Performance Analysis of Spark using k-means

K.Vamsi¹, Dr. Raman Chadha²

¹M.Tech Scholar, ²Professor and Head

¹Department of CSE, Chandigarh Group of Colleges Technical Campus, Jhanjeri, Mohali.

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).

I. INTRODUCTION

A. Apache spark

Apache Spark was developed in 2009 at UC Berkeley AMP Lab and then open sourced in 2010. While MapReduce and DAG Execution engines abstract distributed processing on a cluster Spark provides abstraction over distributed memory on such clusters.

Spark is quite different from distributed memory abstractions like MemCached in that the later is provides for fine grained read and modify operations on mutable objects that is suitable for OLTP systems. In such systems fault tolerance is provided through replication of data, which may be impractical at data volumes of OLAP systems.

Spark provides an abstraction based on coarse-grained transformations that apply same operation to many data items. Spark also keeps track of enough information about the lineage of such transformations, so such they can be recomputed in the event of failures.

B. Spark Ecosystem:

Other than Spark Core API, there are additional libraries that are part of the Spark ecosystem and provide additional capabilities in Big Data analytics and Machine Learning areas. These libraries include:

- **Spark Streaming**: can be used for processing the real-time streaming data. This is based on micro batch style of computing and processing. It uses the DStream which is basically a series of RDDs, to process the real-time data.

- **Spark SQL**: Spark SQL provides the capability to expose the Spark datasets over JDBC API and allow running the SQL like queries on Spark data using traditional BI and visualization tools. Spark SQL allows the users to ETL their data from different formats it’s currently in (like JSON, Parquet, a Database), transform it, and expose it for ad-hoc querying.

- **Spark MLlib**: MLlib is Spark’s scalable machine learning library consisting of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as underlying optimization primitives.

- **Spark GraphX**: GraphX is the new (alpha) Spark API for graphs and graph-parallel computation. At a high level, GraphX extends the Spark RDD by introducing the Resilient Distributed Property Graph: a directed multi-graph with properties attached to each vertex and edge. To support graph computation, GraphX exposes a set of fundamental operators (e.g., subgraph, join Vertices, and aggregate Messages) as well as an optimized variant of the Pregel API. In addition, GraphX includes a growing collection of graph algorithms and builders to simplify graph analytics tasks. Outside of these libraries, there are others like BlinkDB and Tachyon.

BlinkDB is an approximate query engine and can be used for running interactive SQL queries on large volumes of data. It allows users to trade-off query accuracy for response time. It works on large data sets by running queries on data samples and presenting results annotated with meaningful error bars. Tachyon is a memory-centric distributed file system enabling reliable file sharing at memory-speed across cluster frameworks, such as Spark and MapReduce. It caches working set files in memory, thereby avoiding going to disk to load datasets that are frequently read. This enables different jobs/queries and frameworks to access cached files at memory speed.

And there are also integration adapters with other products like Cassandra (Spark Cassandra Connector) and R (SparkR). With Cassandra Connector, you can use Spark to access data stored in a Cassandra database and perform data analytics on that data.

Following diagram (Figure 1) shows how these different libraries in Spark ecosystem are related to each other.

We’ll explore these libraries in future articles in this series. Spark Architecture Spark Architecture includes following three main components:

- **Data Storage**
- **API**
- **Management Framework**

Let’s look at each of these components in more detail.

---

¹kvamsik.ece@gmail.com, ²dr.ramanchadha@gmail.com
Data Storage: Spark uses HDFS file system for data storage purposes. It works with any Hadoop compatible data source including HDFS, HBase, Cassandra, etc.

API: The API provides the application developers to create Spark based applications using a standard API interface. Spark provides API for Scala, Java, and Python programming languages.

Resource Management: Spark can be deployed as a Stand-alone server or it can be on a distributed computing framework like Mesos or YARN.

II. LITERATURE SURVEY

A. Hadoop

Traditionally large-scale computation has been processor bound and handled relatively small amount of data (typically several GBs) once the data is copied into memory, processor can perform complex computations, usually taking more time than the time it took the copy the data into memory.

This kind of computation often performed by scaling the Hardware vertically while vertical scaling of hardware was very expensive it reduced the complexity of the application code greatly. Nevertheless expensive nature of such systems limited their adaptation to areas where cost was not a major concern.

Distributed computing was invented to reduce the cost by scaling the hardware horizontally. However programming such systems is very complex and failure of nodes is the defining difference of such systems. Also these systems stored data centrally and had to suffer from limited bandwidth of the network. Arrival of Internet age has spawned large websites and petabyte scale databases, storing such massive amount of data centrally and shuttling them across network for processing has become impractical.

Hadoop was created to handle processing of such massive amount of data using large cluster of desktop class hardware. Hadoop design is based on Google’s GFS (Google File System) and MapReduce framework thus Hadoop is also made up of these two primary components namely HDFS (Hadoop Distributed File System) and MapReduce.

Hadoop distributes the data across the nodes in the cluster in advance and computation on this data is performed locally on those nodes, Hadoop preserves the reliability by replicating this data across 2 or more nodes.

MapReduce applications that process this distributed data are written using higher-level abstractions to free the application developers from the concerns like scheduling, node failure, network access and temporal dependencies.

B. HDFS

HDFS designates one or more nodes in the cluster as Name Node and the rest as Data Nodes. Name Nodes maintain the metadata about the files on the HDFS and the actual data itself reside on one or more Data Nodes according to the replication settings.

Data Nodes also double as Task Executors where actual processing of the data is performed. When a file is copied into HDFS it is split into blocks of 64MBs (or as specified in the configuration) and distributed across the cluster.

C. Map Reduce

Applications that process the data stored on the Hadoop cluster are expressed in terms of Map and Reduce functions. Data in the file is presented to the Map function by the framework as a pair of key and value and the Map function maps this key and value into another pair of key and value. These key and value pairs can be any user defined object representation of the underlying data expressible in languages like Java, C, Python, etc. All values produced by a mapper for a given key is collected into list and sorted by the frame work and such sorted list of values and the key are presented to the Reduce function as value and key respectively.

When a MapReduce application is submitted for execution to Hadoop cluster first mapping function is scheduled to run one or more nodes in the cluster based on number of splits estimated for a given data file.

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].

**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.

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.

**D. Limitations**

While Map Reduce ameliorates the complexity of writing a distributed application greatly it achieves that goal through severe restrictions on input/output model and problem composition.

Map Reduce requires that all solutions be composed into just two phases however it allows chaining of several such MR phases for execution one after another. While this requirement simplifies the application development it also makes iterative, interactive and graph processing workloads highly inefficient.

Requirement that the nodes in the cluster keep the interactions to the minimum and intermediate output are stored in the hard drive and almost all communication interactions to the minimum and intermediate output are highly inefficient.

Such inherent limitations of MapReduce coupled with the ever-cheaper DRAM prices and highly reliable datacenter LAN has engendered new models of processing data on Hadoop clusters most notably Apache Tez and Apache Spark.

**III. OBJECTIVE**

K-means with Spark & Hadoop: The objective of this hand on is to let you reason about the parallelization of the K-Means clustering algorithm and use 2 platforms for implementing it: Spark and Hadoop. In class we will experiment with Spark. Then at home you will:

1. Test other Spark functions like the visualization tools.
2. Implement the algorithm in Hadoop.

**Getting started with Spark**

Start by launching Spark’ python shell:

```
$ pyspark
```

**K-means on Spark**

We are going to use the machine learning module of Spark called MLlib designed to invoke machine learning algorithms on numerical data sets represented in RDD. By using RDD it is possible to interact with other components of Spark. When data are not numerical, MLlib requires additional data types like vectors generated using data transformation algorithms from text to numerical vectors (package pyspark.mlib). MLlib implementation of k-means corresponds to the algorithm called K-Means\$\S 5 which is a parallel version of the original one. The method header is defined as follows:

```
KMeans.train(k, maxIterations, initializationMode, runs)
```

- K: number of desired clusters
- Max Iterations: the maximum number of iterations that the algorithm will perform. The more iterations the more precision in results but the execution time will increase.
- initialization Mode: specifies the type of initialization of the algorithm.
- runs: number of times to execute the algorithm

Since K-means is not sure to find an optimum solution it can be executed many times on the same data set and the algorithm will return the best possible solution found in a given execution.

**IV. EXPERIMENTAL DATA ANALYSIS AND RESULTS**

**A. Resilient Distributed Datasets**

RDD is a read-only, partitioned collection of records that can be created though deterministic operations on data in a stable storage or other RDD [3]. These deterministic operations are generally referred as transformations.

As mentioned early these RDDs have enough information embedded in them describing how they are derived so they can be recomputed going all the back from the stable storage if needed.

Spark allows application programmers to control how these RDD’s are partitioned and persisted based on use case. For an example a RDD that is needed by different application or rerun of the same application can choose to save it on disk.

On the other hand transient datasets or lookup tables are kept entirely in memory similarly datasets that are to be joined can be partitioned using same hash function so they are collocated.
Spark takes liberty to spill over partitions of RDDs that cannot fit in memory or completely destroy on heuristics that it can be trivially recomputed or not needed anymore.

**Spark API**

```scala
Lines = spark.textFile("hdfs://...")

errors = lines.filter(_.startsWith("ERROR"))
errors.persist()
errors.count()
```

// Count errors mentioning MySQL:
// errors.filter(_.contains("MySQL")).count()

// Return the time fields of errors mentioning HDFS as an array (assuming time is field 3 in a tab-separated format):
// errors.filter(_.contains("HDFS")).map(_.split("\t")(3)).collect()

**B. Spark Application Model**

Spark uses DSL like API similar to DryadLINQ [4] written in Scala, it also has bindings for other popular languages like Java, Python, etc. Spark distributions come with Scala or Python based shells that can be used to run short scripts or execute interactive queries. Full-fledged applications can also be built using the one of the language bindings.

Transformations and Actions are two different sets of operations provided by Spark. Transformations are used to define RDDs these operations are lazy they are not realized until one of the terminal action is executed on such RDDs.

**Table 1: Transformations and Actions**

<table>
<thead>
<tr>
<th>Transformations</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>map(f : T =&gt; U)</code></td>
<td><code>count()</code></td>
</tr>
<tr>
<td><code>filter(f : T =&gt; Boolean)</code></td>
<td><code>collect()</code></td>
</tr>
<tr>
<td><code>flatMap(f : T =&gt; Seq[U])</code></td>
<td><code>reduce(f : (T, T) =&gt; T):</code></td>
</tr>
<tr>
<td><code>sample(fraction : Float)</code></td>
<td><code>lookup(k : K)</code></td>
</tr>
<tr>
<td><code>groupByKey()</code></td>
<td><code>save(path : String)</code></td>
</tr>
<tr>
<td><code>reduceByKey(f : (V, V) =&gt; V)</code></td>
<td><code>union()</code></td>
</tr>
<tr>
<td><code>join()</code></td>
<td><code>cogroup()</code></td>
</tr>
<tr>
<td><code>crossProduct()</code></td>
<td><code>mapValues(f : V =&gt; W)</code></td>
</tr>
<tr>
<td><code>sort(c : Comparator[K])</code></td>
<td><code>partitionBy( p : Partitioner[K])</code></td>
</tr>
</tbody>
</table>

Resilient Distributed Dataset (based on Matei’s research paper) or RDD is the core concept in Spark framework. Think about RDD as a table in a database. It can hold any type of data. Spark stores data in RDD on different partitions. They help with rearranging the computations and optimizing the data processing. They are also fault tolerance because an RDD know how to recreate and recompute the datasets. RDDs are immutable. You can modify an RDD with a transformation but the transformation returns you a new RDD whereas the original RDD remains the same.

RDD supports two types of operations:

- **Transformation**
- **Action**

**Transformation**: It don't return a single value, they return a new RDD. Nothing gets evaluated when you call a Transformation function, it just takes an RDD and return a new RDD. Some of the Transformation functions are map, filter, flatMap, groupByKey, reduceByKey, aggregateByKey, pipe, and coalesce.

**Action**: Its operation evaluates and returns a new value. When an Action function is called on a RDD object, all the data processing queries are computed at that time and the result value is returned.

Some of the Action operations are reduce, collect, count, first, take, countByKey, and foreach.

**C. Spark Runtime**

Spark converts all transformations and terminal actions into a DAG and executes it using a DAG execution engine similar to that of Dryad. Spark uses master-slave architecture similar to that of MapReduce to execute such DAG on the cluster. Spark application runtime consist of 2 major components one is called the Driver and the other is Executer. Driver coordinates a large number of Executers running on the cluster. Spark Application is launched on the cluster using an external service called cluster manager these cluster managers are pluggable components. Currently Spark supports Hadoop YARN and Apache Mesas as cluster mangers.
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.

<table>
<thead>
<tr>
<th>Dataset Size</th>
<th>Nodes</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>62MB</td>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>1240MB</td>
<td>1</td>
<td>149</td>
</tr>
<tr>
<td>1240MB</td>
<td>2</td>
<td>85</td>
</tr>
</tbody>
</table>

Table 2: Results for K-Means using Spark (MLib)

V. CONCLUSION

To sum up, Spark helps to simplify the challenging and compute-intensive task of processing high volumes of real-time or archived data, both structured and unstructured, seamlessly integrating relevant complex capabilities such as machine learning and graph algorithms. Spark brings Big Data processing to the masses. Check it out!

VI. 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.

REFERENCES