Apache Spark Quick Start Guide

Quickly learn the art of writing efficient big data applications with Apache Spark

Packt

www.packt.com

Shrey Mehrotra and Akash Grade

Apache Spark Quick Start Guide

Quickly learn the art of writing efficient big data applications with Apache Spark

Shrey Mehrotra
Akash Grade

Packt

BIRMINGHAM - MUMBAI

Apache Spark Quick Start Guide

Commissioning Editor: Amey Varangaonkar
Acquisition Editor: Siddharth Mandal
Content Development Editor: Smit Carvalho
Technical Editor: Aishwarya More
Copy Editor: Safis Editing
Project Coordinator: Pragati Shukla
Proofreader: Safis Editing
Indexer: Pratik Shirodkar
Graphics: Alishon Mendonsa
Production Coordinator: Deepika Naik

First published: January 2019

Production reference: 1310119

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

ISBN 978-1-78934-910-8

`www.packtpub.com`

Contributors

About the authors

Shrey Mehrotra has over 8 years of IT experience and, for the past 6 years, has been designing the architecture of cloud and big-data solutions for the finance, media, and governance sectors. Having worked on research and development with big-data labs and been part of Risk Technologies, he has gained insights into Hadoop, with a focus on Spark, HBase, and Hive. His technical strengths also include Elasticsearch, Kafka, Java, YARN, Sqoop, and Flume. He likes spending time performing research and development on different big-data technologies. He is the coauthor of the books *Learning YARN* and *Hive Cookbook*, a certified Hadoop developer, and he has also written various technical papers.

Akash Grade is a data engineer living in New Delhi, India. Akash graduated with a BSc in computer science from the University of Delhi in 2011, and later earned an MSc in software engineering from BITS Pilani. He spends most of his time designing highly scalable data pipeline using big-data solutions such as Apache Spark, Hive, and Kafka. Akash is also a Databricks-certified Spark developer. He has been working on Apache Spark for the last five years, and enjoys writing applications in Python, Go, and SQL.

About the reviewer

Nisith Kumar Nanda is a passionate big data consultant who loves to find solutions to complex data problems. He has around 10 years of IT experience working on multiple technologies with various clients globally. His core expertise involves working with open source big data technologies such as Apache Spark, Kafka, Cassandra, HBase, to build critical next generation real-time and batch applications. He is very proficient in various programming languages such as Java, Scala, and Python. He is passionate about AI, machine learning, and NLP.

I would like to thank my family and especially my wife, Samita, for their support. I will also take this opportunity to thank my friends and colleagues who helped me to grow professionally.

Table of Contents

Preface

Apache Spark is a flexible in-memory framework that allows the processing of both batch and real-time data in a distributed way. Its unified engine has made it quite popular for big data use cases.

This book will help you to quickly get started with Apache Spark 2.x and help you write efficient big data applications for a variety of use cases. You will get to grip with the low-level details as well as core concepts of Apache Spark, and the way they can be used to solve big data problems. You will be introduced to RDD and DataFrame APIs, and their corresponding transformations and actions.

This book will help you learn Spark's components for machine learning, stream processing, and graph analysis. At the end of the book, you'll learn different optimization techniques for writing efficient Spark code.

Who this book is for

If you are a big data enthusiast and love processing huge amounts of data, this book is for you. If you are a data engineer and looking for the best optimization techniques for your Spark applications, then you will find this book helpful. This book will also help data scientists who want to implement their machine learning algorithms in Spark. You need to have a basic understanding of programming languages such as Scala, Python, or Java.

What this book covers

`Chapter 1`, *Introduction to Apache Spark*, provides an introduction to Spark 2.0. It provides a brief description of different Spark components, including Spark Core, Spark SQL, Spark Streaming, machine learning, and graph processing. It also discusses the advantages of Spark compared to other similar frameworks.

`Chapter 2`, *Apache Spark Installation*, provides a step-by-step guide to installing Spark on an AWS EC2 instance from scratch. It also helps you install all the prerequisites, such as Python, Java, and Scala.

`Chapter 3`, *Spark RDD*, explains **Resilient Distributed Datasets** (**RDD**) APIs, which are the heart of Apache Spark. It also discusses various transformations and actions that can be applied on an RDD.

`Chapter 4`, *Spark DataFrame and Dataset*, covers Spark's structured APIs: DataFrame and Dataset. This chapter also covers various operations that can be performed on a DataFrame or Dataset.

`Chapter 5`, *Spark Architecture and Application Execution Flow*, explains the interaction between different services involved in Spark application execution. It explains the role of worker nodes, executors, and drivers in application execution in both client and cluster mode. It also explains how Spark creates a **Directed Acyclic Graph** (**DAG**) that consists of stages and tasks.

`Chapter 6`, *Spark SQL*, discusses how Spark gracefully supports all SQL operations by providing a Spark-SQL interface and various DataFrame APIs. It also covers the seamless integration of Spark with the Hive metastore.

`Chapter 7`, *Spark Streaming, Machine Learning, and Graph Analysis*, explores different Spark APIs for working with real-time data streams, machine learning, and graphs. It explains the candidature of features based on the use case requirements.

`Chapter 8`, *Spark Optimizations*, covers different optimization techniques to improve the performance of your Spark applications. It explains how you can use resources such as executors and memory in order to better parallelize your tasks.

To get the most out of this book

Use a machine with a recent version of Linux or macOS. It will be useful to know the basic syntax of Scala, Python, and Java. Install Python's NumPy package in order to work with Spark's machine learning packages.

Download the example code files

You can download the example code files for this book from your account at `www.packt.com`. If you purchased this book elsewhere, you can visit `www.packt.com/support` and register to have the files emailed directly to you.

You can download the code files by following these steps:

1. Log in or register at `www.packt.com`
2. Select the **SUPPORT** tab
3. Click on **Code Downloads and Errata**
4. Enter the name of the book in the **Search** box and follow the onscreen instructions

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR/7-Zip for Windows
- Zipeg/iZip/UnRarX for Mac
- 7-Zip/PeaZip for Linux

The code bundle for the book is also hosted on GitHub at `https://github.com/PacktPublishing/Apache-Spark-Quick-Start-Guide`. In case there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here: `https://www.packtpub.com/sites/default/files/downloads/9781789349108_ColorImages.pdf`.

Conventions used

There are a number of text conventions used throughout this book.

`CodeInText`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "Mount the downloaded `WebStorm-10*.dmg` disk image file as another disk in your system."

Any command-line input or output is written as follows:

```
$ mkdir css
$ cd css
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "Select **System info** from the **Administration** panel."

Warnings or important notes appear like this.

Tips and tricks appear like this.

Get in touch

Feedback from our readers is always welcome:

General feedback: If you have questions about any aspect of this book, mention the book title in the subject of your message and email us at `customercare@packtpub.com`.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packt.com/submit-errata`, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Piracy: If you come across any illegal copies of our works in any form on the Internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packt.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit `packt.com`.

1
Introduction to Apache Spark

Apache Spark is an open source framework for processing large datasets stored in heterogeneous data stores in an efficient and fast way. Sophisticated analytical algorithms can be easily executed on these large datasets. Spark can execute a distributed program 100 times faster than MapReduce. As Spark is one of the fast-growing projects in the open source community, it provides a large number of libraries to its users.

We shall cover the following topics in this chapter:

- A brief introduction to Spark
- Spark architecture and the different languages that can be used for coding Spark applications
- Spark components and how these components can be used together to solve a variety of use cases
- A comparison between Spark and Hadoop

What is Spark?

Apache Spark is a distributed computing framework which makes big-data processing quite easy, fast, and scalable. You must be wondering what makes Spark so popular in the industry, and how is it really different than the existing tools available for big-data processing? The reason is that it provides a unified stack for processing all different kinds of big data, be it batch, streaming, machine learning, or graph data.

Spark was developed at UC Berkeley's AMPLab in 2009 and later came under the Apache Umbrella in 2010. The framework is mainly written in Scala and Java.

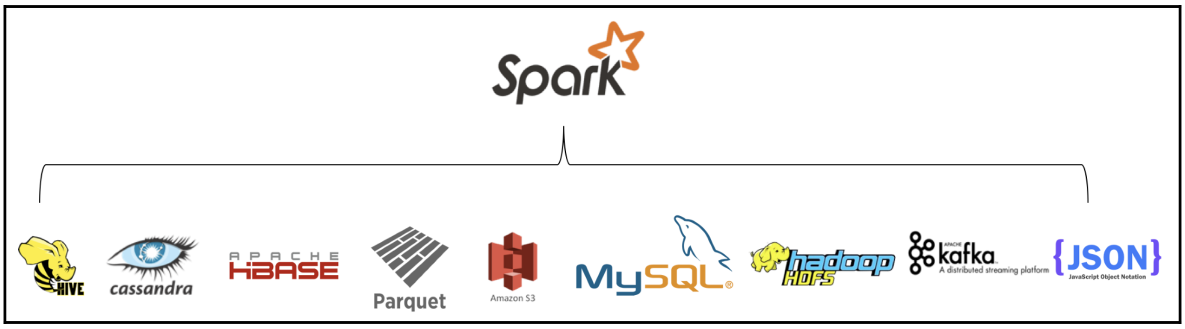
Spark provides an interface with many different distributed and non-distributed data stores, such as **Hadoop Distributed File System** (**HDFS**), Cassandra, Openstack Swift, Amazon S3, and Kudu. It also provides a wide variety of language APIs to perform analytics on the data stored in these data stores. These APIs include Scala, Java, Python, and R.

The basic entity of Spark is **Resilient Distributed Dataset** (**RDD**), which is a read-only partitioned collection of data. RDD can be created using data stored on different data stores or using existing RDD. We shall discuss this in more detail in `Chapter 3`, *Spark RDD*.

Spark needs a resource manager to distribute and execute its tasks. By default, Spark comes up with its own standalone scheduler, but it integrates easily with Apache Mesos and **Yet Another Resource Negotiator** (**YARN**) for cluster resource management and task execution.

One of the main features of Spark is to keep a large amount of data in memory for faster execution. It also has a component that generates a **Directed Acyclic Graph** (**DAG**) of operations based on the user program. We shall discuss these in more details in coming chapters.

The following diagram shows some of the popular data stores Spark can connect to:



Data stores

Spark is a computing engine, and should not be considered as a storage system as well. Spark is also not designed for cluster management. For this purpose, frameworks such as Mesos and YARN are used.

Spark architecture overview

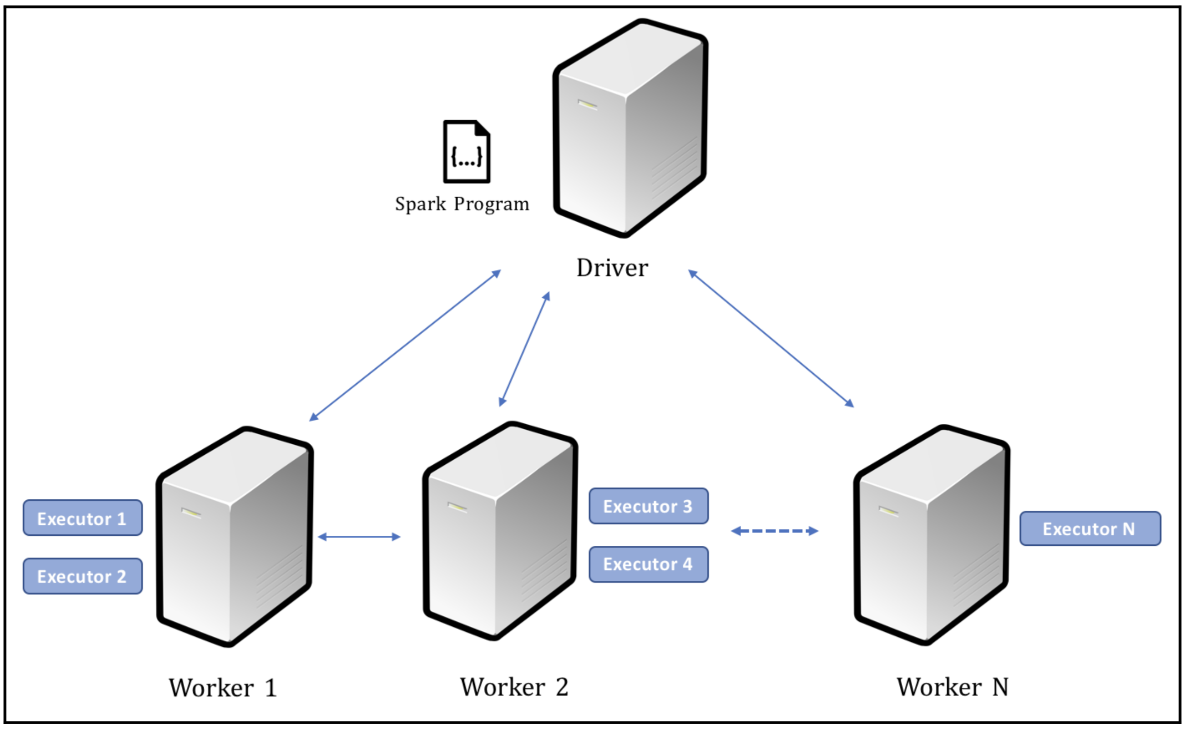
Spark follows a master-slave architecture, as it allows it to scale on demand. Spark's architecture has two main components:

- **Driver Program**: A driver program is where a user writes Spark code using either Scala, Java, Python, or R APIs. It is responsible for launching various parallel operations of the cluster.
- **Executor:** Executor is the **Java Virtual Machine** (**JVM**) that runs on a worker node of the cluster. Executor provides hardware resources for running the tasks launched by the driver program.

As soon as a Spark job is submitted, the driver program launches various operation on each executor. Driver and executors together make an *application*.

The following diagram demonstrates the relationships between **Driver**, **Workers**, and **Executors**. As the first step, a **driver** process parses the user code (**Spark Program**) and creates multiple executors on each worker node. The driver process not only forks the executors on work machines, but also sends tasks to these executors to run the entire application in parallel.

Once the computation is completed, the output is either sent to the driver program or saved on to the file system:



Driver, Workers, and Executors

Spark language APIs

Spark has integration with a variety of programming languages such as Scala, Java, Python, and R. Developers can write their Spark program in either of these languages. This freedom of language is also one of the reasons why Spark is popular among developers. If you compare this to Hadoop MapReduce, in MapReduce, the developers had only one choice: Java, which made it difficult for developers from another programming languages to work on MapReduce.

Scala

Scala is the primary language for Spark. More than 70% of Spark's code is written in **Scalable Language** (**Scala**). Scala is a fairly new language. It was developed by Martin Odersky in 2001, and it was first launched publicly in 2004. Like Java, Scala also generates a bytecode that runs on JVM. Scala brings advantages from both object-oriented and functional-oriented worlds. It provides dynamic programming without compromising on type safety. As Spark is primarily written in Scala, you can find almost all of the new libraries in Scala API.

Java

Most of us are familiar with Java. Java is a powerful object-oriented programming language. The majority of big data frameworks are written in Java, which provides rich libraries to connect and process data with these frameworks.

Python

Python is a functional programming language. It was developed by Guido van Rossum and was first released in 1991. For some time, Python was not popular among developers, but later, around 2006-07, it introduced some libraries such as **Numerical Python** (**NumPy**) and **Pandas**, which became cornerstones and made Python popular among all types of programmers. In Spark, when the driver launches executors on worker nodes, it also starts a Python interpreter for each executor. In the case of RDD, the data is first shipped into the JVMs, and is then transferred to Python, which makes the job slow when working with RDDs.

R

R is a statistical programming language. It provides a rich library for analyzing and manipulating the data, which is why it is very popular among data analysts, statisticians, and data scientists. Spark R integration is a way to provide data scientists the flexibility required to work on big data. Like Python, SparkR also creates an R process for each executor to work on data transferred from the JVM.

SQL

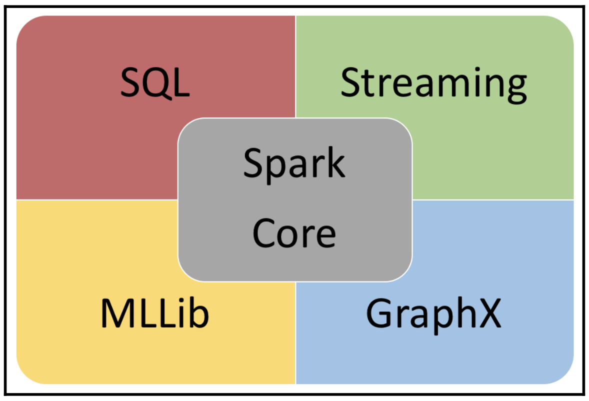
Structured Query Language (**SQL**) is one of the most popular and powerful languages for working with tables stored in the database. SQL also enables non-programmers to work with big data. Spark provides Spark SQL, which is a distributed SQL query engine. We will learn about it in more detail in `Chapter 6`, *Spark SQL*.

Spark components

As discussed earlier in this chapter, the main philosophy behind Spark is to provide a unified engine for creating different types of big data applications. Spark provides a variety of libraries to work with batch analytics, streaming, machine learning, and graph analysis.

It is not as if these kinds of processing were never done before Spark, but for every new big data problem, there was a new tool in the market; for example, for batch analysis, we had MapReduce, Hive, and Pig. For **Streaming**, we had Apache Storm, for machine learning, we had Mahout. Although these tools solve the problems that they are designed for, each of them requires a learning curve. This is where Spark brings advantages. Spark provides a unified stack for solving all of these problems. It has components that are designed for processing all kinds of big data. It also provides many libraries to read or write different kinds of data such as JSON, CSV, and Parquet.

Here is an example of a Spark stack:

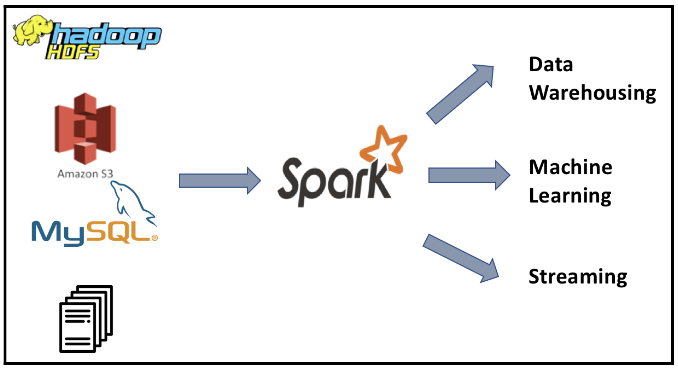


Spark stack

Having a unified stack brings lots of advantages. Let's look at some of the advantages:

- First is code sharing and reusability. Components developed by the data engineering team can easily be integrated by the data science team to avoid code redundancy.
- Secondly, there is always a new tool coming in the market to solve a different big data usecase. Most of the developers struggle to learn new tools and gain expertise in order to use them efficiently. With Spark, developers just have to learn the basic concepts which allows developers to work on different big data use cases.
- Thirdly, its unified stack gives great power to the developers to explore new ideas without installing new tools.

The following diagram provides a high-level overview of different big-data applications powered by Spark:



Spark use cases

Spark Core

Spark Core is the main component of Spark. Spark Core defines the following:

- The basic components, such as RDD and DataFrames
- The APIs available to perform operations on these basic abstractions
- Shared or distributed variables, such as broadcast variables and accumulators

We shall look at them in more detail in the upcoming chapters.

Spark Core also defines all the basic functionalities, such as task management, memory management, basic I/O functionalities, and more. It's a good idea to have a look at the Spark code on GitHub (`https://github.com/apache/spark`).

Spark SQL

Spark SQL is where developers can work with structured and semi-structured data such as Hive tables, MySQL tables, Parquet files, AVRO files, JSON files, CSV files, and more. Another alternative to process structured data is using Hive. Hive processes structured data stored on HDFS using **Hive Query Language** (**HQL**). It internally uses MapReduce for its processing, and we shall see how Spark can deliver better performance than MapReduce. In the initial version of Spark, structured data used to be defined as schema RDD (another type of an RDD). When there is data along with the schema, SQL becomes the first choice of processing that data. Spark SQL is Spark's component that enables developers to process data with **Structured Query Language** (**SQL**).

Using Spark SQL, business logic can be easily written in SQL and HQL. This enables data warehouse engineers with a good knowledge of SQL to make use of Spark for their **extract, transform, load** (**ETL**) processing. Hive projects can easily be migrated on Spark using Spark SQL, without changing the Hive scripts.

Spark SQL is also the first choice for data analysis and data warehousing. Spark SQL enables the data analysts to write ad hoc queries for their exploratory analysis. Spark provides Spark SQL shell, where you can run the SQL-like queries and they get executed on Spark. Spark internally converts the code into a chain of RDD computations, while Hive converts the HQL job into a series of MapReduce jobs. Using Spark SQL, developers can also make use of caching (a Spark feature that enables data to be kept in memory), which can significantly increase the performance of their queries.

Spark Streaming

Spark Streaming is a package that is used to process a stream of data in real time. There can be many different types of a real-time stream of data; for example, an e-commerce website recording page visits in real time, credit card transactions, a taxi provider app sending information about trips and location information of drivers and passengers, and more. In a nutshell, all of these applications are hosted on multiple web servers that generate event logs in real time.

Spark Streaming makes use of RDD and defines some more APIs to process the data stream in real time. As Spark Streaming makes use of RDD and its APIs, it is easy for developers to learn and execute the use cases without learning a whole new technology stack.

Spark 2.x introduced **structured streaming**, which makes use of DataFrames rather than RDD to process the data stream. Using DataFrames as its computation abstraction brings all the benefits of the DataFrame API to stream processing. We shall discuss the benefits of DataFrames over RDD in coming chapters.

Spark Streaming has excellent integration with some of the most popular data messaging queues, such as Apache Flume and Kafka. It can be easily plugged into these queues to handle a massive amount of data streams.

Spark machine learning

It is difficult to run a machine-learning algorithm when your data is distributed across multiple machines. There might be a case when the calculation depends on another point that is stored or processed on a different executor. Data can be shuffling across executors or workers, but shuffle comes with a heavy cost. Spark provides a way to avoid shuffling data. Yes, it is caching. Spark's ability to keep a large amount of data in memory makes it easy to write machine-learning algorithms.

Spark **MLlib** and **ML** are the Spark's packages to work with machine-learning algorithms. They provide the following:

- Inbuilt machine-learning algorithms such as Classification, Regression, Clustering, and more
- Features such as pipelining, vector creation, and more

The previous algorithms and features are optimized for data shuffle and to scale across the cluster.

Spark graph processing

Spark also has a component to process graph data. A graph consists of vertices and edges. Edges define the relationship between vertices. Some examples of graph data are customers's product ratings, social networks, Wikipedia pages and their links, airport flights, and more.

Spark provides **GraphX** to process such data. GraphX makes use of RDD for its computation and allows users to create vertices and edges with some properties. Using GraphX, you can define and manipulate a graph or get some insights from the graph.

GraphFrames is an external package that makes use of DataFrames instead of RDD, and defines vertex-edge relation using a DataFrame.

Cluster manager

Spark provides a *local* mode for the job execution, where both driver and executors run within a single JVM on the client machine. This enables developers to quickly get started with Spark without creating a cluster. We will mostly use this mode of job execution throughout this book for our code examples, and explain the possible challenges with a cluster mode whenever possible. Spark also works with a variety of schedules. Let's have a quick overview of them here.

Standalone scheduler

Spark comes with its own scheduler, called a **standalone scheduler**. If you are running your Spark programs on a cluster that does not have a Hadoop installation, then there is a chance that you are using Spark's default standalone scheduler.

YARN

YARN is the default scheduler of Hadoop. It is optimized for batch jobs such as MapReduce, Hive, and Pig. Most of the organizations already have Hadoop installed on their clusters; therefore, Spark provides the ability to configure it with YARN for the job scheduling.

Mesos

Spark also integrates well with Apache Mesos which is build using the same principles as the Linux kernel. Unlike YARN, Apache Mesos is general purpose cluster manager that does not bind to the Hadoop ecosystem. Another difference between YARN and Mesos is that YARN is optimized for the long-running batch workloads, whereas Mesos, ability to provide a fine-grained and dynamic allocation of resources makes it more optimized for streaming jobs.