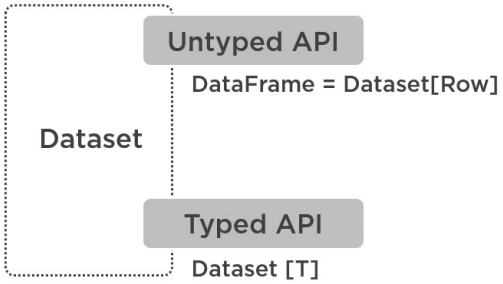
Developing Spark Applications Using Scala & Cloudera

WHY SPARK WITH SCALA & CLOUDERA?

Xavier Morera
HELPING DEVELOPERS UNDERSTAND SEARCH & BIG DATA
@xmorera www.xaviermorera.com



WHY SPARK WITH SCALA & CLOUDERA?

Spark 2

Scala

cloudera

But Why Apache Spark?

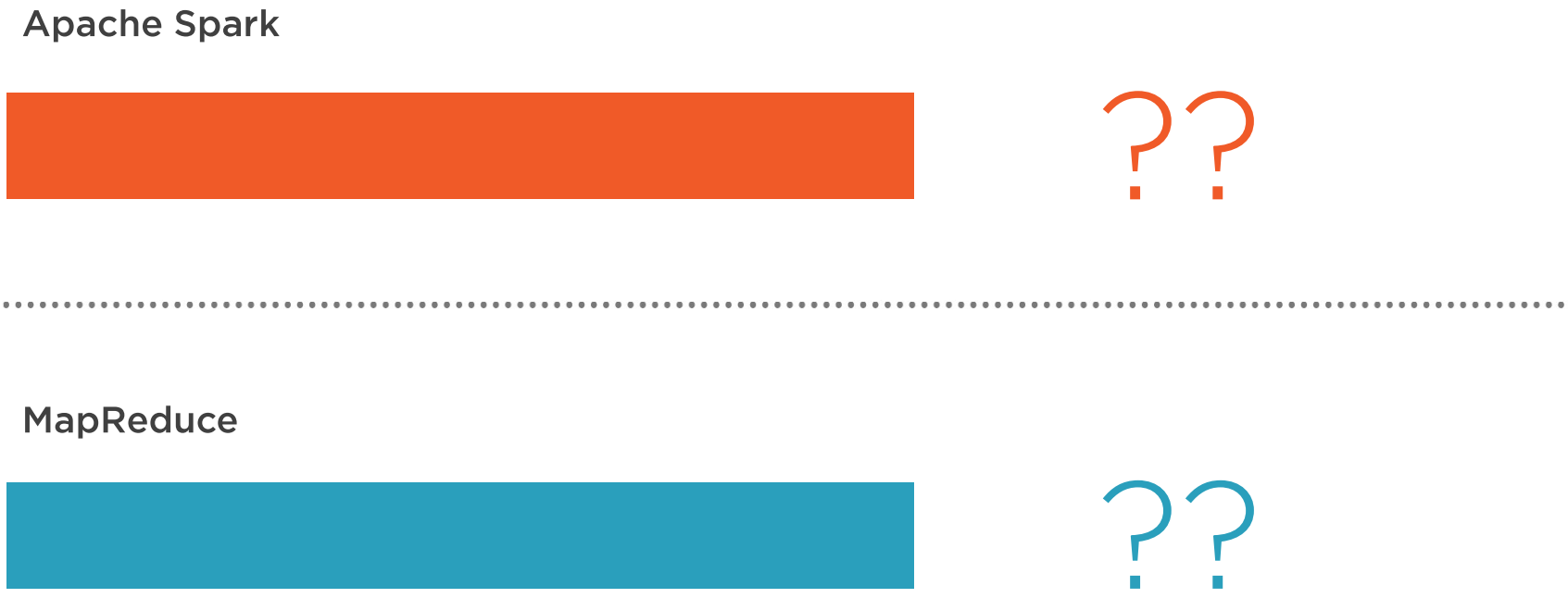
Why Should I Use Apache Spark?


BIG DATA

Speed

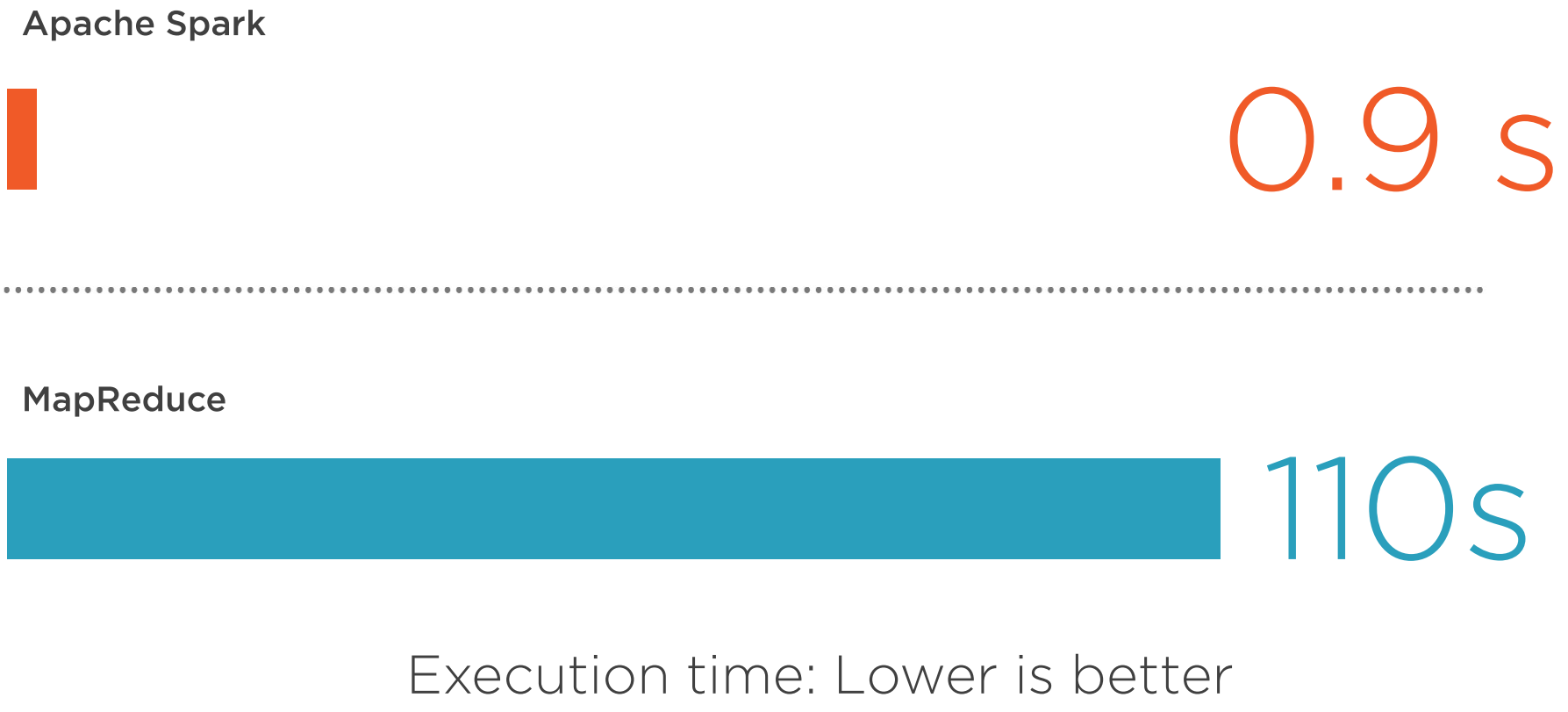
But Why Apache Spark?

Logistic Regression: Hadoop vs. Spark



Execution time: Lower is better

Logistic Regression: Hadoop vs. Spark



Execution time: Lower is better

Ease of Use

But Why Apache Spark?

```
val lines = sc.textFile("/user/cloudera/sparkcourse/")

lines.flatMap(line => line.split(" "))
     .map(word => (word, 1))
     .reduceByKey(_ + _)
```

Word Count Example

Many lines in Hadoop

A couple of lines in Spark

Easy to learn and full of features

Unified Engine for Big Data

But Why Apache Spark?

Spark SQL

Spark Streaming

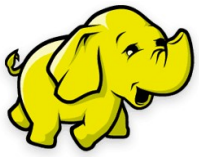
MLlib

GraphX

spark-packages.org



1









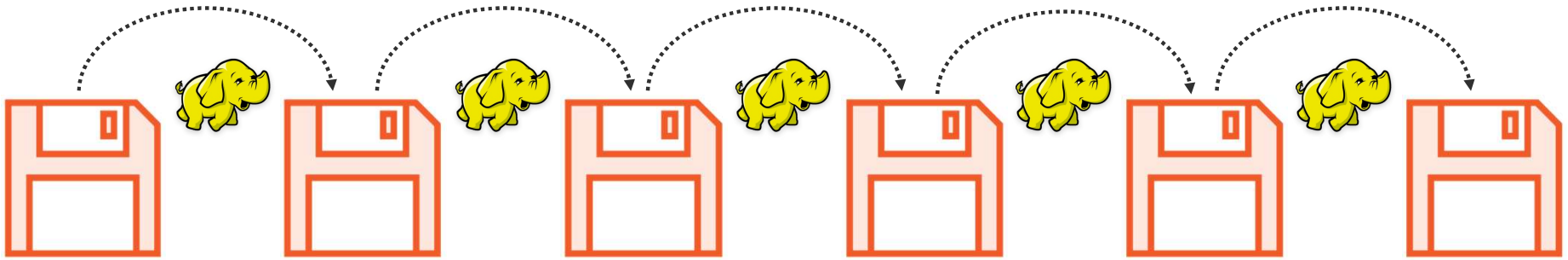


iterative algorithms

+

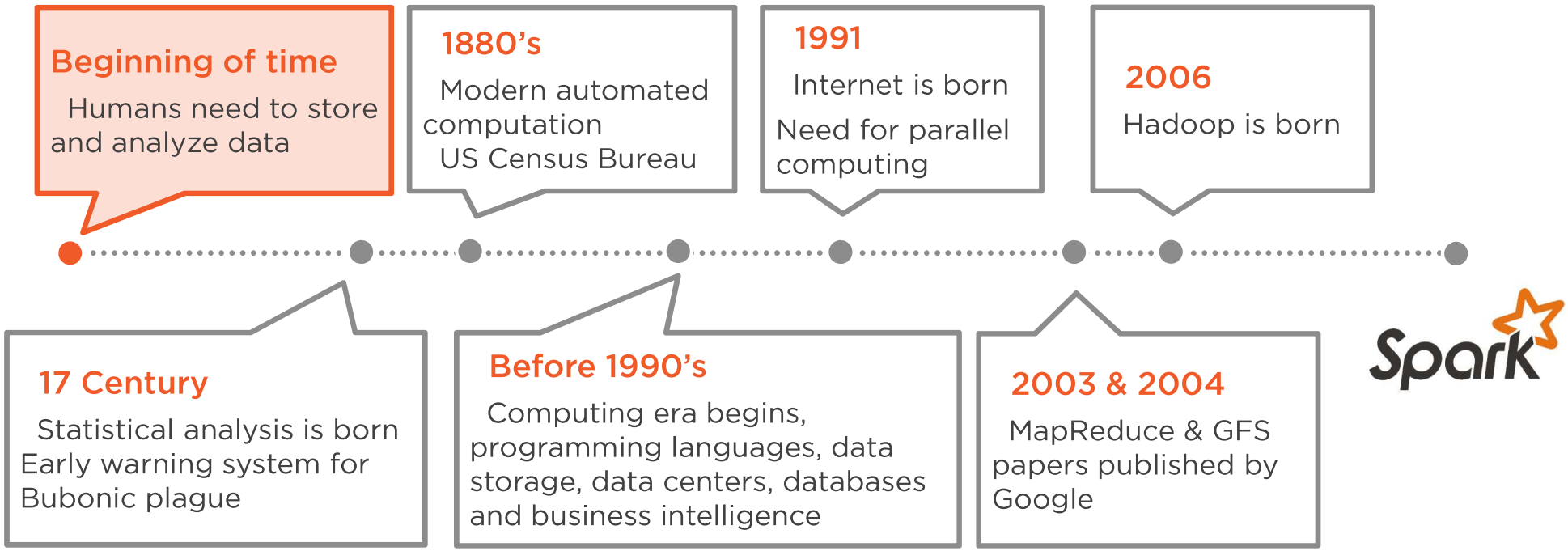
interactive data mining tools

Motivation behind Spark, as stated in the original paper

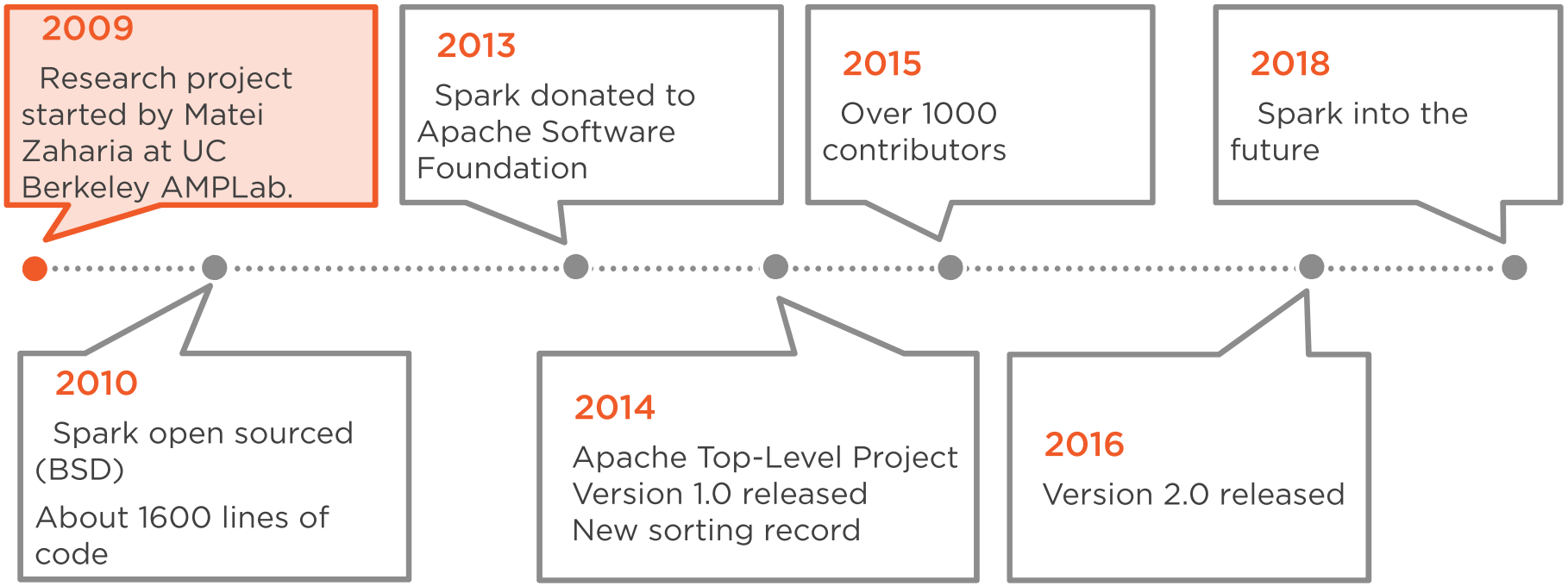



Spark

History of "What Led to Spark" in Big Data



History of Apache Spark



Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing

Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica

University of California, Berkeley

Abstract

We present Resilient Distributed Datasets (RDDs), a distributed memory abstraction that lets programmers perform in-memory computations on large clusters in a fault-tolerant manner. RDDs are motivated by two types of applications that current computing frameworks handle inefficiently: iterative algorithms and interactive data mining tools. In both cases, keeping data in memory can improve performance by an order of magnitude. To achieve fault tolerance efficiently, RDDs provide a restricted form of shared memory, based on coarse-grained transformations rather than fine-grained updates to shared state. However, we show that RDDs are expressive enough to capture a wide class of computations, including recent specialized programming models for iterative jobs, such as Pregel, and new applications that these models do not capture. We have implemented RDDs in a system called Spark, which we evaluate through a variety of user applications and benchmarks.

1 Introduction

Cluster computing frameworks like MapReduce [10] and Dryad [19] have been widely adopted for large-scale data analysis. These systems let users write parallel computations using a set of high-level operators, without having to worry about work distribution and fault tolerance.

Although current frameworks provide numerous abstractions for accessing a cluster's computational resources, they lack abstractions for leveraging distributed memory. This makes them inefficient for an important class of emerging applications: those that reuse intermediate results across multiple computations. Data reuse is common in many iterative machine learning and graph algorithms, including PageRank, K-means clustering, and logistic regression. Another compelling use case is interactive data mining, where a user runs multiple ad-hoc queries on the same subset of the data. Unfortunately, in most current frameworks, the only way to reuse data between computations (e.g., between two MapReduce jobs) is to write it to an external stable storage system, e.g., a distributed file system. This incurs substantial overheads due to data replication, disk I/O, and serialization, which can dominate application execution times.

Recognizing this problem, researchers have developed specialized frameworks for some applications that require data reuse. For example, Pregel [22] is a system for iterative graph computations that keeps intermediate data in memory, while HaLoop [7] offers an iterative MapReduce interface. However, these frameworks only support specific computation patterns (e.g., looping a series of MapReduce steps), and perform data sharing implicitly for these patterns. They do not provide abstractions for more general reuse, e.g., to let a user load several datasets into memory and run ad-hoc queries across them.

In this paper, we propose a new abstraction called *resilient distributed datasets* (RDDs) that enables efficient data reuse in a broad range of applications. RDDs are fault-tolerant, parallel data structures that let users explicitly persist intermediate results in memory, control their partitioning to optimize data placement, and manipulate them using a rich set of operators.

The main challenge in designing RDDs is defining a programming interface that can provide fault tolerance efficiently. Existing abstractions for in-memory storage on clusters, such as distributed shared memory [24], key-value stores [25], databases, and Piccolo [27], offer an interface based on fine-grained updates to mutable state (e.g., cells in a table). With this interface, the only ways to provide fault tolerance are to replicate the data across machines or to log updates across machines. Both approaches are expensive for data-intensive workloads, as they require copying large amounts of data over the cluster network, whose bandwidth is far lower than that of RAM, and they incur substantial storage overhead.

In contrast to these systems, RDDs provide an interface based on coarse-grained transformations (e.g., map, filter and join) that apply the same operation to many data items. This allows them to efficiently provide fault tolerance by logging the transformations used to build a dataset (its *lineage*) rather than the actual data.[1] If a partition of an RDD is lost, the RDD has enough information about how it was derived from other RDDs to recompute

[1] Checkpointing the data in some RDDs may be useful when a lineage chain grows large, however, and we discuss how to do this [34].

2012

http://tiny.bigdatainc.org/rddpaper

Index of /releases

[ICO]	Name	Last modified	Size	Description
[PARENTDIR]	Parent Directory		-	
[TXT]	spark-release-0-3.html	2017-10-18 05:57	11K	
[TXT]	spark-release-0-5-0.html	2017-10-18 05:57	10K	
[TXT]	spark-release-0-5-1.html	2017-10-18 05:57	9.6K	
[TXT]	spark-release-0-5-2.html	2017-10-18 05:57	8.1K	
[TXT]	spark-release-0-6-0.html	2017-10-18 05:57	13K	
[TXT]	spark-release-0-6-1.html	2017-10-18 05:57	9.3K	
[TXT]	spark-release-0-6-2.html	2017-10-18 05:57	9.7K	
[TXT]	spark-release-0-7-0.html	2017-10-18 05:57	15K	
[TXT]	spark-release-0-7-2.html	2017-10-18 05:57	10K	
[TXT]	spark-release-0-7-3.html	2017-10-18 05:57	10K	
[TXT]	spark-release-0-8-0.html	2017-10-18 05:57	20K	
[TXT]	spark-release-0-8-1.html	2017-10-18 05:57	15K	
[TXT]	spark-release-0-9-0.html	2017-10-18 05:57	22K	
[TXT]	spark-release-0-9-1.html	2017-10-18 05:57	16K	
[TXT]	spark-release-0-9-2.html	2017-10-18 05:57	12K	
[TXT]	spark-release-1-0-0.html	2017-10-18 05:57	23K	
[TXT]	spark-release-1-0-1.html	2017-10-18 05:57	16K	
[TXT]	spark-release-1-0-2.html	2017-10-18 05:57	13K	
[TXT]	spark-release-1-1-0.html	2017-10-18 05:57	25K	
[TXT]	spark-release-1-1-1.html	2017-10-18 05:57	14K	
[TXT]	spark-release-1-2-0.html	2017-10-18 05:57	29K	
[TXT]	spark-release-1-2-1.html	2017-10-18 05:57	14K	
[TXT]	spark-release-1-2-2.html	2017-10-18 05:57	12K	
[TXT]	spark-release-1-3-0.html	2017-10-18 05:57	28K	
[TXT]	spark-release-1-3-1.html	2017-10-18 05:57	14K	
[TXT]	spark-release-1-4-0.html	2017-10-18 05:57	33K	
[TXT]	spark-release-1-4-1.html	2017-10-18 05:57	16K	

Spark Release 0.3

Spark Release 0.6.0

Spark Release 0.7.0

Spark Release 0.8.0

Spark Release 1.0.0

Spark Release 1.3.0

Spark Release 2.0.0

Spark Release 2.2.0

Apache Spark 2.2.0 is the third release on the 2.x line. This release removes the experimental tag from Structured Streaming. In addition, this release focuses more on usability, stability, and polish, resolving over 1100 tickets.

Additionally, we are excited to announce that PySpark is now available in pypi. To install just run `pip install pyspark`.

To download Apache Spark 2.2.0, visit the downloads page. You can consult JIRA for the detailed changes. We have curated a list of high level changes here, grouped by major modules.

- Core and Spark SQL
- Structured Streaming
- MLlib
- SparkR

Where does the name 'Spark' come from?

What We Will Cover in This Training

Getting an Environment & Data: CDH + StackOverflow

Refreshing Your Knowledge: Scala Fundamentals for This Course

Understanding Spark: An Overview

Getting Technical with Spark

What We Will Cover in This Training

Learning the Core of Spark: RDDs

Going Deeper into Spark Core

Increasing Proficiency with Spark: DataFrames & Spark SQL

Continuing the Journey on DataFrames and Spark SQL

Understanding a Typed API: Datasets

Picking a Spark Supported Language

Scala: type safe, runs on Java VM, concise (good for interactive use)

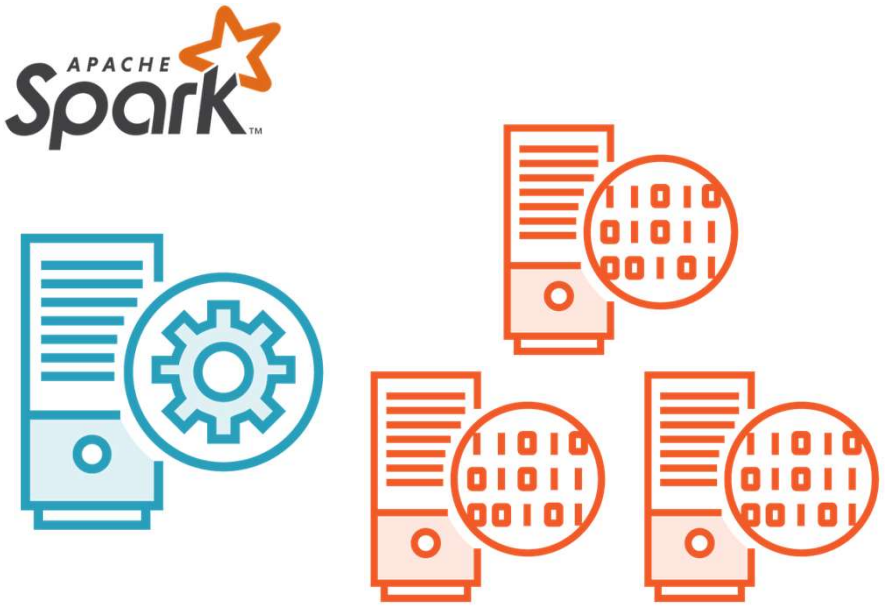
Python: easy to use/read, popular, elegant syntax, REPL, many libraries

Java: language of choice for open source and Big Data

R: statistical analysis, many libraries, taught widely, used in Data Science

What Do You Need for This Course?

Spark Running Somewhere



cloudera

Why Cloudera?

CDH + Tools

On-prem & Cloud

Director

Cloud

Altus

Platform-as-a-Service

A few courses that might help you

Takeaway

Spark + Scala + Cloudera

History of Spark: From RDDs to today

Overview of what's coming

A few related courses that might help