International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 6 Issue 06, June - 2017

Survey on Join Algorithms and Task Scheduling Algorithms for Hadoop MapReduce Platform

Amol R. Rayamane,
M Tech (CSE), KLSGIT,
Belagavi, India.

Prabhuling S. Biradar
M Tech (CSE), KLSGIT,
Belagavi, India.

Bhavana A. Kurade,
M Tech (CSE), KLSGIT,
Belagavi, India.

***Abstract*— Today's Digital era causes a rapid increase of datasets. These datasets are termed as "Big Data" due to its massive amount of volume, variety and velocity and is stored in distributed file system architecture. There are the following techniques that are used to analyze massive amounts of data: MapReduce paradigm, parallel DBMSs, column-wise store, and various combinations. We focus on a MapReduce environment. Hadoop is framework that supports Hadoop Distributed File System (HDFS)for storing and MapReduce for processing of large data sets in a distributed computing environment. Task assignment is possible by schedulers. In this paper we compare join algorithms as well as task scheduling algorithms for Hadoop platform.**

I. INTRODUCTION

Data-intensive applications include large-scale data warehouse systems, cloud computing, data-intensive analysis. Applications for large-scale data analysis use such techniques as parallel DBMS, MapReduce (MR) paradigm, and columnar storage. Applications of this type process multiple data sets. This implies need to perform several join operation. It's known join operation is one of the most expensive operations in terms both I/O and CPU costs. Unfortunately, join algorithms is not directly supported in MapReduc[18]. There are some approaches to solve this problem by using a high-level language PigLatin, HiveQL for SQL queries or implementing algorithms from research papers. Millions of users are using applications based on Internet services work with sheer volume of data has lead to parallel computing on clusters. Processing and storing giant amount of data in parallel manner become a challenge to computing globe. Hadoop[16] is a open source Java based framework which can run applications in the cluster that consist of reasonably priced hardware, for processing and storing large amount of data in distributed computing environment. Hadoop uses HDFS (Hadoop Distributed File System) for storing data and to process these data it uses MapReduce Programming model introduced by Google. One of the fascinating matter is their task scheduling. There are three important scheduling issues in MapReduce such as locality, synchronization and fairness[17].

II. JOIN ALGORITHMS

In this section we consider various techniques of two-way joins in MapReduce framework. Join algorithms can be divided into two groups: Reduce-side join and Map-side join.

A. Reduce-Side Join

Reduce-side join is an algorithm which performs data pre-processing in Map phase, and direct join is done during the Reduce phase. Join of this type is the most general without any restriction on the data. Reduce-side join is the most time-consuming, because it contains an additional phase and transmits data over the network from one phase to another. In addition, the algorithm has to pass information about source of data through the network. The main objective of the improvement is to reduce the data transmission over the network from the Map task to the Reduce task by filtering the original data through semi-joins. Another disadvantage of this class of algorithms is the sensitivity to the data skew, which can be addressed by replacing the default hash partitioner with a range partitioner.

There are three algorithms in this group:

- General Reducer-Side Join,
- Optimized Reducer-Side Join,
- The Hybrid Hadoop join.

1) General Reducer-Side Join is the simplest one. The same algorithms are called Standard Repartition Join in [2]. This algorithm has both Map and Reduce phases. In the Map phase, data are read from two sources and tags are attached to the value to identify the source of a key/value pair. As the key is not effecting by this tagging, so we can use the standard hash partitioner. In Reduce phase, data with the same key and different tags are joined with nested-loop algorithm. The problems of this approach are that the reducer should have sufficient memory for all records with a same key; and the algorithm sensitivity to the data skew.

2) Optimized Reducer-Side Join enhances previous algorithm by overriding sorting and grouping by the key, as well as tagging data source. Also known as Improved Repartition Join in [2], Default join in [4]. In the algorithm all the values of the first tag are followed by the values of the second one. In contrast with the General reducer-side join, the tag is attached to both a key and a value. Due to the fact that the tag is attached to a key, the partitioner must be overridden in order to split the nodes by the key only. This case requires buffering for only one of input sets.

3) Hybrid Join combines the Map-side and Reduce-side joins[1]. In Map phase, we process only one set and the second set is partitioned in advance. The pre-partitioned set is pulled out of blocks from a distributed system in the Reduce phase, where it is joined with another data set that came from the Map phase. The similarity with the Map-side join is the restriction that one of the sets has to be split in advance with

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 6 Issue 06, June - 2017

the same partitioner, which will split the second set. Unlike Map-side join, it is necessary to split in advance only one set. The similarity with the Reduce-side join is that algorithm requires two phases, one of them for preprocessing of data and one for direct join. In contrast with the Reduce-side join we do not need additional information about the source of data, as they come to the Reducer at a time.

B. Map-Side join

Map-Side Join is an algorithm without Reduce phase. This kind of join can be divided into two groups. First of them is partition join, when data previously partitioned into the same number of parts with the same partitioner. The relevant parts will be joined during the Map phase. This map-side join is sensitive to the data skew. The second is in memory join, when the smaller dataset send whole to all mappers and bigger dataset is partitioned over the mappers. The problem with this type of join occurs when the smaller of the sets cannot fit in memory.

There are three methods to avoid this problem:

- JDBM-Based Map Join,
- Multi-Phase Map Join,
- Reversed Map Join.

Map-side partition join algorithm assumes that the two sets of data pre-partitioned into the same number of splits by the same partitioner. Also known as default maps join. At the Map phase one of the sets is read and loaded into the hash table, then two sets are joined by the hash table. This algorithm buffers all records with the same keys in memory, as is the case with skew data may fail due to lack of enough memory.

1) Map-Side Partition Merge Join (MSPMJ) is an improvement of the previous version of the join.If data sets in addition to their partition are sorted by the same ordering, we apply merge join. The advantage of this approach is that the reading of the second set is on-demand, but not completely, thus memory overflow can be avoided. As in the previous cases, for optimization can be used the semi-join filtering and range partitioner.

2) In-Memory Join (IMMJ) does not require to distribute original data in advance unlike the versions of map joins discussed above. The same algorithms are called Map-side replication join in [3], Broadcast Join in [2], Memory-backed joins [1], Fragment- Replicate join in [4]. The IMMJ algorithm has a strong restriction on the size of one of the sets: it must fit completely in memory. The advantage of this approach is its resistance to the data skew because it sequentially reads the same number of tuples at each node.

There are two options for transferring the smaller of the sets:

- Using a distributed cache,
- Reading from a distributed file system.

3) JDBM-Based Map Join is presented in [5]. In this case, JDBM library automatically swaps hash table from memory to disk.

4) Multi-Phase Map Join [5] is algorithm where the smaller of the sets is partitioned into parts that fit into memory, and for each part runs In-Memory join. The problem be put in the memory is increased twice, the execution time of this join is also doubled. It is important to note that the set, which will not be loaded into memory, will be read many times from the disk.

5) Idea of Reversed Map Join [5] approach is that the bigger of the sets, which is partitions during the Map phase, loading in the hash table. Also known as Broadcast Join in [2].The second dataset is read from a file line by line and joined using a hash table.

C. Semi-Join

Sometimes a large portion of the data set does not take part in the join. Deleting of tuples that will not be used in join significantly reduces the amount of data transferred over the network and the size of the dataset for the join. This preprocessing can be carried out using semi-joins by selection or by a bitwise filter.

There are three ways to implement the SEMI-JOIN operation:

- Semi-Join Using Bloom-Filter,
- Semi-Join Using Selection,
- An Adaptive Semi-Join.

1) Bloom-Filter is a bit array that defines a membership of element in the set. False positive answers are possible, but there are no false-negative responses in the solution of the containment problem. The accuracy of the containment problem solution depends on the size of the bitmap and on the number of elements in the set. These parameters are set by the user. It is known that for a bitmap of fixed size *m* and for the data set of n tuples, the optimal number of hash functions is k=0.6931*m/n. In the context of MapReduce, the semi-join is performed in two jobs. The first job consists of the Map phase, in which keys from one set are selected and added to the Bloom-filter. The Reduce phase combines several Bloom filters from first phase into one. The second job consists only of the Map phase, which filters the second data set with a Bloom-filter constructed in previous job. The accuracy of this approach can be improved by increasing the size of the bitmap. The advantage of this method is its the compactness. The performance of the semi-join using Bloom-filter highly depends on the balance between the Bloom-filter size, which increases the time needed for its reconstruction of the filter in the second job, and the number of false positive responses in the containment solution.

2) Semi-Join With Selection extracts unique keys and constructs a hash table. The second set is filtered by the hash table constructed in the previous step. In the context of Map Reduce, the semi-join is performed in two jobs. Unique keys are selected during the Map phase of the first job and then they are combined into one file during the Map phase. The second job consists of only the Map phase, which filters out the second set. The semi-join using selection has some limitations. Hash table in memory, based on records of unique keys, can be very large, and depends on the key size and the number of different keys.

3) Adaptive Semi Join is performed in one job, but filters the original data on the flight during the join. Similar to the

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 6 Issue 06, June - 2017

Reduce-side join at the Map phase the keys from two data sets are read and values are set equal to tags which identify the source of the keys. At the Reduce phase keys with different tags are selected. The disadvantage of this approach is that additional information about the source of data is transmitted over the network.

D. Comparative Analysis of Join Algorithms

The features of join algorithms are presented in the Table 1. The approaches with pre-processing is good when data is prepared in advance for example come from another MapReduce job. Algorithms with one phase and without tagging is more preferable due to the fact that no additional transferring data through the network are needed. Approaches that sensitive to the data skew may be improved by optimizations with range partitioner. In case of data low selectivity semi-join algorithms can improve performance and reduce the possibility of memory overflow.

III. TASK SCHEDULING ALGORITHMS

In this section, we will discuss several task scheduling algorithms such as FIFO, Fairshare, Capacity, Delay and IWRR .

A. FIFO Scheduling Algorithm

This is a default scheduler used by Hadoop which operates using a queue. In this approach job is first partitioned into individual tasks, and afterward loaded into the queue and assigned to free slots on Task Tracker (slave). Each job would exploit the complete cluster, as a result jobs be obliged to wait for their turn. The major disadvantage of this algorithm is that once the previous job with the scheduler, the major drawback is that only after finishing the previous job, subsequently jobs in the job queue will be assigned. The scheduler implementation is straightforward and proficient[8].

B. Fair Share Scheduling Algorithm

The Fair Scheduler was developed at Facebook to manage access to their Hadoop cluster [9]. The Fair Scheduler gives equivalent share of cluster capacity to each user. Users may assign jobs to pool, with each pool allocated a definite minimum number of Map and Reduce slots [10] [15].If there are free slots in pools then they may be allocated to other pools, while excess capacity within a pool is shared among jobs. The Fair Scheduler is a preemptive that is to say if a pool has not received its fair share for a certain period of time, then the scheduler will destroy tasks in pools running over capacity in order to give the slots to the pool running underneath capacity. Seeing that jobs have their tasks allocated to Task Tracker for computation, the scheduler track the shortfall between the amount of time actually used and the ideal fair allocation for that job. When slots become free, the next task from the job with the highest time shortfall is make sure that jobs receive approximately equivalent amounts of resources. Shorter jobs are allocated sufficient resources to finish quickly. Simultaneously, longer jobs are assured to not be starved of resources.

TABLE 1. THE FEATURES OF JOIN ALGORITHM

Algorithm	Pre-Processing	Number Of Phases	Tags	Sensitive To Data Skew	Need Distributed Cache	Memory Overflow
GRSJ	-	2	To Value	Yes	-	Number of tuples for the same key is large
ORSJ	-	2	To Key and Value	Yes	-	Number tuples for the same key is big
HYB	1 Data	2	-	Yes	-	Part size is large
MSPJ	2 Data	1	-	Yes	-	Part size is large
MSPMJ	2 Data + Sort	1	-	Yes	-	-
IMMJ	-	1*Part	-	-	Yes	Size of smaller dataset is large
MUL	1 Data		-	-	Yes	-
JDBM	-	1	-	-	Yes	-
REV	-	1	-	-	Yes	Part size is big band number of tuples for the same key is big

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 6 Issue 06, June - 2017

TABLE 2. COMPARATIVE ANALYSIS OF EXISTING SCHEDULING ALGORITHM

Parameter	FIFO Scheduling	FairShare Scheduling	Capacity Scheduling	Delay Scheduling	IWRR Scheduling
Mode	Non Preemptive	Preemptive	Non preemptive When job fails	Preemptive	Preemptive
Implementation	Simple	Less complex	Complex	Simple	Simple
Resource Utilization	Low	High	High	High	High
Response Time	Low for short jobs	High	High	High	High
Performance	High for small clusters	High for both large and small clusters	High for large clusters	High for small clusters	High for both large and small clusters
Execution	Serial	Parallel	Parallel	Parallel	Parallel
Load Balancing	No	Yes	Yes	Yes	Yes

C. Capacity Scheduling Algorithm

Capacity Scheduling algorithm was developed to manage fair distribution of resources among huge mass of users. The Capacity Scheduler allocates jobs based on the submitting user to queues with configurable numbers of Map and Reduce slots [11][14][15]. Queues which have jobs are given their configured capacity where as free capacity in a queue is shared between other queues. Scheduling is driven on a customized priority queue basis with specific user restrictions within a queue, with priorities adjusted based on the time a job was submitted, and the priority setting allocated to that user and category of job[14]. When a Task Tracker slot becomes free, the queue with the lowest load is selected, from which the oldest lasting job is nominated. A task is next scheduled from that job.

D. Delay Scheduling

Authors of [12] and [13] have discussed delay scheduling algorithm. Fair scheduling algorithm was developed to allocate fair share of capacity to all the users. The scheduler launches a task from a job on a node. If task is run on the node that contain the data, it is most efficient, but when it is not possible, running on a node on the same rack is faster than running off-rack. Delay scheduling is used to improve data locality by asking jobs to wait for its turn for scheduling on a node with local data. When a node requests a task and if the head-of-line job cannot launch a local task then it is skipped and looked at next jobs. But if a job has been skipped for long enough then non-local tasks are allowed to launch to prevent starvation. In this algorithm although the first slot given to a job is not likely to have data for it, but tasks come to an end very quickly that some other slot contain data for it will free within a small amount of time.

E. Improved Weighted Round Robin Scheduling Algorithm

In [6] authors have proposed IWRR scheduling , based on the analysis of WRR algorithm. Under the unweighted conditions, tasks of each job are submitted to Task Tracker in turn. Under the weighted conditions, multiple tasks of the larger weight job will run in a round, and the job's weight will be changed along with the increase or decrease of jobs number. At times, if the number of tasks of the smaller weight job becomes more, while the number of the larger Weight job is less, then the weight of the smaller weight job will be increased correspondingly, so the number of tasks which are assigned to Task Tracker will be relatively increased, and the weight of the larger weight job will be appropriately decreased, the number of tasks which are assigned to Task Tracker will be relatively decreased. However, the relationship between them remains the same in order to achieve load balance. This algorithm used weight update rules to reduce workload and to balance tasks' allocation. The algorithm is easy to be implemented with low cost and suitable for the Hadoop platform that uses the only Job Tracker to schedule.

E. Comparative Analysis of Existing Scheduling Algorithm

Mainly three scheduling issues be taken into consideration: fairness, locality and synchronization. Fairness finiteness has trade-offs between the locality and dependency between the map and reduce phases. Locality is defined as the distance between the input data node and task-assigned node. Synchronization be the process of transmitting the intermediate output of the map processes to the reduce processes as input is also considered as a factor which affects the performance [7] . Task Scheduling is an aspect that directly affecting the overall performance of Hadoop platform and utilization of system resources. There are various algorithms to resolve this issue with different techniques and approaches as we discussed previous. The comparative analysis of Scheduling algorithms is given in Table 2.

IV. CONCLUSIONS

The paper gives an overall idea about different massive parallel processing join algorithms and task scheduling algorithms for Hadoop Mapreduce. We analyzed features of Join algorithms such as preprocessing, number of phases, tags, sensitive to data skew , need distributed cache and memory overflow. We analyzed properties of various task schedulers based on working mode, response time, performance, data locality and fairness provision, execution style, resource utilization, load balancing.

REFERENCES

[1] Fariha Atta. Implementation and analysis of join algorithms to handle skew for the Hadoop MapReduce framework. Master's thesis, MSc Informatics, School of Informatics, University of Edinburgh, 2010.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 6 Issue 06, June - 2017

[2] Spyros Blanas, Jignesh M. Patel, Vuk Ercegovac, Jun Rao, Eugene J. Shekita, and Yuanyuan Tian. A comparison of Join algorithms for log processing in MapReduce. In Proceedings of the 2010 international conference on Management of data, SIGMOD '10, pages 975–986, New York, NY, USA, 2010. ACM.
[3] A Chatzistergiou. Designing a parallel query engine over map/reduce. Master's thesis, MSc Informatics, School of Informatics, University of Edinburgh, 2010.
[4] Alan F Gates. Programming Pig. O'Reilly Media, 2011.
[5] Gang Luo and Liang Dong. Adaptive join plan generation in hadoop. Technical report, Duke University, 2010.
[6] Jilan Chen,Dan Wang and Wenbing Zhao, "A Task Scheduling Algorithm for Hadoop Platform", Journal of computers, Vol. 8,Issue 4,pp 929-936,April 2010.
[7] Seyed Reza Pakize," A Comprehensive View Of Hadoop Map Reduce Scheduling Algorithms", International Journal of computer networks and communications security, Vol. 2, Issue 9, pp.308-317,September 2014.
[8] Jisha S Manjaly, Chinnu Edwin A," A Relative Study on Task Schedulers in Hadoop MapReduce", International Journal of Advanced Research in Computer Science and Software Engineering", Vol. 3, Issue 5, pp 744-747, May 2013.
[9] B. Thirmala Rao, N. V. Sridevei, V. Krishna Reddy, LSS.Reddy, "Performance Issues of Heterogeneous Hadoop Clusters in Cloud Computing", Global Journal Computer Science & Technology Vol. 11, Issue 8, pp.81-87,May 2011.
[10] DeWitt & Stonebraker, "MapReduce: A major step backwards", 2008 .
[11] Dean, J. and Ghemawat, S., "MapReduce: a flexible data processing tool", communication of ACM,Vol. 53,Issue 1,pp72-77, January 2010.
[12] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling",5th European conference on Computer systems, ACM, pages 265–278, 2010.
[13] Matei Zaharia, Hruba Bortha kur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, Ion Stoica, "Job Scheduling for Multi-User MapReduce Clusters", Electrical Engineering and Computer Sciences, University of California at Berkeley ,Technical Report No. UCB/EECS-2009-55,2009.
[14] B.Thirumala Rao, Dr. L.S.S.Reddy, "Survey on Improved Scheduling in Hadoop MapReduce in Cloud Environments", International Journal of Computer Applications (0975-8887) ,Vol. 34,Issue 9,pp 29-33, November 2011.
[15] Joel Wolf, Andrey Balmin, Deepak Rajan, Kirsten Hildrum,Rohit Khandekar, Sujay Parekh,Kun-Lung Wu,Rares Vernica "CIRCUMFLEX: A Scheduling Optimizer for MapReduce Workloads With Shared Scans", ACM SIGOPS Operating Systems Review ,Vol. 46 Issue 1,pages 26-32,January 2012.
[16] Apache Hadoop, "Hadoop home page", http://hadoop.apache.org/
[17] Hiral M. Patel, "A Comparative Analysis of MapReduce Scheduling Algorithms forHadoop", International Journal of Innovative and Emerging Research in Engineering Volume 2, Issue 2, 2015.
[18] A. Pigul,"Comparative Study Parallel Join Algorithms for MapReduce environment", Saint Petersburg State University.