International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 4 Issue 02, February-2015

Shuffle Performance in Apache Spark

Nirali Rana
Department of Computer Engineering
GTU PG School, Gujarat Technological University
Ahmedabad , India.

Shyam Deshmukh
Assistance Professor: dept. of IT
Pune Institute of Computer Technology,
Pune, India.

***Abstract*— Apache Spark is a cluster computing framework that performs in memory computing and responsible for Scheduling, Distributing and Monitoring Applications. Two types of factors to improving Spark performance: Optimization and Latency Hiding. In distributed data processing platforms is common to collect data in many fashion, a stage traditionally known as shuffle Phase. Using this two factors compare the performance between Hadoop and Spark. In this paper, Apache Spark Shuffle is faster than Hadoop Shuffle.**

Keywords—Apache Sparkt; Shuffle;Hadoop

I. INTRODUCTION

Apache Spark is an open-source analytics cluster computing framework developed in AMP Lab at UC Berkeley [11]. Apache Spark is a general purpose cluster computing system with the goal of outperforming disk-based engine like Hadoop. Spark is an implementation of Resilient Distributed Datasets[RDD]. It provides high level APIs in Java, Scala and Python. Mostly Scala is use in Spark programming. Spark enables applications in Hadoop clusters to run up to 100x faster in memory and 10x faster running on disk. It comes with a built-in set of over 80 high-level operators. Apache Spark has been used by many companies including Amazon, Facebook, Yahoo and GroupOn.

Spark Stack offers an integrated framework for advanced analytics including machine learning library (MLLib), a graph engine (GraphX), a streaming analytics engine (Spark Streaming) and a fast interactive query tool (Shark). Apche Spark is based on two key concepts: Resilient distributed Datasets (RDD) and Directed Acyclic Graph (DAG). An RDD is a read-only collection of objects partitioned across a set of machines that can be built if a partition is lost. Spark performance feature is low Latency computations by caching the working dataset in memory and then performing computations at memory speeds. Spark is capable of reading from HBase, Hive, Cassandra and HDFS data source.

Spark Driver controls the workflow and Spark workers launches executors responsible for executing part of the job submitted to spark driver through cluster node. Spark driver has few components: 1) RDD 2) Scheduler 3) Serializer 4) Shuffle. Spark Worker has two components: 1) Task and 2) Block Manager.

1) RDD: Resilient Distributed Datasets (RDD) is a basic abstraction in Spark. RDD represents a partitioned collection of elements that can be operated on in parallel [8].

2) Scheduler: Spark's scheduler uses representation of RDDs. Scheduler assigns task to machines based on data locality using delay scheduling.

3) Serializer: Spark sterilizer that uses Java's built-in serializer. It is used for stream of reading serializer object.

4) Shuffle: In Spark, Shuffle creates a large number of shuffles (M*R). Shuffle refers to maintaining a shuffle file for each partition which is the same as the number of reduce task R per core C rather than per Map task M. Every machine needs to handle only C*R number of shuffle rather than M*R.

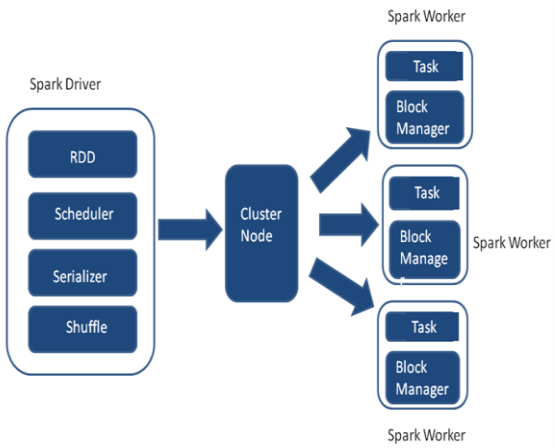

Fig. 1. Spark Driver Execution flow

II. OPTIMIZATION AND LATENCY HIDING

A. *Optimization in Spark*

In Apache Spark, Optimization implements using Shuffling techniques. In this paper we use shuffling technique for optimization. The optimize shuffle performance two possible approaches are 1) To emulate Spark behavior by merging intermediate 2) To create large shuffle files 3) Use columnar compression to shift bottleneck to CPU. For shuffling shuffle data is required. Spark compressibility was low and involves significant computation and metadata maintenance for splitting data into columns on map side and reconstructing them into rows on reduce side.

One of the key optimization factors was the Shuffle. With other two factors was sorting algorithm and the external sorting service. In this paper choose optimize shuffle file performance in the Spark distributed computing platform.

Optimization Limitation:

- Lack of iteration support
- High latency due to persisting intermediate data onto diak

B. *Latency Hididng in Spark*

Apache Spark which has taken the strength of Hadoop and made improvements in a Hadoop's weakness and provides more efficient batch processing framework with a much lower latency. Prefetching is a latency hiding technique. It has to increase parallelism of tasks so as to keep the CPU busy by fetching the data before it is required.

Prefetching has several factors. Emerging trends such as putting data and services into the cloud and services will suffer from latency. A real-time data prefetching algorithm based on when pushing shuffle files prior to the completion of the map phase.

III. SHUFFLE OVERVIEW

Shuffle Phase is a component of Spark Driver. A shuffle is a communication between one input RDD and an Output RDD. Each shuffle has a fixed number of mappers and a fixed number of reduce partitions. Shuffle writer and Shuffle reader handle the I/O for a particular task, operating on iteration of RDD elements. When a shuffle has an Aggtrgator, the Shuffle Manager and its readers and writers are responsible for external spilling.

Shuffle works in two stages: 1) Shuffle writes intermediate files to disk and 2) fetch by the next stage of tasks.

Shuffle operation is implemented differently in Spark compared to Hadoop. The values of M and R in Hadoop are much lesser. The number of shuffle files in Spark scales with M*R , a smaller number of map task and reduce task may provide more justification for the way Spark handles Shuffle files on the map side [11].

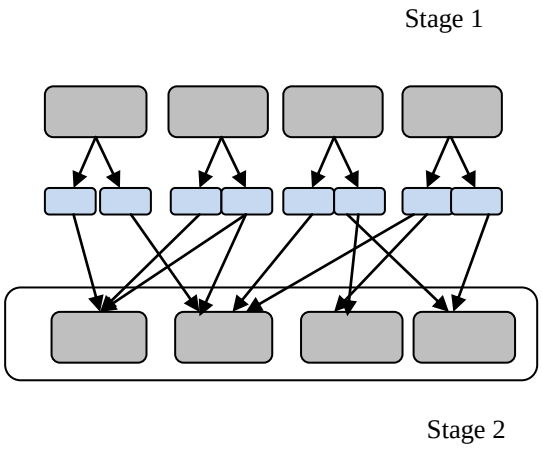

Fig.2. Shuffle Work

A. *Map Side Shuffle*

Each map task in Spark writes outs a shuffle file for every reducer. These files are not intermediately in the sense that Spark does not merge them into large partition. Since scheduling overhead in Spark is much lesser the no. of mappers (M) and reducers ® is higher than in Hadoop. Thus, shipping M*R files to the respective reduces could result in significant overheads.

Spark also provides a parameter to specify compression libraries to compress map outputs. It could be default. Default uses 33KB of buffer for each open file and significantly reduces risk of encountering out-of-memory error.

B. *Reduce side Shuffle*

A major difference between Hadoop and Spark is on the reduce side. Spark requires to all shuffle data to fit in memory of the corresponding reduce task. Where the reducer task demands all shuffle data for a GroupByKey or a ReduceByKey operation. Spark throws an out-of-memory exception. The Shuffle is a pull operation in spark compared to push operation in Hadoop.

Each Reducer should also maintain a network buffer to map outputs. Size of the buffer is Specified through the parameter.

IV. SHUFFLE TECHNIQUES

Spark Shuffling uses two techniques: 1) Sort-based Shuffle 2) Hash-based Shuffle.

A. *Sort-based Shuffle*

A sort-based Shuffle can be more scalable than Spark's current hash-based one because it doesn't require writing a separate file for each reduce task from each mapper.

Each map task will produce one or more output files sorted by a key's partition ID, then merge-sort them to a single output file. Once the map tasks produce files, reduces will be able to request ranges of files to get their particular data. An index file for each output file saying where each partition is located and update the Block Manager to support using this index.

Map tasks will write data through a SortedFileWriter that creates one or more sorted files merge them and then creates an index file for the merged file. The SortedFileWriter must reset compression and serialization streams when writing each range.

SortedFileWriter will works as follows:

1) Given a stream of incoming key-value pairs, first write them into buckets in memory based on their partition ID. This bucket can be ArrayBuffers for each partition ID.

2) When the total size of the buckets gets too large, write the current in-memory output to a new file. This intermediate file will contain a header saying at which position each partition ID.

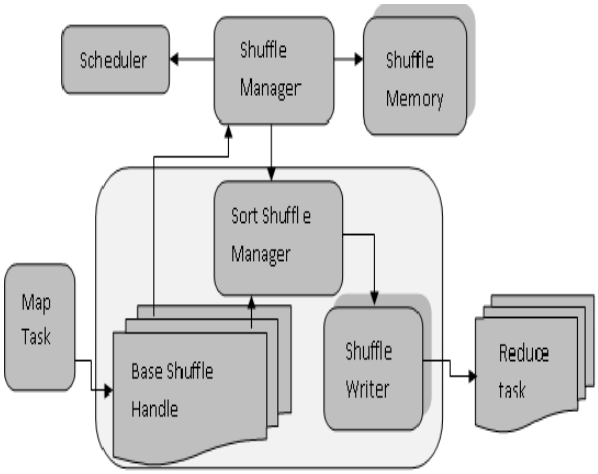

Fig. 2.Sort-Based Shuffle

3) After all intermediate files are written, merge-sort them into a final file.

4) When writing the final file reset the serialization and compression streams after writing each partition and track the byte position of each partition to create an index file.

Reduce Tasks will fetch and hash together data the same way use ExternalAppendOnlyMap. They merge all spilled files at once if there are a huge amount of files.

B. Hash-based Shuffle

A hash-based shuffle is default in shuffling data but starting in spark 1.1. There is an experimental sort-based shuffle that is more memory-efficient in environments with small executors. The mapper reduce the amount of the increase based on the performance of hash-based realization of shuffle sort of performance.

Hash shuffle into a set of 64 subdirectories created on each disk. Hash shuffle large number of random writes when each shuffle is relatively small. A hash-based shuffle required a hash shuffle reader to read the intermediate file from mapper side. Hash-based shuffle are use to BlockStoreShuffle to store the shuffle file and resize into the shuffle.

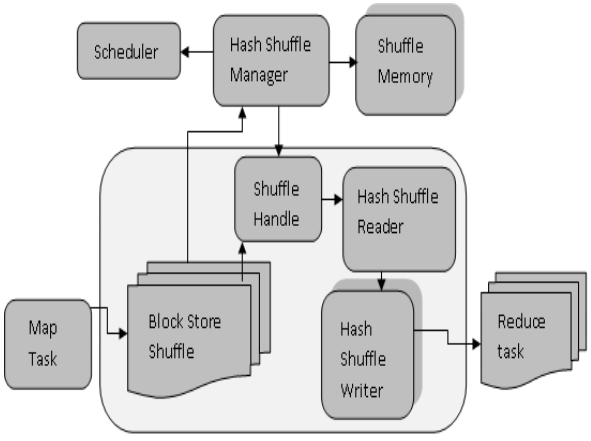

Fig. 3.Hash-Based Shuffle

V. RELATED WORK

Spark Shuffle actually outperformed Hadoop. Hadoop's performance is more expensive shuffle operation compared to Spark. Hadoop's Map phase being significantly slower than Spark's with Shuffle. Shuffle data is prefeched by reduces while the Map phase is running. Each map task in Spark writes outs a shuffle file for every reducer. In Hadoop Pull-based, Push-based and Hybrid shuffle technologies used for Shuffle Performance [7]. Spark uses these two techniques to improve Shuffle performance. That why Spark is increase performance rather than Hadoop shuffle.

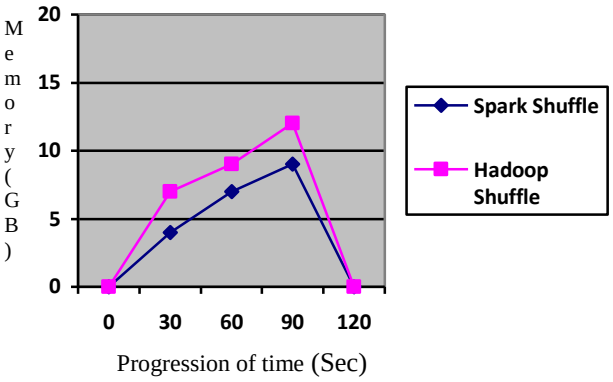

Fig. 4.Compare Shuffle Perfromance between Hadoop and Spark

VI. CONCLUSION

Spark Shuffle performance is increase using large number of shuffle file. Also learning shuffle works in Spark and Techniques. In this paper also understand Spark execution model. Spark Driver is the most important concept in Spark and also learning Spark driver works. Mostly in shuffling in Spark Sort-based is used. Using Shuffle Techniques compare the Spark Shuffle performance with Hadoop.

In future, Distributed Hash Shuffle Algorithm and Shuffle is used to improve better performance in Apache Spark.

REFERENCES

[1] Shantenu Jha, Judy Qiu, Andre Luckow, Pradeep Mantha, Geoffrey C.Fox. A Tale of Two Data-Intensive Paradigms: Applications, Abstractions, and Architectures.Indiana University, USA, International Congress on Big Data 2014 IEEE.

[2] Xiaoyi Lu, Md. Wasi-ur-Rahman, Nusrat Islam, Dipti Shankar and Dhabaleswar K. (DK) Panda, "Accelerating Spark with RDMA for Big Data Processing: Early Experiences" The Ohio State University, 2014 IEEE.

[3] Aaron Davidson, Andrew Or "Optimizing Shuffle Performance in Spark" UC Berkeley 2013.

[4] Jingui Li, Xuelian Lin , Xiaolong Cui , Yue Ye. Improving the Shuffle of Hadoop Map Reduce. IEEE 5th International Conference on Cloud Computing Technology and Science.

[5] Matei Zaharia, Tathagata Das, Haoyuan Li Discretized Streams: Fault-Tolerant Streaming Computation at Scale University of California, Berkeley 2013.

[6] Lei Gu, Huan Li , "Memory or Time: Performance Evaluation for Iterative Operation on Hadoop and Spark", Beihang University, Beijing, China 2013 IEEE.

[7] Yanfei Guo, Jia Rao, and Xiaobo Zhou. iShuffle: Improving Hadoop Performance with Shuffle-on-Write.Department of Computer ScienceUniversity of Colorado, USA 2012.

[8] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das,Ankur Dave, Justin Ma, Murphy McCauley, Michael Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing.In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, pages 2{2}. USENIXAssociation, 2012.

[9] Kichul Kim. Shuffle Memory System. School of Electrical Engineering,University of Seoul, Korea1992.

[10] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica "Spark: Cluster Computing with Working Sets", University of California, Berkeley 2010.

[11] Apache Spark. http://spark.apache.org/

[12] Apache hadoop. http://apache.hadoop.org/

[13] A Deeper Understanding of Spark Internals. http://spark-summit.org/2014/talk/a-deeper-understanding-of-spark-internals/