The R in Spark

Javier Luraschi

2018-02-21

Contents

The R in Spark

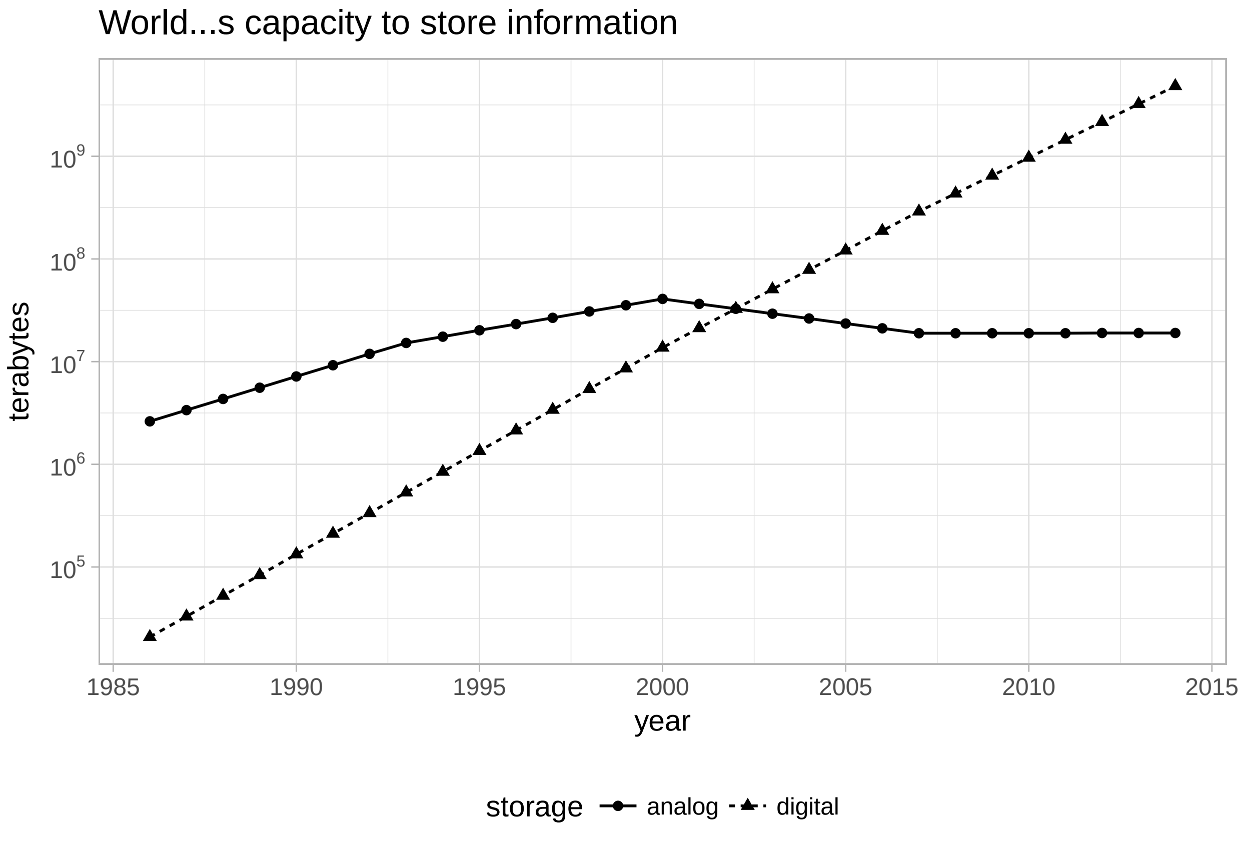
In this book you will learn Apache Spark using R. The book intends to take someone unfamiliar with Spark or R and help them become intermediate users by teaching a set of tools, skills and practices applicable to data science.

Chapter 1

Introduction

Humans have been storing, retrieving, manipulating, and communicating information since the Sumerians in Mesopotamia developed writing in about 3000 BC. Based on the storage and processing technologies employed, it is possible to distinguish four distinct phases development: pre-mechanical (3000 BC – 1450 AD), mechanical (1450–1840), electromechanical (1840–1940), and electronic (1940–present).

As humanity moves from traditional industries to an economy based on information technology our footprint of digital information has kept growing at exponential rates:



With the ambition to provide a searchable tool to all this new digital information, many companies attempted to provide such functionality with what we now know as web search or search engines. Managing information at this scale was a challenging problem that companies had to tackle from the very beginning. Given the vast amount of digital information, search engiens were unble to store all the web page information required to support web searches in a single computer. This meant that they had to split information across many machines, which was accomlished by splitting this data and storing it as many files across many machines, this approach became known as the Google File System from a research paper published in 2003 by Google which has served for others to build on.

One year later, in 2004, Google published a new paper describing how to perform operations across the Google File System, this approach came to be known as **MapReduce**. As you would expect, there are two operations in MapReduce: Map and Reduce. We can think of the mapping operation as a way to transform each file into a new file and, reuduce as a way of combining two files into a new one. It happens to be the case that using these two operations is sufficient to perform interesting operations; for instance, MapReduce can be used to rank web pages efficietly across a cluster of machines.

Since the papers were released by Google, a team in Yahoo worked on implementing the Google File System and MapReduce as free open source projects. This project was released in 2006 as **Hadoop** and the Google File System became implemented as the Hadoop File System, or HDFS for short. The Hadoop project made distributed file-based computing accessible to many users and organizations.

While Hadoop provided support to perform map/reduce operations over a distributed file system, it still required each map/reduce operation to be written with code every time a data analysys was run. The **Hive** project, released in 2008 by Facebook, brought Structured Query Language (SQL) support to Hadoop. This meant that data analysis could now be performed at large-scale without the need to write code for each map/reduce operation, but instead, one could write generic data analysis statements that are much easier to understand and write.

1.1 Spark

While Hadoop with Hive was a powerful tool, it was still working over a distributed file system and was dependent on map/reduce operations. This meant that it was running using disk drives which tend to be significantly slower than using a computer's memory. In 2009, the **Apache Spark** projects starts in Berkeley to improve over Hadoop. Specifically, by making use of memory (instead of disk drives) and by providing a richer set of verbs beyond map/reduce, this allowed it to be much faster and generic than its predecessor. For instance, one can sort 100TB of data in 72min and 2100 computers using Hadoop, but only 206 computers in 23min using Spark. Spark was build using the Scala programming language, but interfaces to other programming languages are also provided today. Spark was released as an open source project in 2010 with the scope of the project defined as follows:

> "Apache Spark is a fast and general engine for large-scale data processing."
>
> — spark.apache.org

meaning that Spark is a tool designed to support:

- **Data Processing**: Data processing is the collection and manipulation of items of data to produce meaningful information (French, 1996).
- **Large-Scale**: What *large* means is hard to quantify, but one can interpret this as cluster-scale instead, which represents a set of connected computers that work together.
- **General**: Spark optimizes and executes parallel generic code, as in, there is no restriction as to what type of code one can write in Spark.
- **Fast**: Spark is much faster than it's predecesor by making efficient use of memory to speed data access while running algorithms at scale.

Spark is good at tackling large-scale data processing problems, this usually known as **big data** (data sets that are more voluminous and complex that traditional ones, but also is good at tackling large-scale computation problems, known as **big compute** (tools and approaches using a large amount of CPU and memory resources in a coordinated way). There is a third problem space where data nor compute are necessarily large scale and yet, there are significant bennefits from using the same tools.

Big data and big compute problems are usually easy to spot, if the data does not fit into a single machine, you might have a big data problem; if the data fits into a single machine but a process over the data takes days, weeks or months to compute, you might have a big compute problem.

For the third problem space, there are a few use cases this breaks to:

1. **Velocity**: One can have a dataset of 10GB in size and a process that takes 30min to run over this data, this is by no means big-compute nor big-data; however, if a data scientist is researching ways to improve accuracy for their models, reducing the runtime down to 3min it's a 10X improvement, this improvement can lead to significant advances and productivity gains by increasing the velocity at which one can analize data.

2. **Variery**: One can have an efficient process to collect data from many sources into a single location, usually a database, this process could be already running efficiently and close to realtime. Such processese are known at ETL (Extract-Transform-Load); data is extracted from multiple sources, transformed to the required format and loaded in a single data store. While this has worked for years, the tradeoff from this system is that adding a new data source is expensive, the system is centralized and tightly controlled. Since making changes to this type of systems could cause the entire process to come to a halt, adding new data sources usually takes long to be implemented. Instead, one can store all data its natural format and process it as needed using cluster computing, this architecture is currently known as a data lake.

Some people reffer to some of these bennefits as the four 'V's of big data: Velocity, Variety, Volume and Veracity. Others have gone as far as expending this to five or even as the 10 Vs of Big Data. Mnemonics set aside, cluster computing is being used today in more innovative ways and and is not uncommon to see organizations experimenting with new workflows and a vareity of tasks that were traditionally uncommon for cluster computing. Much of the hype attributed to big data falls into this space, where some will argue that everything should be considered big data and where others will argue than almost nothing should. My hope is that this book will help you understand the opportunities and limitations of Apache Spark with R.

1.2 R

R is a computing language with it's inception dating back to Bell Laboratories. At that time, computing was done by calling Fortran subroutines which, apparently, were not pleasant to deal with. The S computing language was designed as an interface language to support higher abstractions to perform statistical computing over existing subroutines:

R is a modern and free implementation of S, specifically:

> R is a programming language and free software environment for statistical computing and graphics.
>
> — The R Project for Statistical Computing

There are two strong arguments for choosing R over other computing languages while working with data:

- The **R Language** was designed by statisticians for statisticians, meaning, this is one of the few successful languages designed for non-programmers; so learning R will probably feel more natural. Additioanlly, since the R language was designed to be an interface to other tools and languages, R allows you to focus more on modeling and less on the pecularities of computer science and engineering.
- The **R Community** provives a rich package archive provided by CRAN (The Comprehensive R Archive Network) which allows you to install ready-to-use packages to perform many tasks, most notably, high-quality statistic models with many only available in R. In addition, the R community is a welcoming and active group of talented individuals motivited to help you succeed. Many packages provided by the R community make R, by far, the place to do statistical computing.

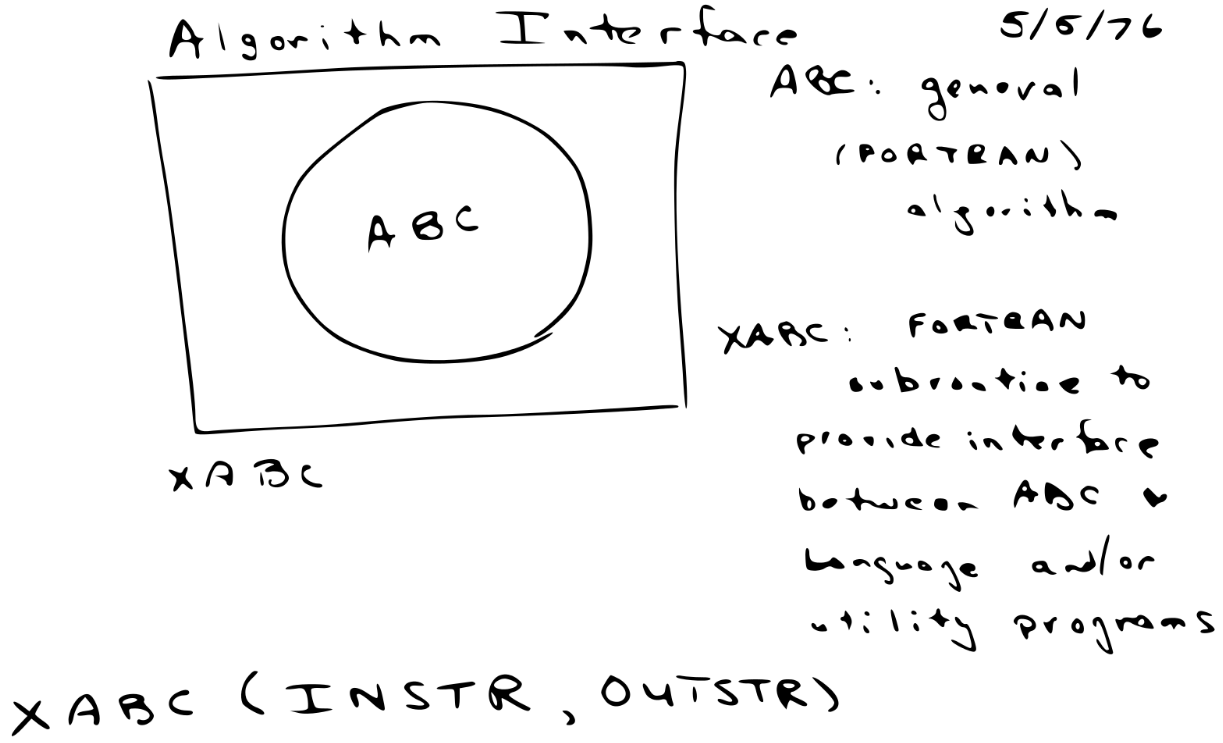
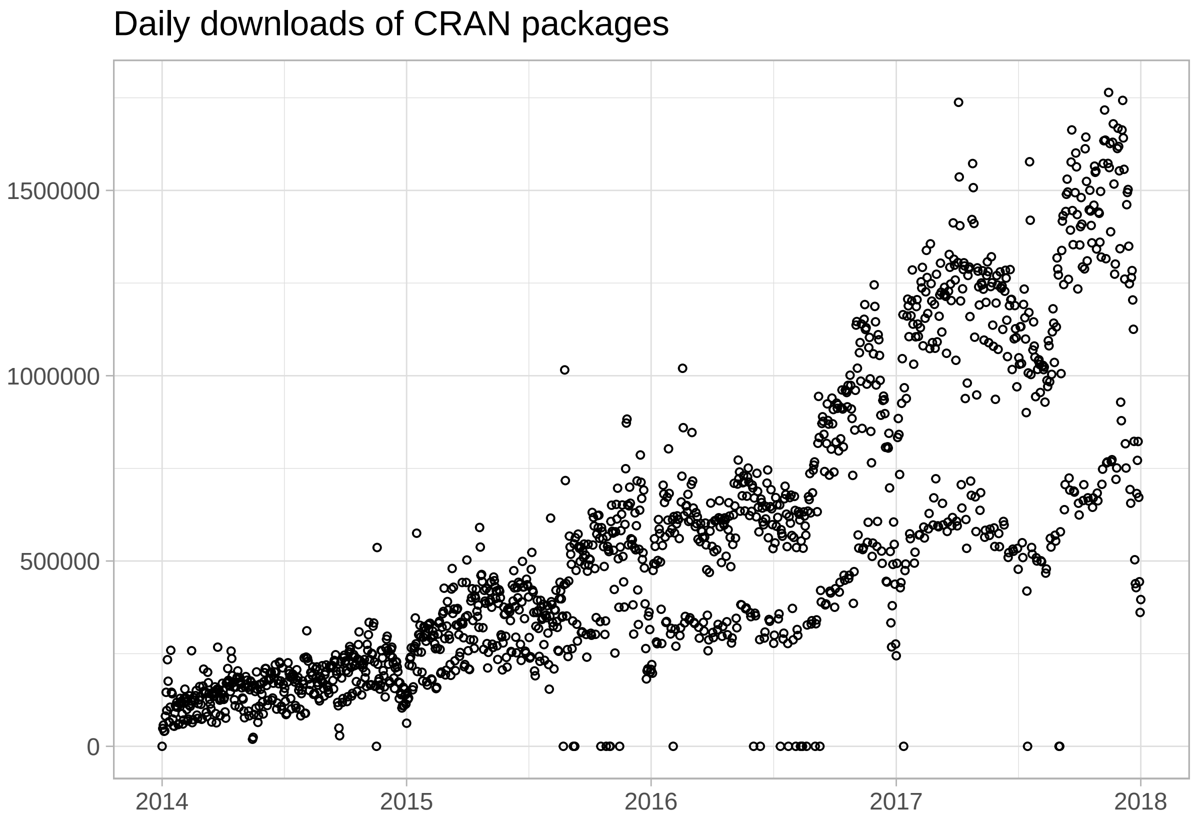

Algorithm Interface
5/6/76
ABC
XABC
ABC: general (FORTRAN) algorithm
XABC: FORTRAN subroutine to provide interface between ABC & language and/or utility programs
XABC (INSTR, OUTSTR)

Figure 1.1: Diagram by John Chambers - useR 2016

One can argue to what degree other fields, like machine learning, overlap with statistics; so far, most people will argue that the overlap is non-trivial. Similar arguments can be made for data science, big data, deep learning and beyond. With the continuous rise of popularity of R, I can only expect R's influence and scope to keep growing over time; we can take a look at the historic downloads of R packages in CRAN to get some sense of R's recent growth:



1.3 sparklyr

Back in 2016, there was a need in the R community to support Spark through a clean interface compatible with other R packages and available in CRAN. To this end, development of `sparklyr` started in 2016 by RStudio under JJ Allaire, Kevin Ushey and Javier Luraschi, version 0.4 was released in summer during the *useR!* conference, this first version added support for `dplyr`, `DBI`, modeling with MLlib and an extenible API that enabled extensions like H2Os rsparkling package. Since then, many new features have been added and support across many Spark distributions and Cloud services hosting Apache Spark has made available.

Officially,

> `sparklyr` is an R interface for Apache Spark.
>
> —github.com/rstudio/sparklyr

It's available in CRAN and works like any other CRAN package, meaning that: it's agnostic to versions, it's easy to install, it serves the R community, it embraces other packages and practices from the R community and so on. It's hosted in GitHub under https://github.com/rstudio/sparklyr and licensed under Apache 2.0 which is allows you to clone, modify and contribute back to this project.

While thinking of who and why should use `sparklyr`, the following roles come to mind:

- **New Users**: For new users, I'm going to argue that `sparklyr` is the best way to get started with

Spark. My hope is that the first chapters of this book will get you up running with ease and set you up for long term success.

- **Data Scientists**: I do believe, strongly, that `sparklyr` in combination with many other R packages and tools is the most productive environment for the modern data scientists. `sparklyr` allows support for high-level tasks and low-level extensibility mechanisms to match the needs and skills of every data scientists.
- **Expert Users**: For those users that are already immersed in Spark and can write code natevely in Scala, I'm going to argue that making their work available as an `sparklyr` extension is very desirable for them and the community. The R community is one of the most welcoming and supportive communities I've known, so I can't think of better ways of helping the expert users share their work and knowledge than by making it available in CRAN to R community.

This book is titled "The R in Spark" as a way to describe and teach that area of overlap between Spark and R. The R package that represents this overlap is `sparklyr`; however, the overlap goes beyond a package. It's an overlap of communities, expectations, future directions, packages and package extensions as well. Naming this book `sparklyr` or "Introduction to sparklyr" would have left behind a much more exciting opportunity, an opportunity to present this book as an intersection of the R and Spark communities. Both are solving very similar problems with a set of different skills and backgrounds; therefore, it is my hope that `sparklyr` can be a fertile ground for innovation, a welcoming place to newcomers, a productive place for experienced data scientists and an open community where cluster computing and modeling can come together.

Here are some resources to help you get involved:

- **Documentation**: This should be your entry point to learn more about sparklyr, the documentation is kept up to date with examples, reference functions and many more relevant resources (https://spark.rstudio.com).
- **Github**: If you believe something needs to get fixed, open a GitHub issue or send us a pull request (https://github.com/rstudio/sparklyr).
- **Stack Overflow**: For general questions, Stack Overflow is a good place to start (stackoverflow.com/tags/sparklyr).
- **Gitter**: For urgent issues or to keep in touch you can chat with us in Gitter (https://gitter.im/rstudio/sparklyr).

Chapter 2

Getting Started

For those already familiar with R, installing and launchiing Spark is as easy as running:

```r
library(sparklyr)

spark_install()
sc <- spark_connect(master = "local")
```

For those new to R or to for those who want to gain deeper understanding of the code above, this chapter will describe in detail how to get started with R and Spark.

2.1 Installation

2.1.1 Install R

2.1.2 Install RStudio

2.1.3 Install Spark

2.2 Connection

2.3 Tools

Chapter 3

Data Analysis

3.1 dplyr

3.2 DBI

Chapter 4

Modeling

4.1 mllib

Chapter 5

Clusters

5.1 On Prem

5.2 Cloud

5.3 Livy

Chapter 6

Data Sources

6.1 CSV

6.2 Text

6.3 Parquet

6.4 JDBC

6.5 Others

Chapter 7

Extensions

7.1 Using H2O

7.2 Writting Extensions

Chapter 8

Distributed R

8.1 Use Cases

8.2 Troubleshooting

Bibliography

French, C. (1996). *Data Processing and Information Technology.* Cengage Learning Business Press.