O'REILLY®

Learning Spark

LIGHTNING-FAST DATA ANALYSIS

Holden Karau, Andy Konwinski,
Patrick Wendell & Matei Zaharia

O'REILLY®

Learning Spark

Data in all domains is getting bigger. How can you work with it efficiently? This book introduces Apache Spark, the open source cluster computing system that makes data analytics fast to write and fast to run. With Spark, you can tackle big datasets quickly through simple APIs in Python, Java, and Scala.

Written by the developers of Spark, this book will have data scientists and engineers up and running in no time. You'll learn how to express parallel jobs with just a few lines of code, and cover applications from simple batch jobs to stream processing and machine learning.

- Quickly dive into Spark capabilities such as distributed datasets, in-memory caching, and the interactive shell
- Leverage Spark's powerful built-in libraries, including Spark SQL, Spark Streaming, and MLlib
- Use one programming paradigm instead of mixing and matching tools like Hive, Hadoop, Mahout, and Storm
- Learn how to deploy interactive, batch, and streaming applications
- Connect to data sources including HDFS, Hive, JSON, and S3
- Master advanced topics like data partitioning and shared variables

"*Learning Spark* is at the top of my list for anyone needing a gentle guide to the most popular framework for building big data applications."

—Ben Lorica
Chief Data Scientist, O'Reilly Media

Holden Karau, a software development engineer at Databricks, is active in open source and the author of *Fast Data Processing with Spark* (Packt Publishing).

Andy Konwinski, co-founder of Databricks, is a committer on Apache Spark and co-creator of the Apache Mesos project.

Patrick Wendell is a co-founder of Databricks and a committer on Apache Spark. He also maintains several subsystems of Spark's core engine.

Matei Zaharia, CTO at Databricks, is the creator of Apache Spark and serves as its Vice President at Apache.

PROGRAMMING LANGUAGES/SPARK

US $39.99 CAN $45.99
ISBN: 978-1-449-35862-4

9 781449 358624 53999

Learning Spark

Holden Karau, Andy Konwinski, Patrick Wendell, and Matei Zaharia

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Learning Spark
by Holden Karau, Andy Konwinski, Patrick Wendell, and Matei Zaharia

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editors: Ann Spencer and Marie Beaugureau
Production Editor: Kara Ebrahim
Copyeditor: Rachel Monaghan
Proofreader: Charles Roumeliotis
Indexer: Ellen Troutman
Interior Designer: David Futato
Cover Designer: Ellie Volckhausen
Illustrator: Rebecca Demarest

February 2015: First Edition

Revision History for the First Edition
2015-01-26: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781449358624* for release details.

978-1-449-35862-4

[LSI]

Table of Contents

CHAPTER 1

Introduction to Data Analysis with Spark

This chapter provides a high-level overview of what Apache Spark is. If you are already familiar with Apache Spark and its components, feel free to jump ahead to Chapter 2.

What Is Apache Spark?

Apache Spark is a cluster computing platform designed to be *fast* and *general-purpose*.

On the speed side, Spark extends the popular MapReduce model to efficiently support more types of computations, including interactive queries and stream processing. Speed is important in processing large datasets, as it means the difference between exploring data interactively and waiting minutes or hours. One of the main features Spark offers for speed is the ability to run computations in memory, but the system is also more efficient than MapReduce for complex applications running on disk.

On the generality side, Spark is designed to cover a wide range of workloads that previously required separate distributed systems, including batch applications, iterative algorithms, interactive queries, and streaming. By supporting these workloads in the same engine, Spark makes it easy and inexpensive to *combine* different processing types, which is often necessary in production data analysis pipelines. In addition, it reduces the management burden of maintaining separate tools.

Spark is designed to be highly accessible, offering simple APIs in Python, Java, Scala, and SQL, and rich built-in libraries. It also integrates closely with other Big Data tools. In particular, Spark can run in Hadoop clusters and access any Hadoop data source, including Cassandra.

A Unified Stack

The Spark project contains multiple closely integrated components. At its core, Spark is a "computational engine" that is responsible for scheduling, distributing, and monitoring applications consisting of many computational tasks across many worker machines, or a *computing cluster*. Because the core engine of Spark is both fast and general-purpose, it powers multiple higher-level components specialized for various workloads, such as SQL or machine learning. These components are designed to interoperate closely, letting you combine them like libraries in a software project.

A philosophy of tight integration has several benefits. First, all libraries and higher-level components in the stack benefit from improvements at the lower layers. For example, when Spark's core engine adds an optimization, SQL and machine learning libraries automatically speed up as well. Second, the costs associated with running the stack are minimized, because instead of running 5–10 independent software systems, an organization needs to run only one. These costs include deployment, maintenance, testing, support, and others. This also means that each time a new component is added to the Spark stack, every organization that uses Spark will immediately be able to try this new component. This changes the cost of trying out a new type of data analysis from downloading, deploying, and learning a new software project to upgrading Spark.

Finally, one of the largest advantages of tight integration is the ability to build applications that seamlessly combine different processing models. For example, in Spark you can write one application that uses machine learning to classify data in real time as it is ingested from streaming sources. Simultaneously, analysts can query the resulting data, also in real time, via SQL (e.g., to join the data with unstructured logfiles). In addition, more sophisticated data engineers and data scientists can access the same data via the Python shell for ad hoc analysis. Others might access the data in standalone batch applications. All the while, the IT team has to maintain only one system.

Here we will briefly introduce each of Spark's components, shown in Figure 1-1.

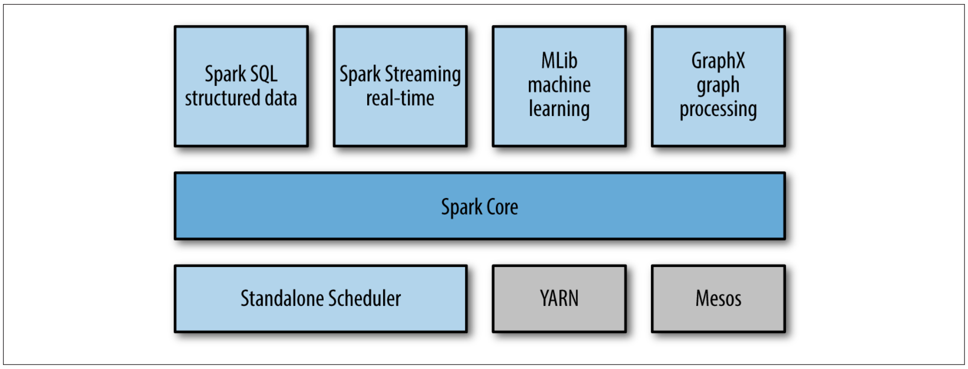

Figure 1-1. The Spark stack

Spark Core

Spark Core contains the basic functionality of Spark, including components for task scheduling, memory management, fault recovery, interacting with storage systems, and more. Spark Core is also home to the API that defines *resilient distributed datasets* (RDDs), which are Spark's main programming abstraction. RDDs represent a collection of items distributed across many compute nodes that can be manipulated in parallel. Spark Core provides many APIs for building and manipulating these collections.

Spark SQL

Spark SQL is Spark's package for working with structured data. It allows querying data via SQL as well as the Apache Hive variant of SQL—called the Hive Query Language (HQL)—and it supports many sources of data, including Hive tables, Parquet, and JSON. Beyond providing a SQL interface to Spark, Spark SQL allows developers to intermix SQL queries with the programmatic data manipulations supported by RDDs in Python, Java, and Scala, all within a single application, thus combining SQL with complex analytics. This tight integration with the rich computing environment provided by Spark makes Spark SQL unlike any other open source data warehouse tool. Spark SQL was added to Spark in version 1.0.

Shark was an older SQL-on-Spark project out of the University of California, Berkeley, that modified Apache Hive to run on Spark. It has now been replaced by Spark SQL to provide better integration with the Spark engine and language APIs.

Spark Streaming

Spark Streaming is a Spark component that enables processing of live streams of data. Examples of data streams include logfiles generated by production web servers, or queues of messages containing status updates posted by users of a web service. Spark

Streaming provides an API for manipulating data streams that closely matches the Spark Core's RDD API, making it easy for programmers to learn the project and move between applications that manipulate data stored in memory, on disk, or arriving in real time. Underneath its API, Spark Streaming was designed to provide the same degree of fault tolerance, throughput, and scalability as Spark Core.

MLlib

Spark comes with a library containing common machine learning (ML) functionality, called MLlib. MLlib provides multiple types of machine learning algorithms, including classification, regression, clustering, and collaborative filtering, as well as supporting functionality such as model evaluation and data import. It also provides some lower-level ML primitives, including a generic gradient descent optimization algorithm. All of these methods are designed to scale out across a cluster.

GraphX

GraphX is a library for manipulating graphs (e.g., a social network's friend graph) and performing graph-parallel computations. Like Spark Streaming and Spark SQL, GraphX extends the Spark RDD API, allowing us to create a directed graph with arbitrary properties attached to each vertex and edge. GraphX also provides various operators for manipulating graphs (e.g., `subgraph` and `mapVertices`) and a library of common graph algorithms (e.g., PageRank and triangle counting).

Cluster Managers

Under the hood, Spark is designed to efficiently scale up from one to many thousands of compute nodes. To achieve this while maximizing flexibility, Spark can run over a variety of *cluster managers*, including Hadoop YARN, Apache Mesos, and a simple cluster manager included in Spark itself called the Standalone Scheduler. If you are just installing Spark on an empty set of machines, the Standalone Scheduler provides an easy way to get started; if you already have a Hadoop YARN or Mesos cluster, however, Spark's support for these cluster managers allows your applications to also run on them. Chapter 7 explores the different options and how to choose the correct cluster manager.

Who Uses Spark, and for What?

Because Spark is a general-purpose framework for cluster computing, it is used for a diverse range of applications. In the Preface we outlined two groups of readers that this book targets: data scientists and engineers. Let's take a closer look at each group and how it uses Spark. Unsurprisingly, the typical use cases differ between the two, but we can roughly classify them into two categories, *data science* and *data applications*.

Of course, these are imprecise disciplines and usage patterns, and many folks have skills from both, sometimes playing the role of the investigating data scientist, and then "changing hats" and writing a hardened data processing application. Nonetheless, it can be illuminating to consider the two groups and their respective use cases separately.

Data Science Tasks

Data science, a discipline that has been emerging over the past few years, centers on analyzing data. While there is no standard definition, for our purposes a *data scientist* is somebody whose main task is to analyze and model data. Data scientists may have experience with SQL, statistics, predictive modeling (machine learning), and programming, usually in Python, Matlab, or R. Data scientists also have experience with techniques necessary to transform data into formats that can be analyzed for insights (sometimes referred to as *data wrangling*).

Data scientists use their skills to analyze data with the goal of answering a question or discovering insights. Oftentimes, their workflow involves ad hoc analysis, so they use interactive shells (versus building complex applications) that let them see results of queries and snippets of code in the least amount of time. Spark's speed and simple APIs shine for this purpose, and its built-in libraries mean that many algorithms are available out of the box.

Spark supports the different tasks of data science with a number of components. The Spark shell makes it easy to do interactive data analysis using Python or Scala. Spark SQL also has a separate SQL shell that can be used to do data exploration using SQL, or Spark SQL can be used as part of a regular Spark program or in the Spark shell. Machine learning and data analysis is supported through the MLLib libraries. In addition, there is support for calling out to external programs in Matlab or R. Spark enables data scientists to tackle problems with larger data sizes than they could before with tools like R or Pandas.

Sometimes, after the initial exploration phase, the work of a data scientist will be "productized," or extended, hardened (i.e., made fault-tolerant), and tuned to become a production data processing application, which itself is a component of a business application. For example, the initial investigation of a data scientist might lead to the creation of a production recommender system that is integrated into a web application and used to generate product suggestions to users. Often it is a different person or team that leads the process of productizing the work of the data scientists, and that person is often an engineer.

Data Processing Applications

The other main use case of Spark can be described in the context of the engineer persona. For our purposes here, we think of engineers as a large class of software developers who use Spark to build production data processing applications. These developers usually have an understanding of the principles of software engineering, such as encapsulation, interface design, and object-oriented programming. They frequently have a degree in computer science. They use their engineering skills to design and build software systems that implement a business use case.

For engineers, Spark provides a simple way to parallelize these applications across clusters, and hides the complexity of distributed systems programming, network communication, and fault tolerance. The system gives them enough control to monitor, inspect, and tune applications while allowing them to implement common tasks quickly. The modular nature of the API (based on passing distributed collections of objects) makes it easy to factor work into reusable libraries and test it locally.

Spark's users choose to use it for their data processing applications because it provides a wide variety of functionality, is easy to learn and use, and is mature and reliable.

A Brief History of Spark

Spark is an open source project that has been built and is maintained by a thriving and diverse community of developers. If you or your organization are trying Spark for the first time, you might be interested in the history of the project. Spark started in 2009 as a research project in the UC Berkeley RAD Lab, later to become the AMPLab. The researchers in the lab had previously been working on Hadoop MapReduce, and observed that MapReduce was inefficient for iterative and interactive computing jobs. Thus, from the beginning, Spark was designed to be fast for interactive queries and iterative algorithms, bringing in ideas like support for in-memory storage and efficient fault recovery.

Research papers were published about Spark at academic conferences and soon after its creation in 2009, it was already 10–20× faster than MapReduce for certain jobs.

Some of Spark's first users were other groups inside UC Berkeley, including machine learning researchers such as the Mobile Millennium project, which used Spark to monitor and predict traffic congestion in the San Francisco Bay Area. In a very short time, however, many external organizations began using Spark, and today, over 50 organizations list themselves on the Spark PoweredBy page, and dozens speak about their use cases at Spark community events such as Spark Meetups and the Spark Summit. In addition to UC Berkeley, major contributors to Spark include Databricks, Yahoo!, and Intel.

In 2011, the AMPLab started to develop higher-level components on Spark, such as Shark (Hive on Spark)[1] and Spark Streaming. These and other components are sometimes referred to as the Berkeley Data Analytics Stack (BDAS).

Spark was first open sourced in March 2010, and was transferred to the Apache Software Foundation in June 2013, where it is now a top-level project.

Spark Versions and Releases

Since its creation, Spark has been a very active project and community, with the number of contributors growing with each release. Spark 1.0 had over 100 individual contributors. Though the level of activity has rapidly grown, the community continues to release updated versions of Spark on a regular schedule. Spark 1.0 was released in May 2014. This book focuses primarily on Spark 1.1.0 and beyond, though most of the concepts and examples also work in earlier versions.

Storage Layers for Spark

Spark can create distributed datasets from any file stored in the Hadoop distributed filesystem (HDFS) or other storage systems supported by the Hadoop APIs (including your local filesystem, Amazon S3, Cassandra, Hive, HBase, etc.). It's important to remember that Spark does not require Hadoop; it simply has support for storage systems implementing the Hadoop APIs. Spark supports text files, SequenceFiles, Avro, Parquet, and any other Hadoop InputFormat. We will look at interacting with these data sources in Chapter 5.

1 Shark has been replaced by Spark SQL.