Spark

PRESENTED BY

SHREYASH SHUKLA

TABLE OF CONTENTS

Introduction to Apache Spark 3

  Components of Apache Spark 4

    Spark Core 4

    Spark SQL 4

    Spark Streaming 4

    MLlib (Machine Learning) 4

    GraphX (Graph Computation) 4

    SparkR 4

Spark Features 4

  Resilient Distributed Dataset 5

    The origins of RDD 5

    The features of RDDs: 5

    RDD operations: 5

  Fast Processing: 5

  Analytics 5

    Map function 6

    Reduce Function 6

  Real Time Processing 6

  Active and Expanding Community 6

  Different modes of using apache spark 6

    Local mode 6

    Standalone mode 6

    Cluster mode 6

Drawbacks of Apache Spark: 7

  No Support for Real-time Processing 7

  No File Management System 7

  Expensive 7

  Manual Optimization 7

  Iterative Processing 7

  Latency 7

  Window Criteria 7

SQl vs Apache Spark: 8

Basic Queries 8

Querying Programmatically on Spark 8
Implementation 9
Spark Live Streaming 9
Spark SQL 10
Conclusion: 11

INTRODUCTION TO APACHE SPARK

With the increase in the need of trying to understand how to store vast amounts of information and make use of that, a lot of platforms have been developed and reinvented to come up with an optimum solution to do the same. Apache Spark is one of the latest iterations of this. It is a fast and general engine for large-scale data processing.

It was initially designed specifically for data science, but evolved to support more use cases and now also includes real-time stream event processing. Spark supports Java, Scala, Python, and R natively, as well as ANSI Standards SQL. It offers 80 high-level operators that make it fast and easy to build applications, including parallelization and streaming. Many popular analytics platforms like Tableau can connect directly using SQL. Besides running in nearly any environment, one can access the data in the Hadoop distributed file system, Hive, or any other Hadoop data source. It can also directly connect to traditional relational databases using dataframes in Python and Scala. The idea here is that for Spark to be successful, it must play well with others.

The four main advantages of Spark are:

- speed
- Ease of use
- Generality
- platform agnostic.

Spark has an advanced Dag engine that can achieve up to hundred times faster processing than traditional MapReduce on Hadoop. This is largely due to how Spark distributes the execution across its cluster and performs many of its operations in memory.

COMPONENTS OF APACHE SPARK

SPARK CORE

Main component on which all the other components are based. It includes task distribution, scheduling, and the input/output operations.

SPARK SQL

This component supports the ANSI standard SQL language. We need to create or at least define table structures in Spark before being able to use in Spark. One advantage of having a strong SQL support is that it enables tools like Tableau to easily integrate with Spark.

SPARK STREAMING

Spark Streaming runs micro batches, which operate just like normal batch operations, just more frequently on smaller datasets. This works using a Lamba architecture.

MLLIB (MACHINE LEARNING)

This component includes many of the most common machine learning functions.

GRAPHX (GRAPH COMPUTATION)

Used for running graph database jobs where one can identify relationships between entities in the data and then query that using those relationships.

SPARKR

Provides an interface for connecting your Spark cluster from the R statistical package. This package provides distributed DataFrames, which are comparable to DataFrames in R.

SPARK FEATURES

RESILIENT DISTRIBUTED DATASET

Resilient Distributed Dataset is the primary data abstraction in Apache Spark. This data abstraction, allows you to write programs that transform these distributed datasets.

RDD can be described as:

- Immutable Distributed
- Collections of objects spread across a cluster, stored in RAM or on Disk
- Built through parallel transformations
- Automatically rebuilt on failure

THE ORIGINS OF RDD

The original paper that gave birth to the concept of RDD is Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing by Matei Zaharia.

THE FEATURES OF RDDS:

- Resilient, i.e. fault-tolerant with the help of RDD lineage graph and so able to recomputed missing or damaged partitions due to node failures.
- Distributed with data residing on multiple nodes in a cluster.
- Dataset is a collection of partitioned data with primitive values or values of values, e.g. tuples or other objects (that represent records of the data you work with).

RDD OPERATIONS:

ACTIONS

- Creating RDD
- Storing RDD
- Extracting data from RDD on the fly

TRANSFORMATIONS

- Restructure or transform RDD data

FAST PROCESSING:

When it comes to Big Data processing, speed always matters. Sparks allows to run up to 100 times in memory and 10 times on disk. Spark makes it possible by reducing number of read/write to disc. It stores this intermediate processing data in-memory. It uses the concept of an Resilient Distributed Dataset (RDD), which allows it to transparently store data on memory and persist it to disc only it’s needed. This helps to reduce most of the disc read and write which in tune is the main time-consuming factor of data processing.

ANALYTICS

In addition to simple “map” and “reduce” operations, Spark supports SQL queries, streaming data, and complex analytics such as machine learning and graph algorithms out-of-the-box. Not only that, users can combine all these capabilities seamlessly in a single workflow.

MAP FUNCTION

Map () operation applies to each element of RDD and it returns the result as new RDD. It is a transformation action. In the Map, operation developer can define his own custom business logic.

REDUCE FUNCTION

Spark reduce operation is almost similar as reduce method in Scala. It is an action operation of RDD which means it will trigger all the lined up transformation on the base RDD (or in the DAG) which are not executed and then execute the action operation on the last RDD.

REAL TIME PROCESSING

Spark can handle real time streaming. Map-reduce mainly handles and process the data stored already. However, Spark can also manipulate data in real time using Spark Streaming.

ACTIVE AND EXPANDING COMMUNITY

Apache Spark is built by a wide set of developers from over 50 companies. The project started in 2009 and as of now more than 250 developers have contributed to Spark already! It has active mailing lists and JIRA for issue tracking.

DIFFERENT MODES OF USING APACHE SPARK

LOCAL MODE

In the local mode the spark runs in a non-distributable single Java Virtual Machine (JVM). Spark deploys all its executions components – drivers, executors, and the master in the same single JVM. Parallelism is achieved by setting the number of threads in the master URL. This is the only mode where the driver executes the task.

STANDALONE MODE

Spark Standalone mode has it’s built in cluster environment. This is the easiest way to run a Spark application in most of the cases. This mode consists of two main nodes – the Standalone Master and the Standalone Worker. The Standalone Master is a resource manager and is responsible for keeping track of the workers, mapping the ids to the applications, track of the work progress of each worker, mapping the id to the workers, keeping track of the drivers and ending the process. There can be one or more Standalone Worker. They are the worker nodes for the application. They receive the jobs from the master node and perform the operations and report the progress to the master node. Parallelism is achieved since jobs are distributed among different workers.

CLUSTER MODE

In the cluster mode spark applications run as independent group of process on a cluster. The application is distributed among various clusters. There is a cluster manager to manage all the various clusters. There are various Worker nodes that are distributed among various clusters and the cluster manager is responsible for managing these nodes. Each worker node has its own executer which executes the process that are sent by the cluster manager. The benefit of the mode is that each application gets its own executor and runs until the end of the application. Therefore, isolation is achieved. Since it is distributed across various cluster, handling of failing of any node is also attained. Also, the executions remain consistent since all the executors and the cluster manager are connected to the same SparkContext

DRAWBACKS OF APACHE SPARK:

NO SUPPORT FOR REAL-TIME PROCESSING

In Spark Streaming, the arriving live stream of data is divided into batches of the pre-defined interval, and each batch of data is treated like Spark Resilient Distributed Database (RDDs). Then these RDDs are processed using the operations like map, reduce, join etc. The result of these operations is returned in batches. Thus, it is not real-time processing, but Spark is near real-time processing of live data. Micro-batch processing takes place in Spark Streaming.

NO FILE MANAGEMENT SYSTEM

Apache Spark does not have its own file management system; thus, it relies on some other platform like Hadoop or another cloud-based platform which is one of the Spark known issues.

EXPENSIVE

In-memory capability can become a bottleneck when we want cost-efficient processing of big data as keeping data in memory is quite expensive, the memory consumption is very high, and it is not handled in a user-friendly manner. Apache Spark requires lots of RAM to run in-memory, thus the cost of Spark is quite high.

MANUAL OPTIMIZATION

The Spark job requires to be manually optimized and is adequate to specific datasets. If we want to partition and cache in Spark to be correct, it should be controlled manually.

ITERATIVE PROCESSING

In Spark, the data iterates in batches and each iteration is scheduled and executed separately.

LATENCY

Apache Spark has higher latency as compared to Apache Flink.

WINDOW CRITERIA

Spark does not support record based window criteria. It only has time-based window criteria.

SQL VS APACHE SPARK:

SQL, or Structured Query Language, is a standardized language for requesting information (querying) from a datastore, typically a relational database. SQL is supported by almost all relational databases of note, and is occasionally supported by other products that aren't relational databases. Like most standards, the implementations vary somewhat, so the specifics depend a little on which database you're using. SQL is a language, or a standard for a language, not a specific piece of software.

Spark is an apache open-source product for "big data", a "data processing" engine. It's not a language or a database, rather a cluster computing middleware framework that sits between the data source and the querying. It supports different data sources and different querying models, including a SQL variant, "Spark SQL".

So, what's the difference between them? Practically everything. SQL is a standardized language used primarily in relational databases. Spark is a cluster computing product used to analyze large datasets.

BASIC QUERIES

Query Description	SQL	Spark
Displaying the entire table	SELECT * FROM usertweets.tweetsuser_11_20_2017;	tweetsuser_11_20_2017.show
Group by	SELECT isRetweet,count(*) FROM usertweets.tweetsuser_11_20_2017 group by isRetweet;	Tweets.groupBy("isRetweet").count().show()
Table structure	describe tweetsuser_11_20_2017;	Tweets.printSchema()
Specific column/s	SELECT textOfTweet, userScreenName FROM tweetsuser_11_20_2017;	Tweets.select("textOfTweet", "userScreenName").show()
Where clause	select * from tweetsuser_11_20_2017 where isRetweet = "true";	Tweets.filter(Tweets("isRetweet").equalTo("true")).show()

QUERYING PROGRAMMATICALLY ON SPARK

Apart from throwing queries using the Spark SQL or Scala, We can also use SQL to get the results.

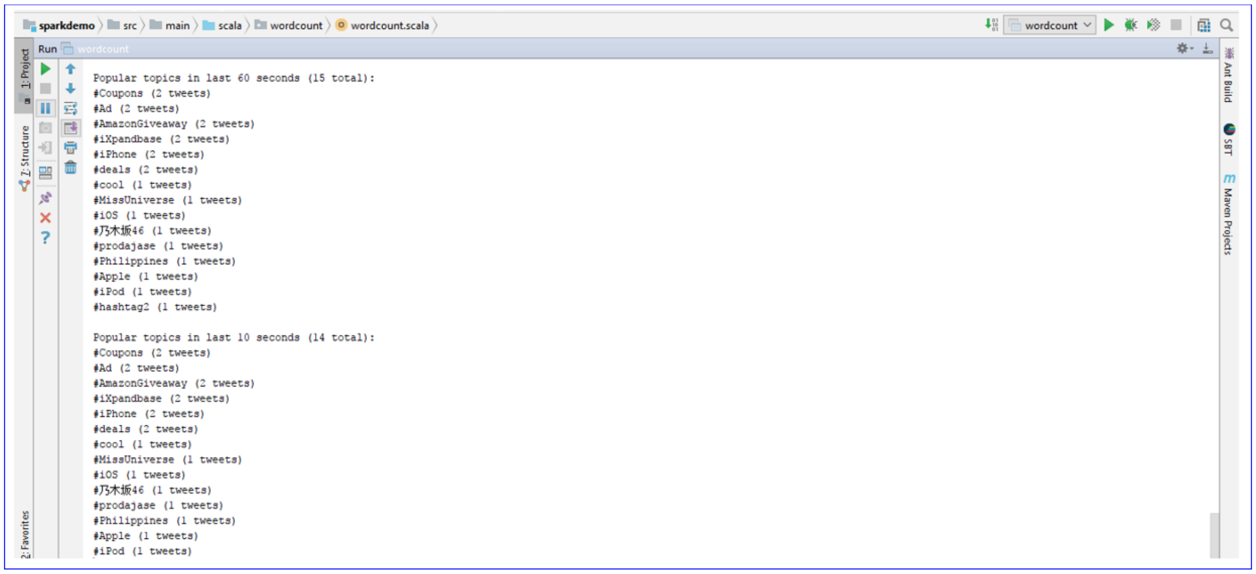
Query Description	SQL query in Spark
Displaying the entire table	sqlContext.sql("SELECT * FROM Tweets_Spark").show()
Group by	sqlContext.sql("SELECT isRetweet,count(*) FROM Tweets_Spark group by isRetweet").show()
Table structure	sqlContext.sql("describe Tweets_Spark").show()
Specific column/s	sqlContext.sql("SELECT textOfTweet, userScreenName FROM Tweets_Spark").show()
Where clause	sqlContext.sql("select * from Tweets_Spark where isRetweet = \"true\" ").show()

IMPLEMENTATION

We are implementing two components of Spark, Spark Live Streaming and SparkSQL.

SPARK LIVE STREAMING

For the Spark Live Streaming, we have used the Twitter App created for the optional Homework 2. Here, we are passing the Twitter credentials as the arguments as well as the filter, i.e. the tweets will be retrieved which are closely related to the filter. For example, you we want the tweets related to Football, I'll pass football as the argument as well. We are running the program on a local computer and local[2] means it uses 2 processors. We have set the Dstream window to 2 seconds, which means it will fetch the tweets every 2 seconds. Dstreams are converted into RDDs. For Hashtags, we are filtering the stream and splitting it by the space and are only capturing the hashtags. We are then finding what are the top tweets in the last 60 seconds and 10 seconds and using map reduce to map them. Then for each RDD, we are printing the tag and its respective count for that 10 second window. Then we are starting the streaming context and the program runs until we manually stop the execution.



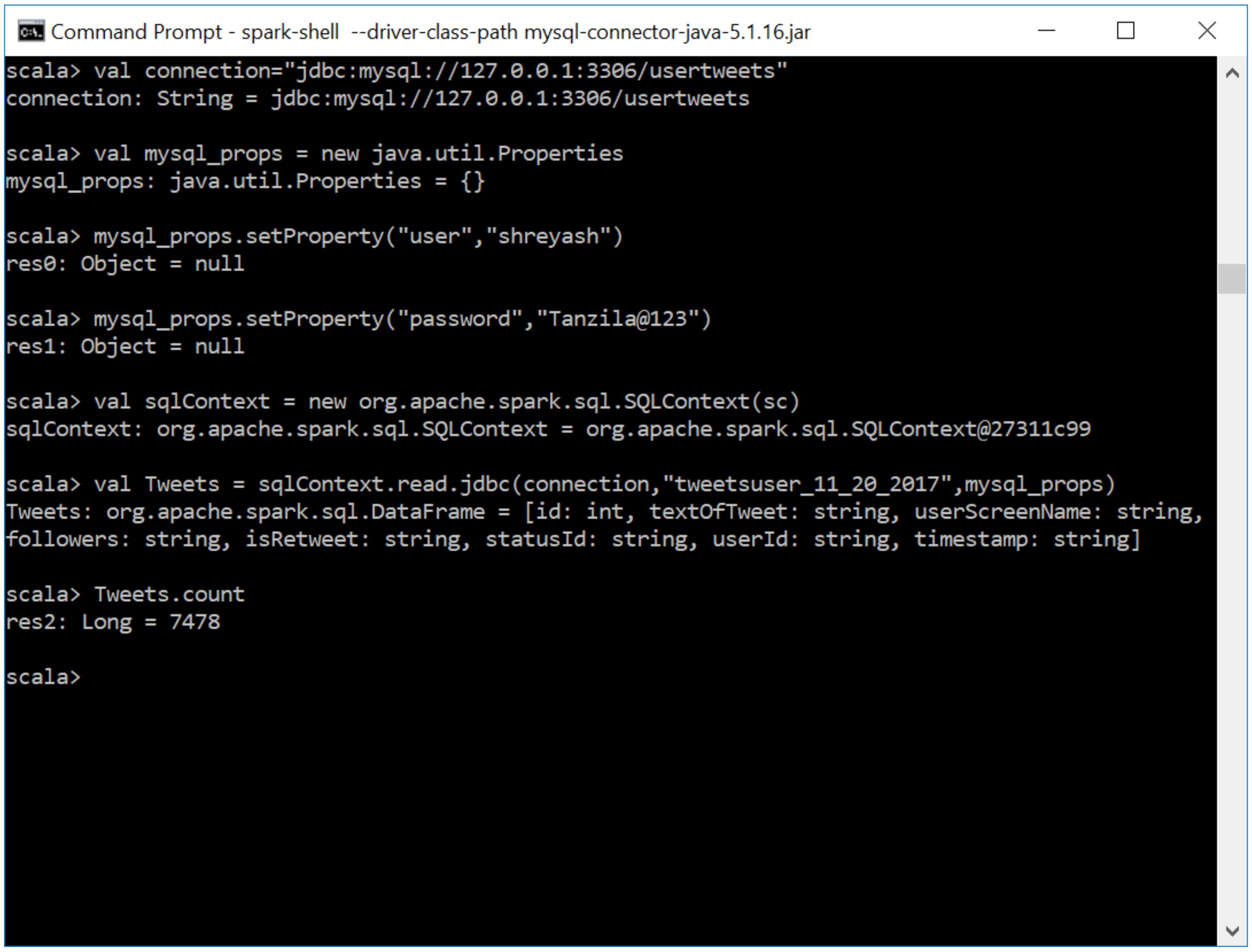
```
Popular topics in last 60 seconds (15 total):
#Coupons (2 tweets)
#Ad (2 tweets)
#AmazonGiveaway (2 tweets)
#iXpandbase (2 tweets)
#iPhone (2 tweets)
#deals (2 tweets)
#cool (1 tweets)
#MissUniverse (1 tweets)
#iOS (1 tweets)
#乃木坂46 (1 tweets)
#prodajase (1 tweets)
#Philippines (1 tweets)
#Apple (1 tweets)
#iPod (1 tweets)
#hashtag2 (1 tweets)

Popular topics in last 10 seconds (14 total):
#Coupons (2 tweets)
#Ad (2 tweets)
#AmazonGiveaway (2 tweets)
#iXpandbase (2 tweets)
#iPhone (2 tweets)
#deals (2 tweets)
#cool (1 tweets)
#MissUniverse (1 tweets)
#iOS (1 tweets)
#乃木坂46 (1 tweets)
#prodajase (1 tweets)
#Philippines (1 tweets)
#Apple (1 tweets)
#iPod (1 tweets)
```

SPARK SQL

For Spark SQL, we have integrated MySQL with apache spark to extract the data from the SQL server and save it on Apache Spark. We have noticed that, the dataframe which is created in apache spark gets updated if there are any changes in the SQL table. We have followed the below steps for this implementation.

1. Connecting to spark using the MySQL connector driver.
2. Creating a connection variable to hold the MySQL JDBC URL as a string.
3. Creating connection properties object with username and password.
4. Loading DataFrame with JDBC data-source (URL, table name, properties).

We have used the java project code(CollectUserTweets_demo) provided by Prof. Tafti for extracting the tweets and saving it in the SQL server. Once we have integrated apache spark with MySQL server, we can play around with the dataframe in Realtime as the tweets are been extracted and saved in the MySQL server.


Command Prompt - spark-shell --driver-class-path mysql-connector-java-5.1.16.jar

```
scala> val connection="jdbc:mysql://127.0.0.1:3306/usertweets"
connection: String = jdbc:mysql://127.0.0.1:3306/usertweets

scala> val mysql_props = new java.util.Properties
mysql_props: java.util.Properties = {}

scala> mysql_props.setProperty("user","shreyash")
res0: Object = null

scala> mysql_props.setProperty("password","Tanzila@123")
res1: Object = null

scala> val sqlContext = new org.apache.spark.sql.SQLContext(sc)
sqlContext: org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@27311c99

scala> val Tweets = sqlContext.read.jdbc(connection,"tweetsuser_11_20_2017",mysql_props)
Tweets: org.apache.spark.sql.DataFrame = [id: int, textOfTweet: string, userScreenName: string,
followers: string, isRetweet: string, statusId: string, userId: string, timestamp: string]

scala> Tweets.count
res2: Long = 7478

scala>
```

CONCLUSION:

Apache Spark is a cluster computing platform designed to be fast and extends the popular MapReduce model to efficiently support more types of computations, including interactive queries and stream processing. Spark Integrates closely with other Big Data tools, this ability helps us to build applications that seamlessly combine different models. Spark is for fit wide range of cases because of its versatility, integration and a rich set of different libraries.

CITATIONS

https://courses.cs.ut.ee/MTAT.08.036/2014_fall/uploads/Main/L4_Spark_Lecture.pdf

https://spark.apache.org/docs/0.8.1/api/core/org/apache/spark/rdd/RDD.html

https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-rdd.htmlhttps://www.infoq.com/articles/apache-spark-introduction

https://dzone.com/articles/6-sparkling-features-apache

http://backtobazics.com/big-data/spark/apache-spark-reduce-example/

https://data-flair.training/blogs/limitations-of-apache-spark/