# Comparing Apache Spark and Map Reduce with Performance Analysis using K-Means

Satish Gopalani    Rohan Arora

## ABSTRACT

Big Data has long been the topic of fascination for Computer Science enthusiasts around the world, and has gained even more prominence in the recent times with the continuous explosion of data resulting from the likes of social media and the quest for tech giants to gain access to deeper analysis of their data. This paper discusses two of the comparison of - Hadoop Map Reduce and the recently introduced Apache Spark – both of which provide a processing model for analyzing big data. Although both of these options are based on the concept of Big Data, their performance varies significantly based on the use case under implementation. This is what makes these two options worthy of analysis with respect to their variability and variety in the dynamic field of Big Data. In this paper we compare these two frameworks along with providing the performance analysis using a standard machine learning algorithm for clustering (K-Means).

## General Terms

Big Data, Machine Learning, K Means.

## Keywords

Big data, Hadoop, HDFS, Map Reduce, Spark, Mahout, MLib, Machine learning, K-Means.

## 1. INTRODUCTION

Apache Hadoop [1] is an open source framework that provides solutions for handling big data along with extensive processing and analysis. It was created by Doug Cutting in 2005 when he was working for Yahoo at the time for the Nutch search engine project. Hadoop has two major components named HDFS (Hadoop Distributed File System) [2] and the Map Reduce [3] framework. Hadoop Distributed File System is said to be inspired by Google’s The Google File System (GFS) [4] and provides a scalable, efficient, and replica based storage of data at various nodes that form a part of a cluster.

HDFS is based on a master slave architecture where ‘namenode’ is the master and ‘datanodes’ are the slave nodes where the actual data resides (quite possibly replicated data). The replication factor by default is of three, but can be configured as per the need of the user and the usage type. The second vital component, which is Map Reduce is the processing model for Apache Hadoop which allows successful processing of the replicated data in parallel based on the former programming language techniques of map and reduce. Map is the phase which is implemented to distributed portions of a dataset to various ‘mappers’ that work in parallel to provide the achievability for the essence of big data computation. The outputs from these mappers are exposed to sorting and shuffling which takes the flow to the next phase, called the ‘Reduce’ phase where data is aggregated to find out the result to our initial problem statement [5].

Although recently, the world of Big Data has seen a dynamic shift from this computing model with the introduction and stable release of Apache Spark [6], which provides a user friendly programming interface to decrease coding efforts and provide better performance in a majority of the cases with problems related to big data. Spark not just provides an alternative to Map Reduce, but also has options for SQL like querying with Shark and a machine learning library called MLib. The performance and working of spark is considerably different from that of map reduce, but is also dependent on the constraints of parallelism, the types of problems in context, and the resources available.

Apache Spark [7] started as a research project at UC Berkeley in the AMPLab, was started with a goal to design a programming model that supports a much wider class of applications than MapReduce, while maintaining its automatic fault tolerance.

Spark offers an abstraction called Resilient distributed Datasets (RDDs) [8] to support these applications efficiently. RDDs can be stored in memory between queries without requiring replication. Instead, they rebuild lost data on failure using lineage: each RDD remembers how it was built from other datasets (by transformations like map, join or groupBy) to rebuild itself. RDDs allow Spark to outperform existing models by up to 100x in multi-pass analytics. RDDs can support a wide variety of iterative algorithms, as well as interactive data mining and a highly efficient SQL engine Shark [9].

## 2. DIFFERENCE BETWEEN MAPREDUCE AND SPARK

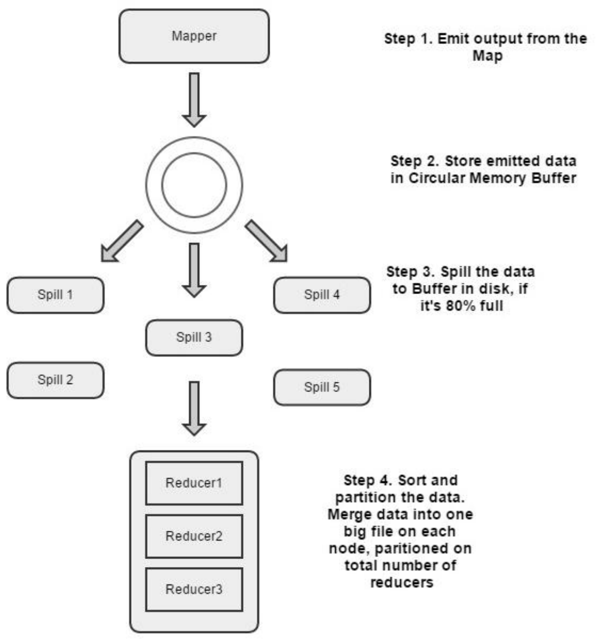
Mapper
Step 1. Emit output from the Map
Step 2. Store emitted data in Circular Memory Buffer
Spill 1
Spill 2
Spill 3
Spill 4
Spill 5
Step 3. Spill the data to Buffer in disk, if it's 80% full
Reducer1
Reducer2
Reducer3
Step 4. Sort and partition the data. Merge data into one big file on each node, paritioned on total number of reducers

**Fig 3: Map phase in Map Reduce**

The above diagram shows the Map phase of Hadoop Map Reduce [10]. The steps for the same are explained below:

- The Map phase outputs the result in the form of (Key, Value) pairs.
- The output would be stored in a circular buffer in memory.
- When the circular buffer fills 80% (configurable), then it data is spilled onto disk.
- All the spill files are combined into a single big file which is partitioned and sorted depending upon the reducers.

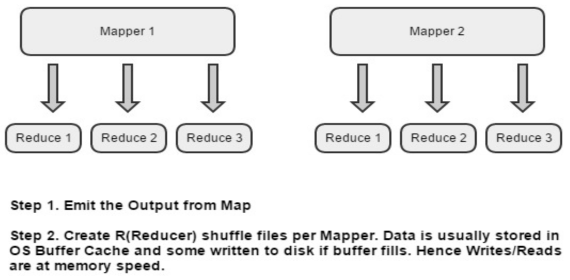
Mapper 1
Mapper 2
Reduce 1
Reduce 2
Reduce 3
Reduce 1
Reduce 2
Reduce 3
Step 1. Emit the Output from Map
Step 2. Create R(Reducer) shuffle files per Mapper. Data is usually stored in OS Buffer Cache and some written to disk if buffer fills. Hence Writes/Reads are at memory speed.

**Fig 2: Map phase in Spark**

The above diagram shows the Map phase of Hadoop Map Reduce [10]. The steps for the same are explained below:

- In contrast to Map Reduce, the output of map phase is written to OS Buffer Cache.
- Operating System decides whether the data will stay in buffer or will be spilled onto disk.
- Unlike Map Reduce, Spark does not merge or partition spill files, with the only difference being that the map output from the same cores are merged into a single file.
- Each Map task/core outputs as many spill files as number of reducers [11][12][13].

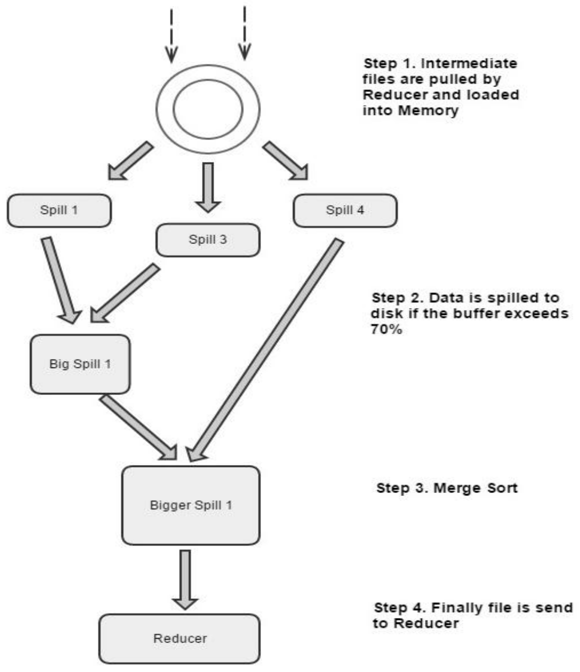
Step 1. Intermediate files are pulled by Reducer and loaded into Memory
Spill 1
Spill 3
Spill 4
Step 2. Data is spilled to disk if the buffer exceeds 70%
Big Spill 1
Bigger Spill 1
Step 3. Merge Sort
Reducer
Step 4. Finally file is send to Reducer

**Fig 3: Reduce phase in Map Reduce**

Reduce side of Hadoop MR:

- The data (intermediate files) created by the map phase is pulled by the reducers and loaded into the memory.
- If buffer reaches 70% (configurable), it is spilled onto disk.
- The data spilled to the disk is then merged into larger files, and the reduce function is initiated.

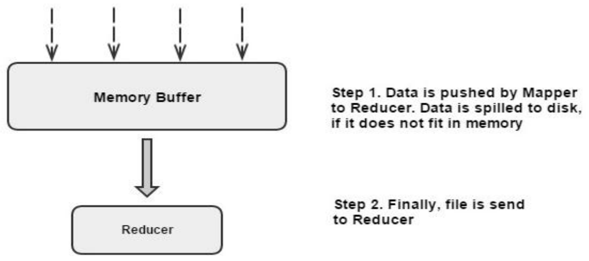
Memory Buffer
Step 1. Data is pushed by Mapper to Reducer. Data is spilled to disk, if it does not fit in memory
Reducer
Step 2. Finally, file is send to Reducer

**Fig 4: Reduce phase in Spark**

- The map phase pushes the data in the form of intermediate (shuffle) files to the reducers.
- These files are written to reducer's memory and reduce functionality is invoked.

## 2.1 Reasons to choose Spark

- Spark uses the concept of RDD which allows us to store data on memory and persist it as per the requirements. This allows a massive increase in batch processing job performance (up to ten to hundred times as much as that of conventional Map Reduce).
- Spark also allows us to cache the data in memory, which is beneficial in case of iterative algorithms such as those used in machine learning.
- Traditional MapReduce and DAG engines are suboptimal for these applications because they are based on acyclic data flow: an application has to run as a series of distinct jobs, each of which reads data from stable storage (e.g. a distributed file system) and writes it back to stable storage. They incur significant cost loading the data on each step and writing it back to replicated storage.
- Spark allows us to perform stream processing with large input data and deal with only a chunk of data on the fly. This can also be used for online machine learning, and is highly appropriate for use cases with a requirement for real time analysis which happens to be an almost ubiquitous requirement in the industry.
- In particular, MapReduce is inefficient for multi-pass applications that require low-latency data sharing across multiple parallel operations. These applications are quite common in analytics, and include:
    - Iterative algorithms, including many machine learning algorithms and graph algorithms like PageRank.
    - Interactive data mining, where a user would like to load data into RAM across a cluster and query it repeatedly.
    - Streaming applications that maintain aggregate state over time.

### 2.2 Why should one stick to MapReduce

The prominent benefits of MapReduce over Spark are highlighted as below:

- The main component of Spark happens to be Scala, along with ported Java API's. Map Reduce might be friendlier and more native for Java based developers.
- If the functionality is implemented only in Mapper, with no reducers, then it would hardly give any benefit to move to Spark, since Spark has benefits due to its in-memory handling of data, and processing in Mapper, data is held in memory even in Map Reduce.
- People experience with Hadoop are familiar with MapReduce yet Spark is a totally new paradigm.
- When it comes to data-parallel, ETL tasks, Map Reduce emerges as the winner when compared to Spark.
- Spark on YARN is considerably new and may not be the best option for many people familiar with YARN already.

## 3. MACHINE LEARNING AND K-MEANS

### 3.1 Machine Learning Introduction

Machine learning is an active branch of artificial intelligence that allow computers to learn new patterns and instructions from data rather than being explicitly coded by a developer. Machine learning allows systems to enhance themselves based on new data that is added and to generate more efficient new patterns or instructions for new data [14].

### 3.2 K-Means Algorithm

K Means clustering is a non-hierarchical approach of grouping items into different number of clusters/groups. The number of clusters/groups is defined by the user which he chooses based on his/her use-case and data in question. K-Means works by forming cluster of data points by minimizing the sum of squared distances between the data points and their centroids. A centroid is a central point to a group of data points in the dataset. There are various ways of choosing initial centroid, but in many cases it is done using random allocation. The algorithm [14] is as follows:

1. Firstly, select randomly chosen 'k' cluster centroids.
2. Cluster Assignment: In this step, assign each of the data points in the dataset to one of the centroids, selecting centroid which is closest to the data-point.
3. Centroid Movement: For each centroid, compute the average of all the data-points that are allocated to each centroid. This computed average is the new value of the particular centroid.
4. Calculate the sum of square of distance that each centroid has moved from its previous value, repeat steps 2 and 3 until this value is not less than or equal to threshold value (usually 0.01) or the number of iterations reaches maximum iterations specified, either of which is satisfied.

## 4. COMPARISON

In order to come to a conclusion about the practical comparison of Apache Spark and Map Reduce, we performed a comparative analysis using these frameworks on a dataset that allows us to perform clustering using the K-Means algorithm.

### 4.1 Dataset Description

The Data Set includes sensor data of size 1240 MB collected over the years, and includes latitude and longitude values of the respective records. A sample of the data records is shown as below: The data record contains:

1. Date
2. Device Name
3. Device ID
4. Status
5. Latitude
6. Longitude

Sample Record:

2014-03-15:13:10:20|Titanic 2500|15e758be-8624-46aa-80a3-b6e08e979600|77|70|40|22|13|0|enabled|connected|enabled|38.9253917959|-122.78959506

### 4.2 Performance Analysis and Description

Post working on the K-Means algorithm on the described data set, we achieved the following results for comparison (shown in the tables on the right).

To gain a varied analysis, we considered 64MB, 1240 MB with a single node and 1240MB with two nodes and monitored the performance in terms of the time taken for clustering as per our requirements using K-Means algorithm. The machines used had a configuration as follows:

• 4GB RAM
• Linux Ubuntu
• 500 GB Hard Drive

The results clearly showed that the performance of Spark turn out to be considerably higher in terms of time, where each of the dataset size results in a decrease in the processing time of up to three times as compared to that of Map Reduce.

Although there exists a minor fluctuation in this result, this is due to the random nature of the K-Means algorithm and does not affect the analysis to a large extent.

**Table 1. Results for K-Means using Spark (MLib)**

| Dataset Size | Nodes | Time (s) |
|---|---|---|
| 62MB | 1 | 18 |
| 1240MB | 1 | 149 |
| 1240MB | 2 | 85 |

**Table 2. Results for K-Means using Map Reduce (Mahout)**

| Dataset Size | Nodes | Time (s) |
|---|---|---|
| 62MB | 1 | 44 |
| 1240MB | 1 | 291 |
| 1240MB | 2 | 163 |

## 5. CONCLUSION

This paper provides an overview of both the frameworks and also compares these on various parameters followed by a performance analysis using K-Means algorithm. Our results for this analysis show that Spark is a very strong contender and would definitely bring about a change by using in-memory processing. Observing Spark's ability to perform batch processing, streaming, and machine learning on the same cluster and looking at the current rate of adoption of Spark throughout the industry, Spark will be the de facto framework for a large number of use cases involving Big Data processing.

## 6. FUTURE WORK

Although most of the algorithms on Mahout till now have been based on Map Reduce, Spark's consistent improvements and increasing user base has lead Mahout to adopt Spark for their base framework replacing Map Reduce for their future implementations. This is one of the many instances where Spark is proving out to gain predominance over Map Reduce.

## 7. REFERENCES

[1] Apache Hadoop Documentation 2014 http://hadoop.apache.org/.

[2] Shvachko K., Hairong Kuang, Radia S, Chansler, R The Hadoop Distributed File System Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium

[3] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In OSDI'04: Sixth Symposium on Operating System Design and Implementation, 2004.

[4] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. In 19th Symposium on Operating Systems Principles, pages 29–43, Lake George, New York, 2003.

[5] HortonWorks documentation 2014 http://docs.hortonworks.com/HDPDocuments/HDP1/HDP-1.2.4/bk_getting-started-guide/content/ch_hdp1_getting_started_chp2_1.html

[6] Apache Spark documentation 2014 https://spark.apache.org/documentation.html.

[7] Apache Spark Research 2014 https://spark.apache.org/research.html.

[8] Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. Technical Report UCB/EECS-2011-82, EECS Department, University of California, Berkeley, 2011

[9] Reynold Xin, Joshua Rosen, Matei Zaharia, Michael J. Franklin, Scott Shenker, Ion Stoica. Shark: SQL and Rich Analytics at Scale. SIGMOD 2013. June 2013.

[10] Tom White, Hadoop the definitive guide chapter 06

[11] Spark Internals - Spark Summit 2014 http://spark-summit.org/wp-content/uploads/2014/07/A-Deeper-Understanding-of-Spark-Internals-Aaron-Davidson.pdf

[12] Spark Job Flow – Databricks https://databricks-training.s3.amazonaws.com/slides/advanced-spark-training.pdf

[13] Aaron Davidson, Andrew Or. Optimizing Shuffle Performance in Spark. Technical Report http://www.cs.berkeley.edu/~kubitron/courses/cs262a-F13/projects/reports/project16_report.pdf

[14] Machine Learning, Wikipedia, 2014 http://en.wikipedia.org/wiki/Machine_learning

[15] Machine learning with Spark - Spark Summit 2013 https://spark-summit.org/2013/exercises/machine-learning-with-spark.html