Learning Spark

Introduction to Apache Spark

💻 Master's degree, Universitat de Lleida

/bluetab

an IBM Company

The genesis of spark

Big data and distributed computing at Google

- Google is synonymous with scale. (googol:1 plus 100 zeros!)
- Google File System (GFL), MapReduce(MR) and Bigtable
- GFS: fault-tolerant and distributed file system across many hardware servers in cluster farm
- Bigtable: scalable storage of structured GFS.
- MR introduced a new parallel programming paradigm, based on functional programming, for large-scale processing of data distributed over GFS and Bigtable.
  - MR applications, interact with the MR system that sends computation code (MR functions) to where the data resides.
  - This favours data locality and cluster affinity rather than bringing data to our applications.
  - Workers in the cluster aggregate and reduce the intermediate computations and produce a final appended output from the reduce function, which is the written to the distributed storage where it is accessible to our application.

The genesis of spark

Hadoop at Yahoo!

- The solutions expressed in Google's GFS paper, provided a blueprint for the Hadoop File System(HDFS), including MR implementation as a framework for distributed computing.
- HDFS was donated to the Apache Software Foundation (ASF) in April 2006, and it became part of the Apache Hadoop framework.
- MR framework on HDFS had few shortcomings.
  - /It was hard to manage and administer, with cumbersome operational complexity.
  - /Its general batch-processing MR API was verbose and required a lot of repetitive setup code, with fragile fault tolerance.
  - /With large batches of data jobs with many pairs of MR tasks, each par's intermediate computed result is written to the local disk for the subsequent stage of its operation. Large MR jobs could run for hours or days.
  - /Even though Hadoop MR was conductive to large-scale jobs for general batch processing, it fell short for combining other workloads such as machine learning, streaming or interactive SQL-like queries.
  - /To handle these new workloads, engineers developed a lot of systems, each with their APIs and cluster configurations, adding complexity to the learning curve for developers.

The genesis of spark

Spark's Early Years at AMPLab

- Researchers at Berkeley who previously worked with Hadoop MR starts with a project called ***spark***
- They discovered that MR was inefficient for interactive or iterative computing jobs and a complex framework to learn
- So, they embraced the idea of making Spark **simpler, faster and easier.**
- Early paper demonstrated Spark was 10 to 20 times faster than MR for certain jobs. Today is many orders of magnitude faster.
- Spark ideas:
  - /make it highly fault tolerance and parallel
  - /support in-memory storage for intermediate results between iterative and interactive map and reduce computations
  - /offer easy and composable API in multiple languages
  - /support other workloads in a unified manner
- 2013- Some of the original creators of spark, donated the project to de ASF and formed a company called Databricks.
- May 2014- Spark 1.0

What is Apache Spark?

- Apache Spark is a unified engine designed for large-scale distributed data processing on premises in data center or in the cloud.
- Spark provides
  - in memory storage for intermediate computations
  - Libraries with compasable APIs for:
    - ML (MLlib)
    - SQL for interactive queries (Spark SQL)
    - stream processing (Structured Streaming)
    - graph processing (GraphX)
- Spark design philosophy:
  - Speed
  - Easy of use
  - Modularity
  - Extensibility

What is Apache Spark?

Speed

- Spark has pursued the goal of speed in several ways:
  - its internal implementation benefits from the hardware industry's recent huge strides in improving the price and performance of CPUs and memory.
  - Spark builds its query computations as a directed acyclic graph (DAG)
    - its DAG scheduler and query optimizer construct an efficient computational graph that can usually be decomposed into tasks that are executed in parallel across workers on the cluster.
    - its physical execution engine (Tungsten) uses whole-stage code generation to generate compact code for execution.
  - With all the intermediate results retained in memory and its limited disk I/O, thus gives it a huge performance boost.

What is Apache Spark?

Ease of use

- Providing a fundamental abstraction of a simple logical data structure called Resilient Distributed Dataset (RDD), upon which all other higher-level structured data abstractions, such as DataFrames and Datasets, are constructed.
- By providing a set of *transformations* and *actions* as *operations*, Spark offers a simple programming model that we can use to build big data applications in familiar languages.

Modularity

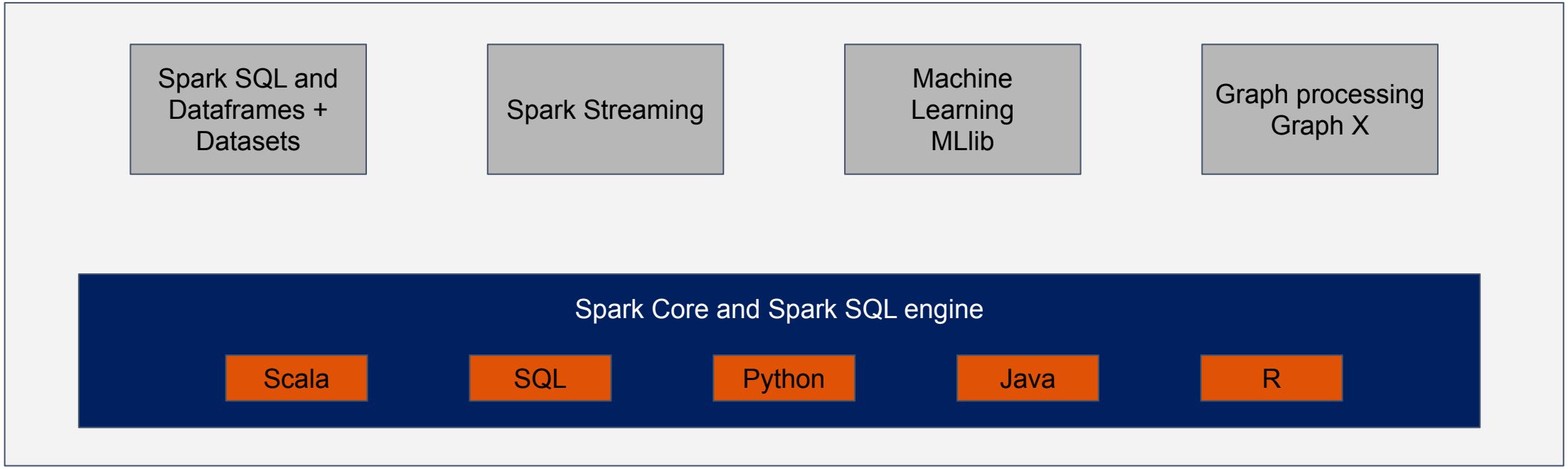
- Spark operations can be applied across many type of workloads and expressed in many languages (Scala, Java, Python, SQL and R)
- Well-documented APIs of Spark and its modules (Spark SQL, Spark Structured Streaming, Spark MLlib and GraphX)
- We can write a single Spark application that can do it all

Extensibility

- Rich ecosystem of packages for accessing third party sources (Apache Kafka, Kinesis, Azure Storage, Amazon S3…)

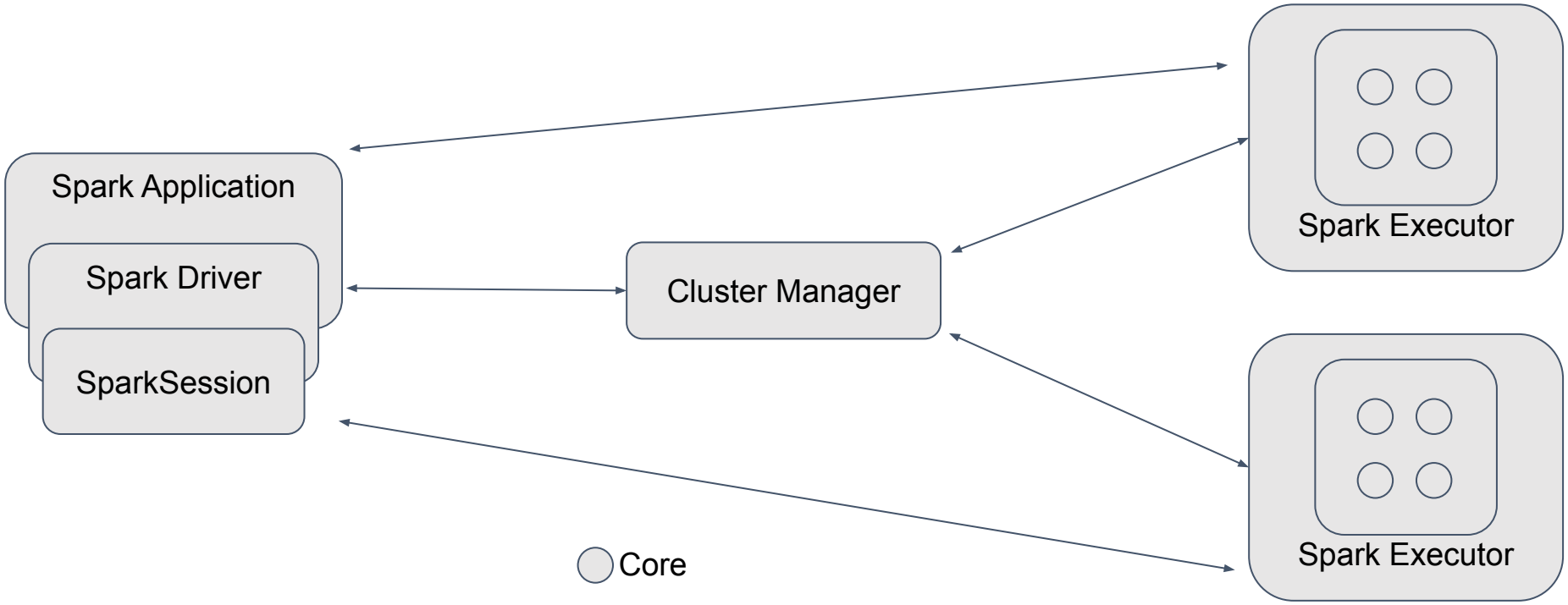
Unified Engine for Big Data Processing

Components



Unified Engine for Big Data Processing

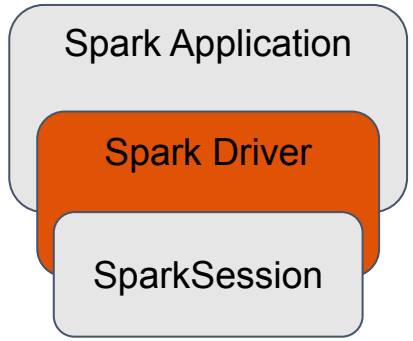
Apache Spark's distributed Execution



Unified Engine for Big Data Processing

Spark Driver

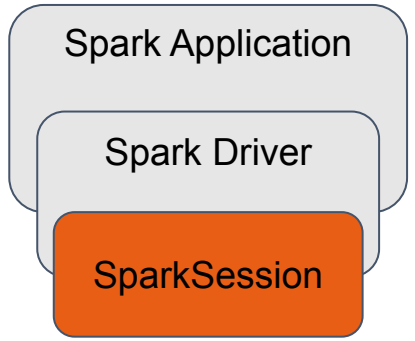
- Responsible for instantiating a SparkSession
- Has multiple roles:
  - Communicates with the cluster manager
  - Request resources (CPU, memory, etc) for the Spark's executors (JVMs)
  - Transforms all the Spark operations into DAG computations, schedule them, and distributes their execution as tasks across the Spark executors
  - Once the resources are allocated, it communicates directly with the executors



Unified Engine for Big Data Processing

SparkSession

- In Spark 2.0, the SparkSession became a unified conduit to all Spark operations and data
- Not only did it substitut previous entry points to Spark, but it also made working with Spark simpler and easier.
- Through this one conduit, you can create JVM runtime, define DataFrames and Datasets, read from data sources, access catalog metadata, issue Spark SQL queries…




```
// Create a SparkSession. No need to create SparkContext
// You automatically get it as part of the SparkSession
val warehouseLocation = "file:${system:user.dir}/spark-warehouse"
val spark = SparkSession
   .builder()
   .appName("SparkSessionZipsExample")
   .config("spark.sql.warehouse.dir", warehouseLocation)
   .enableHiveSupport()
   .getOrCreate()

//set new runtime options
spark.conf.set("spark.sql.shuffle.partitions", 6)
spark.conf.set("spark.executor.memory", "2g")
//get all settings
val configMap:Map[String, String] = spark.conf.getAll()
```

```
//fetch metadata data from the catalog
spark.catalog.listDatabases.show(false)
spark.catalog.listTables.show(false)

//create a Dataset using spark.range starting from 5 to 100, with increments of 5
val numDS = spark.range(5, 100, 5)

val zipsDF = spark.read.json(jsonFile)

zipsDF.createOrReplaceTempView("zips_table")
zipsDF.cache()
val resultsDF = spark.sql("SELECT city, pop, state, zip FROM zips_table")
```

Unified Engine for Big Data Processing

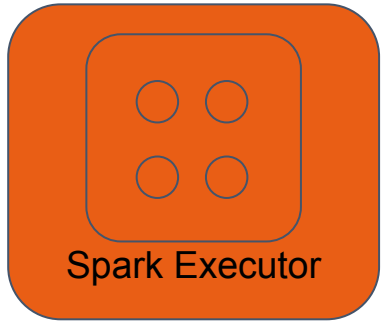
Cluster manager

- The cluster manager is responsible for managing and allocate resources for the cluster of nodes on which your Spark application runs.
- Spark supports four cluster managers:
  - the build-in standalone cluster manager
  - Apache Hadoop YARN
  - Apache Mesos
  - Kubernetes

Unified Engine for Big Data Processing

Spark executor

- A Spark executor runs on each worker node in the cluster.
- The executors communicate with the driver program and are responsible for executing tasks on the workers.
- In most deployment modes, only a single executor runs per node.



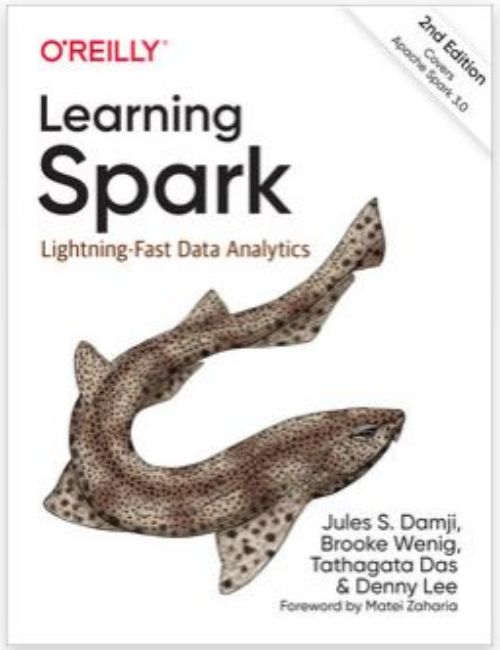
Mode	Spark driver	Spark executor	Cluster manager
Local	Runs on a single JVM, like a laptop or single node	Runs on the same JVM as the driver	Runs on the same host
Standalone	Can run on any mode in the cluster	Each node in the cluster will launch its own executor JVM	Can be allocated arbitrarily to any host in the cluster
YARN (client)	Runs on a client, not part of the cluster	YARN's NodeManager container	YARN's Resource Manager works with YARN's Application Master to allocate the containers on NodeManager for executors
YARN (cluster)	Runs with the YARN Application Master	Same as Yarn client mode	Same as YARN client mode
Kubernetes	Runs in a Kubernetes pod	Each worker runs within its own pod	Kubernetes Master

Unified Engine for Big Data Processing

Distributed data and partitions

- Physical data is distributed across storage as partitions residing in either HDFS or cloud storage.
- While the data is distributed as partitions across the physical cluster, Spark treats partition as logical data abstractions (DataFrames).
- Each Spark executor is preferably allocated a task that requires it to read the partition closest to it in the network, observing data locality (not always possible).
- Partitioning allows for efficient parallelism.
- Each executor’s core is assigned its own data partition to work on.

Bibliography



Learning Spark, 2nd Edition

by Jules S. Damji, Brooke Wenig, Tathagata Das, Denny Lee
Released July 2020
Publisher(s): O'Reilly Media, Inc.
ISBN: 9781492050049

Explore a preview version of *Learning Spark, 2nd Edition* right now.

O'Reilly members get unlimited access to live online training experiences, plus books, videos, and digital content from 200+ publishers.

/bluetab
an IBM Company

¡Gracias!

alba.lamas@bluetab.net

¡Síguenos!

https://bluetab.net/

https://www.linkedin.com/company/bluetab/