**DS-610 Big Data Analytics**

**Reading Assignment 1**

**Topic: Resilient Distributed Datasets and Spark**

*Madhumita Duvvuri*

**SPARK and RDDs**

**Overview:**

SPARK is an open source platform for cluster computing mostly known for its speed, in-memory computations performed on large datasets and its fault tolerance efficiency. We can say that it is extended from MapReduce to improve the efficiency of numerous types of computations which includes interactive queries and real-time streaming of data. It is fast, easily available through many APIs in SQL, Python, Java and SCALA and inexpensive general purpose engine for Big Data processing.

RDDs or Resilient Distributed Datasets is a read-only data structure which distributes large amounts of data over a group of machines (generally called a cluster) and these machines are made fault-tolerant. The major aspect for using a RDD is that it allows reuse of data efficiently and also allows users to persist intermediate results in memory.

**Discussion:**

We will discuss in this section about the article “Resilient Distributed Datasets: A Fault Tolerant Abstraction for In-Memory Cluster Computing”. The key points here are (1) motivation behind the introduction of SPARK, (2) challenges to achieve a fault tolerant and efficient distributed memory abstraction and (3) Features of SPARK and its components.

The main motivation behind the introduction of SPARK is the limitations of MapReduce to process large datasets. Data scientists used MapReduce to analyze huge datasets or clusters and this programming model immensely simplified big data analysis. As they had to analyze large clusters they needed to perform complex applications which meant they had to use multiple MapReduce stages and interactive data mining. But this made the system slow because to perform complex computations and interactive queries, data sharing is required which is done is a sequential way similar to a distributed file system (DFS).

Now for example if we run a iterative algorithm on Hadoop distributed file system (HDFS), each time a MapReduce stage has been completed we need to write this to HDFS (intermediary storage) and then proceed to the next stage. This reduces the efficiency, does not provide fault tolerance and does not allow us to reuse data until all the MapReduce stages have been processed.

So from the article we can say that there goal is to provide in-memory data sharing which allows faster performance of computations on these MapReduce up to 20X times. The challenge here is to provide both fault-tolerant and efficient (i.e. high speed) distributed file systems. To achieve this goal the concept of RDDs was introduced. RDDs are partitioned data structures which are distributed across the clusters and can be built from coarse grained transformations (i.e. these can be applied to all the dataset elements at once) like map, filter and join. We can say that the most powerful aspect of an RDD is how we can recover data using lineage which means that each RDD is built using few operations and if we lose data then we can recover this missing dataset easily by performing these operations and do not need to create the entire RDD again.

Another advantage of using RDD is that we can use parallel algorithms which mean that since they are parallel algorithms they are applied on the operations rather than the datasets. So the datasets are in-memory and can be used by other processes or RDDs.

SPARK uses the RDDs in its abstraction and provides an API in SCALA programming language. The operations used in SPARK are transformations (to create RDDs) and actions (to perform computations and get results). Each RDD in Spark is represented through five interfaces: set of partitions, set of dependencies, operations for performing computations on a dataset, metadata about partitioning scheme and the data placement. RDD implementations in Spark are done through HDFS files, maps, joins, union and sample.

Spark is considered as a computational engine which follows a tight integration of scheduling, distributing and monitoring applications which perform many complex and iterative computations which are distributed across many machines or clusters. Spark consists of the following components: Spark Core, Spark SQL, Spark Streaming, MLib, GraphX and Cluster Managers (known as Standalone Scheduler).

**Conclusion:**

From the discussion here we can say that RDDs are the most efficient in cluster computing to provide a fault-tolerant abstraction for data sharing. RDDs can be used in parallel applications where iterative algorithms and interactive querying is required. Implementation of RDDs in Spark makes it faster in iterative computations (by 20X times faster than done in Hadoop) and also be used for interactive querying of large datasets (up to hundreds of GBs). Data Scientists and Engineers use Spark to do interactive data analysis on large data sizes.