Apache Spark Architecture Simplified

Document by – Siddhartha Subudhi

Visit my LinkedIn profile

What is Spark?

Spark Architecture, an open-source, framework-based component that processes a large amount of unstructured, semi-structured, and structured data for analytics, is utilized in Apache Spark. Apart from Hadoop and map-reduce architectures for big data processing, Apache Spark's architecture is regarded as an alternative.

The RDD and DAG, Spark's data storage and processing framework, are utilized to store and process data, respectively. Spark architecture consists of four components, including the spark driver, executors, cluster administrators, and worker nodes. It uses the Dataset and data frames as the fundamental data storage mechanism to optimize the Spark process and big data computation.

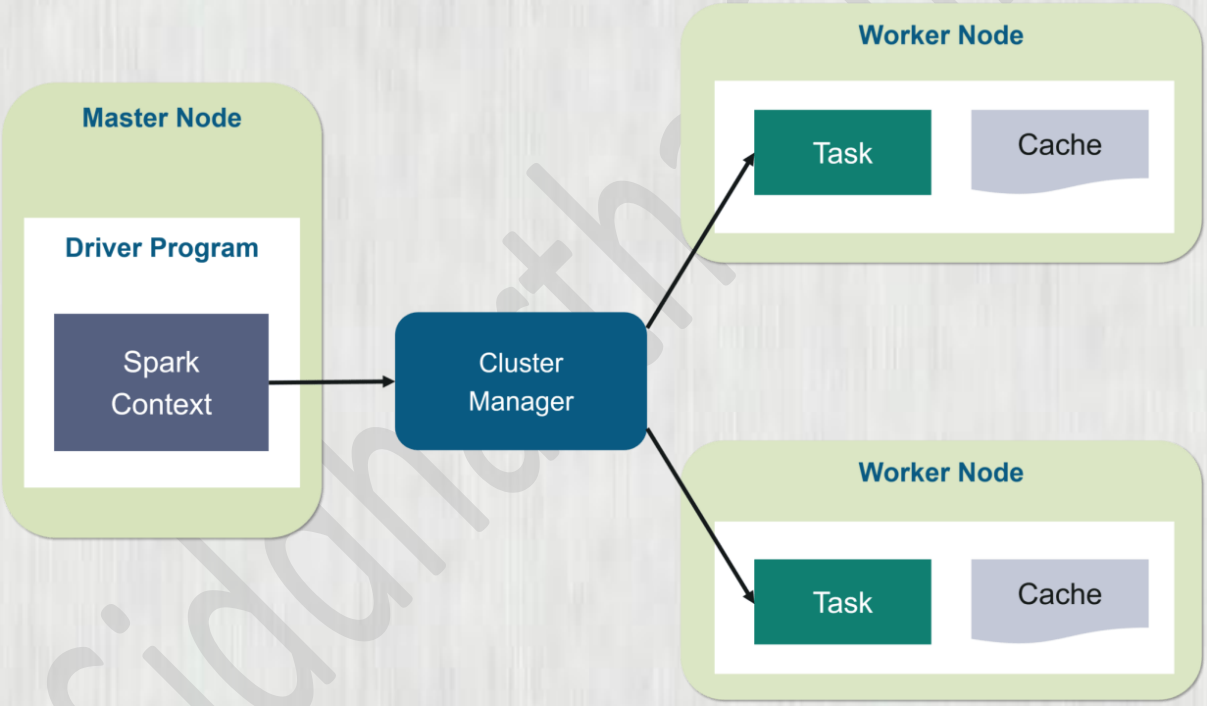
Spark Architecture Overview

Apache Spark has a well-defined layered architecture where all the spark components and layers are loosely coupled. This architecture is further integrated with various extensions and libraries. It is used for parallel data processing on computer clusters and has become a standard tool for any Developer or Data Scientist interested in Big Data.

Spark supports multiple widely-used programming languages like Java, Python, R, and Scala. It runs anywhere from a laptop to a cluster of thousands of servers making it a beginner-friendly system with a steep learning curve and users can scale up to big data processing or to an incredibly large scale.

Apache Spark Architecture

Ready to gain a competitive advantage with Future Ready Emerging Technologies?



The Spark follows the master-slave architecture. Its cluster consists of a single master and multiple slaves. In your master node, you have the driver program, which drives your application. The code you are writing behaves as a driver program or if you are using the interactive shell, the shell acts as the driver program.

Driver Program

Driver Program in the Apache Spark architecture calls the main program of an application and creates SparkContext. It is a process that runs the main() function of the application and creates the SparkContext object.

The purpose of SparkContext is to coordinate with the spark applications, running as independent sets of processes on a cluster.

Cluster Manager

The role of the cluster manager is to allocate resources across applications. It maintains a cluster of machines that will run Spark applications. It has its own driver called the “master” and “worker” abstractions. These are tied to physical machines instead of processes like in Spark.

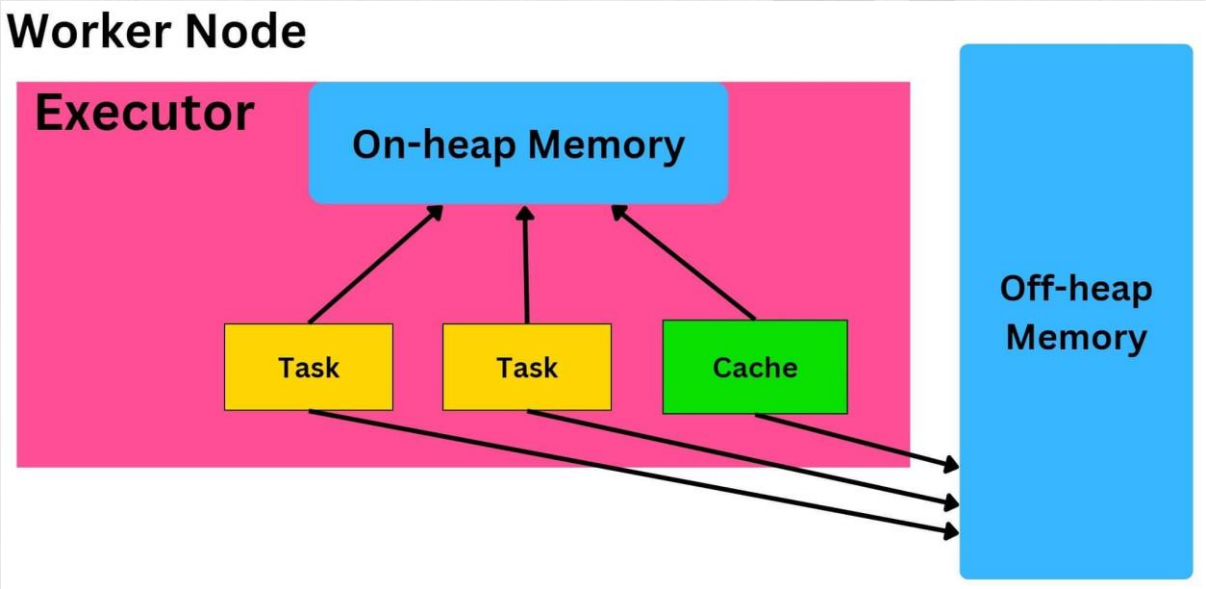
It consists of various types of cluster managers such as Hadoop YARN, Apache Mesos, and Standalone Scheduler. Here, the Standalone Scheduler is a standalone spark cluster manager that facilitates to the installation of Spark on an empty set of machines.

Worker Nodes

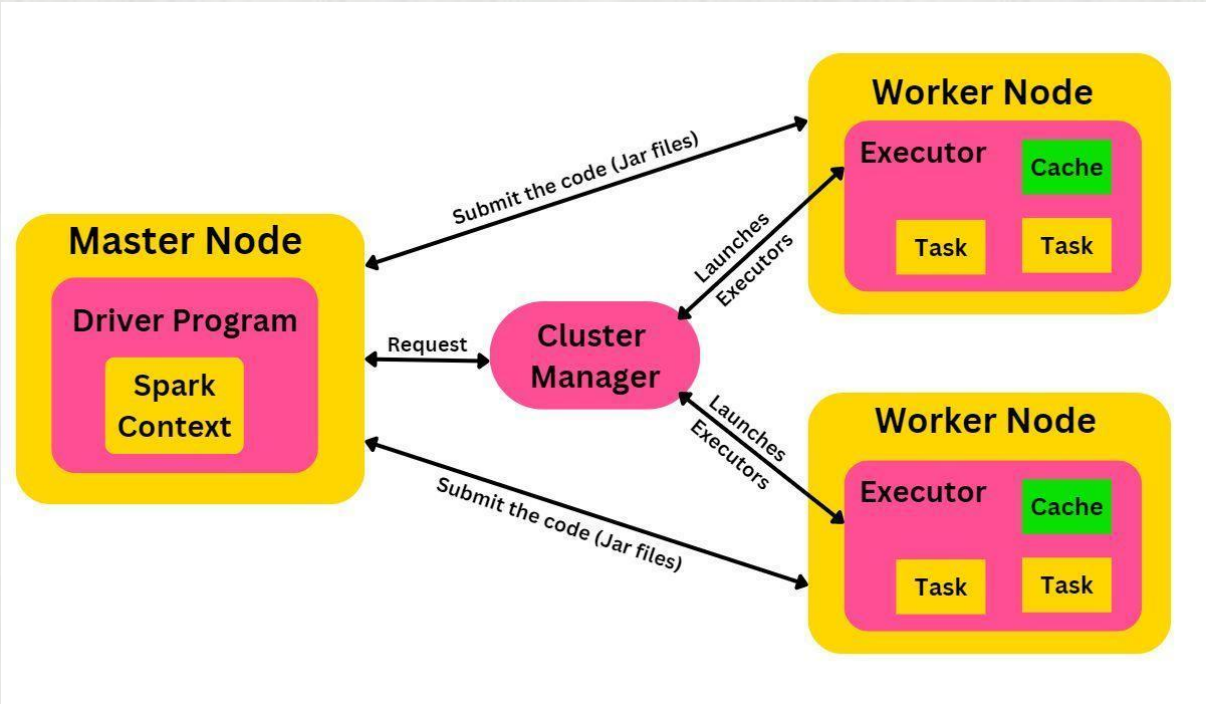
Worker nodes are the slave nodes whose job is to basically execute the tasks. These tasks are then executed on the partitioned RDDs in the worker node and hence return back the result to the Spark Context.

Spark Context takes the job, breaks the job into tasks, and distributes them to the worker nodes. These tasks work on the partitioned RDD, perform operations, collect the results and return to the main Spark Context.

If you increase the number of workers, then you can divide jobs into more partitions and execute them parallelly over multiple systems. It will be a lot faster. With the increase in the number of workers, memory size will also increase & you can cache the jobs to execute them faster.



Working of Spark Architecture:-



When the Driver Program in the Apache Spark architecture executes, it calls the real program of an application and creates a SparkContext. SparkContext contains all of the basic functions. The Spark Driver includes several other components, including a DAG Scheduler, Task Scheduler, Backend Scheduler, and Block Manager, all of which are responsible for translating user-written code into jobs that are actually executed on the cluster.

The Cluster Manager manages the execution of various jobs in the cluster. Spark Driver works in conjunction with the Cluster Manager to control the execution of various other jobs. The cluster Manager does the task of allocating resources for the job. Once the job has been broken down into smaller jobs, which are then distributed to worker nodes, SparkDriver will control the execution.
Many worker nodes can be used to process an RDD created in SparkContext, and the results can also be cached.

The Spark Context receives task information from the Cluster Manager and enqueues it on worker nodes.

The executor is in charge of carrying out these duties. The lifespan of executors is the same as that of the Spark Application. We can increase the number of workers if we want to improve the performance of the system. In this way, we can divide jobs into more coherent parts.