Tutorial 8: Spark Framework

Question Set:

Q1. What is Apache Spark (and how does it compare with Apache Hadoop [optional])?

ANSWER:

Apache Spark is an open-source distributed general-purpose cluster-computing framework. It provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. Originally developed at the University of California, Berkeley's AMPLab. Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. [1]

Apache Spark VS. Apache Hadoop

Apache Spark is an open-source, in-memory computing framework that is designed to enhance computational speed. Apace Hadoop, on the other hand, reads and writes from the disk, as a result, it slows down the computation. While Spark can run on top of Hadoop and provides a better computational speed solution [2].

	Spark	**Hadoop**
Speed	100x times faster than Hadoop	Faster than the traditional system
Language	Scala	Java
Data Processing	Batch/Real-time/iterative/interactive/Graph	Batch processing
Ease of Use	Compact and easier than Hadoop	Complex and lengthy
Caching	Caches the data in memory and enhance the system performance	N/A.

Q2. What is Resilient Distributed Data (RDD)?

ANSWER:

Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark. It is an immutable distributed collection of objects. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster.

Formally, an RDD is a read-only, partitioned collection of records. RDDs can be created through deterministic operations on either data on stable storage or other RDDs. RDD is a fault-tolerant collection of elements that can be operated on in parallel.

There are two ways to create RDDs:

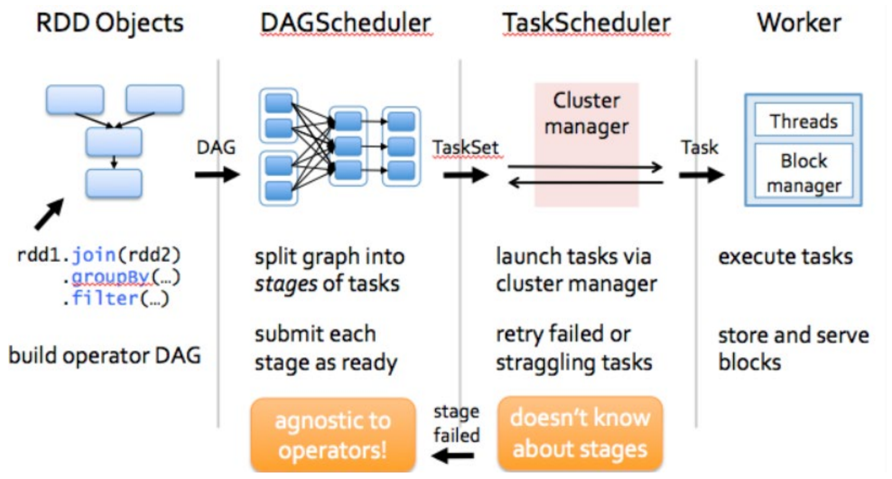
- Parallelizing an existing collection in your driver program.
- Referencing a dataset in an external storage system.
- Transforming a new RDD from an existing RDD.

Spark makes use of the concept of RDD to achieve faster and more efficient MapReduce operations.

Q3. Please discuss how Spark works.

ANSWER:

Spark is a small codebase, and the system is divided into various layers. Each layer has some responsibilities. The layers are independent of each other.



1. The first layer is the **interpreter**, Spark uses a Scala interpreter, with some modifications.
2. As you enter your code in the Spark console (creating RDD objects and applying operators), Spark creates an operator graph.
3. When the user runs an action (like collect), the Graph is submitted to a **DAG Scheduler**. The DAG scheduler divides the operator graph into (map and reduce) stages.
4. A stage is comprised of tasks based on partitions of the input data. The DAG scheduler pipelines operators together to optimize the graph. E.g., Many map operators can be scheduled in a single stage. This optimization is key to Spark's performance. The final result of a DAG scheduler is a set of stages.
5. The stages are passed on to the **Task Scheduler**. The task scheduler launches tasks via the **cluster manager**. (Spark Standalone/Yarn/Mesos). The task scheduler doesn't know about dependencies among stages.
6. The **Worker** executes the tasks. A new JVM is started per job. The worker knows only about the code that is passed to it.

- **Job**: A piece of code that reads some input from HDFS or local, performs some computation on the data and writes some output data.
- **Stages**: Jobs are divided into stages. Stages are classified as a Map or reduced stages (It's easier to understand if you have worked on Hadoop and want to correlate). Stages are divided based on computational boundaries; all computations (operators) cannot be updated in a single Stage. It happens over many stages.
- **Tasks**: Each stage has some tasks, one task per partition. One task is executed on one partition of data on one executor (machine).
- **DAG**: DAG stands for Directed Acyclic Graph, in the present context it's a DAG of operators.
- Executor: The process responsible for executing a task.
- **Driver**: The program/process responsible for running the Job over the Spark Engine

- **Master**: The machine on which the Driver program runs
- **Slave**: The machine on which the Executor program runs

References

[1] Apache Spark, "Spark 2.4.3 released", https://spark.apache.org/.
[2] Dataflair, Apache Spark vs Hadoop MapReduce – Feature Wise Comparison, https://data-flair.training/blogs/spark-vs-hadoop-mapreduce/
[3] Arush Kharbanda, Apache Spark A Look under the Hood, https://www.sigmoid.com/apache-spark-internals/