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.

“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

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

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.
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

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.

—Ben Lorica
Chief Data Scientist, O’Reilly Media

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.

Holden Karau, Andy Konwinski, Patrick Wendell & Matei Zaharia

LIGHTNING-FAST DATA ANALYSIS
Holden Karau, Andy Konwinski, Patrick Wendell, and Matei Zaharia
# Table of Contents

Foreword ................................................................. ix
Preface ................................................................. xi

1. **Introduction to Data Analysis with Spark.** ........................................ 1
   - What Is Apache Spark? ........................................... 1
   - A Unified Stack .................................................. 2
     - Spark Core ................................................. 3
     - Spark SQL .................................................. 3
     - Spark Streaming .......................................... 3
   - MLlib .......................................................... 4
   - GraphX ......................................................... 4
   - Cluster Managers ............................................ 4
   - Who Uses Spark, and for What? ................................ 4
     - Data Science Tasks ....................................... 5
     - Data Processing Applications ............................ 6
   - A Brief History of Spark ........................................ 6
   - Spark Versions and Releases ................................ 7
   - Storage Layers for Spark ...................................... 7

2. **Downloading Spark and Getting Started.** ........................................ 9
   - Downloading Spark ........................................... 9
   - Introduction to Spark’s Python and Scala Shells .................. 11
   - Introduction to Core Spark Concepts .......................... 14
   - Standalone Applications ...................................... 17
     - Initializing a SparkContext .................................. 17
     - Building Standalone Applications ......................... 18
   - Conclusion ..................................................... 21
3. Programming with RDDs
   RDD Basics 23
   Creating RDDs 25
   RDD Operations 26
      Transformations 27
      Actions 28
      Lazy Evaluation 29
   Passing Functions to Spark 30
      Python 30
      Scala 31
      Java 32
   Common Transformations and Actions 34
      Basic RDDs 34
      Converting Between RDD Types 42
   Persistence (Caching) 44
   Conclusion 46

4. Working with Key/Value Pairs
   Motivation 47
   Creating Pair RDDs 48
   Transformations on Pair RDDs 49
      Aggregations 51
      Grouping Data 57
      Joins 58
      Sorting Data 59
   Actions Available on Pair RDDs 60
   Data Partitioning (Advanced) 61
      Determining an RDD’s Partitioner 64
      Operations That Benefit from Partitioning 65
      Operations That Affect Partitioning 65
      Example: PageRank 66
      Custom Partitioners 68
   Conclusion 70

5. Loading and Saving Your Data
   Motivation 71
   File Formats 72
      Text Files 73
      JSON 74
      Comma-Separated Values and Tab-Separated Values 77
      SequenceFiles 80
      Object Files 83

iv | Table of Contents
10. **Spark Streaming**................................................................. 183
    - A Simple Example .................................................. 184
    - Architecture and Abstraction ...................................... 186
    - Transformations ..................................................... 189
      - Stateless Transformations ....................................... 190
      - Stateful Transformations ......................................... 192
    - Output Operations .................................................. 197
    - Input Sources ....................................................... 199
      - Core Sources ...................................................... 199
      - Additional Sources .............................................. 200
    - Multiple Sources and Cluster Sizing ................................ 204
    - 24/7 Operation ....................................................... 205
      - Checkpointing .................................................... 205
      - Driver Fault Tolerance ........................................... 206
      - Worker Fault Tolerance ........................................... 207
      - Receiver Fault Tolerance ......................................... 207
      - Processing Guarantees ........................................... 208
    - Streaming UI .......................................................... 208
    - Performance Considerations ....................................... 209
      - Batch and Window Sizes ........................................... 209
      - Level of Parallelism ............................................... 210
    - Garbage Collection and Memory Usage ................................ 210
    - Conclusion ............................................................ 211

11. **Machine Learning with MLlib**.............................................. 213
    - Overview ............................................................... 213
    - System Requirements ................................................ 214
    - Machine Learning Basics ............................................ 215
      - Example: Spam Classification ...................................... 216
    - Data Types ............................................................. 218
      - Working with Vectors .............................................. 219
    - Algorithms ............................................................. 220
      - Feature Extraction .................................................. 221
      - Statistics .............................................................. 223
      - Classification and Regression .................................... 224
      - Clustering ............................................................. 229
      - Collaborative Filtering and Recommendation .................. 230
      - Dimensionality Reduction ......................................... 232
      - Model Evaluation ................................................... 234
    - Tips and Performance Considerations ................................ 234
      - Preparing Features ................................................. 234
      - Configuring Algorithms ......................................... 235

Table of Contents | vii
Index
In a very short time, Apache Spark has emerged as the next generation big data processing engine, and is being applied throughout the industry faster than ever. Spark improves over Hadoop MapReduce, which helped ignite the big data revolution, in several key dimensions: it is much faster, much easier to use due to its rich APIs, and it goes far beyond batch applications to support a variety of workloads, including interactive queries, streaming, machine learning, and graph processing.

I have been privileged to be closely involved with the development of Spark all the way from the drawing board to what has become the most active big data open source project today, and one of the most active Apache projects! As such, I’m particularly delighted to see Matei Zaharia, the creator of Spark, teaming up with other longtime Spark developers Patrick Wendell, Andy Konwinski, and Holden Karau to write this book.

With Spark’s rapid rise in popularity, a major concern has been lack of good reference material. This book goes a long way to address this concern, with 11 chapters and dozens of detailed examples designed for data scientists, students, and developers looking to learn Spark. It is written to be approachable by readers with no background in big data, making it a great place to start learning about the field in general. I hope that many years from now, you and other readers will fondly remember this as the book that introduced you to this exciting new field.

—Ion Stoica, CEO of Databricks and Co-director, AMPlab, UC Berkeley
As parallel data analysis has grown common, practitioners in many fields have sought easier tools for this task. Apache Spark has quickly emerged as one of the most popular, extending and generalizing MapReduce. Spark offers three main benefits. First, it is easy to use—you can develop applications on your laptop, using a high-level API that lets you focus on the content of your computation. Second, Spark is fast, enabling interactive use and complex algorithms. And third, Spark is a general engine, letting you combine multiple types of computations (e.g., SQL queries, text processing, and machine learning) that might previously have required different engines. These features make Spark an excellent starting point to learn about Big Data in general.

This introductory book is meant to get you up and running with Spark quickly. You’ll learn how to download and run Spark on your laptop and use it interactively to learn the API. Once there, we’ll cover the details of available operations and distributed execution. Finally, you’ll get a tour of the higher-level libraries built into Spark, including libraries for machine learning, stream processing, and SQL. We hope that this book gives you the tools to quickly tackle data analysis problems, whether you do so on one machine or hundreds.

**Audience**

This book targets data scientists and engineers. We chose these two groups because they have the most to gain from using Spark to expand the scope of problems they can solve. Spark’s rich collection of data-focused libraries (like MLlib) makes it easy for data scientists to go beyond problems that fit on a single machine while using their statistical background. Engineers, meanwhile, will learn how to write general-purpose distributed programs in Spark and operate production applications. Engineers and data scientists will both learn different details from this book, but will both be able to apply Spark to solve large distributed problems in their respective fields.
Data scientists focus on answering questions or building models from data. They often have a statistical or math background and some familiarity with tools like Python, R, and SQL. We have made sure to include Python and, where relevant, SQL examples for all our material, as well as an overview of the machine learning and library in Spark. If you are a data scientist, we hope that after reading this book you will be able to use the same mathematical approaches to solve problems, except much faster and on a much larger scale.

The second group this book targets is software engineers who have some experience with Java, Python, or another programming language. If you are an engineer, we hope that this book will show you how to set up a Spark cluster, use the Spark shell, and write Spark applications to solve parallel processing problems. If you are familiar with Hadoop, you have a bit of a head start on figuring out how to interact with HDFS and how to manage a cluster, but either way, we will cover basic distributed execution concepts.

Regardless of whether you are a data scientist or engineer, to get the most out of this book you should have some familiarity with one of Python, Java, Scala, or a similar language. We assume that you already have a storage solution for your data and we cover how to load and save data from many common ones, but not how to set them up. If you don’t have experience with one of those languages, don’t worry: there are excellent resources available to learn these. We call out some of the books available in “Supporting Books” on page xii.

How This Book Is Organized

The chapters of this book are laid out in such a way that you should be able to go through the material front to back. At the start of each chapter, we will mention which sections we think are most relevant to data scientists and which sections we think are most relevant for engineers. That said, we hope that all the material is accessible to readers of either background.

The first two chapters will get you started with getting a basic Spark installation on your laptop and give you an idea of what you can accomplish with Spark. Once we’ve got the motivation and setup out of the way, we will dive into the Spark shell, a very useful tool for development and prototyping. Subsequent chapters then cover the Spark programming interface in detail, how applications execute on a cluster, and higher-level libraries available on Spark (such as Spark SQL and MLlib).

Supporting Books

If you are a data scientist and don’t have much experience with Python, the books *Learning Python* and *Head First Python* (both O’Reilly) are excellent introductions. If
you have some Python experience and want more, *Dive into Python* (Apress) is a
great book to help you get a deeper understanding of Python.

If you are an engineer and after reading this book you would like to expand your data
analysis skills, *Machine Learning for Hackers* and *Doing Data Science* are excellent
books (both O’Reilly).

This book is intended to be accessible to beginners. We do intend to release a deep‐
dive follow-up for those looking to gain a more thorough understanding of Spark’s
internals.

**Conventions Used in This Book**

The following typographical conventions are used in this book:

*Italic*  
Indicates new terms, URLs, email addresses, filenames, and file extensions.

*Constant width*  
Used for program listings, as well as within paragraphs to refer to program ele‐
ments such as variable or function names, databases, data types, environment
variables, statements, and keywords.

*Constant width bold*  
Shows commands or other text that should be typed literally by the user.

*Constant width italic*  
Shows text that should be replaced with user-supplied values or by values deter‐
mined by context.

This element signifies a tip or suggestion.

This element indicates a warning or caution.

**Code Examples**

All of the code examples found in this book are on GitHub. You can examine them
and check them out from [https://github.com/databricks/learning-spark](https://github.com/databricks/learning-spark). Code exam‐
pies are provided in Java, Scala, and Python.
Our Java examples are written to work with Java version 6 and higher. Java 8 introduces a new syntax called *lambdas* that makes writing inline functions much easier, which can simplify Spark code. We have chosen not to take advantage of this syntax in most of our examples, as most organizations are not yet using Java 8. If you would like to try Java 8 syntax, you can see the Databricks blog post on this topic. Some of the examples will also be ported to Java 8 and posted to the book’s GitHub site.

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you’re reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O’Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product’s documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: “Learning Spark by Holden Karau, Andy Konwinski, Patrick Wendell, and Matei Zaharia (O’Reilly). Copyright 2015 Databricks, 978-1-449-35862-4.”

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at permissions@oreilly.com.

**Safari® Books Online**

*Safari Books Online* is an on-demand digital library that delivers expert content in both book and video form from the world’s leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of plans and pricing for enterprise, government, education, and individuals.

Members have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O’Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at http://bit.ly/learning-spark.

To comment or ask technical questions about this book, send email to bookquestions@oreilly.com.

For more information about our books, courses, conferences, and news, see our website at http://www.oreilly.com.

Find us on Facebook: http://facebook.com/oreilly

Follow us on Twitter: http://twitter.com/oreillymedia

Watch us on YouTube: http://www.youtube.com/oreillymedia

Acknowledgments

The authors would like to thank the reviewers who offered feedback on this book: Joseph Bradley, Dave Bridgeland, Chaz Chandler, Mick Davies, Sam DeHority, Vida Ha, Andrew Gal, Michael Gregson, Jan Jopeen, Stephan Jou, Jeff Martinez, Josh Mahonin, Andrew Or, Mike Patterson, Josh Rosen, Bruce Szalwinski, Xiangrui Meng, and Reza Zadeh.

The authors would like to extend a special thanks to David Andrzejewski, David Butler, Juliet Hougland, Marek Kolodziej, Taka Shinagawa, Deborah Siegel, Dr. Normen Müller, Ali Ghodsi, and Sameer Farooqui. They provided detailed feedback on the majority of the chapters and helped point out many significant improvements.

We would also like to thank the subject matter experts who took time to edit and write parts of their own chapters. Tathagata Das worked with us on a very tight schedule to finish Chapter 10. Tathagata went above and beyond with clarifying
examples, answering many questions, and improving the flow of the text in addition to his technical contributions. Michael Armbrust helped us check the Spark SQL chapter for correctness. Joseph Bradley provided the introductory example for MLlib in Chapter 11. Reza Zadeh provided text and code examples for dimensionality reduction. Xiangrui Meng, Joseph Bradley, and Reza Zadeh also provided editing and technical feedback for the MLlib chapter.
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.
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 10–20x 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.