Ciência de Dados em Larga Escala

Inês Dutra and Zafeiris Kokkinogenis

DCC-FCUP
room 1.31
ines@dcc.fc.up.pt
zafeiris.kokkinogenis@gmail.com

23/24

Apache Spark 2.0

(Based on book Learning Spark 2.0)

- unified engine for large-scale distributed data processing
- on-premises in data centers or in the cloud
- provides in-memory storage for intermediate computations (faster than Hadoop)
- incorporates libraries with composable APIs for machine learning (MLlib)
- SQL for interactive queries (Spark SQL)
- stream processing (Structured Streaming)
- graph processing (GraphX)

Apache Spark history

- Both Hadoop and Spark are distributed systems that let you process data at scale. They can recover from failure if data processing is interrupted for any reason.
- To store, manage, and process big data, Apache Hadoop separates datasets into smaller subsets or partitions. It then stores the partitions over a distributed network of servers. Likewise, Apache Spark processes and analyzes big data over distributed nodes to provide business insights.
- Apache Spark relies on a special data processing technology called Resilient Distributed Dataset (RDD). With RDD, Apache Spark remembers how it retrieves specific information from storage and can reconstruct the data if the underlying storage fails.

Spark components

Apache Spark runs with the following components:

- Spark Core coordinates the basic functions of Apache Spark. These functions include memory management, data storage, task scheduling, and data processing.
- Spark SQL allows you to process data in Spark's distributed storage.
- Spark Streaming and Structured Streaming allow Spark to stream data efficiently in real time by separating data into tiny continuous blocks.
- Machine Learning Library (MLlib) provides several machine learning algorithms that you can apply to big data. GraphX allows you to visualize and analyze data with graphs.

YARN

- Apache Hadoop YARN is the resource management and job scheduling technology in the open source Hadoop distributed processing framework. One of Apache Hadoop's core components, YARN is responsible for allocating system resources to the various applications running in a Hadoop cluster and scheduling tasks to be executed on different cluster nodes.
- YARN stands for Yet Another Resource Negotiator, but it's commonly referred to by the acronym alone; the full name was self-deprecating humor on the part of its developers. The technology became an Apache Hadoop subproject within the Apache Software Foundation (ASF) in 2012 and was one of the key features added in Hadoop 2.0, which was released for testing that year and became generally available in October 2013.

YARN

- The addition of YARN significantly expanded Hadoop's potential uses. The original incarnation of Hadoop closely paired the Hadoop Distributed File System (HDFS) with the batch-oriented MapReduce programming framework and processing engine, which also functioned as the big data platform's resource manager and job scheduler. As a result, Hadoop 1.0 systems could only run MapReduce applications – a limitation that Hadoop YARN eliminated.
- Before getting its official name, YARN was informally called MapReduce 2 or NextGen MapReduce. But it introduced a new approach that decoupled cluster resource management and scheduling from MapReduce's data processing component, enabling Hadoop to support varied types of processing and a broader array of applications. For example, Hadoop clusters can now run interactive querying, streaming data and real-time analytics applications on Apache Spark and other processing engines simultaneously with MapReduce batch jobs.

Apache Spark main characteristics

- speed
- ease of use
- modularity
- extensibility

Apache Spark components and architecture

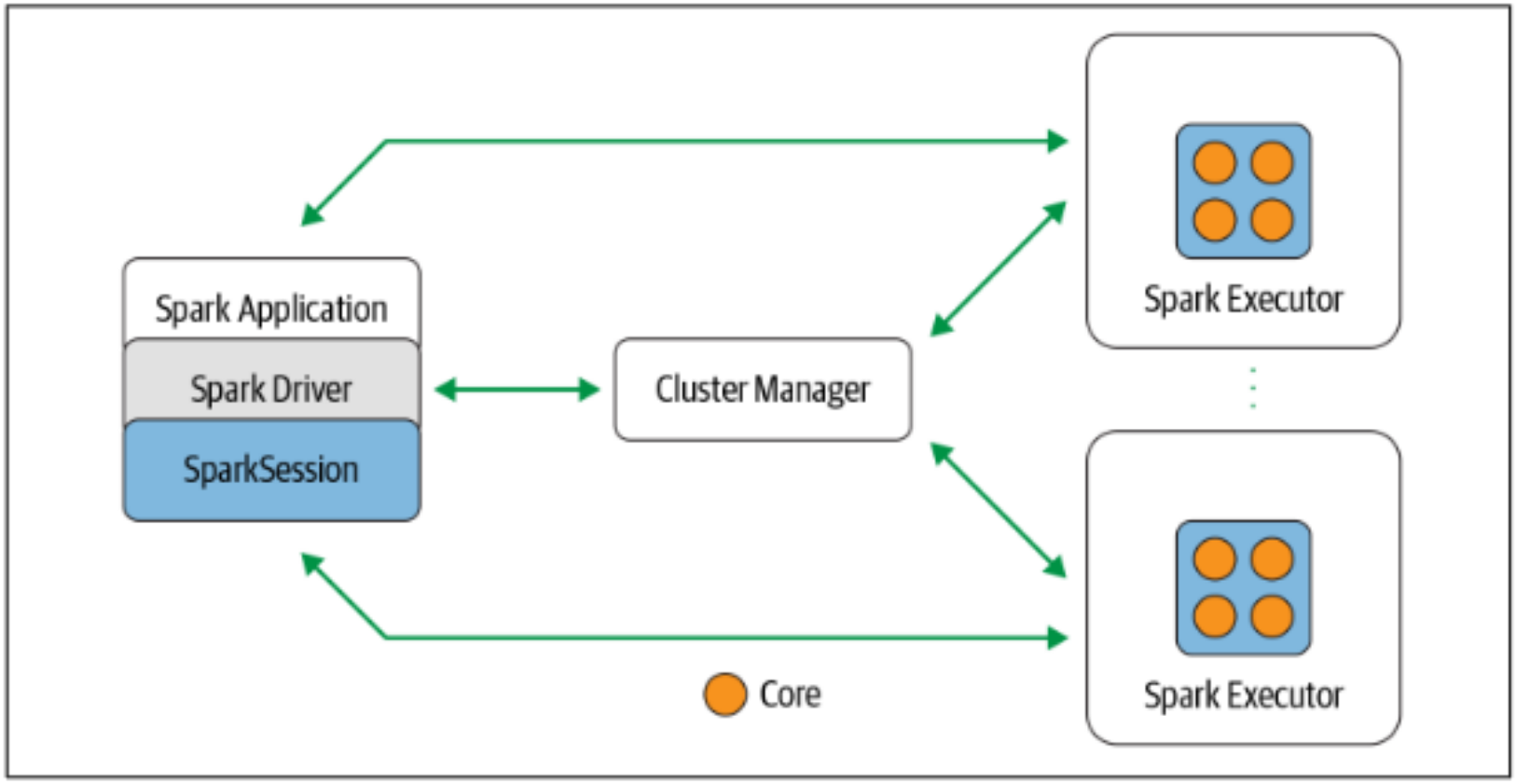

Figure 1-4. Apache Spark components and architecture

Apache Spark components and architecture

- Spark application: driver program responsible for orchestrating parallel operations on the Spark cluster
- Spark driver: responsible for instantiating a SparkSession, with roles:
  - communicates with the cluster manager
  - requests resources (CPU, memory, etc.) from the cluster manager for Spark's executors (JVMs)
  - transforms all the Spark operations into DAG computations
  - schedules them, and distributes their execution as tasks across the Spark executors
  - once the resources are allocated, it communicates directly with the executors

Apache Spark components and architecture

- Spark session: unifies all Spark operations and data
- Aggregates previous Spark 1.0 versions of entry points such as SparkContext, SQLContext, HiveContext, SparkConf and SreamingContext
- User can:
  - create JVM runtime parameters
  - define DataFrames and Datasets
  - read from data sources
  - access catalog metadata
  - issue Spark SQL queries

Apache Spark components and architecture

Spark session example in Scala

```scala
// In Scala
import org.apache.spark.sql.SparkSession

// Build SparkSession
val spark = SparkSession
  .builder
  .appName("LearnSpark")
  .config("spark.sql.shuffle.partitions", 6)
  .getOrCreate()
...
// Use the session to read JSON
val people = spark.read.json("...")
...
// Use the session to issue a SQL query
val resultsDF = spark.sql("SELECT city, pop, state, zip FROM table_name")
```

Apache Spark components and architecture

- Cluster manager: responsible for managing and allocating resources for the cluster of nodes on which your Spark application runs
- Support for:
  - built-in standalone cluster manager
  - Apache Hadoop YARN
  - Apache Mesos
  - Kubernetes

Apache Spark components and architecture

- Spark executor: runs on each worker node
- communicate with the driver program
- responsible for executing tasks on the workers
  (most deployment models run only a single executor per node)

A data sharing abstraction

- Resilient Distributed Dataset (RDD): data sharing abstraction for fault-tolerant, parallel data structures
- RDD allows a user to keep intermediate results and optimizes their placement in the memory of a large cluster
- Data storage in memory significantly improves performance
  - The write bandwidth throughput for both hard disks and solid-state disks is three orders of magnitude lower than the memory bandwidth
  - Only the random access latency of solid-state disks is much lower than the latency of hard disks, their sequential I/O bandwidth is not larger
- The RDD user interface exposes:
  - Partitions, atomic pieces of the dataset.
  - Dependencies on parent RDD.
  - A function for constructing the dataset.
  - Metadata about data location.