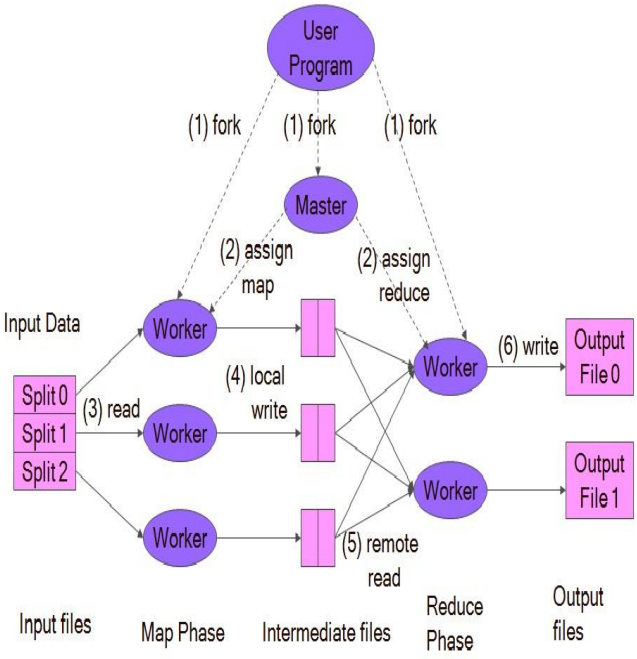

Fig 2.1. Overview of MapReduce job

1. The input data is split into M pieces of 16 megabytes to 64 megabytes each. Then many copies of the program are started on a cluster of machines.

2. One copy of the program is the master and the rest are slaves that are assigned work by the master. M map tasks and R reduce tasks are there to be assigned. The idle workers are picked by the master program and each one is assigned with a map task or a reduce task.

3. The contents of the corresponding input split is read by the worker who is assigned a map task. After parsing the key/value pairs out of the input data, it passes each pair to the user defined Map function. The intermediate key/value pairs which are produced by the Map function are buffered in memory.

4. The buffered pairs are written to local disk and partitioned into R regions by the partitioning function periodically. The locations of these buffered pairs on the local disk are passed back to the master who takes the responsibility of forwarding them to the reduce workers.

5. While the master notifies about these locations to a reduce worker, remote procedure calls are used by it to read the buffered data from the local disks of the map workers. After reading all intermediate data, a reduce worker sorts it by the intermediate keys in order to group together all occurrences of the same key. Typically many different keys map to the same reduce task and hence the sorting is needed. An external sort is used if the amount of intermediate data happens to be too large to fit in the memory.

6. The reduce worker interacts over the stored intermediate data. It passes the key and the corresponding set of intermediate values to the users Reduce function for each unique intermediate key encountered. For the reduce partition, the output of the Reduce function is appended to a final output file.

7. After all map tasks and reduce tasks are completed the master wakes up the user program and the MapReduce call returns to the user program.

The R output files will have the output of the MapReduce execution after the successful completion of the call. The users do not need to combine the R output files into a single file, they usually pass these files as input to another MapReduce call [3].

3. MAPREDUCE SCHEDULING ALGORITHMS

3.1 MapReduce Scheduling Algorithm in Hadoop

Apache Hadoop [4] is an open source implementation of the Google's MapReduce parallel processing framework. Hadoop framework allows programmers to write parallel processing programs hiding the details of its parallel execution and focuses on the computation problems.

The main parts in Hadoop architecture is the Hadoop Distributed File System (HDFS) [5] which stores huge amounts of data across multiple machines with high throughput access to the data on clusters and the Hadoop MapReduce framework which performs the distributed processing of data on clusters. HDFS achieves reliability and fault tolerance of MapReduce framework by storing and replicating the inputs and outputs of Hadoop job. A scheduling policy must be used in order to determine when a job can execute its task since the Hadoop jobs share the cluster resources.

By default Hadoop uses First in First out (FIFO) scheduling algorithm where the jobs are executed in the order of their submission. The user job after partitioning into individual tasks is loaded into a queue and is assigned to free slots in TaskTracker nodes as they become available. One drawback of the FIFO scheduler is that it cannot identify tasks which are needed to be re-executed on fast nodes correctly. Speculative execution is introduced in Hadoop in order to minimize job's response time.

Usually when a node has an empty slot, Hadoop chooses a task from one of the three categories

1) Any failed tasks with highest priority.
2) Any Non-running tasks are considered.
3) Any slow tasks which are needed to be executed speculatively are considered.

Inorder to select a speculative task, Hadoop checks the progress of a task using a progress score whose value ranges between 0 and 1.The progress score is the fraction of input data read for a map task and for reduce task it is 1/3 of the three phases of execution. The copy phase is that where a task fetches a map output, the sort phase is that where the map outputs are sorted by using the key and the reduce phase is where a user defined function is applied to a list of map outputs with a key.

PSavg denotes the average progress score of a job [5]. PS[i] denotes the Progress Score of the ith task. Suppose T is the number of tasks which are being executed , N is the number of key/value pairs that need to be processed in a task ,M is the number of key/value pairs that have been processed

successfully in a task , the map task spends negligible time in the order stage (i.e., M1=1 and M2=0) and the reduce task has finished K stages and each stage takes the same amount of time (i.e., R1=R2=R3=1/3). Hadoop calculates PS according to the Eq. (3.1) and Eq. (3.2), and launches backup tasks according to Eq. (3.3). If Eq. (3.3) is satisfied, then the task T_i needs a backup task.

$$PS = \begin{cases} \frac{M}{N} & \text{For MT} \\ \frac{1}{3} * \left(K + \frac{M}{N}\right) & \text{For RT} \end{cases} \quad (3.1)$$

$$PS_{avg} = \sum_{i=1}^{T} \frac{PS[i]}{T} \quad (3.2)$$

$$\text{For task } T_i: PS[i] < PS_{avg} - 20\% \quad (3.3)$$

3.2 Longest Approximate Time to End (LATE) MapReduce Scheduling Algorithm

LATE [6] algorithm improves the execution in Hadoop by finding real slow tasks. For this it computes the remaining time of all the tasks and selects a set of tasks with longer remaining time when compared to all the nodes and considers them as real slow tasks. In this algorithm it does not depend on the data locality property for launching a speculative map task.

In this algorithm the task which is speculatively executed is one which will finish farthest into the future because this task provides the greatest opportunity for a speculative copy to overtake the original and reduce the job's response time [6].If nodes run at consistent speeds and if there is no cost to launch a speculative task on an otherwise idle node, this policy is optimal.

Usually speculative tasks are launched only on fast nodes and not on stragglers in order to beat the original task with the speculative task. For this a heuristic is followed, i.e., don't launch speculative tasks on nodes that are below some threshold(SlowNodeThreshold) of total work performed. Another method is to allow more than one speculative copy of each task, but it is just the wastage of resources. Inorder to handle the fact that speculative tasks cost resources, the algorithm is implemented with two heuristics:

- A SpeculativeCap which is a limit on the number of speculative tasks that can be running at once.
- A SlowTaskThreshold that a task's progress rate is compared with, to determine whether it is "slow enough" to be speculated upon. This is done to prevent needless speculation when only fast tasks are running.

In a nutshell the LATE algorithm works as follows:

- If a node asks for a new task and there are less speculative tasks running than the SpeculativeCap:
  - When the node's total progress is below SlowNodeThreshold defined, the request is ignored.
  - Running tasks that are not speculated by its time to finish are ranked currently.
  - A copy of the highest-ranked task with progress rate below SlowTaskThreshold is launched.

LATE MapReduce scheduling algorithm also uses Eq.3.1 to calculate task's progress score, but it launches backup tasks for those tasks that will finish farthest into future. Suppose a task T has run T_r seconds. Let PR denotes the progress rate of T, and TTE denotes how much time remains until T is finished. LATE algorithm computes ProgressRate (PR) and TimeToEnd (TTE) according to Eqs. (3.4) and (3.5).

$$PR = \frac{PS}{T_r} \quad (3.4)$$

$$TTE = \frac{1.0 - PS}{PR} \quad (3.5)$$

3.3 A Self-Adaptive MapReduce Scheduling Algorithm (SAMR)

SAMR [7] scheduling algorithm uses the historical information to get more accurate PSs of all the tasks. Since SAMR finds real slow tasks by using the accurate PSs it decreases the execution time compared with Hadoop and LATE. So by using SAMR algorithm the execution time and the system resources are saved. Inorder to tune parameters SAMR combines historical information recorded on each node and dynamically find the slow tasks. SAMR classifies slow nodes into map slow nodes and reduce slow nodes for saving the system resources. Map or reduce slow nodes means nodes which execute map/reduce task using a longer time than most other nodes in the system.

As in the case of LATE, SAMR also estimates the remaining execution time to identify the slow tasks. Unlike Hadoop default and LATE schedulers, which assume M1, M2, R1, R2, and R3 values to be 1, 0, 1/3, 1/3, and 1/3 respectively, SAMR records M1, M2, R1, R2, and R3 values on each TaskTracker node and uses these historical information to facilitate more accurate estimation of task's TTE.

The process of SAMR algorithm includes reading the historical information and tuning parameters, finding the slow tasks, finding the slow TaskTracker, launching backup tasks, collecting results and updating the historical information.

The Fig 3.1 shows the process of using and updating the historical information [7]. First, TTs read historical information from the nodes where they are running on. These historical information includes historical values of M1, M2, R1, R2 and R3. Now TTs tune M1, M2, R1, R2 and R3 according to historical information and information collected from the current running system. Consequently, TTs collect values of M1, M2, R1, R2 and R3 according to the real running information after the tasks finished. Finally, TTs write these updated historical information into the xml file in each of the nodes.

SAMR computes PS by tuning parameters, to calculate the correct remaining time of tasks.

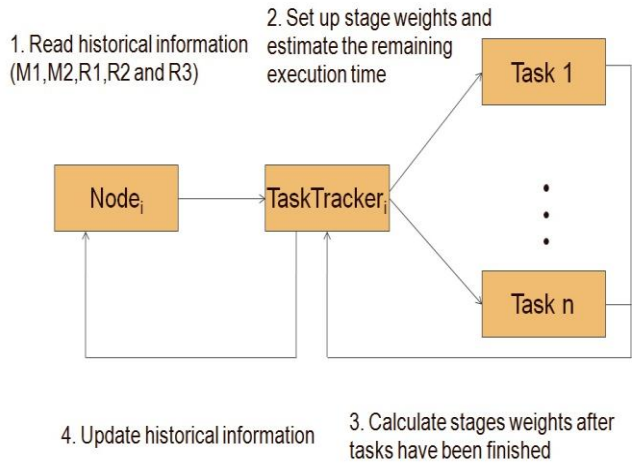

Fig 3.1 The way to use and update historical information

3.4 ESAMR: An Enhanced Self-Adaptive MapReduce Scheduling Algorithm

Like SAMR, ESAMR [8] scheduling algorithm also considers the fact that slow tasks extends the execution time of the whole job and due to hardware heterogeneity different amounts of time are needed to complete the same task on different nodes. ESAMR records historical information for each node as in case of SAMR and it adopts a k-means clustering algorithm to dynamically tune stage weight parameters and to find slow tasks accurately. ESAMR significantly improves the performance of MapReduce scheduling in terms of estimating task execution time and launching backup tasks [9].

ESAMR uses a machine learning technique to classify the historical information stored on every node into k clusters. If a running job has completed some map tasks on a node then ESAMR records the job's temporary map phase weight (i.e., M1) on the node according to the job's map tasks completed on the node and uses the temporary M1 weight to find the cluster whose M1 weight is the closest. ESAMR then uses the cluster's stage weights to estimate the job's map tasks TimeToEnd on the node and identify slow tasks that need to be re-executed.

In the reduce stage, after a job has finished, ESAMR calculates the job's stage weights on every node and saves these new weighs as a part of the historical information. Finally, ESAMR applies k-means algorithm [10], [11] , to re-classify the historical information stored on every worker node into k clusters and saves the updated average stage weights for each of the k clusters. Since ESAMR make use of more accurate stage weights to estimate the TimeToEnd of running tasks, it can identify slow tasks more accurately than SAMR, LATE, and Hadoop default scheduling algorithms.

4. COMPARISON

The speculative task scheduling with a simple heuristic method which compares the progress of each task to the average progress is implemented by the Hadoop scheduler. For re-execution, tasks with the lowest progress compared to the average are selected. However in a heterogeneous environment where each node has different computing power the heuristic method is not well suited.

To improve the response time of Hadoop in heterogeneous environments Longest Approximate Time to End (LATE) scheduling is devised. The task progress with the progress rate, rather than simple progress score is estimated by this scheduling scheme. By attempting to find real slow tasks by computing remaining time of all the tasks it tries to improve Hadoop. Data locality for launching speculative map tasks is not taken into account by it.

With similar idea of LATE MapReduce scheduling algorithm SAMR is developed. By using historical information SAMR gets more accurate PSs of all the tasks. In terms of saving time of execution as well as system resources SAMR improves MapReduce.

ESAMR, like SAMR is inspired by the fact that slow tasks prolong the execution time of the whole job and different amounts of time are needed to complete the same task on different nodes due to hardware heterogeneity. ESAMR records historical information for each node it adopts a k-means cluster identification algorithm to dynamically tune stage weight parameters and find slow tasks accurately. In terms of estimating task execution time and launching backup tasks ESAMR significantly improves the performance of MapReduce scheduling. ESAMR can identify slow tasks more accurately than SAMR, LATE, and Hadoop default scheduling algorithms By utilizing more accurate stage weights to estimate the TimeToEnd of running tasks.

Table 1. Comparison of different algorithms

Algorithm	Advantages	Disadvantages
Hadoop Default Scheduler	➢ Reduces response time due to speculative execution. ➢ Works well in case of short jobs.	➢ Static time weights for task stages. ➢ Uses a fixed threshold for selecting tasks to re-execute. ➢ May launch backup tasks for fast tasks. ➢ Hadoop always launches backup tasks for those tasks of which PSs are less than PSavg– 20%.

LATE	➢ Robust to node heterogeneity. ➢ Takes into account node heterogeneity when deciding where to run speculative tasks. ➢ Speculatively executes only tasks that will improve job response time, rather than any slow tasks.	➢ LATE does not approximate TTE of running tasks correctly. So chooses wrong tasks for re-execute. ➢ It does not distinguish map slow nodes and reduce slow nodes. ➢ Poor performance due to static manner in which it computes progress of tasks.
SAMR	➢ Uses historical information to tune weights of map and reduce stages**.**	➢ It does not consider that the dataset sizes and the job types can also affect the stage weights of map and reduce tasks.
ESAMR	➢ Can identify slow tasks more accurately. ➢ Improves the performance in terms of estimating task execution time and launching backup tasks.	➢ Little overhead due to K-means algorithm. ➢ Allows only one speculative copy of a task to run on a node at a time.

5. CONCLUSION

A comparison of four different map reduce scheduling algorithms are attempted through this paper. Hadoop default scheduler takes care of only homogeneous clusters. It includes speculative execution by comparing the progress of each task to the average progress. The response time of Hadoop in heterogeneous environments is improved by the Longest Approximate Time to End (LATE) scheduling. SAMR has the similar idea of LATE MapReduce scheduling algorithm. However SAMR gets more accurate PSs of all the tasks by using historical information. Historical information for each node is recorded by ESAMR and a k-means cluster identification algorithm is adopted to dynamically tune stage weight parameters and find slow tasks accurately. ESAMR can identify slow tasks more accurately than SAMR, LATE, and Hadoop default scheduling algorithms by utilizing more accurate stage weights to estimate the TimeToEnd of running tasks.

6. REFERENCES

[1] J. Dean and S. Ghemawat, "Mapreduce: simplied data processing on large clusters," (2004) in OSDI 2004: Proceedings of 6th Symposium on Operating System Design and Implemention,(New York), pp. 137–150, ACM Press.

[2] J. Dean and S. Ghemawat, "MapReduce: a flexible data processing tool,"(2010) Communications of the ACM, vol. 53, no. 1, pp. 72–77.

[3] C. Jin and R. Buyya, "MapReduce programming model for .NET-based distributed computing," (2009) in Proceedings of the 15th European Conference on Parallel Processing.

[4] " Apache Hadoop." http://hadoop.apache.org.

[5] Hadoop Distributed File System, http://hadoop.apache.org/hdfs.

[6] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," (2008) in 8th Usenix Symposium on Operating Systems Design and Implementation, (New York), pp. 29–42, ACM Press.

[7] Quan Chen; Daqiang Zhang; Minyi Guo; Qianni Deng; Song Guo; , "SAMR: A Self-adaptive MapReduce Scheduling Algorithm in Heterogeneous Environment,"(2010) Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on , vol., no., pp.2736-2743.

[8] Xiaoyu Sun, Chen He and Ying Lu "ESAMR: An Enhanced Self-Adaptive MapReduce Scheduling Algorithm"(2012) IEEE 18th International Conference on Parallel and Distributed Systems.

[9] R. Nanduri, N. Maheshwari, A. Reddyraja, and V. Varma, "Job aware scheduling algorithm for mapreduce framework,"(2011) in Proceedings of the 3rd International Conference on Cloud Computing Technology and Science, CLOUDCOM '11, (Washington, DC, USA), pp. 724–729, IEEE Computer Society.

[10] "K-means." http://en.wikipedia.org/wiki/K-means clustering.

[11] G. Hamerly and C. Elkan, "Alternatives to the k-means algorithm that find better clusterings,"(2002) in Proceedings of the 11th international conference on Information and knowledge management, CIKM '02, (New York, NY, USA), pp. 600–607, ACM.