A COMPARATIVE REVIEW OF JOB SCHEDULING FOR MAPREDUCE

Dongjin Yoo, Kwang Mong Sim

Multi-Agent and Cloud Computing Systems Laboratory,
School of Information and Communication,
Gwangju Institute of Science and Technology (GIST), Gwangju, Republic of Korea
dongjinyoo@gist.ac.kr, prof_sim_2002@yahoo.com

Abstract

MapReduce is an emerging paradigm for data intensive processing with support of cloud computing technology. MapReduce provides convenient programming interfaces to distribute data intensive works in a cluster environment. The strengths of MapReduce are fault tolerance, an easy programming structure and high scalability. A variety of applications have adopted MapReduce including scientific analysis, web data processing and high performance computing. Data Intensive computing systems, such as Hadoop and Dryad, should provide an efficient scheduling mechanism for enhanced utilization in a shared cluster environment. The problems of scheduling map-reduce jobs are mostly caused by locality and synchronization overhead. Also, there is a need to schedule multiple jobs in a shared cluster with fairness constraints. By introducing the scheduling problems with regards to locality, synchronization and fairness constraints, this paper reviews a collection of scheduling methods for handling these issues in MapReduce. In addition, this paper compares different scheduling methods evaluating their features, strengths and weaknesses. For resolving synchronization overhead, two categories of studies; asynchronous processing and speculative execution are discussed. For fairness constraints with locality improvement, delay scheduling in Hadoop and Quincy scheduler in Dryad are discussed.

Keywords: MapReduce, Job Scheduling, Locality, Fairness

1 Introduction

MapReduce [5] is currently the most famous framework for data intensive computing. MapReduce is motivated by the demands of processing huge amounts of data from a web environment. MapReduce provides an easy parallel programming interface in a distributed computing environment. Also MapReduce deals with fault tolerance issues for managing multiple processing nodes. The most powerful feature of MapReduce is its high scalability that allows user to process a vast amount of data in a short time. There are many fields that benefit from MapReduce such as bioinformatics [13], machine learning [4], scientific analysis [6], web data analysis, astrophysics [12], and security [12]

The structure of MapReduce is primarily based on the master-slave architecture. A single master node monitors the status of the slave nodes and assigns jobs to them. A task assigned to slave nodes has two phases of processing; map and reduce. From the map process, intermediate results are generated and transferred to the reduce process as input. The reduce process sorts the intermediate results by keys then merges them as one final output. There is a synchronization step between the map and reduce processes. The synchronization phase is a data communication step between the mapper and reducer nodes to launch the reduce process. The architecture of the MapReduce model follows a shared nothing structure [14] where each task on a node has no knowledge of other tasks. This leads to simplicity of data processing and fault tolerance.

There are some implemented systems for data intensive computing such as Hadoop[2] , Dryad [9] and Sphere [8]. An open source framework, Hadoop resembles the original MapReduce [5]. The internal data processing model of Dryad is different from the MapReduce model. Dryad supports a graph represented data processing model like DAG (Direct Acyclic Graph).

To manage multiple nodes with multiple map-reduce jobs in a shared cluster environment, an efficient scheduling mechanism is essential to achieve the best performance. There are several main factors that affect scheduling issues such as locality, synchronization and fairness constraints. Locality is about placing a task on the node containing the input data. Scheduling considering locality can save network I/O cost, since network

978-1-61284-204-2/11/$26.00 ©2011 IEEE

bandwidth is limited in a shared cluster. Synchronization overhead is caused by late mappers, which cause the entire delay in the reduce processes and an underutilization problem. Another important issue is how to allocate jobs with fairness among multiple jobs and multiple users.

The contributions of this paper are; 1) review a collection of approaches for scheduling jobs for MapReduce considering scheduling issues including locality, synchronization overhead and fairness constraints, and 2) compare their features, strengths and weaknesses.

The rest of this paper is organized as follows: section 2 will describe main issues affecting the performance of scheduling in MapReduce. In Section 3, several approaches answering the effects from each major scheduling issue will be discussed. Section 4 will compare the methods for job scheduling and discuss unaddressed issues to be resolved for scheduling map-reduce jobs in a shared cluster environment.

2 Scheduling issues in MapReduce

In this section, the main scheduling issues in MapReduce such as locality (Section 2.1), synchronization barrier (Section 2.2), and fairness criterion (Section 2.3) will be discussed.

2.1 Locality

One of the complicating issues of MapReduce is scheduling considering locality (the distance between a node with the input data and the task-assigned node). When input data is nearer to the computation node, it has a lower data transfer cost. Locality is a very crucial issue affecting performance in a shared cluster environment, due to limited network bisection bandwidth [5]. In fact, high locality enhances the throughput of tasks. The most efficient case of locality is processing a task on a node holding the data, called node locality. Sometimes when achieving node locality is impossible, executing a task on the same rack, called rack locality, is preferred. If locality is not fulfilled, data transferring I/O costs can seriously affect performance because of the shared network bandwidth. Most methods of scheduling map-reduce jobs have a policy of attempting to assign tasks to a place near the input data to save network cost. Locality is regarded as a basic principle to scheduling jobs with other scheduling constraints.

2.2 Synchronization

The synchronization phase, which is the process of transferring intermediate output of the map processes to the reduce processes as input, is also a factor affecting performance. Multiple mappers have to wait until all the map processes are finished to start sending intermediate output. Due to the dependency between the map and reduce phases, a single node can slow down the entire process, causing the other nodes to wait until it is finished.

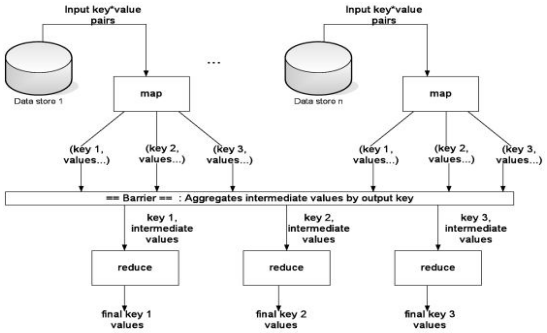

Figure 1 The synchronization barrier [19]

There are various reasons that result in performance degradation in the synchronization step such as heterogeneity of the cluster, miss-configuration, node failures, and serious overhead of the I/O cost. In a heterogeneous cluster environment, each node has different computation capability and/or network bandwidth varies among nodes in the same data center. Therefore, nodes may not perform tasks at an equal rate. This causes inefficient nodes to sit idle while waiting for a reduce slot. Also, failures can delay synchronization. Rather than delays caused by some slow nodes, the shuffle phase itself can cause bottlenecks in network bandwidth or I/O overhead in several operations, for example PageRank [16] and iterative operations of map-reduce jobs. In such operations, the I/O cost is even higher than the computation cost. We will discuss how some approaches handle these kinds of issues in section 3.1.

2.2 Fairness criterion

Multiple map-reduce jobs are performed in a shared data warehouses of cloud computing enterprises like Amazon, Google and Yahoo. A map-reduce job with a heavy workload may dominate utilization of the shared clusters, so some short computation jobs may not have the desired response time. The demands of the workload can be elastic, so fair a workload to each job sharing cluster should be considered.

Fairness constraints have tradeoffs with locality and dependency between the map and reduce phases. When each map-reduce job has roughly an equal share of the nodes and the input files are spread in a distributed file system, some map processes have to load data from networks. This causes degradation of throughput and response time. Also, synchronization overhead could influence fairness; for example, reduce processes have to wait for the

completion of map processes, which leads to idle nodes and starvation of other jobs. This problem causes a poor utilization situation.

3 Scheduling methods in MapReduce

In this section, methods and algorithms handling issues of map-reduce job scheduling will be described. In section 3.1, methods dealing with synchronization overhead will be described. Algorithms regarding fairness issues will be illustrated in Section 3.2.

3.1 Methods for synchronization overhead

MapReduce frameworks such as Hadoop [2] enforce a synchronization barrier between the map-process and the reduce-process. The reduce phase does not start until the entire map phase is finished, and this causes synchronization overhead for reasons such as heterogeneous environment, and the map-reduce jobs requiring a heavy I/O operation.

A. Asynchronous Processing

One of the ways to handle this issue is asynchronous techniques to process the map and reduce phases. From works by Elteir et al. [7], two approaches to asynchronous data processing were introduced; hierarchical reduction and incremental reduction. To solve the underutilization problem caused by node heterogeneity or alternation of map computation and I/O operation in the map phase, these two methods asynchronously process intermediate results from the map phase. Hierarchical reduction aggregates and reduces partial results from the map process along a tree like order. In incremental reduction, as described in Fig. 2, the reduce phase is started as soon as a certain amount of the map process is finished, and every reducer incrementally reduces output from the map process. Performance releasing overhead by incremental reduction achieves better results than hierarchical reduction. As a result, incremental reduction outperforms Hadoop [2] in terms of speedup in word-count and grep applications by 35.33% and 57.98%, respectively. However, this method is only applicable to recursively reducible jobs, for example, grep and word-count.

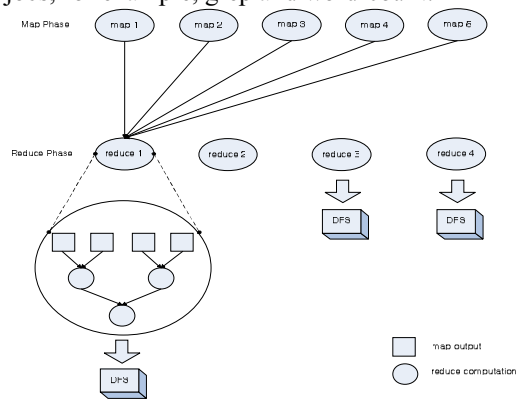

Figure 2 Incremental reduction by [7]

Another approach to solve the problem of synchronization overhead in an asynchronous fashion is partial synchronization of the map and reduce phases [11]. The initiative of this method is relieving global synchronization overhead caused by the strict synchronization barrier and iterative operation requiring heavy I/O loads. According to this asynchronous algorithm, there are two levels of processing for each map and reduce phase; local and global map-reduce, as described in Fig. 3. By replacing global synchronization with two levels of partial synchronization, applications requiring iterative operation can be enhanced with locality improvement. The benefit of this approach is that it solves overhead problems in both the map and reduce phases for some applications requiring iterative operations. However, this algorithm does not address the problem caused by heterogeneity of nodes in a cluster. Like [7], this approach is only applicable for some applications admitting the asynchronous process.

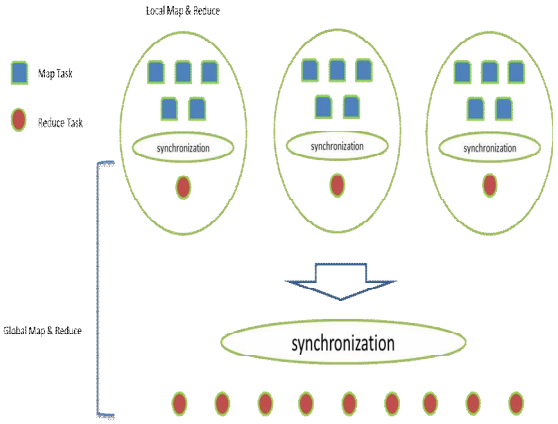

Figure 3 Partial synchronization by [11]

B. Speculative Execution

Another approach is applying speculative task scheduling. Speculative execution considers late nodes (called stragglers) that create a slowdown in the entire synchronization phase. This mechanism is basically adopted by Hadoop [2]. In Hadoop, JobTracker monitors the progress of each task from TaskTrackers and estimates straggler tasks. If any straggler tasks are estimated, the scheduler assigns and starts copies of these tasks on other nodes. The estimation scheme is made by a progress score (between 0 and 1) of each map and reduce task. If there is a task with a progress score relatively low compared to the others, it is marked as a straggler task. The reasons for occurrence of stragglers could be hardware degradation issues, software problems, or failed re-executing tasks.

Since the Hadoop scheduler assumes a homogeneous environment, speculative execution doesn't work properly in a heterogeneous

environment. Zaharia et al. [19] proposed a scheduling algorithm based upon the speculative execution scheduler in Hadoop, called LATE (Longest Approximate Time to End) for resolving the limitation of Hadoop scheduler in a heterogeneous environment. The LATE algorithm uses a *SlowTaskThreshold* meaning slow enough to prevent unnecessary speculation. The LATE algorithm ranks tasks by estimated time remaining and starts a copy of the highest ranked task that has a progress rate lower than the *SlowTaskThreshold*. The advantage of the LATE algorithm is robustness to node heterogeneity, since only some of the slowest (not all) speculative tasks are restarted. This method does not break the synchronization phase between the map and reduce phases, but only takes action on appropriate slow tasks.

3.2 Methods for fairness with locality improvement

There is a growing need to manage resources with fairness constraints for multiple users in a shared cluster. The major issue when dealing with fairness constraints is a conflict with locality. There are two approaches for achieving fairness with locality enhancement; they are Delay Scheduling [18] with Hadoop [2] or Quincy [10] with Dryad [9].

A. Delay Scheduling with FairScheduler

Delay scheduling [18] is currently applied to FairScheduler [1] in Hadoop. FairScheduler uses a two level scheduling hierarchy. Each user has its own pool in a shared cluster, and a minimum share is assigned to each pool. In each user's pool, multiple jobs share the pool's time slots. Among multiple pools, each pool does not share an equal amount of time slots, and the degree of fairness is weighted. A minimum share is the least amount of slots that should be guaranteed. If a pool doesn't use its slots, other pools can use the idle slots instead. If there is no slot remaining in a pool so that a user cannot get the minimum share, preemption occurs, which reassigns slots used by another job to the user who wants the slots. There are two ways of preemption; killing the running task immediately or waiting for the running task to finish. Killing the task sacrifices the time it had been running, but the waiting method sacrifices a degree of fairness.

The delay scheduling mechanism uses the waiting approach to enhance locality, because rescheduling a task, which was previously assigned to another user for fairness, loses the opportunity of placing tasks on the node with the data. The delay scheduling algorithm does not immediately reassign a task for the unfair job, but reassign the task with some delay. The aim of delay scheduling is enhancing locality with relaxed fairness. Also delay scheduling resolves other locality problems such as head-of-line scheduling and sticky slots. Head-of-line scheduling problems occur when small jobs are unlikely assigned to a node without the data. The sticky slot problem occurs when tasks of a job are likely finished earlier than others, and the job does not leave the original slot. The sticky slots occur for relatively large jobs. The benefit of this method is achieving fairness with locality improvement. By relaxing fairness slightly and not trying to allocate tasks with a purely greedy approach, the delay scheduling eventually removes suffering from the overhead of transferring costs over the network.

B. Quincy Scheduler

Quincy [10] is another approach to accomplishing fairness with multiple jobs on a Dryad [9] framework. Dryad is a data intensive processing framework that supports a data flow operation with DAG (Directed Acyclic Graph). Due to the distinct architecture between MapReduce [5] and Dryad, allocation mechanisms cannot be compared under the same conditions. However, Quincy also deals with achieving fairness with locality improvement. Quincy basically adopts flow based scheduling, whereas queue based scheduling is used in Hadoop [2]. Every form of job scheduling is represented in graph form in Dryad, so the primary approach is optimizing the cost of the data flow graph. This method is called the min-cost flow. Via this method, the tasks of workers are connected to the graph where each node represents a computation node, rack aggregator, cluster aggregator or sink node. The way to optimize the cost of data processing is to minimize the cost of the data flow graph. The weights of nodes in the graph represent the degree of locality and fairness.

To achieve fairness constraints, four policies are used; fair sharing with preemption, fair sharing without preemption, unfair sharing with preemption, and unfair sharing without preemption. By changing the capacities of the edge of the flow graph, the four policies can be achieved. The Quincy algorithm has more information about the cluster than Hadoop, but the optimization approach is more complex than the delay scheduling algorithm [18].

4 Comparison and unaddressed issues

There are several points to discuss regarding the various methods for job scheduling in MapReduce. First, there are many variances to resolving synchronization overhead in regards to heterogeneity and applicability. Karthik et al.[11] does not focus on resolving overhead by late nodes causing the entire process to slow down, but this method specifically relieves overhead from a task

Table 1 Comparison of Approaches

Methods	Description	Strength	Weakness
Asynchronous Processing	allowing asynchronous processing	eagerly resolving overheads	limited application
Karthik et al.[11]	two levels of partial synchronization - local/global map and reduce	relieving synchronization overheads from both map and reduce phase	limited application no considerations of heterogeneity
Elteir et al.[7]	the reduce process doesn't wait until entire the map process - incremental reduction	consideration of heterogeneity	applicable only for recursively reducible jobs
Speculative Execution	monitoring whether a task is stragglers - duplicate and launch straggler task	no limitation of applicability	not eagerly solve synchronization overheads
pure speculation [2]	restart duplicates of all slow tasks	no particular	no consideration of heterogeneity
LATE [19]	restart some slowest tasks considering node heterogeneity	robustness to node heterogeneity	no particular
Scheduler for fairness			
Delay scheduling [18] with FairScheduler [1]	queue based scheduling relaxing fairness for locality enhancement	simplicity of scheduling algorithm	no particular
Quincy scheduler [10]	flow based scheduling optimization using min cost flow mechanism	better choice of dataflow	more complex scheduling algorithm

with heavy I/O operations, which can cause potential synchronization overhead. On the other hand, the primary approaches of speculative execution with LATE [19] and the method of Elteir et al. [7] focus on late nodes and node heterogeneity. In terms of applicability, asynchronous processing approaches ([11], [7]) cannot be applied to some operations that do not allow asynchrony, whereas speculative execution ([2],[19]) has no limitations on applicability. Also, the asynchronous processing methods ([11], [7]) directly remove effects from late nodes or potential effects caused by the late nodes. However, the speculative execution methods ([2], [19]) only estimate late nodes and restart copies of the late tasks. Second, in terms of fairness constraints, the two methods ([10], [18]) are distinct in how to achieve fairness. Quincy [10] considers every schedule problem as an optimization problem with graph data flow, and the schedulers have more knowledge of the nodes than the schedulers in Hadoop [2]. Thus, Quincy [10] may make better choices for data flow considering fairness than the delay scheduling method. However, the optimization process is more complex than delay scheduling. Both of the methods achieve fairness and locality enhancement, and use a preemption mechanism. In summary, comparisons of the features, strengths and weaknesses of various scheduling mechanism are presented in Table 1.

There have been a number of studies researching the scheduling of map-reduce jobs. Beyond job scheduling mechanisms, there is a need for scheduling workflow consisting of multiple map-reduce jobs with precedence dependency. Workflow can be represented as a form of DAG. Oozie [17] provides interfaces of an application composed of map-reduce jobs, Pig [3] operation, and HDFS with precedence constraints. Oozie supports DAG representation of workflow, but it does not provide an optimization mechanism for scheduling workflow considering scheduling issues. In other research, Kepler [15] provides a scientific workflow application by integrating with Hadoop. According to Kepler's experimental results with Hadoop, overhead by scientific workflow with a Hadoop scheduler were observed. Since there is no optimization scheme or consideration of scheduling issues in workflow, the performance of Kepler with Hadoop is lower than pure map-reduce jobs in Hadoop. The next step toward job scheduling in MapReduce could be optimizing workflow schedules considering scheduling issues like locality and synchronization overhead.

5 Conclusions

This paper reviewed the following issues for scheduling map-reduce jobs in a shared cluster: 1) locality, 2) synchronization overhead and 3) fairness constraints. Several studies handling these issues were introduced. To resolve synchronization overhead, two approaches were described: 1) asynchronous processing and 2) speculative

execution. Also, for the fairness constraint in a shared cluster environment, two methods were presented: 1) delay scheduling with fair scheduler and 2) Quincy in Dryad. Comparisons among the different approaches with features, strengths and weaknesses were discussed.

Acknowledgements

This work was supported by a Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea Government (MEST 2009-0065329) and the DASAN International Faculty Fund (project code: 140316).

References

[1] Apache Software Foundation. FairScheduler, http://hadoop.apache.org/mapreduce/docs/r0.21.0/fair_scheduler.html

[2] Apache Software Foundation. Hadoop, http://hadoop.apache.org/core/

[3] Apache Software Foundation. Pig, http://pig.apache.org/

[4] C. Chu, S. Kim, Y. Lin, Y. Yu, G. Bradski, A. Ng, and K. Olukotun, Map-Reduce for Machine Learning on Multicore. http://www.cs.stanford.edu/people/ang//papers/nips06-mapreducemulticore.pdf. 2006.

[5] J. Dean and S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters. In proceeding of the 6th Symposium on Operating systems Design and Implementation (OSDI 2004), pp 137-150. USENIX Association, 2004.

[6] J. Ekanayake, S. Pallickara, G. Fox, MapReduce for Data Intensive Scientific Analyses. Proceeding ESCIENCE '08 Proceedings of the 2008 Fourth IEEE International Conference on eScience. 2008.

[7] M. Elteit, H. Lin, and W. Feng, Enhancing Mapreduce via Asynchronous Data Processing. Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS, pp. 397-405. 2010.

[8] Y. Gu, R. L. Grossman. Sector and sphere: The design and of a high-performance data cloud. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences Volume 367, Issue 1897, 28 June 2009, pp 2429-2445.

[9] M. Isard, M. Budiu, Y. Yu, A. Birell and D. Fetterly, Dryad: Distributed data-parallel programs from sequential building blocks. In EuroSys 2007, pp. 59-72.

[10] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar and A. Goldberg, Quincy: Fair scheduling for distributed computing clusters. In SOSP 2009, 2009

[11] K. Kambatla, N. Rapolu, S. Jagannathan, and A. Grama, Asynchronous Algorithm in MapReduce. Proceedings - IEEE International Conference on Cluster Computing, ICCC, art. No. 5600303, pp. 245-254. 2010

[12] G. Mackey, S. Sehrish, J. Bent, J. Lopez, S. Habib, J. Wang, Introducing map-reduce to high end computing. Proceedings of the 2008 3rd Petascale Data Storage Workshop, PDSW 2008.

[13] A. Matsunaga, M. Tsugawa, and J. Fortes, CloudBLAST: combining MapReduce and virtualization on distributed resources for bioinformatics applications. Proceedings - 4th IEEE International Conference on eScience, eScience 2008, pp. 222-229.

[14] M. Stonebraker, The Case for Shared Nothing. http://db.cs.berkeley.edu/papers/hpts85-nothing.pdf. 1986.

[15] J. Wang, D. Crawl, and I. Altintas, Kepler + Hadoop: A general architecture facilitating data-intensive applications in scientific workflow systems. Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science, WORKS '09, in Conjunction with SC 2009.

[16] Wikipedia. PageRank. http://en.wikipedia.org/wiki/PageRank

[17] Yahoo, Oozie. http://yahoo.github.com/oozie/ .

[18] M. Zaharia, D. Borthankur, J. Sarma, K. Elmellegy, S. Shenker, and I. Stoica, Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in cluster Scheduling. In EuroSys 2010, pp. 265-278. ACM, New York (2010)

[19] M. Zaharia, A. Kowinski, A. Joseph, R. Katz and I. Stoica, Improving mapreduce performance in heterogeneous environments. USENIX OSDI, 2008