Understand The Internal Working of Apache Spark

Dhanya Thailappan — August 26, 2021

Beginner Data Engineering Programming Python Spark

This article was published as a part of the Data Science Blogathon

In this fast-paced digitized world, the size of data generation is increasing every second. This data cannot be thrown away as this may help to get important business insights. Apache Spark is the largest open-source project for data processing.

In this article, I am going to discuss the internals of Apache Spark and the high level of Spark architecture.

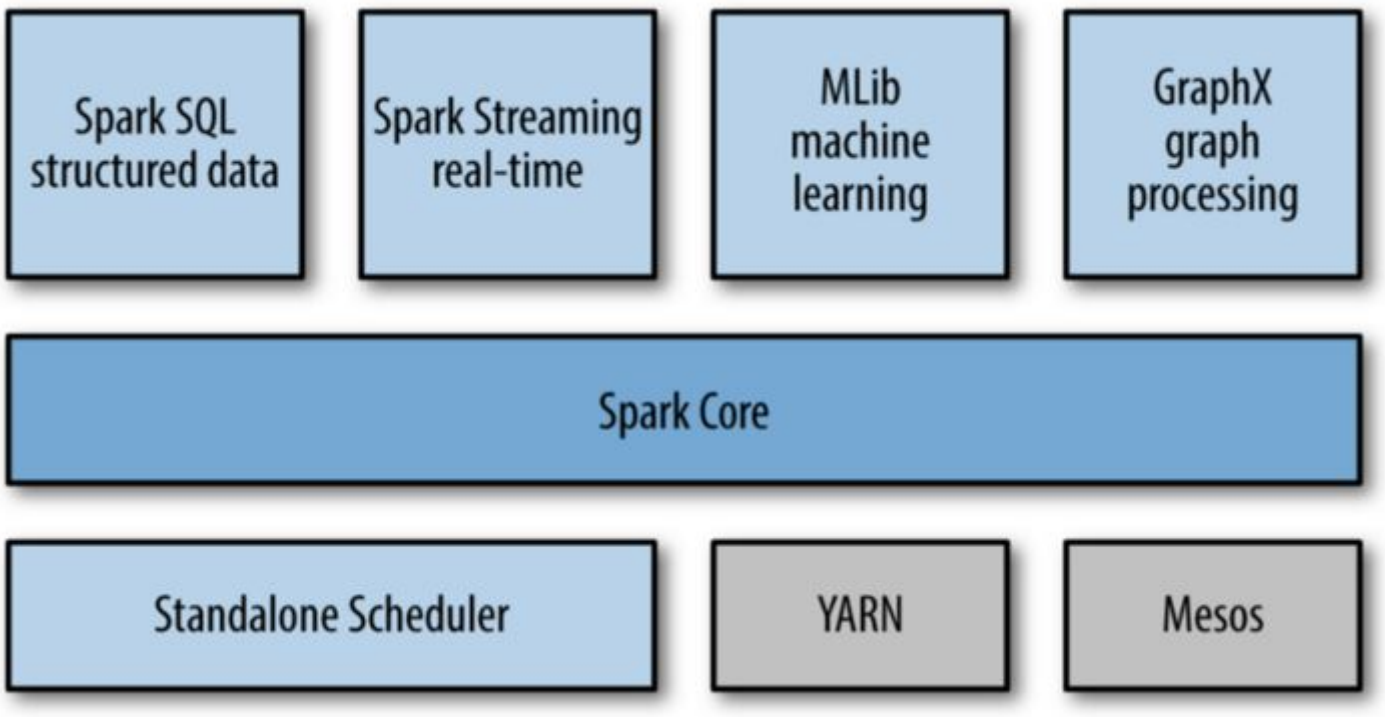
Overview

Apache Spark is an open-source distributed big data processing engine. It provides a common processing engine for both streaming and batch data. It provides parallelism and fault tolerance. Apache Spark provides high-level APIs in four languages such as Java, Scala, Python and R. Apace Spark was developed to eliminate the drawbacks of Hadoop MapReduce. Spark works on the concept of in-memory computation which makes it around a hundred times faster than Hadoop MapReduce.

Table of contents

Let’s see the different components of Spark.

Spark Components



Spark consists of one of the bigger components called Spark core which contains the majority of the libraries. On top of this Spark core, there are four different components. They are

- Spark SQL
- Spark Streaming
- MLLIB
- GraphX

Spark SQL

It provides a SQL like interface to do the data processing with Spark as a processing engine. It can process both structured and semi-structured data. Using Spark SQL, you can query in SQL and HQL too. It can process data from multiple sources.

Spark Streaming

It provides support for processing a large stream of data. The real-time streaming is supported by using micro-batching. The incoming live data is converted into small batches and processed by the processing engine.

MLLIB

This component provides libraries for machine learning towards the statistic and dynamic analysis of the data. It also includes clustering, classification, regression and many other machine learning algorithms. It delivers high-quality algorithms with high speed.

GraphX

It provides a distributed graph computation on top of the Spark core. It consists of several Spark RDD API which helps in creating directed graphs whose vertices and edges are linked with arbitrary properties. Using GraphX, traversal, searching and pathfinding can be done.

Cluster Managers

Spark also consists of three pluggable cluster managers.

Standalone

It is a simple cluster manager which is easier to set up and execute the tasks on the cluster. It is a reliable cluster manager which can handle failures successfully. It can manage the resources based on the application requirements.

Apache Mesos

It is a general cluster manager from the Apache group that can also run Hadoop MapReduce along with Spark and other service applications. It consists of API for most of the programming languages.

Hadoop YARN

It is a resource manager which was provided in Hadoop 2. It stands for Yet Another Resource Negotiator. It is also a general-purpose cluster manager and can work in both Hadoop and Spark.

These are the three cluster managers supported by the Spark ecosystem.

Now, let me give a glimpse of the most essential part of Apache Spark. i.e., Spark RDD.

Spark RDD

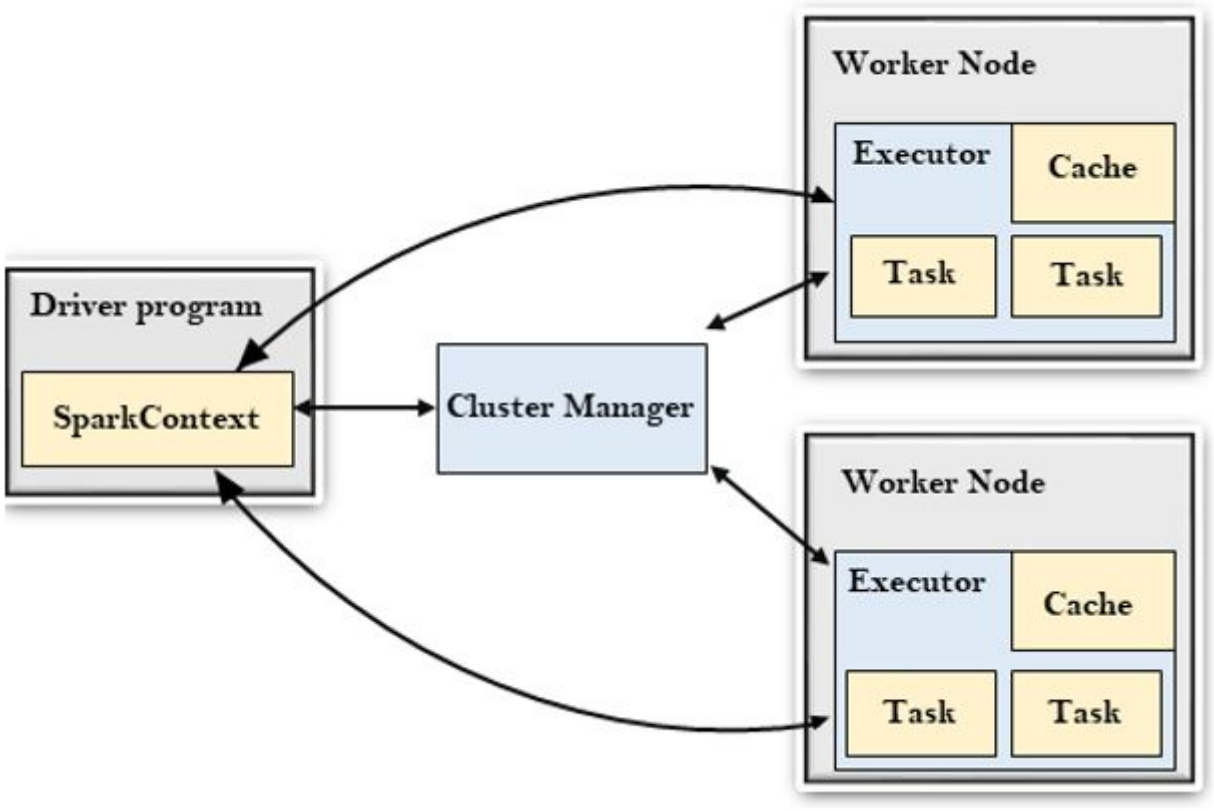
Apache Spark has a fundamental data structure called Resilient Distributed Dataset (RDD) which forms the backbone of

create new RDD based on the operations we perform. Actions are the processes that we perform on the newly created RDD to get the desired results. Examples of actions are collect(), countByKey(). Action returns the result to the driver.

The high-level architecture of Spark

Now, let's look into the high-level architecture of the Spark execution program.

When we run any job
with Spark, this is how underlying execution happens.



Source

There would be one driver program that works with the cluster manager to schedule tasks on the worker nodes. So, once these tasks are completed, they will return the result to the driver program.

Let me explain this
process in detail.

Spark Architecture run-time components

Spark Driver

The first and foremost activity of the Spark driver is to call the main method of the program. Whenever you write any execution code in Spark, you will give the main class which is the entry point to your program and that point is executed by the Spark driver. The Spark driver creates the Spark context or Spark session depends on which version of Spark you are working in. The driver is the process that runs the user code which eventually creates RDD data frames and data units which are data unit abstractions in the Spark world. The driver performs all the different transformations and executes the actions in the form of tasks that are executed on different executors. The two main functions performed by the driver are

- Converting the user program into tasks
- Scheduling those tasks on the executors with the help of the cluster manager.

Spark cluster manager

Spark execution is agnostic to the cluster manager. You can plug in any of the three available cluster managers or supported cluster managers and the execution behaviour don't change that it doesn't change based on which cluster manager you are using.

Spark lies on the cluster manager to launch the executors. This is the prerogative of the cluster manager to schedule and launch executors. Resources are allocated by the cluster manager for the execution of the tasks. The cluster manager is a pluggable component as we have seen we have a choice out of three available and supported cluster managers and we can plug any one of them based on our use case and needs. Cluster manager can dynamically adjust your resource used by the Spark application depending on the workload. The cluster manager can increase the number of executors or decrease the number of executors based on the kind of workload data processing needs to be done.

Now, let's see what are the different activities performed by Spark executors.

Spark executor

The individual tasks in the given Spark job run in the Spark executor. The Spark executors run the actual programming logic of data processing in the form of tasks. The executors are launched at the beginning of the Spark application when you submit to do the jobs and they run for the entire lifetime of an application. The two main roles of the executors are

- To run the tasks and return the results to the driver.
- It provides in-memory storage for the RDD data set and data frames that are cached by the user.

So,

the executor is the actual unit of execution that performs tasks for the data processing.

Spark sample program

Consider the following sample program written in PySpark.

```
from pyspark import SparkContext
sc = SparkContext("local", "count app")
words = sc.parallelize (
   ["scala",
   "java",
   "hadoop",
   "spark",
   "akka",
   "spark vs hadoop",
   "pyspark",
   "pyspark and spark"]
)
words_filter = words.filter(lambda x: 'spark' in x)
counts = words_filter.count()
print "Number of elements filtered  -> %i" % (counts)
```

```
//sample output
Number of elements
filtered -> 4
```

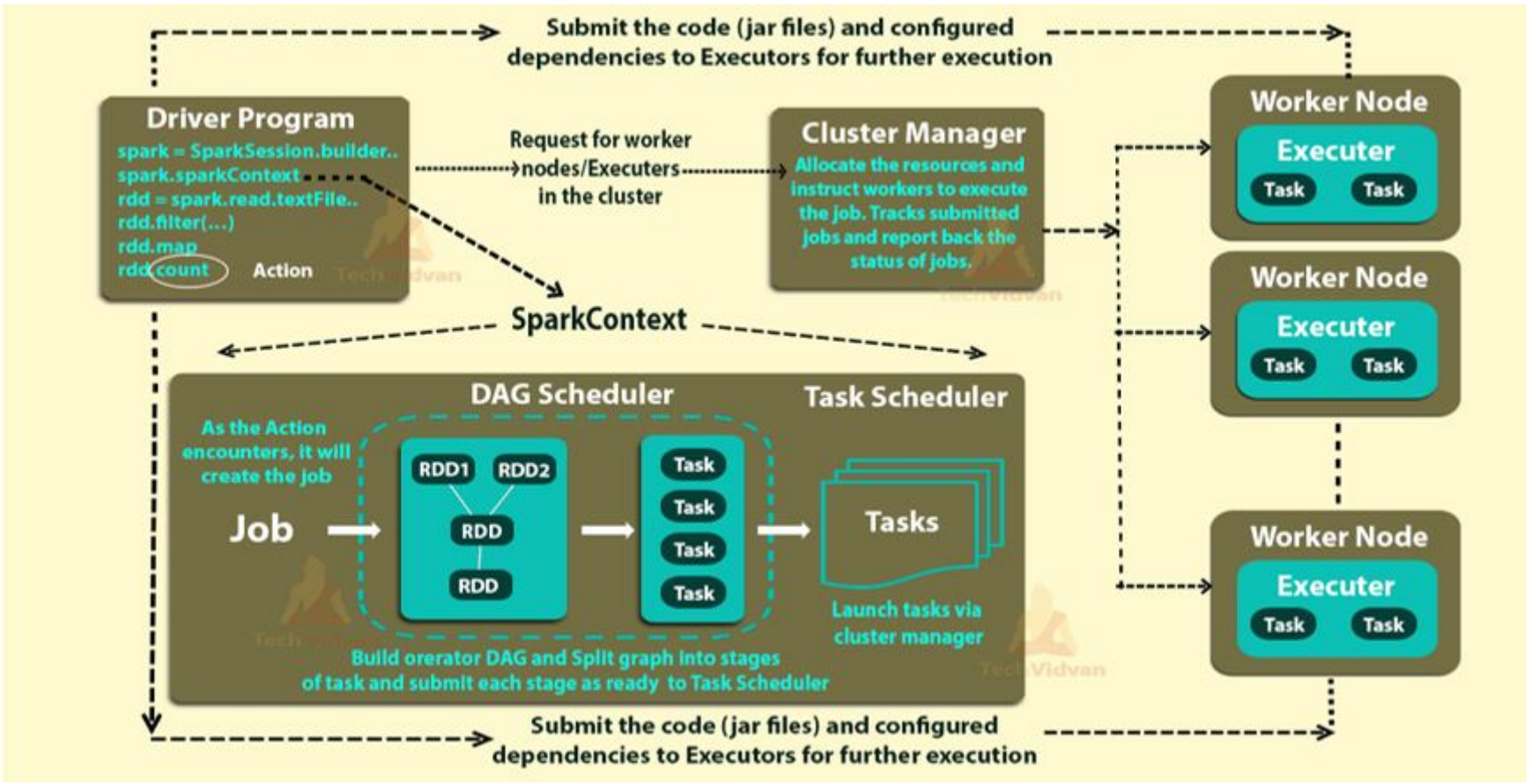
First, we are importing the SparkContext which is the entry point of Spark functionality. The second line creates a SparkContext object. Next, we are creating a sample RDD 'words' by the parallelize method. The words containing the string 'spark' is filtered and stored in words_filter. Count() function is used to count the number of words filtered and the result is printed.

Here, the process of applying a filter to the data in RDD is transformation and counting the number of words is action.

Now, let's take a

further deep dive into what happens when you do submit a particular Spark job

This is the detailed diagram of Spark Architecture.



Source

The first block you see is the driver program. Once you do a Spark submit, a driver program is launched and this requests for resources to the cluster manager and at the same time the main program of the user function of the user processing program is initiated by the driver program.

Based on that, the execution logic is processed and parallelly Spark context is also created. Using the Spark context, the different transformations and actions are processed. So, till the time the action is not encountered, all the transformations will go into the Spark context in the form of DAG that will create RDD lineage.

Once the action is called job is created. Job is the collection of different task stages. Once these tasks are created, they are launched by the cluster manager on the worker nodes and this is done with the help of a class called task scheduler.

The conversion of RDD lineage into tasks is done by the DAG scheduler. Here DAG is created based on the different transformations in the program and once the action is called these are split into different stages of tasks and submitted to the task scheduler as tasks become ready.

Then these are launched on the different executors in the worker node through the cluster manager. The entire resource allocation and the tracking of the jobs and tasks are performed by the cluster manager.

As soon as you do a Spark submit, your user program and other configuration mentioned are copied onto all the available nodes in the cluster. So that the program becomes the local read on all the worker nodes. Hence, the parallel executors running on the different worker nodes do not have to do any kind of network routing.

This is how the
entire execution of the Spark job happens.

Endnotes

Now you have a basic idea about the internal working of Apache Spark and how it is applied in solving big data problems.

I hope you enjoyed reading this article.

Thanks for reading, cheers!

Please take a look at

my other articles on dhanya_thailappan, Author at Analytics Vidhya.

The media shown in this article are not owned by Analytics Vidhya and are used at the Author's discretion.

Apache Spark blogathon Data Engineering

About the Author

Dhanya Thailappan

Data Science and AI enthusiast

Our Top Authors

Download

Analytics Vidhya App for the Latest blog/Article

Previous Post

An Introduction to Natural Language Processing with Transformers

Next Post

Understanding Gradient Descent Algorithm and the Maths Behind It

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name*

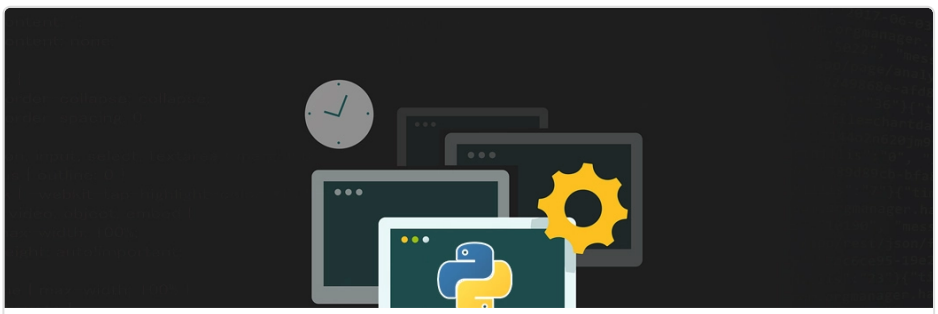
Email*

Website

Notify me of follow-up comments by email.

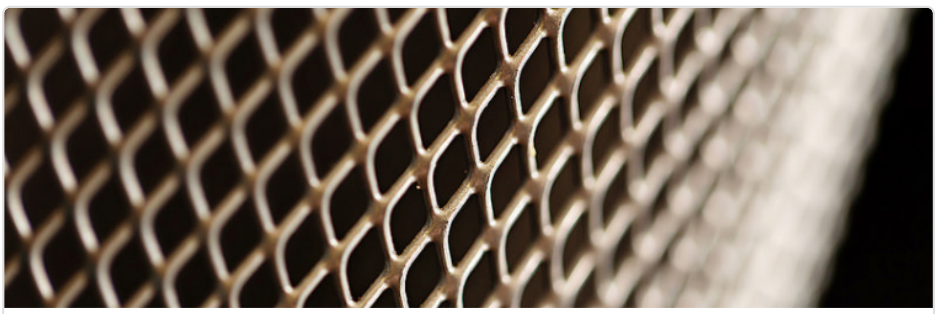
Submit

Top Resources



Python Tutorial: Working with CSV file for Data Science

Harika Bonthu - AUG 21, 2021



40 Questions to test a Data Scientist on Clustering Techniques..

Sauravkaushik8 Kaushik - FEB 05, 2017

30 Questions to test a data scientist on K-Nearest Neighbors..

Sunil Ray - SEP 04, 2017

3 Interesting Python Projects With Code for Beginners!

Gaurav Sharma - JUL 18, 2021