So far, I’ve learned useful things about Spark during these past 3 assignments, which has made me want to learn more about it. Even though I know we had just seen the basics of this tool, it’s been easy to follow up. In contrast, I have no experience with MapReduce, however it seems to be a data processing tool which is used to process the data parallelly in a distributed form (based on Java).

Differences:

* Programming language: I’ve seen that spark has a more user-friendly API and it also supports many programming languages like Python, SQL (currently using both), Scala, R, and Java. MapReduce is a tool based on Java and even though it might support other programming languages like C++, it’s always best to use Java to write MapReduce jobs. This can create some limitations while working with MapReduce.
* Fault tolerance: RDDs (Spark) offer better fault tolerance than MapReduce’s HDFS.
* Colab: It looks like both tools can be used in Google Colab.
* Processing and Speed: Spark is way faster than MapReduce. MapReduce is suitable for batch processing and linear data processing, but it lacks real-time processing abilities. Spark is ideal for real-time processing and processing live unstructured data streams.
* Machine learning applications: Spark is better in this field because it includes MLlib and tools for regression, classification, pipeline constructions, evaluation etc.

Even though it might seem that Spark may have the upper hand in use and flexibility, it all depends on the task on hand. I’m sure MapReduce can be much more useful in some scenarios than Spark like big batch-processing task.

Sources:

<https://www.javatpoint.com/mapreduce>

<https://www.integrate.io/blog/apache-spark-vs-hadoop-mapreduce/#:~:text=Processing%20speed%3A%20Apache%20Spark%20is,data%20processing%20and%20iterative%20analytics>.

<https://www.quora.com/Can-I-run-MapReduce-jobs-using-various-programming-languages>

SparkContext is an entry point to Spark, and it is used to programmatically create RDD accumulators and broadcast variables on the cluster. Its object “sc” is default available in spark-shell and can be programmatically created using the class SparkContext. Most of the operations/methods or functions used in Spark comes from SparkContext for example accumluators, broadcast variables, parallelize. It is more focused on low level operations than SparkSession.

SparkSession is also an entry point to underlying Spark functionality to programmatically create RDD, DataFrame and DataSet. It’s object “spark” is default available in spark-shell and it can be created programmatically using SparkSession builder pattern. We can also replace SparkSession with SQLContext and HiveContext. This is more related to high level data manipulation and access.

SparkContext:

* is an entry point to Spark, and it is used to programmatically create RDD accumulators and broadcast variables on the cluster.
* Connection to a Spark cluster.
* Sc is the default object available in spark-shell and can be programmatically created using SparkContext class.
* It’s used to create RDDs and performs low level operations on them.
* Most of the operations/methods or functions used in Spark comes from SparkContext for example accumluators, broadcast variables, parallelize.

SparkSession:

* Entry point to underlying Spark functionality to programmatically create RDD, DataFrame and DataSet. Contrary to SparkContext, it performs higher level operations/functionalities on the above examples.
* It provides a more user-friendly API in comparison to SparkContext.
* you can integrate SparkContext, SQL Context, Streaming Context, Hive Context, NoSQL databases, and different files formats etc.
* it also supports SQL language allowing you to perform SQL queries on Datasets and DataFrames.

Hive advantages:

* HiveContext in PySpark provides powerful SQL-like interface for working with data stored in Hive. This is beneficial to individuals who use SQL in a daily basis without having to learn complex concepts.
* Allows you to easily query and analyze large datasets, and it provides a way to write data back to Hive tables.
* HiveContext is used by spark to enhance the query parsing and accessing to existing Hive tables, and even to persist your result DataFrames / Tables.
* Hive Metastore integratation. This metastore contains all the metadata about the data and tables in the EMR cluster, which allows for easy data analysis.
* Hive provides a more robust SQL parser.

Hive disadvantages:

* HiveContext does not offer real-time queries (It was design for batch processing). The batch behavior is that all inputs must be ready by map before the reduce job starts, which makes MapReduce unsuitable for online and stream processing use cases.
* Compared to relational databases, it does provide limited subquery support.
* It is not designed for the Online transaction processing (read/write operations on individual records).
* Has very high latency (longer delays) due to the batch processing nature.

<http://www.freshers.in/article/spark/pyspark-hivecontext-in-pyspark-a-brief-explanation/>

<https://stackoverflow.com/questions/36188516/spark-sql-query-execution-on-hive>

<https://aws.amazon.com/big-data/what-is-hive/#:~:text=The%20Hive%20metastore%20contains%20all,in%20the%20S3%20data%20lake>.

<https://stackoverflow.com/questions/33666545/what-is-the-difference-between-apache-spark-sqlcontext-vs-hivecontext>

<https://subscription.packtpub.com/book/web-development/9781783558575/1/ch01lvl1sec11/batch-real-time-and-stream-processing>

RDD Advantages:

* RDDs track data lineage information to recover lost data, automatically on failure (fault tolerance).
* RDD can handle structured and unstructured data easily and effectively.
* It offers a lot of flexibility thanks to operators such as map, reduce, and shuffle. These operators allow us to perform different transformations against our data.
* The above operators also offer a lot of performance that might be better than other high-level APIs.
* Even though it can handle both data structures, RDD is preferable on unstructured data, to be used for low-level transformations and actions.

RDD Disadvantages:

* Data does not get loaded in an RDD even if you define it. Transformations are computed when you call action, such as count or collect, or save the output to a file system.
* Data stored in an RDD is in the read-only mode, you cannot edit the data which is present in the RDD.
* RDDs are slow (specifally within PySpark) because whenever a PySpark program is executed using RDDs, there is a potentially large overhead to execute the job.
* RDD does not know the information of the stored data, so the structure of the data is a black box which requires a user to write a very specific aggregation function to complete an execution.
* The above RDDs operators cannot be re-used.
* RDDs take a lot of time to master, and we can potentially right inefficiently code.

<https://intellipaat.com/blog/tutorial/spark-tutorial/programming-with-rdds/?US>

<https://subscription.packtpub.com/book/data/9781788835367/2/ch02lvl1sec25/pitfalls-of-using-rdds>

<https://www.wisewithdata.com/2020/05/rdds-vs-dataframes-vs-datasets-the-three-data-structures-of-spark/#:~:text=RDD%20can%20handle%20structured%20and,operators%20cannot%20be%20re%2Dused>.