

The R in Spark

Javier Luraschi

2018-03-22

Contents

The R in Spark	5
1 Introduction	7
1.1 Spark	9
1.2 R	10
1.3 sparklyr	10
2 Getting Started	15
2.1 Prerequisites	15
2.2 Installing Spark	19
2.3 Connecting to Spark	19
2.4 Using Spark	20
2.5 Disconnecting	21
2.6 Recap	21
3 Data Analysis	23
3.1 dplyr	23
3.2 DBI	23
4 Modeling	25
4.1 mllib	25
5 Clusters	27
5.1 Overview	27
5.2 Cluster Types	27
6 Data Sources	29
6.1 CSV	29
6.2 Text	29
6.3 Parquet	29
6.4 JDBC	29
6.5 Others	29
7 Extensions	31
7.1 Using H2O	31
7.2 Writing Extensions	31
8 Distributed R	33
8.1 Use Cases	33
8.2 Troubleshooting	33
9 Applications	35
9.1	35

9.2	35
Appendix		37
9.3	Worlds Store Capacity	37
9.4	Daily downloads of CRAN packages	37

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.

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 United States License.

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 (see Section 9.3):

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 engines were unable 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 accomplished 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, reduce 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 efficiently 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 analysis 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.

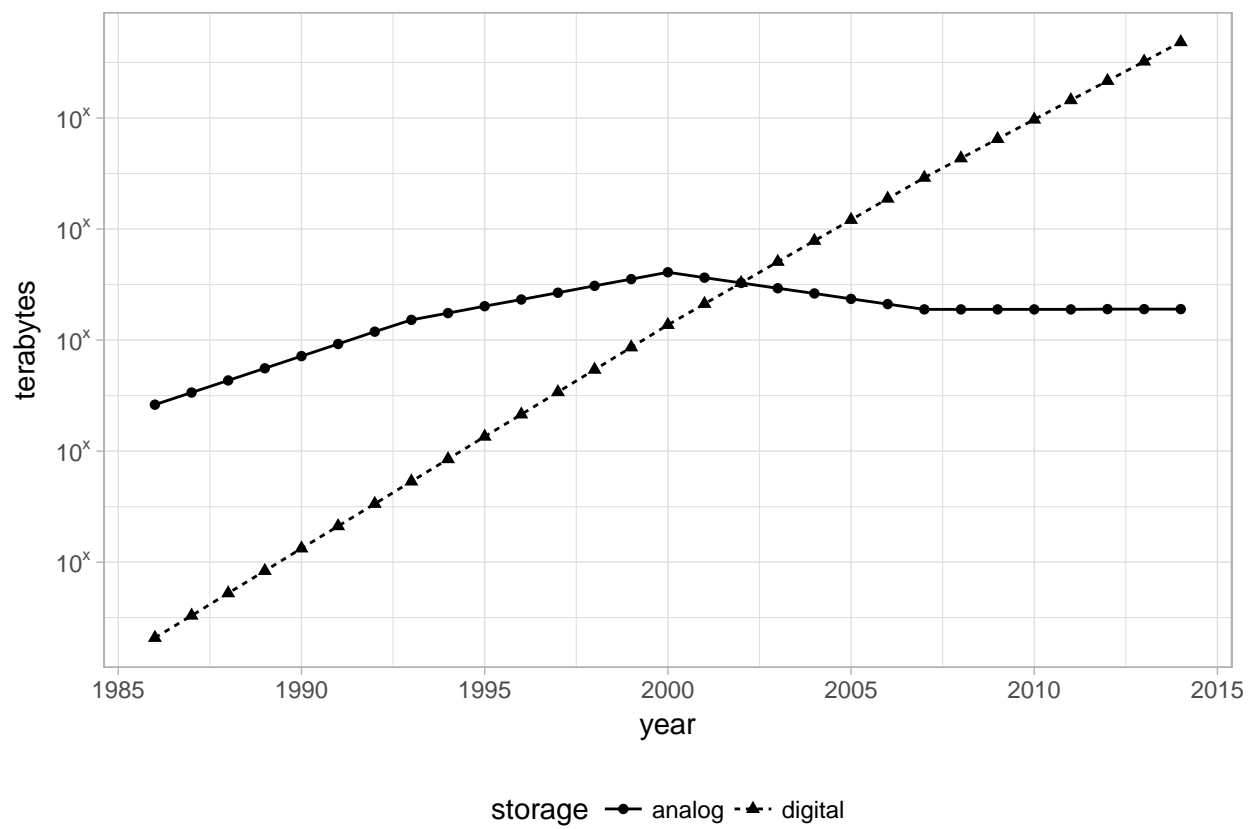


Figure 1.1: World's capacity to store information.

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 its predecessor 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 than 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 benefits 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 analyze data.
2. **Variety:** 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 processes are known as 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 refer to some of these benefits 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 is not uncommon to see organizations experimenting with new workflows and a variety 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 that 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 its 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:

```
knitr::include_graphics("images/01-intro-s-algorithm-interface.png")
```

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. Additionally, 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 peculiarities of computer science and engineering.
- The **R Community** provides 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 motivated to help you succeed. Many packages provided by the R community make R, by far, the place to do statistical computing.

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 (see Section 9.4):

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 extensible API that enabled extensions like H2O's **rsparkling** package. Since then, many new features have been added and support across many Spark distributions and cloud services 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

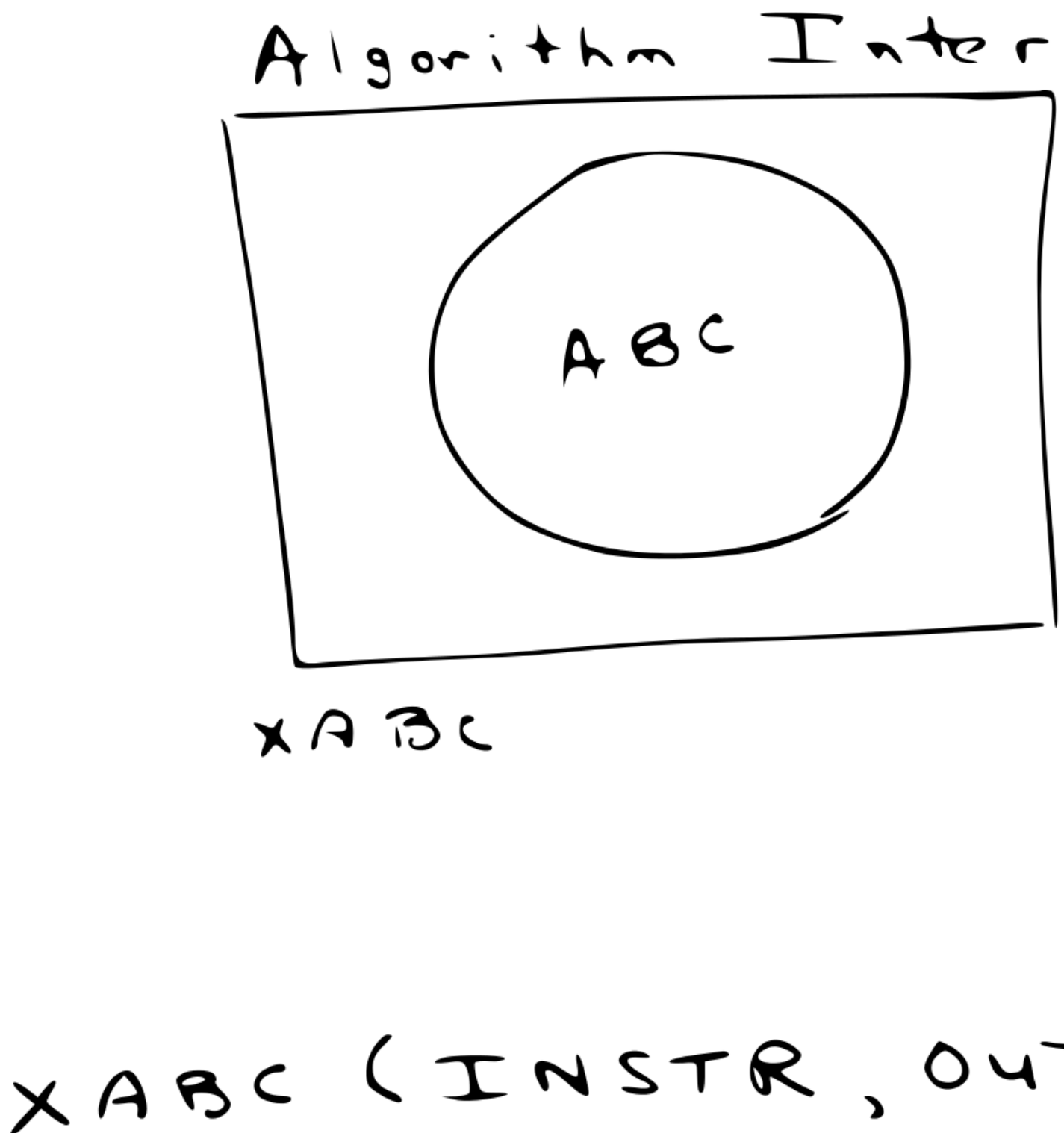


Figure 1.2: Interface language diagram by John Chambers from useR 2016.



Figure 1.3: Daily downloads of CRAN packages.

and so on. It's hosted in GitHub under <https://github.com/rstudio/sparklyr> and licensed under Apache 2.0 which 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 scientist.
- **Expert Users:** For those users that are already immersed in Spark and can write code natively 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

From R, installing and launching a local Spark cluster using `sparklyr` is as easy as running:

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

However, to make sure we can all run the code above and understand it, this section will walk you through installing the prerequisites, installing Spark and connecting to a local Spark cluster.

2.1 Prerequisites

As briefly mentioned in Section 1, R is a programming language that can run in many platforms and environments. Most people making use of a programming language also choose tools to make them more productive in it; for R, RStudio would be such tool. Strictly speaking, RStudio is an Integrated Development Environment or IDE for short, which also happens to support many platforms and environments. R and RStudio are the free software tools this book will make use of and therefore, I strongly recommend you get those installed if you haven't done so already.

Additionally, since Spark is build in the Scala programming language which is run by the Java Virtual Machine, you also need to install Java 7 or newer in your system. It is likely that your system already has Java installed, but is probably worth updating with the steps bellow.

2.1.1 Install R

From r-project.org, download and launch the installer for your platform, Windows, Macs or Linux available.

2.1.2 Install Java 8

From oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html, download and launch the installer for your platform, Windows, Macs or Linux available.

2.1.3 Install RStudio

While installing RStudio is not strictly required to work with `sparklyr` in R, it will make you much more production and therefore, I highly recommend you take the time to install RStudio from [rstu-](http://rstudio.com)

[\[Home\]](#)**Download**[CRAN](#)**R Project**[About R](#)[Logo](#)[Contributors](#)[What's New?](#)[Reporting Bugs](#)[Development Site](#)[Conferences](#)[Search](#)**R Foundation**[Foundation](#)[Board](#)[Members](#)[Donors](#)[Donate](#)**Help With R**[Getting Help](#)**Documentation**[Manuals](#)[FAQs](#)[The R Journal](#)[Books](#)

The R Project for Statistical Computing

Getting Started

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To [download R](#), please choose your preferred [CRAN mirror](#).

If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

News

- [R version 3.5.0 \(Joy in Playing\) prerelease versions](#) will appear starting Friday 2018-03-23. Final release is scheduled for Monday 2018-04-23.
- [R version 3.4.4 \(Someone to Lean On\)](#) has been released on 2018-03-15.
- [useR! 2018](#) (July 10 - 13 in Brisbane) is open for registration at <https://user2018.r-project.org>
- [The R Journal Volume 9/2](#) is available.
- [R version 3.3.3 \(Another Canoe\)](#) has been released on Monday 2017-03-06.
- [useR! 2017](#) took place July 4 - 7 in Brussels <https://user2017.brussels>
- The [R Logo](#) is available for download in high-resolution PNG or SVG formats.

Figure 2.1: The R Project for Statistical Computing.

The screenshot displays the Oracle Java Downloads page for the Java SE Development Kit 8. The page layout includes a top navigation bar with the Oracle logo, a menu icon, a search bar, and links for Sign In, Country/Region, and a Call button. Below the navigation bar is a breadcrumb trail: Oracle Technology Network / Java / Java SE / Downloads.

The main content area is titled "Java SE Development Kit 8 Downloads" and includes a thank-you message for downloading the release. It also provides information about the JDK's purpose and includes a section for "See also:" with links to the Java Developer Newsletter, Java Developer Day, and Java Magazine.


Below the "See also:" section, there are links for JDK 8u161 checksum and JDK 8u162 checksum. The page then presents a table of download links for the Java SE Development Kit 8u161, requiring users to accept the Oracle Binary Code License Agreement for Java SE.

The table lists the following products and their corresponding download links:

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.92 MB	jdk-8u161-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	74.88 MB	jdk-8u161-linux-arm64-vfp-hflt.tar.gz
Linux x86	168.96 MB	jdk-8u161-linux-i586.rpm
Linux x86	183.76 MB	jdk-8u161-linux-i586.tar.gz
Linux x64	166.09 MB	jdk-8u161-linux-x64.rpm
Linux x64	180.97 MB	jdk-8u161-linux-x64.tar.gz
macOS	247.12 MB	jdk-8u161-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	139.99 MB	jdk-8u161-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.29 MB	jdk-8u161-solaris-sparcv9.tar.gz
Solaris x64	140.57 MB	jdk-8u161-solaris-x64.tar.Z
Solaris x64	97.02 MB	jdk-8u161-solaris-x64.tar.gz
Windows x86	198.54 MB	jdk-8u161-windows-i586.exe
Windows x64	206.51 MB	jdk-8u161-windows-x64.exe

On the right side of the page, there are two sections: "Java SDKs and Tools" and "Java Resources". The "Java SDKs and Tools" section includes links to Java SE, Java EE and Glassfish, Java ME, Java Card, NetBeans IDE, and Java Mission Control. The "Java Resources" section includes links to Java APIs, Technical Articles, Demos and Videos, Forums, Java Magazine, Developer Training, Tutorials, and Java.com.


Figure 2.2: Java Downloads.



[Products](#)
[Resources](#)
[Pricing](#)
[About Us](#)
[Blogs](#)
[Q](#)

Choose Your Version of RStudio

RStudio is a set of integrated tools designed to help you be more productive with R. It includes a console, syntax-highlighting editor that supports direct code execution, and a variety of robust tools for plotting, viewing history, debugging and managing your workspace. [Learn More about RStudio features.](#)



	RStudio Desktop Open Source License	RStudio Desktop Commercial License	RStudio Server Open Source License	RStudio Server Pro Commercial License	RStudio Server Pro + RStudio Connect Commercial License
	FREE	\$995 per year	FREE	\$9,995 per year	\$29,995 per year
	DOWNLOAD	BUY	DOWNLOAD	DOWNLOAD	TALK
	Learn More	Learn More	Learn More	Learn More	Learn More
Integrated Tools for R	●	●	●	●	●
Priority Support		●		●	●
Access via Web Browser			●	●	●

Figure 2.3: RStudio Downloads.

`dio.com/products/rstudio/download/`, then download and launch the installer for your platform, Windows, Macs or Linux available.

After launching RStudio, identify the Console panel since this is where most of the code will be executed in this book. For additional learning resources on R and RStudio consider visiting: `rstudio.com/online-learning/`.

2.1.4 Install sparklyr

First of all, we would want to install `sparklyr`. As many other R packages, `sparklyr` is available in CRAN and can be easily installed as follows:

```
install.packages("sparklyr")
```

The CRAN release of `sparklyr` contains the most stable version and it's the recommended version to use; however, for those that need or might want to try newer features being developed in `sparklyr` you can install directly from GitHub using the `devtools` package. First install the `devtools` package and then `sparklyr` as follows:

```
install.packages("devtools")
devtools::install_github("rstudio/sparklyr")
```

2.2 Installing Spark

Start by loading `sparklyr`,

```
library(sparklyr)
```

This will make all `sparklyr` functions available in R, which is really helpful; otherwise, we would have to run each `sparklyr` command prefixed with `sparklyr::`.

As mentioned, Spark can be easily installed by running `spark_install()`; this will install the latest version of Spark locally in your computer, go ahead and run `spark_install()`. Notice that this command requires internet connectivity to download Spark.

```
spark_install()
```

All the versions of Spark that are available for installation can be displayed with `spark_available_versions()`:

```
spark_available_versions()
```

```
##      spark
## 1  2.3.0
## 2  2.2.1
## 3  2.2.0
## 4  2.1.1
## 5  2.1.0
## 6  2.0.2
## 7  2.0.1
## 8  2.0.0
## 9  1.6.3
## 10 1.6.2
## 11 1.6.1
## 12 1.6.0
```

A specific version can be installed using the Spark version and, optionally, by also specifying the Hadoop version. For instance, to install Spark 1.6.3, we would run `spark_install("1.6.3")`.

You can also check which versions are installed by running:

```
spark_installed_versions()
```

Finally, in order to uninstall a specific version of Spark you can run `spark_uninstall()` by specifying the Spark and Hadoop versions, for instance:

```
spark_uninstall(version = "1.6.0", hadoop = "2.6")
```

2.3 Connecting to Spark

It's important to mention that, so far, we've only installed a local Spark cluster. A local cluster is really helpful to get started, test code and troubleshoot with ease; further chapters will explain where to find, install and connect to real Spark clusters with many machines; but for the first few chapters, we will focus on using local clusters.

Therefore, to connect to this local cluster we simply run:

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

The `master` parameter helps `sparklyr` find which is the “main” machine from the Spark cluster, this machine is often called the driver node. While working with real clusters using many machines, most machines will be

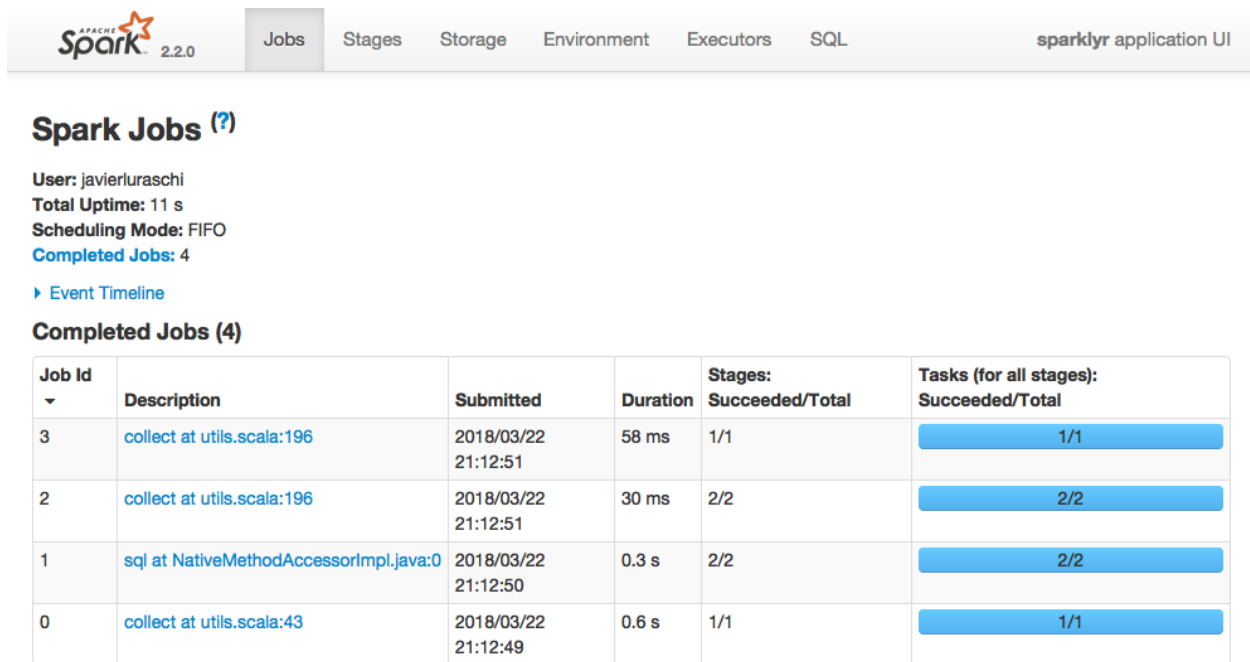


Figure 2.4: Apache Spark Web Interface.

worker machines and one will be the master. Since we only have a local cluster with only one machine, we will default to use "local" for now.

2.4 Using Spark

Now that you are connected, we can run a simple commands. For instance, let's start by loading some text.

First, lets create a text file by running:

```
write("Hello World!", "hello.txt")
```

Which we can read back in Spark by running:

```
spark_read_text(sc, "hello", "hello.txt")
```

```
## # Source:   table<hello> [?? x 1]
## # Database: spark_connection
##   line
##   <chr>
## 1 Hello World!
```

2.4.1 Web Interface

Most of the Spark commands will get started from the R console; however, it is often the case that monitoring and analyzing execution is done through Spark's web interface. This interface is a web page provided by the driver node which can be accessed from `sparklyr` by running:

```
spark_web(sc)
```

2.4.2 Logs

Another common tool is to read through the Spark logs, a log is just a text file where Spark will append information relevant to the execution of tasks in the cluster. For local clusters, we can retrieve the `sparklyr` related log entries by running:

```
spark_log(sc, filter = "sparklyr", n = 5)
```

```
## 18/03/22 21:12:42 INFO SparkContext: Submitted application: sparklyr
## 18/03/22 21:12:42 INFO SparkContext: Added JAR file:/Library/Frameworks/R.framework/Versions/3.4/Resour
## 18/03/22 21:12:49 INFO Executor: Fetching spark://127.0.0.1:56640/jars/sparklyr-2.2-2.11.jar with times
## 18/03/22 21:12:49 INFO Utils: Fetching spark://127.0.0.1:56640/jars/sparklyr-2.2-2.11.jar to /private/v
## 18/03/22 21:12:49 INFO Executor: Adding file:/private/var/folders/fz/v6wfsq2x1fb1rw4f6r0x4jwm0000gn/T/
```

2.5 Disconnecting

For local clusters and, really, any cluster; once you are done processing data you should disconnect by running:

```
spark_disconnect(sc)
```

this will terminate the connection to the cluster but also terminate the cluster tasks as well. If multiple Spark connections are active, or if the connection instance `sc` is no longer available, you can also disconnect all your Spark connections by running `spark_disconnect_all()`.

2.6 Recap

This chapter walked you through installing R, Java, RStudio and `sparklyr` as the main tools required to use Spark from R. We covered installing local Spark clusters using `spark_install()` and learned how to launch the web interface using `spark_web(sc)` and view logs using `spark_log(sc)`.

It is my hope that this chapter will help anyone interested in learning cluster computing using Spark and R to get you started, ready to experiment on your own and ready to tackle actual data analysis and modeling tasks without any major blockers. However, if you hit any installation or connection issues, start by browsing online for the error message or open a GitHub issue under <https://github.com/rstudio/sparklyr/issues> to help you get going.

Chapter 3

Data Analysis

3.1 dplyr

3.2 DBI

Chapter 4

Modeling

4.1 mllib

Chapter 5

Clusters

5.1 Overview

5.1.1 Spark Distributions

5.1.2 Cluster Types

5.1.3 On Prem vs Cloud

5.2 Cluster Types

5.2.1 Standalone

5.2.2 Yarn

5.2.2.1 Client

5.2.2.2 Cluster

5.2.3 Mesos

5.2.4 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

Chapter 9

Applications

9.1

9.2

Appendix

9.3 Worlds Store Capacity

```
library(tidyverse)
read_csv("data/01-worlds-capacity-to-store-information.csv", skip = 8) %>%
  gather(key = storage, value = capacity, analog, digital) %>%
  mutate(year = X1, terabytes = capacity / 1e+12) %>%
  ggplot(aes(x = year, y = terabytes, group = storage)) +
    geom_line(aes(linetype = storage)) +
    geom_point(aes(shape = storage)) +
    scale_y_log10(
      breaks = scales::trans_breaks("log10", function(x) 10^x),
      labels = scales::trans_format("log10", scales::math_format(10^x))
    ) +
    theme_light() +
    theme(legend.position = "bottom")
```

9.4 Daily downloads of CRAN packages

```
downloads_csv <- "data/01-intro-r-cran-downloads.csv"
if (!file.exists(downloads_csv)) {
  downloads <- cranlogs::cran_downloads(from = "2014-01-01", to = "2018-01-01")
  readr::write_csv(downloads, downloads_csv)
}

cran_downloads <- readr::read_csv(downloads_csv)

ggplot(cran_downloads, aes(date, count)) +
  geom_point(colour="black", pch = 21, size = 1) +
  scale_x_date() +
  xlab("") +
  ylab("") +
  theme_light()
```


Bibliography

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