Apache Spark Architecture

dataninja Follow
Jul 4 · 8 min read ★

Photo by Lance Anderson on Unsplash

Spark & its Features

Apache Spark is an open source cluster computing framework for real-time data processing. The main feature of Apache Spark is its ***in-memory cluster computing*** that increases the processing speed of an application. Spark provides an interface for

programming entire clusters with implicit ***data parallelism and fault tolerance***. It is designed to cover a wide range of workloads such as batch applications, iterative algorithms, interactive queries, and streaming.

Features of Apache Spark:

1. **Speed —** Spark runs up to 100 times faster than Hadoop MapReduce for large-scale data processing. It is also able to achieve this speed through controlled partitioning.
2. **Powerful Caching —** Simple programming layer provides powerful caching and disk persistence capabilities.
3. **Deployment —** It can be deployed through ***Mesos, Hadoop via YARN, or Spark's own cluster manager.***
4. **Real-Time —** It offers Real-time computation & low latency because of ***in-memory computation.***
5. **Polyglot —** Spark provides high-level APIs in Java, Scala, Python, and R. Spark code can be written in any of these four languages. It also provides a shell in Scala and Python.

Major components of Apache Spark's distributed architecture.

Spark Architecture Overview

Apache Spark has a well-defined layer architecture which is designed on two main abstractions:

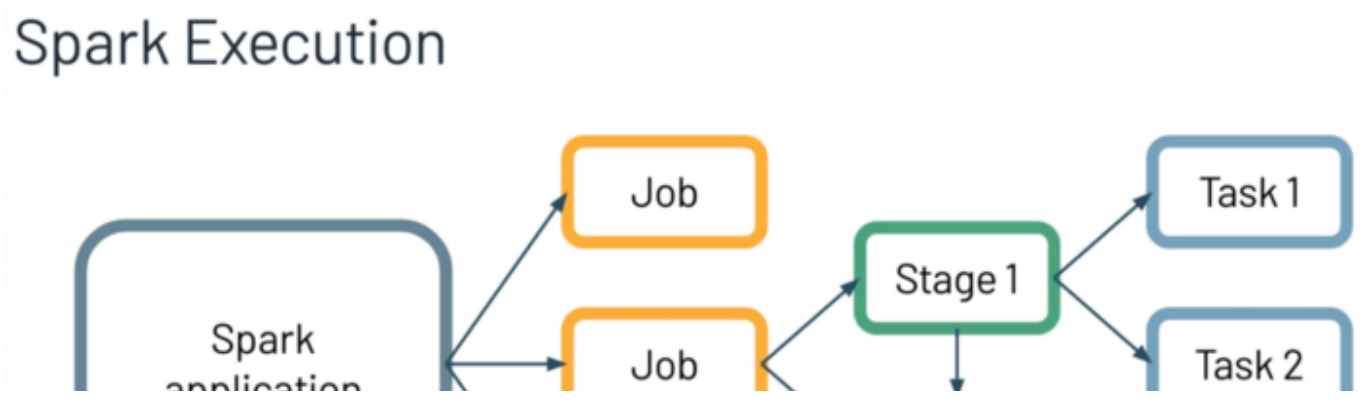
- **Resilient Distributed Dataset (RDD)**: RDD is an immutable (read-only), fundamental collection of elements or items that can be operated on many devices at the same time (parallel processing). Each dataset in an RDD can be divided into logical portions, which are then executed on different nodes of a cluster.
- **Directed Acyclic Graph (DAG)**: DAG is the scheduling layer of the Apache Spark architecture that implements stage-oriented scheduling. Compared to MapReduce that creates a graph in two stages, Map and Reduce, Apache Spark can create DAGs that contain many stages.

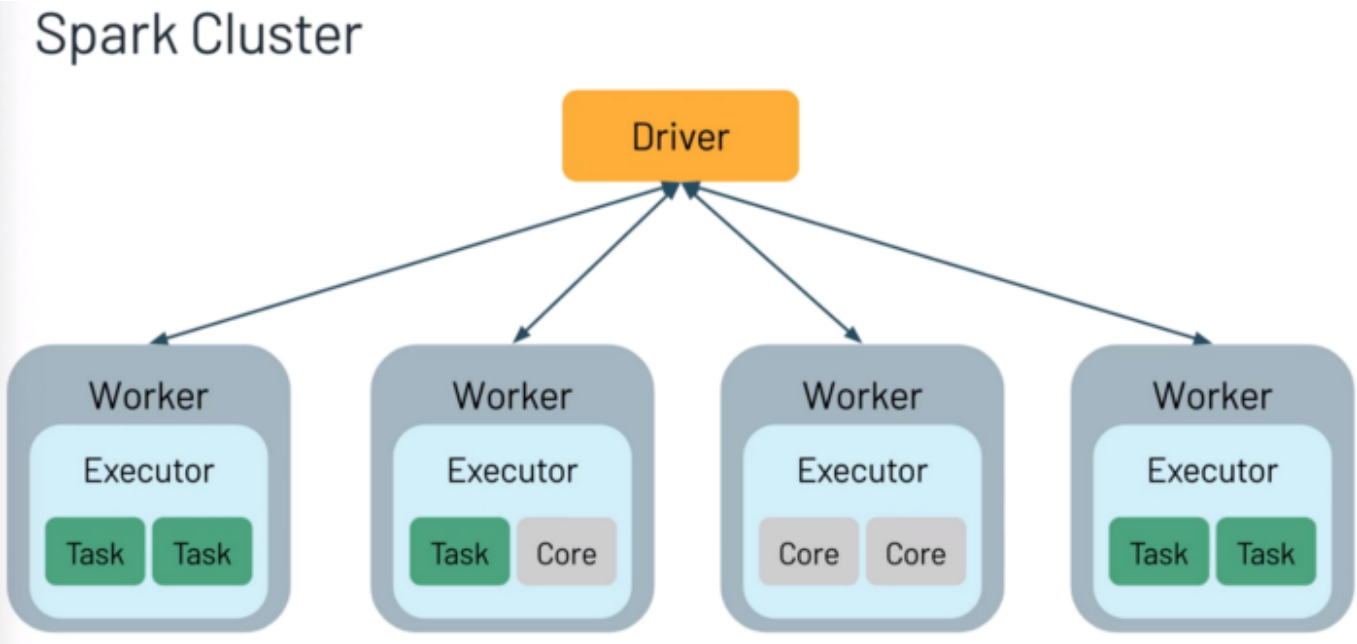
Fig: Spark Architecture

Fig: Spark Architecture

- **Driver Program** in the Apache Spark architecture calls the main program of an application and creates **SparkContext**. A **SparkContext** consists of all the basic functionalities. **Spark Driver** contains various other components such as **DAG Scheduler, Task Scheduler, Backend Scheduler, and Block Manager**, which are responsible for *translating the user-written code into jobs that are actually executed on the cluster*.
- **Spark Driver** and **SparkContext** collectively watch over the job execution within the cluster. **Spark Driver** works with the **Cluster Manager** to manage various other jobs. **Cluster Manager** does the resource allocating work. And then, the job is split into multiple smaller tasks which are further distributed to worker nodes.
- Whenever an **RDD** is created in the **SparkContext**, it can be distributed across many worker nodes and can also be cached there.
- Worker nodes execute the tasks assigned by the **Cluster Manager** and return it back to the **Spark Context**.
- An **executor** is responsible for the **execution** of these tasks. The lifetime of executors is the same as that of the Spark Application. If we want to increase the performance of the system, we can increase the number of workers so that the jobs can be divided into more logical portions.

> *The secret to Spark's performance is parallelism*





Spark interfaces

There are three key Spark interfaces that you should know about.

Resilient Distributed Dataset (RDD)

Apache Spark's first abstraction was the RDD. It is an interface to a sequence of data objects that consist of one or more types that are located across a collection of machines (a cluster). RDDs can be created in a variety of ways and are the "lowest level" API available. While this is the original data structure for Apache Spark, you should focus on the DataFrame API, which is a superset of the RDD functionality. The RDD API is available in the Java, Python, and Scala languages.

DataFrame

These are similar in concept to the DataFrame you may be familiar with in the pandas Python library and the R language. The DataFrame API is available in the Java, Python, R, and Scala languages.

Dataset
A combination of DataFrame and RDD. It provides the typed interface that is available in RDDs while providing the convenience of the DataFrame. The Dataset API is available in the Java and Scala languages.

> *Databricks datasets*
>
> *Databricks includes a variety of datasets within the Workspace that you can use to learn Spark or test out algorithms. You'll see these throughout the getting started guide. The datasets are available in the* `/databricks-datasets` *folder.*

In many scenarios, especially with the performance optimizations embedded in DataFrames and Datasets, it will not be necessary to work with RDDs. But it is important to understand the RDD abstraction because:

- The RDD is the underlying infrastructure that allows Spark to run so fast and provide data lineage.
- If you are diving into more advanced components of Spark, it may be necessary to use RDDs.
- The visualizations within the Spark UI reference RDDs.

When you develop Spark applications, you typically use DataFrames and Datasets.

Spark Eco-System

As you can see from the below image, the spark ecosystem is composed of various components like **Spark SQL, Spark Streaming, MLlib, GraphX**, and **the Core API component.**

1. **Spark Core**
   Spark Core is the base engine for large-scale parallel and distributed data processing. Further, additional libraries which are built on the top of the core allows diverse workloads for streaming, SQL, and machine learning. It is responsible for memory management and fault recovery, scheduling, distributing and monitoring jobs on a cluster & interacting with storage systems. Spark Core API is available in R, SQL, Python, Scala and Java.

2. **Spark Streaming**
   Spark Streaming is the component of Spark which is used to process real-time streaming data. Thus, it is a useful addition to the core Spark API. It enables high-throughput and fault-tolerant stream processing of live data streams.

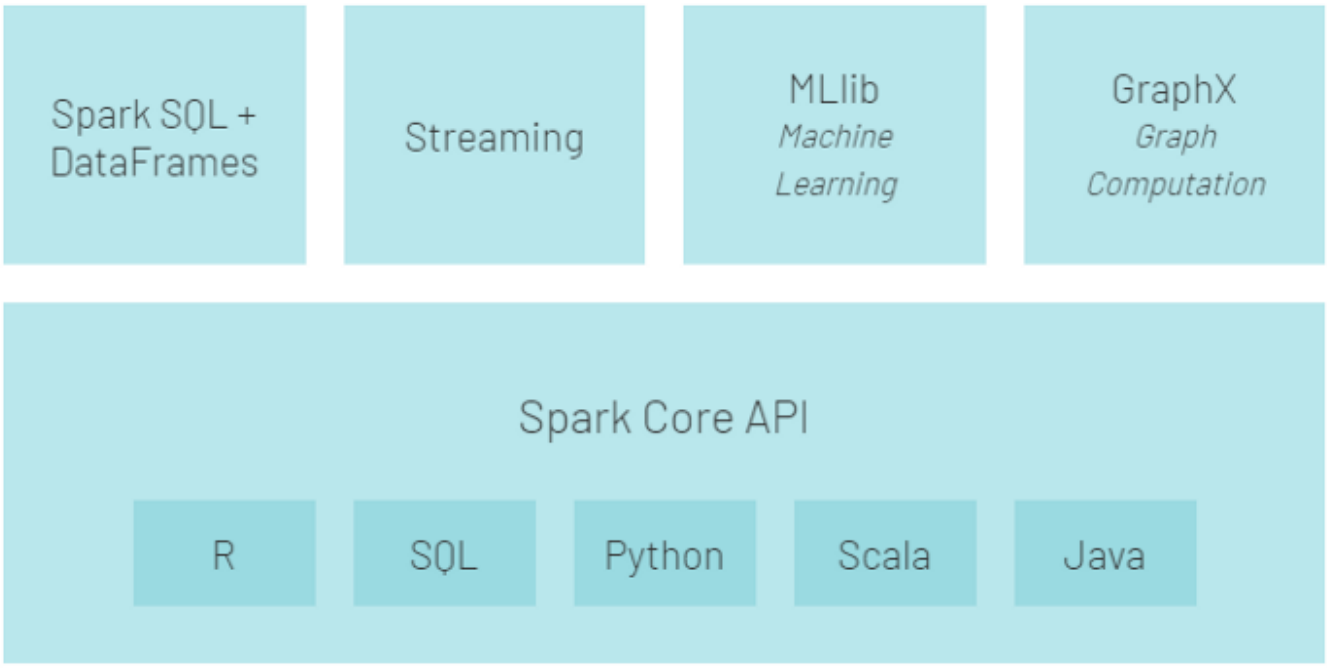
3. **Spark SQL + DataFrames**
   Spark SQL is a new module in Spark which integrates relational processing with Spark's functional programming API. It supports querying data either via SQL or via the Hive Query Language. For those of you familiar with RDBMS, Spark SQL will be

an easy transition from your earlier tools where you can extend the boundaries of traditional relational data processing.

4. **GraphX**
   GraphX is the Spark API for graphs and graph-parallel computation. Thus, it extends the Spark RDD with a Resilient Distributed Property Graph. At a high-level, GraphX extends the Spark RDD abstraction by introducing the Resilient Distributed Property Graph (a directed multigraph with properties attached to each vertex and edge).

5. **MLlib (Machine Learning)**
   MLlib stands for Machine Learning Library. Spark MLlib is used to perform machine learning in Apache Spark.

6. ***SparkR***
   It is an R package that provides a distributed data frame implementation. It also supports operations like selection, filtering, aggregation but on large data-sets.



Key Concepts & Terms

- Shared resources
- Parallelization

- Partitions
- Jobs, Stages, and Tasks
- Drivers
- Executors
- Clusters & Nodes
- Slots/Cores/Threads

Filtering

Distributed

Cluster and Nodes

- A Cluster is a group of nodes
- Nodes are the individual machines within a cluster (generally a VM)
- With Databricks, the driver (a JVM) and each executer (each a JVM) all run in their own nodes

Drivers

- Runs the Spark application
- Assign tasks to slots in an executor
- Coordinates the works between tasks
- Receives the results, if any

Executors

- Provide an environment in which tasks can be run
- Leverages the JVM to execute many threads

Slots/Cores/Threads

- The lowest unit of parallelization

- Generally interchangeable terms, but “slot” is the most accurate term
- Execute a set of transformations against a partition as directed by the driver

Parallelization

- scale horizontally by adding more executors
- scale vertically by adding cores to each executor

Partitions

- a ~128MB chunk of the larger dataset
- each task processes one and only one partition
- the size and record splits are decided by the driver
- The initial size is partially adjustable with various configuration options

Applications, Jobs, Stages, and Tasks

- The hierarchy into which work is subdivided
- One Spark action results in one or more jobs
- The number of stages depends on the operations submitted with the application
- Tasks are the smallest unit of work

General Notes

- Executors share machine level resources
- Tasks share executor (JVM) level resources
- Rarely are significant performance improvements made by tweaking Spark configuration settings

Counting

Stages

- A Stage cannot be completed until all tasks are completed

- Spark 3.x can run some stages in parallel (e.g. inputs to a join)
- One long-running task can delay an entire stage from completing
- The shuffle between two stages is one of the **most expensive operations** in Apache Spark

Distinct

The Shuffle

Shuffling is the processes of rearranging data within a cluster between stages.

Triggered by "wide" operations:

- re-partitioning
- ByKey operations (except counting)
- Joins, the worse being cross joins
- Sorting
- Distinct
- GroupBy

General Notes

- Try to group wide transformations together for the best automatic optimization
- Aim for: narrow, narrow, wide, wide, wide, narrow
- Avoid: narrow, wide, narrow, wide, narrow, wide
- One of the most significant, yet often unavoidable, cause of performance degradation
- Just because it's slow, it doesn't mean that it's bad (don't polish the cannon ball)

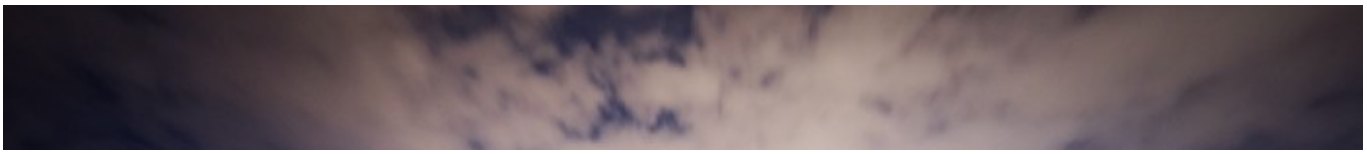


Photo by Erik Mclean on Unsplash

Apache Spark Architecture | Distributed System Architecture Explained | Edureka

Apache Spark is an open-source cluster computing framework which is setting the world of Big Data on fire. According to...

www.edureka.co

Apache Spark Architecture Explained in Detail

"Spark is beautiful. With Hadoop, it would take us six-seven months to develop a machine learning model. Now, we can do...

www.dezyre.com

Apache Spark Tutorial: Getting Started with Apache Spark Tutorial

Apache Spark is a powerful open-source processing engine built around speed, ease of use, and sophisticated analytics...

databricks.com

Sign up for Geek Culture Hits

By Geek Culture

Subscribe to receive top 10 most read stories of Geek Culture — delivered straight into your inbox, once a week. Take a look.

Your email

Get this newsletter

By signing up, you will create a Medium account if you don't already have one. Review our Privacy Policy for more information about our privacy practices.

Apache Spark Data Engineering Pyspark Architecture Programming