Implementing K-Means for Achievement Study between Apache Spark and Map Reduce

Dr.E.Laxmi Lydia,
Associate Professor, Department of Computer Science and Engineering, Vignan's Institute Of Information Technology, Visakhapatnam, Andhra Pradesh, India.

Dr.A.Krishna Mohan,
Professor, Department of Computer Science and Engineering, JNTUK Andhra Pradesh, India.

Dr. M.Ben Swarup,
Professor, Department of Computer Science and Engineering, Vignan's Institute Of Information Technology, Visakhapatnam, Andhra Pradesh, India.

ABSTRACT
Huge Data has for quite some time been the subject of enthusiasm for Computer Science fans around the globe, and has increased much more conspicuousness in the later times with the constant blast of information coming about because of any semblance of online networking and the journey for tech monsters to get entrance to more profound investigation of their information. MapReduce and its variations have been very fruitful in actualizing vast scale information concentrated applications on ware groups. Then again, a large portion of these frameworks are manufactured around a non-cyclic information stream demonstrate that is not suitable for other famous applications. Unique MapReduce executes jobs in a straightforward yet unbending structure design. MapReduce changes step ("map"), a synchronization step ("shuffle"), and a stage to join results from every one of the nodes in a cluster ("reduce"). Accordingly to defeat the inflexible structure of guide and diminish we proposed the as of late presented Apache Spark – both of which give a handling model to breaking down enormous information. The primary contender for "successor to MapReduce" today is Apache Spark. Like MapReduce, it is an extensively helpful engine, be that as it may it is proposed to run various more workloads, and to do in that capacity much speedier than the more prepared framework. In this paper we contrast these two systems along and giving the execution examination utilizing a standard machine considering so as learning calculation for bunching (K-Means) and through considering some different parameters like scheduling delay, speed up, energy consumption than the existing systems.

Keywords:
Spark, MapReduce, Hadoop, Big Data

1. INTRODUCTION

Another model of cluster computing has turned out to be broadly well known, in which data-parallel computations are executed on clusters of questionable machines by systems that consequently give locality-aware scheduling, fault tolerance, and load balancing. MapReduce [11] spearheaded this model, while systems like Dryad [17] and Map-Reduce-Merge [24] summed up the sorts of data streams upheld.

These systems accomplish their scalability and fault tolerance by giving a programming model where the client makes non-cyclic data stream graphs to go input data through an arrangement of operators. This permits the hidden framework to oversee scheduling and to respond to faults without client mediation.

While this data flow programming model is useful for a large class of applications, there are applications that can't be communicated proficiently as non-cyclic data flows. In this paper, we concentrate on one such class of applications: those that reuse a working arrangement of data over numerous parallel operations. This incorporates two use cases where we have seen Hadoop users report that MapReduce is lacking:

Iterative employments: Many normal machine learning algorithms apply a capacity more than once to the same dataset to upgrade a parameter (e.g., through inclination plummet). While every iteration can be communicated as a MapReduce/Dryad work, every employment must reload the data from disk, bringing about a huge performance punishment.

Intelligent analytics: Hadoop is frequently used to run specially appointed exploratory questions on large datasets, through SQL interfaces, for example, Pig [21] and Hive [1]. In a perfect world, a user would have the capacity to stack a dataset of enthusiasm into memory over various machines and inquiry it more than once. Be that as it may, with Hadoop, every inquiry acquires huge inertness (several seconds) because it keeps running as a different MapReduce occupation and peruses data from disk.

This paper shows new cluster computing framework called Spark, which bolsters applications with working sets while giving comparative scalability and fault tolerance properties to MapReduce.

The primary abstraction in Spark is that of a resilient distributed dataset (RDD), which speaks to a read-just accumulation of items partitioned over an arrangement of machines that can be reconstructed if a segment is lost. Clients can expressly store a RDD in memory crosswise over machines and reuse it in numerous MapReduce-like parallel operations. RDDs accomplish fault tolerance

through an idea of genealogy: if an allotment of a RDD is lost, the RDD has enough data about how it was gotten from different RDDs to have the capacity to remake only that parcel. Despite the fact that RDDs are not a general shared memory abstraction, they speak to a sweet-spot between expressivity from one perspective and scalability and reliability then again, and we have discovered them appropriate for an assortment of applications.

Spark is executed in Scala [5], a statically wrote high-level programming language for the Java VM, and uncovered a functional programming interface like DryadLINQ [25]. Likewise, Spark can be utilized intelligently from an altered version of the Scala interpreter, which permits the client to characterize RDDs, functions, variables and classes and utilize them in parallel operations on a cluster. We trust that Spark is the main framework to permit a productive, universally useful programming language to be utilized intelligently to process extensive datasets on a cluster.

Despite the fact that our usage of Spark is still a prototype, early involvement with the framework is empowering. We demonstrate that Spark can beat Hadoop by 10x in iterative machine learning workloads and can be utilized intelligently to filter a 39 GB dataset with sub-second latency.

1.1 HADOOP ALONG WITH SPARK

Hadoop as a big data processing technology has been around for a long time and has ended up being the solution of decision for processing large data sets. MapReduce is an extraordinary solution for one-pass computations, yet not extremely productive for use cases that require multi-pass computations and algorithms. Every progression in the data processing work process has one Map phase and one Reduce phase and you'll have to change over any utilization case into MapReduce pattern to influence this solution. The Job output data between every progression must be put away in the circulated document framework before the following stride can start. Thus, this methodology has a tendency to be moderate because of replication and plate stockpiling. Likewise, Hadoop solutions normally incorporate bunches that are difficult to set up and oversee. It likewise requires the incorporation of a few devices for various big data use cases (like Mahout for Machine Learning and Storm for streaming data processing).

On the off chance that you needed to accomplish something convoluted, you would need to string together a progression of MapReduce jobs and execute them in sequence. Each of those jobs was high-latency, and none could begin until the past occupation had completed totally. Spark permits programmers to create complex, multi-step data pipelines utilizing coordinated non-cyclic diagram (DAG) design. It additionally underpins in-memory data sharing crosswise over DAGs, so that diverse jobs can work with the same data.

Spark keeps running on top of existing Hadoop Distributed File System (HDFS) framework to give improved and extra usefulness. It gives backing to deploying Spark applications in a current Hadoop v1 cluster (with SIMR – Spark-Inside-MapReduce) or Hadoop v2 YARN cluster or even Apache Mesos. We ought to take a gander at Spark as another option to Hadoop MapReduce as opposed to a substitution to Hadoop. It's not proposed to supplant Hadoop but rather to give an extensive and bound together answer for oversee distinctive big data use cases and prerequisites. Figure1 demonstrating the contrast amongst Hadoop and spark.

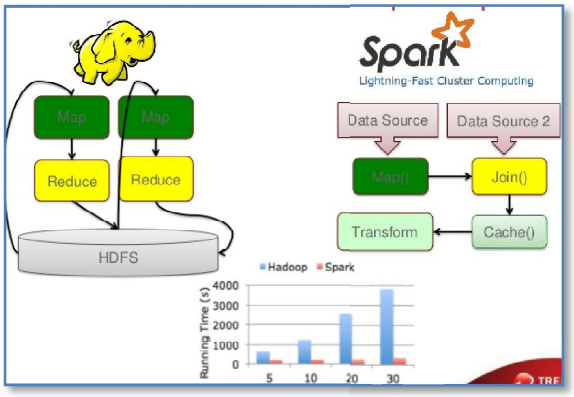

Figure 1 Difference between Hadoop and Spark

1.2 SPARK ARCHITECTURE

Spark Architecture incorporates taking after three principle components:

Data Storage: Spark utilizes HDFS document framework for information stockpiling purposes. It works with any Hadoop perfect information source including HDFS, HBase, Cassandra, and so forth.

Programming interface: The API gives the application developers to make Spark based applications utilizing a standard API interface. Spark gives API to Scala, Java, and Python programming languages.

Resource Management: Spark can be conveyed as a Stand-alone server or it can be on a distributed computing framework like Mesos or YARN. Figure 2 below shows these components of Spark architecture model.

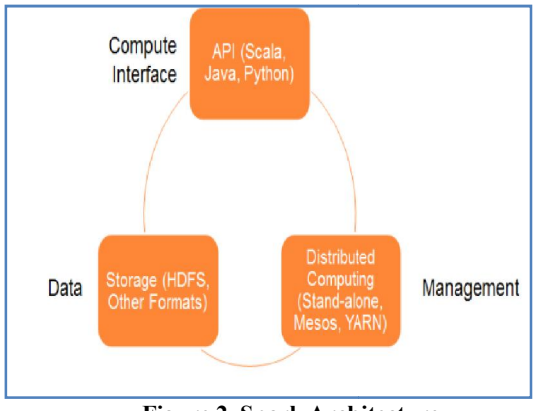

Figure 2. Spark Architecture

1.3 INTELLECTION TO DESIGNATE SPARK

Sparkle utilizes the idea of RDD which permits us to store data on memory and persevere it according to the prerequisites. This permits a massive increment in batch processing job execution (up to ten to hundred times as much as that of routine Map Reduce).

Start additionally permits us to reserve the data in memory, which is valuable if there should arise an occurrence of iterative algorithms, for example, those utilized as a part of machine learning.
Conventional MapReduce and DAG engines are problematic for these applications since they depend on acyclic data stream: an application needs to keep running as a progression of unmistakable jobs, each of which peruses data from stable storage (e.g. a disseminated record framework) and composes it back to stable storage. They bring about noteworthy cost stacking the data on every progression and composing it back to replicated storage.
Flash permits us to perform stream processing with extensive information data and manage just a chunk of data on the fly. This can likewise be utilized for online machine learning, and is very fitting for use cases with a prerequisite for continuous investigation which happens to be a practically universal necessity in the business.
MapReduce is ineffective for multi-pass applications that require low-latency data sharing over multiple parallel operations. These applications are very basic in analytics, and include:

- Iterative algorithms, including numerous machine learning algorithms and graph algorithms like PageRank.
- Interactive data mining, where a client might want to load data into RAM over a bunch and question it more than once.
- Streaming applications that keep up aggregate state after some time.

2. IMPLEMENTATION

2.1 K-MEANS CLUSTERING

K-Means is a simple learning algorithm for clustering analysis. The goal of K-Means algorithm is to find the best division of n entities in k groups, so that the total distance between the group's members and its corresponding centroids, representative of the group, is minimized. The k-means algorithm is used for partitioning where each cluster's centre is represented by the mean value of the objects in the cluster. The Pseudo code is as following:
Step 1: Begin with n clusters, each containing one object and we will number the clusters 1 through n.
Step 2: Compute the between-cluster distance D(r, s) as the between-object distance of the two objects in r and s respectively, r, s =1, 2, …, n. Let the square matrix D = (D(r, s)). If the objects are represented by vectors, we can use the Euclidean distance.
Step 3: Next, find the most similar pair of clusters r and s, such that the distance, D(r, s), is minimum among all the pair wise distances.
Step 4: Merge r and s to a new cluster t and compute the between-cluster distance D(t, k) for any existing cluster $k \neq r, s$. Once the distances are obtained, delete the rows and columns corresponding to the old cluster r and s in the D matrix, since r and s do not exist anymore. Then add a new row and column in D corresponding to cluster t.
Step 5: Repeat Step 3 a total of $n - 1$ times until there is only one cluster left.

3. COMPARISON

Keeping in mind the end goal to arrive at a decision about the useful correlation of Apache Spark and Map Reduce, we performed a near examination utilizing these systems on a dataset that permits us to perform bunching utilizing the K-Means calculation.

3.1 DATASET DESCRIPTION

The Data Set includes healthcare_Sample_datasets size of 3.13 MB collected over the years, and includes patientID, name and other values of the respective records. A sample of the data records is shown as below: The data record is demonstrated in the table1:

Table 1: Healthcare_sample_datasets

PatientID:	int
Name:	chararray
DOB:	chararray
PhoneNumber:	chararray
EmailAddress:	chararray
SSN:	chararray
Gender:	chararray
Disease:	chararray
weight:	float

Sample Record

111	aa1	12/10/1950	1234	aa1@xx.com	11	M	Diabetes	78
112	aa2	12/10/1984	1234	aa2@xx.com	11	F	PCOS	67

3.2 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). To gain a varied analysis, we considered 64MB, 3.13 MB with a single node and 3.13MB 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 2 Results for K-Means using Spark (MLib)

Dataset Size	Nodes	Time (s)
64 MB	1	18
3.13MB	1	149

Table 3. Results for K-Means using Map Reduce (Mahout)

Dataset Size	Nodes	Time (s)
64MB	1	44
3.13 MB	1	291
3.13 MB	2	163

The performance of the spark and Map Reduce are compared with the metrics used for the analysis are: scheduling delay, speed up, energy consumption with respect to the number of nodes in the cluster.

3.2.1 Scheduling Delay: Spark Vs Map Reduce

Figure 3 shows the result of scheduling delay with respect to the spark and map reduce in the Hadoop cluster. The spark is showing the good scheduling length compare to the map reduce.

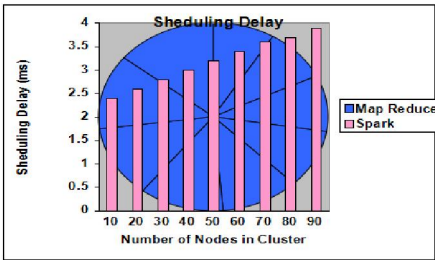

Figure 3. The result of scheduling delay with respect to Spark and Map Reduce in the Hadoop Cluster

3.2.2 Speed up: Spark Vs Map Reduce

The speed up is the ratio of the sequential execution time to the schedule length of the output schedule. Figure 4 shows the result of speed up with respect to spark and Map Reduce. The speed up of spark model is higher than the other Map Reduce approaches, where its value is gradually increasing with regard to the number of clusters.

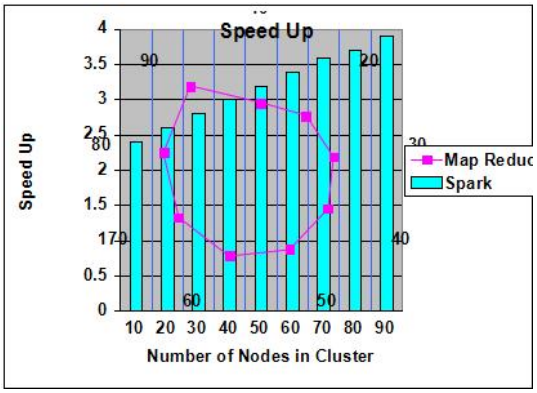

Figure 4: Result of speed up with respect to Spark and Map Reduce

3.2.3 Energy Consumption: Spark Vs Map Reduce

Figure 5 shows the result of energy consumption with respect to the Spark and Map Reduce Model. The Spark consumes less energy than Map Reduce. Its value gradually increases in regards to the number of cluster resource.

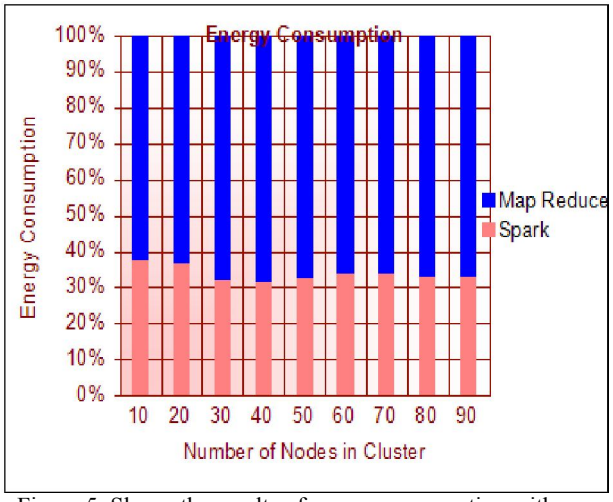

Figure 5: Shows the results of energy consumption with respect to the Spark and Map Reduce

4. CONCLUSION

This research paper gives a review of both the systems furthermore analyzes these on different parameters took after by an execution investigation utilizing K-Means calculation. Our outcomes for this examination demonstrate that Spark is an extremely solid contender and would without a doubt achieve a change by utilizing as a part of memory preparing. Watching Spark's capacity to perform group handling, gushing, and machine learning on the same bunch and taking a gander at the present rate of reception of Spark all through the business, Spark will be the true system for countless cases including Big Data preparing.

REFERENCES

[1] Apache Hive. http://hadoop.apache.org/hive 5Scalaprogramming language. http://www.scala-lang.org.
[2] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data

[3] Y. Yu, M. Isard, D. Fetterly, M. Budiu, U´ . Erlingsson,P. K. Gunda, and J. Currey. DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language. In OSDI '08, San Diego, CA, 2008.

[4] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. Commun. ACM, 51(1):107 113, 2008.

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

[6] B. Nitzberg and V. Lo. Distributed shared memory: a survey of issues and algorithms. Computer, 24(8):52 –60, Aug 1991.

[7] Spark Main Website

[8] Spark Examples

[9]Spark Summit 2014 Conference Presentation and Videos

[10]Spark on Databricks website

