Spark: Unified Framework to manage Big Data

Abstract

Apache Spark is an open source big data framework to build around speed, ease of use and analytics. It was originally developed in 2009 in UC Berkeley's AMPLab, and open sourced in 2010 as an Apache project. Spark is complete framework for big data. Spark comes packaged with higher-level libraries, including support for SQL queries, streaming data, machine learning and graph processing. These standard libraries increase developer productivity and can be seamlessly combined to create complex workflows. Spark enables applications in Hadoop clusters to run up to 100 times faster in memory and 10 times faster even when running on disk.

Introduction

Over the past decade the era of information world has evolving towards the data science, the way of collecting data has been tremendously changes over the period of time. Historically data was being generated an accumulated by workers, company hire employee to enter the user data into the computer and then internet comes along than user can generate their own data which we may called the year of social network, now a days machines are accumulating data. So while collecting those huge amounts of data a single computer didn't handle those data, infinite number of server and parallel processing need to manage those data so the distributed computer system comes along.

Within the past 11 year in big data Hadoop is the only choice and it proven to be the solution to store and manage those huge amount of data. It is an open source project manages by Apache Software Foundation for distributed storage and processing of large data set. It uses the Hadoop distributed file system known as HDFS which is able to store large files across the multiple servers.

Hadoop doesn't meet the requirement of today's need it doesn't maintain the speed in calculating the large amount of data set also it will take more time to run queries.

Spark was introduced by Apache Software foundation which is a fast, in-memory engine of data processing. It is a 10x faster on disk and 100x faster in memory than Hadoop. Although it use the Hadoop file system to store the data but it is not the modified version of Hadoop. Spark can run standalone along with Hadoop. It has its own clustering system it only use Hadoop as storage.

Apache Spark is a powerful open source project written in Scala Programming Language for processing large data set with speed, ease of use, and sophisticated analytics. It provides in-memory analytics which is faster than Hadoop/Hive. It designed for running iterative algorithms and interactive analytics. Spark use Resilient Distributed Datasets (RDD) for its data structure which is an immutable object. While computing in different nodes each dataset in RDD is divided into logical partitions. The RDD is a fault-tolerant collection of elements that can operate on in parallel because of it RDD know how to recreate and recomputed the datasets. RDD supports in-memory processing computation which store the state of memory as an object across the jobs and the object is sharable between those jobs.

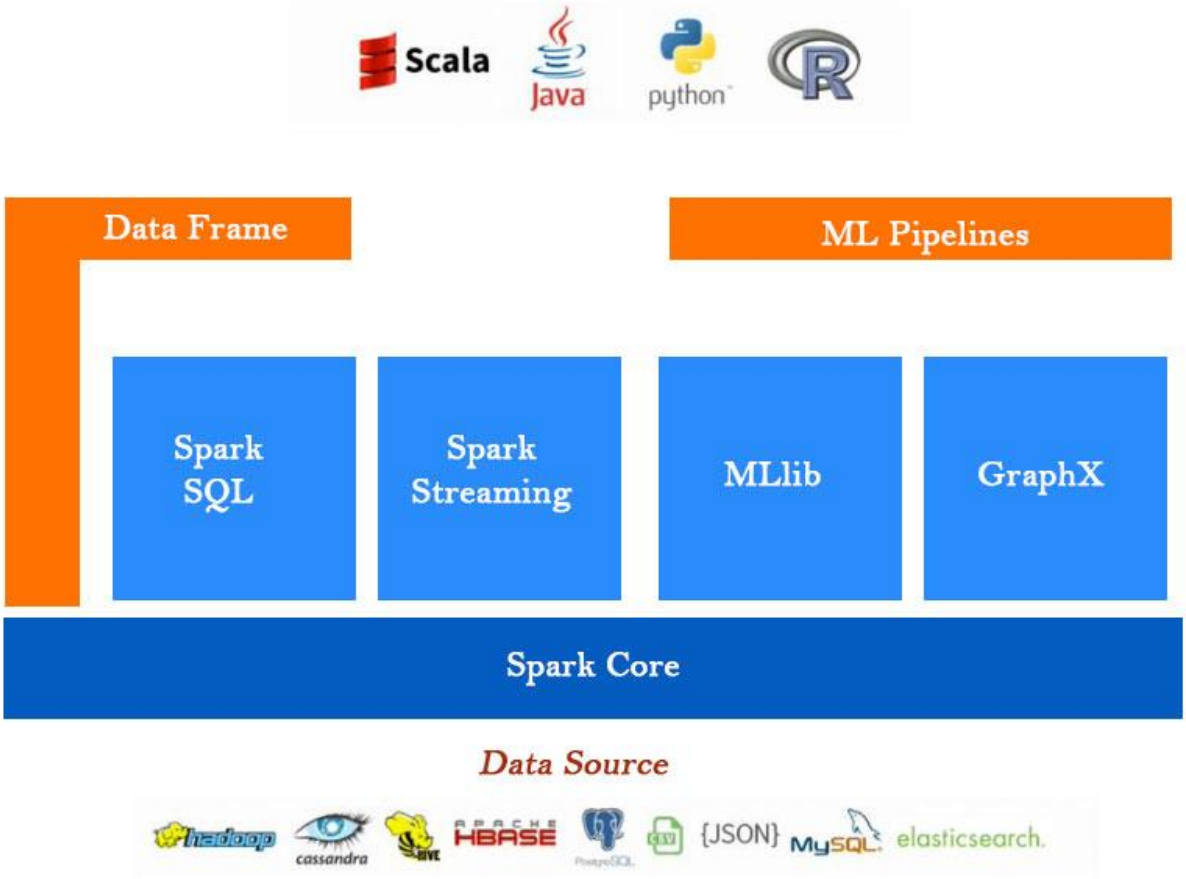


Figure 1: Spark Components

There are additional libraries which is part of the spark other than the spark core that gives more capabilities in machine learning and big data.

Spark Streaming: It is used to process the data in real time typically based on micro batch style of computing and processing. Apache Spark has provided an unified engine that natively supports both batch

and streaming workloads. This is different from other systems that either have a processing engine designed only for streaming, or have similar batch and streaming APIs but compile internally to different engines. Spark's single execution engine and unified programming model for batch and streaming lead to some unique benefits over other traditional streaming systems

Spark SQL: It provides the JDBC API and allows running SQL like queries on Spark. Spark SQL is a component on top of Spark Core that introduces a new data abstraction called SchemaRDD, which provides support for structured and semi-structured data.

Spark MLlib: It consists the algorithms and utilities including regression, clustering, classification and other machine learning. MLlib is a distributed machine learning framework above Spark because of the distributed memory-based Spark architecture. It is, according to benchmarks, done by the MLlib developers against the Alternating Least Squares (ALS) implementations.

Spark GraphX: It includes a collection of graph algorithm and build graph analytics tasks. GraphX is a distributed graph-processing framework on top of Spark. It provides an API for expressing graph computation that can model the user-defined graphs by using Pregel abstraction API.

Spark Program Flow

In cluster spark runs independent processes by the SparkContext object in main program which is called driver program. The applications that are in different types of cluster will connect to SparkContext. It connects the application that occupied the resource of clusters. To run the computation and store data of the application it will executes on nodes in the clusters and then it sends application code to the executors. At the end SparkContext sends tasks to the executors to run.

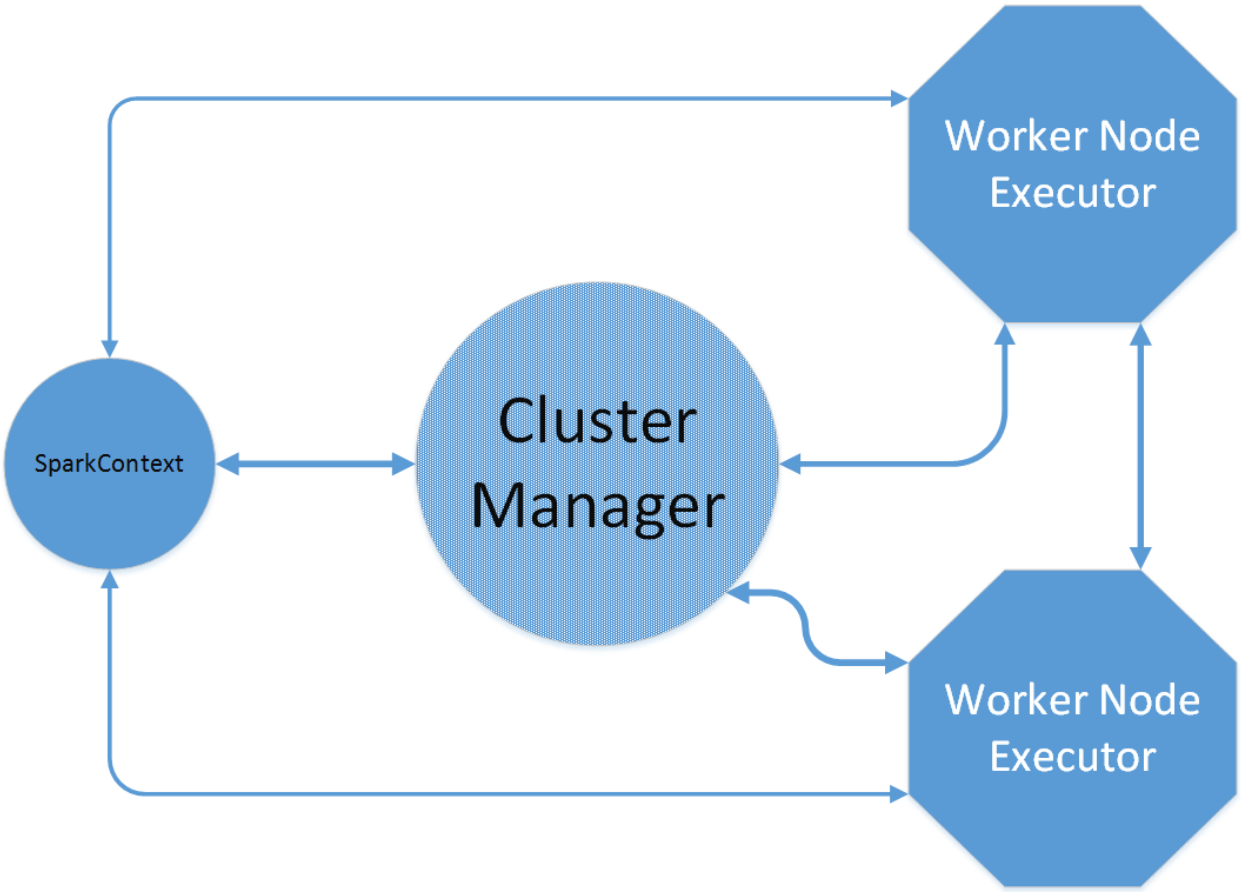


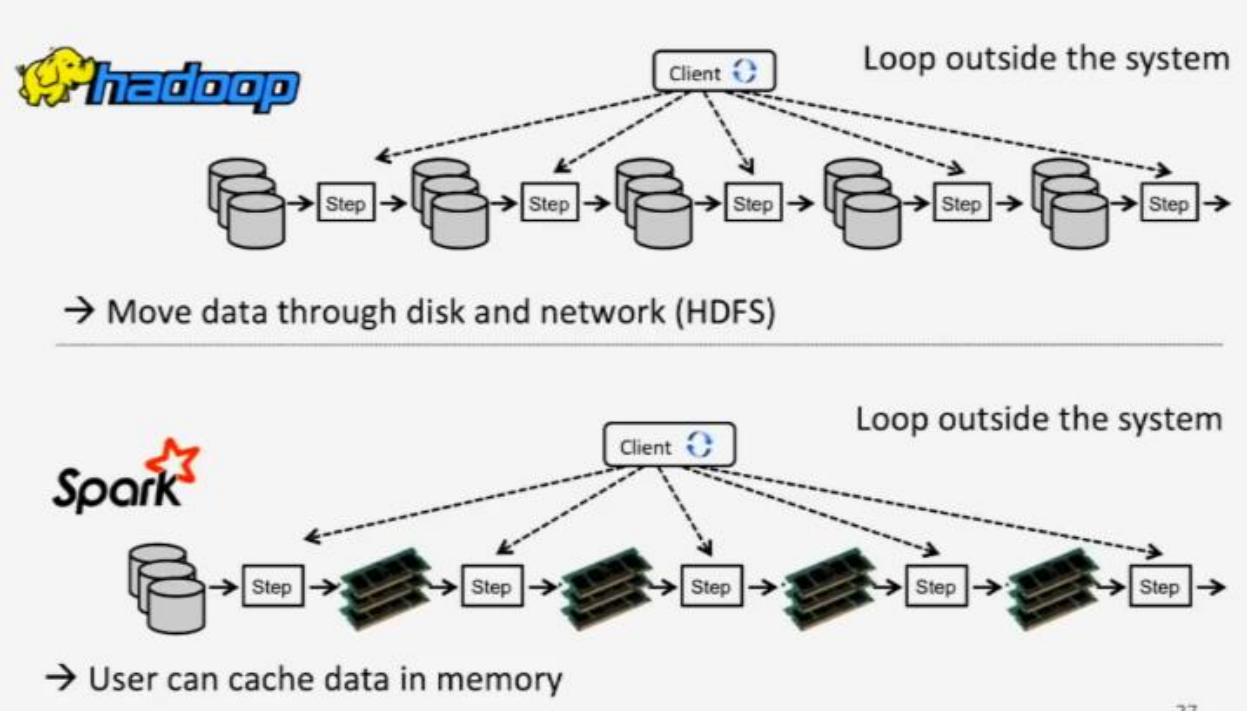
Figure 2 Spark Component

Hadoop MapReduce Vs. Spark

Hadoop and Spark are both different and independent big-data frameworks. Hadoop is a distributed data infrastructure whereas Spark is a data-processing tool that operates those data, It doesn't do distributed storage.

Hadoop comes with storage known as Hadoop Distributed File System and processing module called MapReduce so we can say that it is a complete package, in other hand spark doesn't come with the storage it needs to be connected with the file system, it can use HDFS or other file system. Initially Spark was designed to enhance the Hadoop so that many recommend using HDFS.

Spark is lot faster than MapReduce because it processes the data in memory instead of disk that MapReduce use. MapReduce read data from cluster, perform the task, write result in cluster, read from cluster and again doing the same thing but Spark completes its full task in-memory and real-time. Read data from cluster, perform the task in memory, write back to cluster So that it is 100 times faster than Hadoop.



Spark provides multi language api such as Java, Scala, Python so that developers can quickly develop application and build parallel apps. Along with Spark supports SQL queries, streaming data, and complex analytics along with map and reduce.

Spark can runs on Hadoop, Mesos, standalone or in the cloud. It supports multiple data sources such as HDFS, Cassandra, HBase, S3.

The following is the major cases Spark over Hadoop

- Iterative Algorithms in Machine Learning
- Interactive Data Mining and Data Processing
- Spark is a fully Apache Hive-compatible data warehousing system that can run 100x faster than Hive.
- Stream processing: Log processing and Fraud detection in live streams for alerts, aggregates and analysis
- Sensor data processing: Where data is fetched and joined from multiple sources, in-memory dataset really helpful as they are easy and fast to process.

Upon first glance, it seems that using Spark would be the default choice for any big data application. However, that's not the case. MapReduce has made inroads into the big data market for businesses that need huge datasets brought under control by commodity systems. Spark's speed, agility, and relative ease of use are perfect complements to MapReduce's low cost of operation.

The truth is that Spark and MapReduce have a symbiotic relationship with each other. Hadoop provides features that Spark does not possess, such as a distributed file system and Spark provides real-time, in-memory processing for those data sets that require it. The perfect big data scenario is exactly as the designers intended—for Hadoop and Spark to work together on the same team.

Spark	Hadoop
Apache Spark processes data in-memory	Hadoop MapReduce persists back to the disk after a map or reduces action.
Spark can do great in real-time processing as well as batch processing.	Hadoop Map Reduce is great for batch processing. If you want a real-time option you'll need to use another platform like Storm or Impala, and for graph processing you can use Graph.
Spark is generally a lot faster than MapReduce, 10 times faster than Map Reduce for batch processing and up to 100 times faster for in-memory analytics.	Map Reduce is slow
Spark performs better when all the data fits in the memory.	Hadoop MapReduce is designed for data that doesn't fit in the memory
The process will start from beginning if process crashes in the middle of execution.	The process will continue where it is left off if the process crashes in the middle of execution.
Batch Processing	Real-time processing
Stores Data on Disk	Stores Data in Memory
Written in Java	Written in Scala
Less secured because it is still in early stage of development.	More secured , has security features like kerborose, sentry etc.

How to install Spark

Step 1: Verify Java Installation

First of all you need to verify Java installation which is mandatory for installing Spark. You can verify it by using command

Java –version

In case you have not install Java then do it.

Step 2: verifying Scala installation

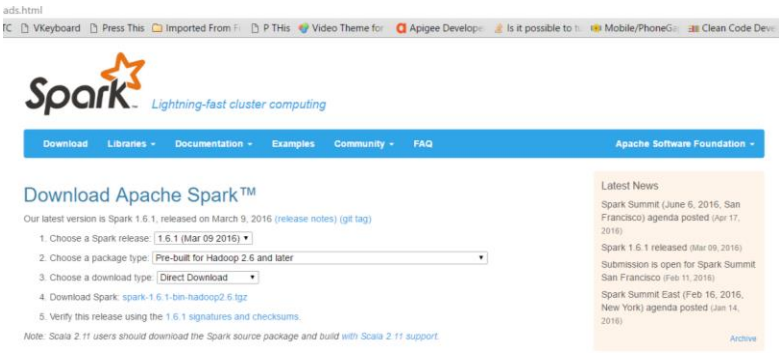
You can verify Scala installation by using this command

Scala –version

In case you don't have Scala installed then proceed to next step.

Step 3: Downloading Apache Spark

Download the latest version of Spark https://spark.apache.org/downloads.html and copy the Spark and the same Scala folder and extract it.



Step 4: Installation Spark

Setting up the environment for Spark

Setx JAVA_HOME

Setx c:\Program Files\Java\jdk1.8.0_71

Step 5: Verifying the Spark Installation

You can use command: spark –shell

If spark is installed successfully then you get out like this:

```
Spark assembly has been built with Hive, including Datanucleus jars on classpath
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
15/06/04 15:25:22 INFO SecurityManager: Changing view acls to: hadoop
15/06/04 15:25:22 INFO SecurityManager: Changing modify acls to: hadoop
15/06/04 15:25:22 INFO SecurityManager: SecurityManager: authentication disabled;
  ui acls disabled; users with view permissions: Set(hadoop); users with modify permissions: Set(hadoop)
15/06/04 15:25:22 INFO HttpServer: Starting HTTP Server
15/06/04 15:25:23 INFO Utils: Successfully started service 'HTTP class server' on port 43292.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.4.0
      /_/

Using Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_71)
Type in expressions to have them evaluated.
```

```
Spark context available as sc
scala>
```

Word Count Examples

Step 1 : Load text file

```
scala> val input = sc.textFile("Z:\\BDA\\homework5\\obamaspeech\\1.txt")
```

Step 2: Use lambda expression to count the grouping text.

```
scala> val wordCounts = input.flatMap(line=>line.split("
")).map(word=>(word,1)).reduceByKey((a,b)=>a+b)
```

Note: With single line of lambda code it can group the text and count the frequency.

Step 3: Collect and Count

```
scala> wordCounts.collect()
```

res9: Array[(String, Int)] = Array((segregation,,1), (ills,1), (country,1), (God,3), (greater,4), (patrol,1), (responsibility,1), (carried,3), (filled,1), (Now,,1), (prosperous,,1), (play,,1), (been,7), (wage,,1), (met.,1), (citizenship.,1), (challenges,2), (over,3), (packed,1), (Guided,1), (make,2), (conflict,1), (tribe,1), (prefer,1), (long,,1), (threat,,1), (month,1), (real.,1), (are,22), (economy,3), (yet,,1), (indifference,1), (scarcely,1), (raise,1), (safely,1), (sacrifices,1), (aims,1), (our,57), (plans.,1), (storms,1), (use;,1), (dust,1), (values,1), (alliances,1), (presidential,1), (come.,1), (Afghanistan.,1), (ancestors.,1), (pursue,1), (Yet,,1), (them,4), (coldest,1), (chosen,1), (sapping,1), (village,1), (pleasures,1), (firm,1), (alongside,1), (ourselves,,1), (pass;,1), (cha.

Spark with R Studio

```
Sys.setenv(SPARK_HOME="C:/Users/suraj/Downloads/spark")
.libPaths(c(file.path(Sys.getenv("SPARK_HOME"),"R","lib"),.libPaths()))
library(SparkR)
sc <-sparkR.init(master="local")
sqlContext <-sparkRSQL.init(sc)
DF <-createDataFrame(sqlContext,faithful)
head(DF)
```

```
> head(DF)
  eruptions waiting
1     3.600      79
2     1.800      54
3     3.333      74
4     2.283      62
5     4.533      85
6     2.883      55
```

```
localDF <-data.frame(name=c("John","Smith","Sarah"),ages=c(19,23,18))
df<-createDataFrame(sqlContext,localDF)
printSchema(df)
path <- file.path(Sys.getenv("SPARK_HOME"), "examples/src/main/resources/people.json")
peopleDF <- jsonFile(sqlContext, path)
printSchema(peopleDF)
```

```
registerTempTable(peopleDF, "people")

# SQL statements can be run by using the sql methods provided by sqlContext
teenagers <- sql(sqlContext, "SELECT name FROM people WHERE age >= 13 AND age <= 19")

teenagersLocalDF <- collect(teenagers)

print(teenagersLocalDF)
```
```
  name
1 Justin
```
```
sparkR.stop()
```

Conclusion

Apache Spark is the sparkly new evolving tool in Big Data. Spark has excellent performance and is highly cost-effective. It's compatible with all Hadoop's data sources and files formats and has friendly APIs that are available in different languages. It also includes graph processing and machine learning capabilities.

It does have a collection of all required tools to work with Big Data and also support third party tool for storage and other processes.

References

1. What is Apache Spark
   https://databricks.com/spark/about

2. Is Apache Spark going to replace Hadoop
   http://aptuz.com/blog/is-apache-spark-going-to-replace-hadoop/

3. Big Data Processing with Apache Spark – Part 1: Introduction
   http://www.infoq.com/articles/apache-spark-introduction

4. Apache Spark RDD
   http://www.tutorialspoint.com/apache_spark/apache_spark_rdd.htm

5. Spark SQL & DataFrames
   http://spark.apache.org/sql/

6. An Overview of Apache Spark
   https://www.youtube.com/watch?v=mL5dQ_1gkiA