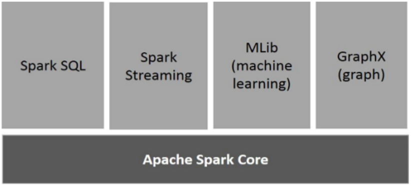
Apache Spark Fundamentals

Introduction

- Spark is an open-source cluster-computing framework. Originally developed at the *UC Berkeley* in 2009, it was later donated to the *Apache Software Foundation*.
- Spark provides
  - Unified computing engine (Spark Core)
  - Set of APIs for data analysis, usable with Scala, Java, Python, R: Spark SQL (structured data) MLlib (machine learning), GraphX (graph analytics), Spark Streaming (streaming analytics). Spark is written in Scala.



Spark SQL

Spark Streaming

MLib (machine learning)

GraphX (graph)

Apache Spark Core

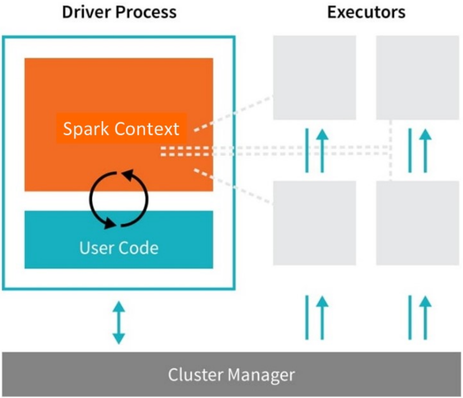
Introduction (cont'd)

- Spark does not come with a storage system (unlike Hadoop) but can run on the *Hadoop Distributed File System (HDFS)* as well as on other systems (e.g., HIVE or Relational DBMS).
- Spark's features:
  - Fault tolerance.
  - In-memory caching, which enables efficient execution of iterative algorithms, with a substantial performance improvement w.r.t. Hadoop.
- Spark can run:
  - On a single machine in the so called local mode. This is what we do in Homeworks 1, 2, and 3.
  - On a cluster managed by a cluster manager such as Spark's Standalone, YARN, Mesos. For Homework 4 we will use the YARN cluster manager on CloudVeneto.

Observations

- A single machine does not have enough power and resources to process big data efficiently.
- A cluster is a group of machines whose power and resources are pooled to provide one powerful execution platform.
- Spark can be viewed as a tool for managing and coordinating the execution of (big data) jobs on a cluster. *Without careful management and coordination the potential added power coming from the combination of individual machines is likely to be wasted.*

Spark Application



Spark Application (cont'd)

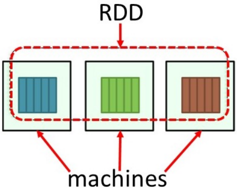
- The driver process runs the main() function and sits on a node in the cluster. It is the heart of the application and is responsible for
  - maintaining information about the application;
  - responding to a users program or input;
  - analyzing, distributing, and scheduling work across the executors.
- The driver process is represented by an object called Spark Context which can be regarded as a channel to access all Spark functionalities. (Starting from Spark 2.0.0, a Spark Session object has been introduced which encapsulates a Spark Context object, providing a wider spectrum of functionalities.)

Spark Application (cont'd)

- The executor processes are responsible for actually executing the work that the driver assigns them. Each executor is responsible for:
  - executing code assigned to it by the driver;
  - reporting the state of its computation back to the driver.
- The cluster manager controls physical machines and allocates resources to applications.
- The driver and executors are simply processes which can live on the same machine or on different machines. In local mode, both run (as threads) on one machine instead of a cluster.
- While executors, for the most part, run Scala code, the driver can be *driven* from different languages through Spark's APIs.

Resilient Distributed Dataset (RDD)

- Fundamental abstraction in Spark. An RDD is a collection of elements of the same type, partitioned and distributed across (possibly) several machines.



- An RDD provides an interface based on coarse-grained transformations (e.g., map, reduce, filter, etc.).
- RDDs ensure fault-tolerance and allow users to cache intermediate data and to control the partitioning.

Resilient Distributed Dataset (cont'd)

- A key ingredient behind the efficiency of Spark is data partitioning: namely, each RDD is broken into chunks called partitions which are distributed among the available machines.
- The number of partitions is typically 2x/3x the number of cores.
- Partitioning is achieved through a partitioner function $p(\cdot)$. If P partitions are desired, an element x is assigned to partition $p(x) \bmod P$. The default partitioner is a hash code.
- Partitioning enables:
  - Data reuse. In iterative (e.g., multi-round) applicatons data are kept in the executors' main memories as much as possible to avoid expensive accesses to secondary storage or shuffles.
  - Parallelism. Some data transformations are applied independently in each partition avoiding expensive data movements.

Resilient Distributed Dataset (cont'd)

- RDDs are immutable (i.e., read-only) and can be created either from data in stable storage (e.g., HDFS) or from other RDDs, through transformations.
- RDDs need not be materialized at all times. Each RDDs maintains enough information regarding the sequence of transformations that generated it (its lineage), which enable the recomputation of its partitions from data in stable storage. In other words, an RDD can always be recostrucuted after a failure (unless the failure affects the stable storage).
- Programmers can control the caching of RDDs.

Operations of RDDs

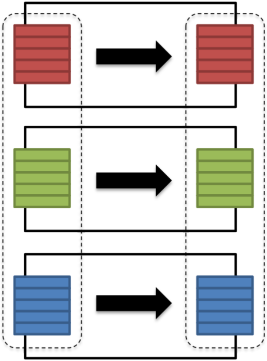
The following types of operations can be performed on an RDD A

- Transformations. A transformation generates a new RDD B starting from the data in A. We distinguish between:
  - Narrow transformations. Each partition of A cantributes to at most one partition of B. Hence, no shuffling of data across machines is needed ($\Rightarrow$ maximum parallelism).
  - Wide transformations. Each partition of A may contribute to many partitions of B. Hence, shuffling of data across machines may be required.
- Actions. An action launches a computation on the data in A which returns a value to the application or exports data to the storage system. It is at this point that the RDD A is acutally materialized (lazy evaluation).

Operations of RDDs (cont'd)

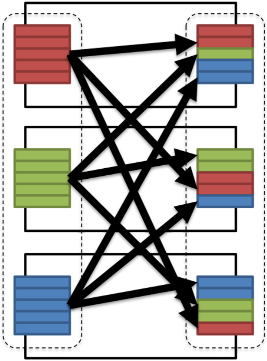
Narrow transformation

- Input and output stays in same partition
- No data movement is needed



Wide transformation

- Input from other partitions are required
- Data shuffling is needed before processing



Examples of theory questions

- What does the driver process do in a Spark Application?
- Briefly describe the two types of operations on RDDs which can be performed in Spark.

References

AS-1 Spark's Web Site: `spark.apache.org`

AS-2 Spark's RDD Programming guide: `spark.apache.org/docs/latest/rdd-programming-guide.html`

CZ18 A Gentle Introduction to Apache Spark. From B. Chambers, M. Zaharia. *Spark: The Definite Guide*, Databricks 2018.

Z+12 M. Zaharia et al. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. NSDI 2012: 15-28