Assignment #1 - Spark Paper

Read the following paper and answer the following questions:

Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing

[https://www.usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf (Links to an external site.) (Links to an external site.)](https://www.usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf)

Questions:

1. In your own words, describe RDD
2. What are the main use cases that Spark was designed for?
3. What are the differences between transformations and actions and what are the advantages of having transformations lazily evaluated?
4. How does RDD handle fault tolerance?
5. What are the different options for storage of persistent RDDs?
6. What are the differences between narrow and wide dependencies?
7. Compare and contrast between RDD and MapReduce?
8. Describe a concept or idea that you really like about Spark and it hasn’t been asked from any of the above questions?

Write your answers on a text file and attach it to your assignment submission.

1. In your own words, describe RDD

- RDD stands for Resilient Distributed Dataset and it is a data structure used in the Apache Spark Analytics Engine. RDDs improve upon two issues in big data computing: iterative operations and interactive operations. It does this by keeping data in memory yet still providing fault tolerance. RDDs utilize coarse-grained transformations like map, filter, and join and logs each of these transformations to compute a dataset creating a lineage that can be traced back and recomputed quickly is data is lost.

2. What are the main use cases that Spark was designed for?

- Spark was created by the graduate student Matei Zaharia during his time at UC Berkeley. Initially he developed the system for research purposes and for several companies. Matei designed the system so with a programming language interface in mind so that computations can be done at incredible speeds for in-memory big data. Spark leverages distributed memory and reuses intermediate results across multiple computations without storing it in a distributed file system that MapReduce jobs would do. This allows Spark to minimize execution times due to data replication, disk input/output, and serialization.

3. What are the differences between transformations and actions and what are the advantages of having transformations lazily evaluated?

- In Spark, transformations are lazy operations that model a new Resilient Distributed Dataset, these operations include map, filter, join, and sort. Actions launch a computation to return a value to the program or write data to an external storage, these operations include count, collect, reduce, and save. When a transformation is performed in Spark, it is lazily evaluated which means it will not be executed until an action is performed. Spark holds these transformation operations in a Directed Acyclic Graph (DAG) classifies them into specific stages. Spark will then formulate a plan and execute these stages in order to optimize efficiency and throughput.

4. How does RDD handle fault tolerance?

- In Spark, RDDs provide fault tolerance by utilizing an interface based on coarse-grained transformations including map, filter, and join which apply operation to multiple data items. Spark holds these transformations in a Directed Acyclic Graph (DAG) which is called a lineage. If a partition of a RDD becomes lost or corrupted, the RDD will be able to rebuild the dataset through the lineage and restore the partition through computation of that partition instead of replication of the dataset across one or more machines.

5. What are the different options for storage of persistent RDDs?

- Spark has three options to store persistent RDDs: in-memory storage as deserialized Java objects, in-memory storage as serialized data, and on-disk storage. In-memory storage as deserialized Java objects provides the fastest performance because the Java VM can access each RDD element natively. In-memory storage as serialized data has lower performance but allows users to choose a more memory-efficient representation than Java object graphs when space is limited. On-disk storage are useful for RDDs that are too large to keep in RAM but costly to recompute on each use.

6. What are the differences between narrow and wide dependencies?

- A dependency in an RDD is a function that computes the dataset based on its parents. A narrow dependency is where each partition of the parent RDD is used by at most one partition of the child RDD, a 1-to-1 mapping between parent and child. A wide dependency is where each partition of the parent RDD is used by one or more partition of the child RDD, a 1-to-many mapping between the parent and child. In narrow dependencies pipelined executions more efficiently, which makes computation between parent and child partitions are straightforward. In wide dependencies data is required from all parent partitions and the data is shuffled between nodes, which makes computation upstream and downstream more complicated.

7. Compare and contrast between RDD and MapReduce?

- RDD implements lazy evaluation computations to reuse intermediate results across multiple computations in a lineage. MapReduce does not have this feature and is slower because of it, it has to write to an external storage system to reuse computations. At the high level, Spark is great for fast iterative processing and graph processing compared to MapReduce however, MapReduce is better for linear processing of much larger datasets.  
  
  
8. Describe a concept or idea that you really like about Spark and it hasn’t been asked from any of the above questions?

- As someone who is new to this, what does Spark provide to helps a new user get started on processing datasets. What makes it an easier analytics framework to use compared to Hadoop MapReduce and other big data engines?

Answer: Hadoop MapReduce requires a higher level of Java programming skills to use efficiently. While Apache Spark uses Scala and is easier because of a built-in interactive mode. Spark also provides richer APIs compared to MapReduce. On top this Spark+Scala is easier to write and debug and it also contains less lines of code which mutually help each other (writing/debugging).