SEAI RD&D 2019

SMARTeBuses

Storing Data in a Hadoop Cluster

Deliverable number: WP2-D1



Seal SUSTAINABLE Strengy AUTHORTY GENERATION OF THE SEAL RD&D Programme. Grand Award Number: 19/RDD/519

Project Acronym:	SMARTeBuses
Project Full Title:	SMART electric Buses
Grant Number:	19/RDD/519
Project URL:	https://smartebuses.github.io/web/
Deliverable type:	Deliverable
Dissemination level:	Public (PU)
Delivery Date:	August 31st, 2020
Number of pages:	65
Keywords:	Big Data, Distributed File System, Large-scale Data Analysis, SMARTe-
	Buses, Artificial Intelligence
Authors:	Dr. Ignacio Castiñeiras, Cork Institute of Technology
Peer review:	Dr. Alejandro Arbelaez, University College Cork

Contents

1	Intro	oductio	n	5
2	Big 2.1	Data Eo	cosystem	6 6
	2.1	2.1.1	Hadoop Distributed File System	6
		2.1.2	Hadoop Yarn	8
		2.1.3	Hadoop MapReduce	8
	2.2	Spark		12
		2.2.1	Data Storage and Resource Manager	12
		2.2.2	Spark Core	13
		2.2.3	Spark SQL	14
3	Con	figurati	ion and Code Examples	16
	3.1	Local C	Cluster	16
	3.2	Java .		16
	3.3	Python		17
	3.4	Hadoop)	19
		3.4.1	Installing Hadoop	19
		3.4.2	Start a Hadoop Cluster	22
		3.4.3	Dataset	23
		3.4.4	HDFS	24
		3.4.5	MapReduce	31
		3.4.6	Stop a Hadoop Cluster	40
	3.5	Spark		41
		3.5.1	Installing Spark	41
		3.5.2	Spark Core	42
		3.5.3	Spark SQL	51
4	Con	clusior	ns and Future Work	62

List of Figures

2.1	HDFS and Local File System
2.2	File to Block and Block Replication
2.3	MapReduce Application Phases
2.4	Map Phase
2.5	Reduce Phase
2.6	Spark Components
2.7	Spark Components
2.8	Spark Core: User Program
2.9	Spark Core: Program Execution
0.1	
3.1	Script openJDK_8.sh \ldots 16
3.2	Script python_ 3_7_7 .sh
3.3	Script Hadoop_2_7_1.sh
3.4	Script Start_Hadoop_Cluster.sh
3.5	Script data_analysis.sh
3.6	Cluster Overview
3.7	HDFS
3.8	Dataset Folder: my_dataset
3.9	Command put: my_dataset from local file system to HDFS
3.10	HDFS With New Folder my_result
3.11	Command get: my_result from HDFS to local file system
3.12	ResourceManager Overview
3.13	ResourceManager: MapReduce Application in Progress
3.14	ResourceManager: MapReduce Application Finished
3.15	HDFS: MapReduce Result in my_result
3.16	HDFS: MapReduce Result Brought Back to Local File System
3.17	my_mapper.py: Import Section
3.18	my_mapper.py: my_map Function
3.19	my_mapper.py: Main Entry Point
3.20	my_mapper.py: Import Section
3.21	my_mapper.py: my_map Function
3.22	my_mapper.py: Main Entry Point
3.23	Script Stop_Hadoop_Cluster.sh
3.24	Script Spark_2_4_5.sh $\ldots \ldots 41$
3.25	mv_Spark_Core_example.pv: Import Section
3.26	my Spark Core_example.py: my spark core_iob Function
3.27	my Spark Core example by: Main Entry Point 45
3.28	inputRDD Content
3.29	all wordsRDD Content
3.30	clean wordsRDD Content 46
3.31	RDDs Content 46

3.32	ResourceManager: Spark Core Application in Progress	47
3.33	ResourceManager: Spark Core Application Finished	48
3.34	HDFS: Spark Core Result in my_result	49
3.35	HDFS: Spark Core Result Brought Back to Local File System	50
3.36	my_Spark_SQL_example.py: Import Section	51
3.37	my_Spark_SQL_example.py: my_spark_sql_job Function	53
3.38	my_Spark_SQL_example.py: my_spark_sql_job Function	54
3.39	my_Spark_SQL_example.py: Main Entry Point	55
3.40	inputDF Content	56
3.41	sentenceDF Content	56
3.42	wordsDF Content	56
3.43	cleanDF Content	57
3.44	lowerDF Content	57
3.45	solutionDF Content	57
3.46	ResourceManager: Spark SQL Application in Progress	58
3.47	ResourceManager: Spark SQL Application Finished	59
3.48	HDFS: Spark SQL Result in my_result	60
3.49	HDFS: Spark SQL Result Brought Back to Local File System	61

List of Acronyms

- **HDFS** Hadoop Distributed File System
- **JRE** Java Runtime Environment
- Yarn Yet Another Resource Negotiator
- **DSL** Domain Specific Language
- **RDD** Resilient Distributed Datasets

1 Introduction

Big Data is the ability of society to harness information in novel ways to produce useful insights or goods and services of significant value [1]. In this work package we focus on the use of Big Data technology to store, analyse and collect online public data related to the bus network (e.g., routes, timetables, delays) and wind power available from wind farms in the country. Furthermore, we focus on developing a realistic reference road network with essential information for a strategic planning and design of an electric bus network in Ireland. In line with the goals of the work package, we introduce the Big Data ecosystem of tools we are going to use for storing and analysis bus and wind power-related large-datasets.

The document is structured as follows:

Chapter 2 presents our ecosystem of tools to store and analyse large-scale datasets. First, Section 2.1 presents Apache Hadoop [2], specifically its components Hadoop Distributed File System (HDFS) [3, 4], Hadoop Yarn [5, 6] and Hadoop MapReduce [7, 8] for storing and analysing datasets. Then, Section 2.2 presents Apache Spark [9] as our primary data analytics tool (relying on HDFS and Yarn for data storage and job management, resp). In particular, it presents its components Spark Core [10, 11] and Spark SQL [12, 13]. While Python is selected as the programming language of choice (with both MapReduce and Spark providing an API for it), the data analytics applications will run on top of the Java Runtime Environment (JRE) [14].

Chapter 3 serves as a tutorial for installing and configuring such ecosystem of tools, as well as presenting some introductory data analysis examples. First, Section 3.1 presents the specs of the local server. Then, sections 3.2, 3.3, 3.4 and 3.5 present the setup of Java, Python, Hadoop and Spark, resp., including some introductory data analysis examples running on the local cluster.

2 Big Data Ecosystem

This chapter presents our ecosystem of tools to store and analyse large-scale datasets.

2.1 Hadoop

Apache Hadoop is a data framework for store and process big data distributed on clusters of commodity machines. Using the definition of its own website: "*The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures*" [2].

The main idea of the framework is for data to be distributed among the nodes of the cluster for both its storage and processing. This way, each individual node works as much as possible on the data it hosts, minimising the need to talk to other nodes of the cluster (and thus minimising the data transferred over the network). Data is replicated among different nodes of the cluster for redundancy, availability and fault tolerant-free computations. If a node is unexpectedly shut down, the system is still ready to store and process the entire dataset without suffering a noticeable penalisation in the performance to achieve it. At failure, the workload of the node is assumed by any other alive node of the cluster, without any loss of data. Thus, the outcome of the computation does not become affected. When the node recovers, it just joins again the set of operative nodes, becoming available again for storing and processing data. Likewise, the addition of new nodes to the cluster is easy, and provides extra storage and processing capacity under demand. As the nodes cooperate on the tasks, the system is capable of dealing with peak workloads, which result only in slightly penalisation for the overall performance.

There are 3 components of Hadoop that we are going to use:

- HDFS: A distributed file system designed to efficiently allocate data across the multiple commodity machines (nodes) of the cluster.
- Yet Another Resource Negotiator (Yarn): A resource manager, responsible for schedule and monitor the execution of our data analysis applications.
- MapReduce: A framework for easily writing applications processing large-scale datasets across a cluster in a reliable, fault-tolerant manner.

2.1.1 Hadoop Distributed File System

We use HDFS as a distributed file system to store our datasets across a cluster. The master/slave architecture of HDFS consists of two types of daemons:

• A single NameNode. This is a master daemon managing the file system namespace and regulating access to files. In brief, the NameNode acts as the table of contents: it contains no single file itself, but it knows all files in the filesystem, as well as their distribution among the nodes.

• A number of DataNodes, usually one per node in the cluster. This is a slave daemon managing the storage of the node it runs on. That is, the DataNode controls all the files stored in the node, but it lacks any knowledge about the data stored in other nodes.

Figure 2.1 presents a cluster with 4 nodes, where 1 node contains the NameNode and the other 3 a DataNode each. As we can see, the memory storage of each node contains its own local file system and its fraction/slice of the HDFS.



Figure 2.1: HDFS and Local File System

HDFS is designed to reliably store very large files. Its approach consists on splitting each file into manageable blocks and replicate each blocks among multiple nodes (typically 3, although this is configurable). Figure 2.2 shows a file split into 3 blocks when being written to HDFS and the replication of each block among the DataNodes.



Figure 2.2: File to Block and Block Replication

We envision our data analysis applications to follow a very simple flow:

• Read the dataset from a logic folder of HDFS (e.g., my_dataset).

• Write the data analysis result to a new logic folder of HDFS (e.g., my_result).

The above requires us to:

- Create a new logic folder my_dataset into HDFS.
- Bring my_dataset from the local file system to the HDFS folder.
- Bring my_result from HDFS to the local file system.

HDFS provide the specific commands mkdir, put and get to accomplish the aforementioned tasks, resp. In brief, any read/write operation is coordinated by the NameNode, who refers to the associated DataNode of the node hosting the block for managing it.

2.1.2 Hadoop Yarn

We use the Yarn system as a resource manager, responsible for schedule and monitor the execution of our data analysis applications.

The key idea of Yarn is to split up the functionalities of resource management and job scheduling/monitoring into separate daemons:

- A single ResourceManager. This is a master daemon distributing the resources of the cluster among all the applications. In an analogy with HDFS, it will play the role of the NameNode, in the sense it executes nothing by itself, but it knows all the resources and tasks to be executed among the nodes of the cluster.
- A NodeManager per node in the cluster. This is a slave daemon monitoring the resource usage (cpu, memory, disk, network) of the node it runs on. In an analogy with HDFS, it will play the role of the DataNode, in the sense it controls the resources of the node and monitors the execution of the tasks assigned to it, but it lacks any knowledge about the resources and execution in other nodes of the cluster.

In brief, given a MapReduce or Spark application, the ResourceManager will provide the application with resources from 1 or more nodes of the cluster. Then, each NodeManager involved in the execution of the application will accomplish its assigned tasks with the resources provided.

2.1.3 Hadoop MapReduce

We use MapReduce as our first framework for easily writing applications processing large-scale datasets across a cluster in a reliable, fault-tolerant manner. A MapReduce application can be seen as a pipeline process operating on files that uses streaming for communication and high-level programming features to isolate the data processing from both the cluster and the pipeline complexities.

As previously stated, we envision a MapReduce application to read the dataset from a HDFS folder (e.g., my_dataset) and produce its results to a novel HDFS folder (e.g., my_result). In doing so, the application is to follow 3 phases: map, sort and reduce. Figure 2.3 presents an introductory word count MapReduce application, viewing its different phases from a logic and a physical point of views, resp.

As we can see, the *Map Phase* processes the entire dataset of $my_dataset$ in a completely parallel manner. On doing so, each data block is processed on its own node as much as possible, with typically a Map process allocated to each block. The Map functionality is to be programmed just once, with all the processes applying it to their respective data slice (cf. middle picture of Figure 2.3). While the right level of parallelism for Map processes seems to be around 10-100 per node, in general the framework figures out the number of processes to be applied, as well as the block allocation for them.

This abstracts the cluster complexity from the user, who just has to focus on the programming of the Map functionality.

To specify the Map functionality we make use of Hadoop Streaming [15], which abstracts the Map process from its underlying programming language by only requiring it to be in the form of an executable command. In our case, we choose to program the Map functionality in the form of a Python script (e.g., my_mapper.py). Figure 2.4 shows my_mapper.py as a black-box. As we can see, it must receive its input from the standard input, and produce a bunch of key-value pairs by writing them to the standard output.

Coming back to Figure 2.3, we see that the *Sort Phase* sorts the outputs of the Map processes, which are then input to the Reduce processes. Again, while the right level of parallelism for Reduce processes seems to be around 1-2 per node, this is abstracted to the user who just has to focus on the programming of the Reduce functionality. What it is ensured is that all sorted entries having the same key are to be assigned to the same Reduce process.

As we did with the Map functionality, we use Hadoop Streaming to program the Reduce functionality in the form of a Python script (e.g., my_reducer.py). Figure 2.5 shows my_reducer.py as a black-box. As we can see, it must receive the key-value pairs produced by Map as input from the standard input, and produce a bunch of key-value pairs by writing them to the standard output.

Finally, the bulk of results from the *Reduce Phase* produce the new HDFS folder my_result. In particular, the output of each Reduce process turns into one or more new blocks in the folder.



Figure 2.3: MapReduce Application Phases



Figure 2.4: Map Phase



Figure 2.5: Reduce Phase

2.2 Spark

We use Apache Spark as our second framework for easily writing applications processing large-scale datasets across a cluster in a reliable, fault-tolerant manner. Using the definition of its own website: *Apache Spark is a unified analytics engine for large-scale data processing* [9]. It is an an open-source, distributed, general-purpose, cluster-computing framework designed for 3 purposes:

- Be easy to use, allowing us to develop applications locally, using a high-level API.
- Be fast, enabling interactive use and complex algorithms.
- Be general, allowing us to combine multiple types of computations, including text, SQL, graph and machine learning processing (both offline and online) that might previously have required different engines.

Spark itself is written in Scala [16], and runs on the Java Virtual Machine (JVM). However, it offers simple APIs in Scala, Java, Python and R. In our case, we choose Python to program our data analysis applications.

Figure 2.6 presents the different components of Spark. We are going to rely on HDFS for data storage and on Yarn for resource management and job scheduling. For developing our data analytics applications we are going to use Spark Core and Spark SQL.



Figure 2.6: Spark Components

2.2.1 Data Storage and Resource Manager

On its own, Spark is not a data storage solution; it performs computations on Java Virtual Machines (JVMs) that last only for the duration of a Spark application. While Spark can be run locally on a single machine with a single JVM (called local mode), this mode is only useful for debugging and testing purposes. Once the application is tested, Spark is designed to efficiently scale up from one to many thousands of compute nodes, so we will use it in distributed mode across a cluster.

When used in a cluster, Spark is used in tandem with a distributed storage system (in our case HDFS) and with a cluster manager (in our case Yarn).

- HDFS is used to provide the input dataset used by Spark application (e.g., the HDFS folder my_dataset), as well as to stable store the results such application produces (e.g., the HDFS folder my_result).
- Yarn is used to schedule the execution of a Spark application, and for assigning and monitoring the resources of each node executing it.



Figure 2.7: Spark Components

Figure 2.7 presents a general view of the execution of a Spark application in a cluster. The master/slave architecture of Spark consists of two types of daemons:

- A single SparkDriver. This is a master daemon coordinating the execution of the User program. In an analogy to HDFS, it plays the role of the NameNode, in the sense that it does not compute anything by itself, but controls who is executing each task and how is this execution going.
- A number of SparkExecutors. This is a slave daemon managing the execution of a single given task. A number of SparkExecutors are distributed among the cluster. While the number of cores per executor can be configured at the user program, typically they correspond to the physical cores on a node, and an executor cannot span cores of different nodes. In an analogy to HDFS, a SparkExecutor plays the role of the DataNode, in the sense that it carries out the computation of the task being given (and reports its status to the SparkDriver), but lacks any knowledge about the tasks executed by other SparkExecutors.

2.2.2 Spark Core

Spark Core contains the basic functionality of Spark, including:

- The main data abstraction being offered to users to express their programs: The Resilient Distributed Datasets (RDD)
- The implementation of the SparkDriver and SparkExecutor daemons needed to execute a Spark application. This includes the Spark driver functionality to generate a logical plan (Direct Acyclic Graph per job, with concrete stages and tasks) and a physical plan (schedule and tracking of the tasks execution).
- Other functionality such as memory management, fault recovery and the interaction with storage systems.

An RDD simply defines a collection of items. It is:

- Indivisible (logically presented as an atomic variable).
- Generic, but statycally-typed (available for any data type T as long as the type T sticks for all its elements).
- Lazily-evaluated (only computed if required, and as much as required).
- Non-mutable (cannot change type nor value).

In this context, we can think of an RDD as a list (in the sense that its elements can be repeated) or a set (in the sense that its elements have no particular default order).

The API of Spark Core offers Creation, Transformation, Persistent and Observer operations. Figure 2.8 presents the life-cycle of a Spark Core user program, based on these operations.





Figure 2.8: Spark Core: User Program

Figure 2.9 presents the execution of a Spark Core application. The application is defined as a set of Jobs triggered by the action operations of the user program. Each Job consists on a sequential execution of stages. Each new stage is caused by a shuffle of information among nodes executing different tasks. While no external communication is needed, each SparkExecutor works locally, performing a pipeline of tasks.



Figure 2.9: Spark Core: Program Execution

2.2.3 Spark SQL

Spark SQL is the module integrating relational processing with the functional programming API of Spark. By using it, we can benefit of a higher-level data abstraction (DataFrames) for ingesting, querying and persisting (semi)structured data using relational queries via a Domain Specific Language (DSL). Under the hood, Spark SQL translates a DataFrame-based program into an equivalent RDD-based one via a catalyst optimiser (which semantically analyse the query expressed by the user) and a Tungsten encoder optimiser (which translates the query to an equivalent binary RDD-based format). An DataFrame simply defines a collection of items. It is:

- Indivisible (logically presented as an atomic variable).
- Structured, in the sense of having a fixed number of fields, each of them of a concrete data type T.
- Lazily-evaluated (only computed if required, and as much as required).
- Non-mutable (cannot change type nor value).

In this context, we can think of a DataFrame as a table in a relational database (in the sense that each Row follows the schema) or a collection in a NoSQL document oriented database (in the sense that the content of the collection is distributed).

Similarly to Spark Core, the API of Spark SQL offers Creation, Transformation, Persistent and Observer operations. Spark SQL provides them via a catalog of DSL operators. This DSL is being updated on each new release of Spark, making Spark SQL more and more expressive, and thus making it easier for the user to develop its data analysis applications.

3 Configuration and Code Examples

This chapter serves as a tutorial for installing and configuring the ecosystem of tools, as well as presenting some introductory data analysis examples.

3.1 Local Cluster

The local cluster used in this project consist on a server with the following specs:

- **Processor:** Intel Xeon W-2175 2.5GHz, 4.3GHz Turbo, 14C, 19.25M Cache, HT, (140W) DDR4-2666.
- RAM: 64GB 4x16GB DDR4 2666MHz RDIMM ECC Memory.
- Hard Drive: M.2 2TB PCIe NVMe Class 40 Solid State Drive.
- Operating System: Ubuntu 20.04 LTS [17].

The server has been purchased with the funding for equipment assigned to the project.

3.2 Java

In this section we discuss how to install and configure Java OpenJDK 8 [18]. While there is a more recent version available (Java OpenJDK 11 [19]), Java OpenJDK 8 is the most recent version supported by the latest stable release of Spark, thus making it our JRE of choice.

Figure 3.1 presents the script to install and configure Java OpenJDK 8. It can be run from a terminal in our local server.

```
(01) #!/bin/bash
(02) sudo apt-get update
(03) sudo apt-get install openjdk-8-jdk
(04) sudo update-alternatives --config java
(05) sudo update-alternatives --config javac
(06) sudo gedit ~/.bashrc
```

Figure 3.1: Script openJDK_8.sh

Next, we present more detailed instructions about the steps followed in the script:

• (01) #!/bin/bash

We indicate bin bash as the interpreter being used.

• (02) sudo apt-get update

We update apt-get, the tool to handle packages in Linux.

- (03) sudo apt-get install openjdk-8-jdk We install Java OpenJDK 8.
- (04) sudo update-alternatives --config java

We ensure that Java OpenJDK 8 is the default JRE. The command shows the different JREs installed in the system, highlighting with a symbol \star the default one. For example, in the case below both Java OpenJDK 8 and Java OpenJDK 11 are installed in the system, with Java OpenJDK 8 being the default one being used.

Selection Path Usr/lib/jvm/java-11-openjdk-amd64/bin/java (usr/lib/jvm/java-11-openjdk-amd64/bin/java 2 /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java

• (05) sudo update-alternatives --config javac

We ensure that Java OpenJDK 8 is the default Java compiler. The command shows the different Java compilers installed in the system, highlighting with a symbol * the default one. For example, in the case below both Java OpenJDK 8 and Java OpenJDK 11 are installed in the system, with Java OpenJDK 8 being the default one being used.

Selection Path 0 /usr/lib/jvm/java-11-openjdk-amd64/bin/javac 1 /usr/lib/jvm/java-11-openjdk-amd64/bin/javac * 2 /usr/lib/jvm/java-8-openjdk-amd64/bin/javac

• (06) sudo gedit ~/.bashrc

We use super user privileges with the text editor gedit so as to modify the content of the configuration file bashrc. In our case we add:

```
export JAVA_HOME="/usr/lib/jvm/java-1.8.0-openjdk-amd64"
export PATH=$PATH:/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin
```

3.3 Python

In this section we discuss how to install and configure Python 3.7.7 [20]. While there is a more recent version available (Python 3.8.2 [21]), Python 3.7.7 is the most recent version supported by the latest stable release of Spark, thus making it our programming language of choice.

Figure 3.2 presents the script to install and configure Python 3.7.7. It can be run from a terminal in our local server.

Next, we present more detailed instructions about the steps followed in the script:

• (01) #!/bin/bash

We indicate bin bash as the interpreter being used.

• (02) sudo apt update

We update apt, the tool for managing deb packages in Ubuntu.

• (03) sudo apt install build-essential zlib1g-dev libncurses5-dev \ libgdbm-dev libnss3-dev \

```
(01) #!/bin/bash
(02) sudo apt update
(03) sudo apt install build-essential zlib1g-dev \
              libncurses5-dev libgdbm-dev libnss3-dev \
              libssl-dev libreadline-dev \
              libffi-dev libsqlite3-dev \
         wget libbz2-dev
(04) wget https://www.python.org/ftp/python/3.7.7/Python-3.7.1tgz
(05) tar -xf Python-3.7.7.tgz
(06) cd Python-3.7.7
(07) ./configure --enable-optimizations
(08) make -j $(nproc)
(09) sudo make altinstall
(10) sudo python3.7 -m pip install -U pip
(11) which python3.7
(12) python3.7
```

Figure 3.2: Script python_3_7_7.sh

libssl-dev libreadline-dev \
libffi-dev libsglite3-dev \

```
wget libbz2-dev
```

We use apt to install the required dependency packages.

- (04) wget https://www.python.org/ftp/python/3.7.7/Python-3.7.7.tgz We download Python 3.7.7 as a Gzipped source tarball.
- (05) tar -xf Python-3.7.7.tgz We extract it.
- (06) cd Python-3.7.7

We move to the extracted Python 3.7.7 folder.

• (07) ./configure --enable-optimizations

We run the configure script, which looks for dependencies.

• (08) make -j \$(nproc)

We run the make accross the number of processors we have in the server.

• (09) sudo make altinstall

We use altinstall instead of install so as to allow Python 3.7.7 to co-exist with other potential versions installed in the server.

• (10) sudo python3.7 -m pip install -U pip

We upgrade pip, the package installer for Python, to its most recent version 20.1 [22].

 (11) which python3.7
 We get the location of Python3.7.7 in our server. In this case it is: /usr/local/bin/python3.7

 (12) python3.7
 We launch Python3.7.7 to ensure it has been installed. In this case we should get the following: Python 3.7.7 (default, May 6 2020, 16:41:07) [GCC 7.5.0] on linux Type "help", "copyright", "credits" or "license" for more information.

3.4 Hadoop

In this section we discuss how to install, configure and manage a Hadoop cluster. We also present a MapReduce application running on the cluster.

3.4.1 Installing Hadoop

We install and configure Hadoop 2.7.1 [23]. While there is a more recent version available (Hadoop 3.2.1 [24]), Hadoop 2.7.x is the most recent version supported by the latest stable release of Spark, thus making it our version of choice.

Figure 3.3 presents the script to install and configure Hadoop 2.7.1 as a Single Node Cluster with Pseudo-Distributed Operation. We choose this mode as our cluster contains just 1 node. Thus, all the HDFS, Yarn, MapReduce and Spark daemons presented in Chapter 2 still take place, each of them running as an independent process. The script can be run from a terminal in our local server.

Next, we present more detailed instructions about the steps followed in the script:

```
• (01) #!/bin/bash
```

We indicate bin bash as the interpreter being used.

• (02) sudo apt-get install openssh-server openssh-client

We need to be able to access localhost by ssh without a passphrase. Thus, first we use apt-get to install OpenSSH, which is the premier connectivity tool for remote login with the ssh protocol [25].

• (03) ssh-keygen -t rsa -P ""

We use ssh-keygen to generate a public/private rsa key pair. If run successfully, it saves the identification and public key in $HOME \$.

• (04) cat \$HOME/.ssh/id_rsa.pub >> \$HOME/.ssh/authorized_keys

We use the command cat to append the public keys (generated by ssh-keygen) to our file of listed authorised keys.

• (05) ssh localhost

We ensure we can connect now by ssh to the localhost without a passphrase.

• (06) wget https://archive.apache.org/dist/hadoop/

common/hadoop-2.7.1/hadoop-2.7.1.tar.gz

We download Hadoop 2.7.1 as a Gzipped source tarball.

• (07) tar -xzvf hadoop-2.7.1.tar.gz We extract it.

Deliverable WP2-D1

```
(01) #!/bin/bash
(02) sudo apt-get install openssh-server openssh-client
(03) ssh-keygen -t rsa -P ""
(04) cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
(05) ssh localhost
(06) wget https://archive.apache.org/dist/hadoop/
                  common/hadoop-2.7.1/hadoop-2.7.1.tar.gz
(07) tar -xzvf hadoop-2.7.1.tar.gz
(08) sudo mv hadoop-2.7.1 /usr/local/hadoop/
(09) sudo gedit ~/.bashrc
(10) sudo gedit /usr/local/hadoop/etc/hadoop/hadoop-env.sh
(11) sudo gedit /usr/local/hadoop/etc/hadoop/core-site.xml
(12) sudo gedit /usr/local/hadoop/etc/hadoop/hdfs-site.xml
(13) sudo gedit
         /usr/local/hadoop/etc/hadoop/mapred-site.xml.template
(14) sudo cp
         /usr/local/hadoop/etc/hadoop/mapred-site.xml.template
         /usr/local/hadoop/etc/hadoop/mapred-site.xml
(15) sudo gedit /usr/local/hadoop/etc/hadoop/yarn-site.xml
(16) hadoop
```

Figure 3.3: Script Hadoop_2_7_1.sh

- (08) sudo mv hadoop-2.7.1 /usr/local/hadoop/ We move the extracted folder to /usr/local/hadoop.
- (09) sudo gedit ~/.bashrc

We use super user privileges with the text editor gedit so as to modify the content of the configuration file bashrc. In our case we add:

```
export PATH=$PATH:/usr/local/hadoop/bin/:/usr/local/hadoop/sbin/
export HADOOP_HOME=/usr/local/hadoop/
export HADOOP_CONF_DIR=/usr/local/hadoop/
export HADOOP_MAPRED_HOME=/usr/local/hadoop/
export HADOOP_COMMON_HOME=/usr/local/hadoop/
export HADOOP_HDFS_HOME=/usr/local/hadoop/
export YARN_HOME=/usr/local/hadoop/
export HADOOP_COMMON_LIB_NATIVE_DIR=/usr/local/hadoop/lib/native
export HADOOP_OPTS="-Djava.library.path=/usr/local/hadoop/lib"
export JAVA_LIBRARY_PATH=$HADOOP_HOME/lib/native:$JAVA_LIBRARY_PATH
```

• (10) sudo gedit /usr/local/hadoop/etc/hadoop/hadoop-env.sh We use super user privileges with the text editor gedit so as to modify the content of the configuration file hadoop-env.sh. In our case we add:

export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64

• (11) sudo gedit /usr/local/hadoop/etc/hadoop/core-site.xml

We use super user privileges with the text editor gedit so as to modify the content of the configuration file core-site.xml. In our case we fill the (originally empty <configuration></configuration>) configuration section with:

```
<configuration>
<property>
<name>fs.defaultFS</name>
<value>hdfs://localhost:9000</value>
</property>
</configuration>
```

• (12) sudo gedit /usr/local/hadoop/etc/hadoop/hdfs-site.xml

We use super user privileges with the text editor gedit so as to modify the content of the configuration file hdfs-site.xml. In our case we fill the (originally empty <configuration></configuration>) configuration section with:

```
<configuration>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
</configuration>
```

• (13) sudo gedit

/usr/local/hadoop/etc/hadoop/mapred-site.xml.template

We use super user privileges with the text editor gedit so as to modify the content of the configuration file mapred-site.xml.template. In our case we fill the (originally empty <configuration></configuration>) configuration section with:

```
<configuration>
<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>
</configuration>
```

• (14) sudo cp

/usr/local/hadoop/etc/hadoop/mapred-site.xml.template
/usr/local/hadoop/etc/hadoop/mapred-site.xml

We use super user privileges to copy the file mapred-site.xml.template to the new file mapred-site.xml.

```
• (15) sudo gedit /usr/local/hadoop/etc/hadoop/yarn-site.xml
```

We use super user privileges with the text editor gedit so as to modify the content of the configuration file yarn-site.xml. In our case we fill the (originally empty <configuration></configuration>) configuration section with:

```
<configuration>
  <property>
      <name>yarn.nodemanager.aux-services</name>
      <value>mapreduce_shuffle</value>
```

(16) hadoop	
We launch Hadoop 2.7.1 to ensure	t has been installed. In this case we should get the following:
Usage: hadoop [config	g confdir] [COMMAND CLASSNAME]
CLASSNAME	run the class named CLASSNAME
or	
where COMMAND is one	of:
fs	run a generic filesystem user client
version	print the version
jar <jar></jar>	run a jar file
	note: please use "yarn jar" to launch
	YARN applications, not this command.
checknative [-a -h]	check native hadoop and compression
libraries availabilit	τy
distcp <srcurl> <dest< td=""><td>turl> copy file or directories recursively</td></dest<></srcurl>	turl> copy file or directories recursively
archive -archiveName	NAME -p <parent path=""> <src>* <dest></dest></src></parent>
create a hadoop arch:	ive
classpath	prints the class path needed to get the
credential	interact with credential providers
	Hadoop jar and the required libraries
daemonlog	get/set the log level for each daemon
trace	view and modify Hadoop tracing settings

Most commands print help when invoked w/o parameters.

3.4.2 Start a Hadoop Cluster

We start the Hadoop Single Node Cluster with Pseudo-Distributed Operation.

Figure 3.4 presents the script to start the cluster. It can be run from a terminal in our local server.

```
(01) #!/bin/bash
(02) ssh localhost
(03) hdfs namenode -format
(04) start-dfs.sh
(05) start-yarn.sh
(06) hdfs dfs -mkdir /user/
(07) hdfs dfs -mkdir /user/my_HDFS/
```

Figure 3.4: Script Start_Hadoop_Cluster.sh

Next, we present more detailed instructions about the steps followed in the script:

• (01) #!/bin/bash

We indicate bin bash as the interpreter being used.

• (02) ssh localhost

We connect by ssh to the localhost without a passphrase.

• (03) hdfs namenode -format

We ensure HDFS has some format. If the command is successfully executed, then we should get the following, where ***MACHINE_NAME*** represents the name of the server:

```
STARTUP_MSG: Starting NameNode
STARTUP_MSG: host = ***MACHINE_NAME***/127.0.1.1
STARTUP_MSG: args = [-format]
STARTUP_MSG: version = 2.7.1
STARTUP_MSG: classpath =
STARTUP_MSG: java = 1.8.0_252
SHUTDOWN_MSG: Shutting down NameNode
at ***MACHINE_NAME***/127.0.1.1
```

• (04) start-dfs.sh

We start the HDFS daemons NameNode and DataNode. If the command is successfully executed, then we should get the following, where ***USER_NAME*** represents the name of the user:

```
Starting namenodes on [localhost]
localhost: starting namenode, logging to /usr/local/hadoop/
logs/hadoop-***USER_NAME***-namenode-***MACHINE_NAME***.out
localhost: starting datanode, logging to /usr/local/hadoop/
logs/hadoop-***USER_NAME***-***MACHINE_NAME***.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/
hadoop/logs/hadoop-***USER_NAME***-secondarynamenode-***MACHINE_NAME***
```

• (05) start-yarn.sh

We start the YARN Job Scheduler, specifically its daemons ResourceManager and NodeManager. If the command is successfully executed, then we should get the following:

```
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/
yarn-***USER_NAME***-resourcemanager-***MACHINE_NAME***.out
localhost: starting nodemanager, logging to /usr/local/hadoop/
logs/yarn-***USER_NAME***-nodemanager-***MACHINE_NAME***.out
```

• (06) hdfs dfs -mkdir /user/ We create the HDFS folder /user.

• (06) hdfs dfs -mkdir /user/my_HDFS/

We create the HDFS subfolder /user/my_HDFS/, where we can place the dataset to be analysed.

3.4.3 Dataset

We use the website Lipsum [26] to generate 4 paragraphs of *Lorem Ipsum* text. We create a dataset $my_dataset$ with 4 small text files, each of them containing one of the generated paragraphs. Figure 3.8 presents the content of $my_dataset$ in the local file system of our server.

3.4.4 HDFS

Once the cluster is started, we can check the status of the HDFS NameNode at and DataNode at http://localhost:50070/

Figures 3.6 and 3.7 show it. As we can see, the folder my_HDFS is empty.

Figure 3.5 presents the script to run a MapReduce or Spark data analysis application on top of HDFS. It can be run from a terminal in our local server.

```
(01) #!/bin/bash
(02) ssh localhost
(03) hdfs dfs -put ./my_dataset/ /user/my_HDFS/my_dataset
(04) # MapReduce or Spark Job Command
(05) hdfs dfs -get /user/my_HDFS/my_result ./
(06) hdfs dfs -rm -r /user/my_HDFS/my_dataset/
(07) hdfs dfs -rm -r /user/my_HDFS/my_result/
```

Figure 3.5: Script data_analysis.sh

Next, we present more detailed instructions about the steps followed in the script:

• (01) #!/bin/bash

We indicate bin bash as the interpreter being used.

• (02) ssh localhost

We connect by ssh to the localhost without a passphrase.

• (03) hdfs dfs -put ./my_dataset/ /user/my_HDFS/my_dataset

We use the HDFS command put to copy my_dataset from the local file system to HDFS. Figure 3.9 shows it.

• (04) # MapReduce or Spark Job Command

We leave out the details of running a MapReduce/Spark Job to the next section. By the moment we assume the MapReduce/Spark Job succeeds, producing the results in the new HDFS folder my_result. Figure 3.10 shows it.

- (05) hdfs dfs -get /user/my_HDFS/my_result ./ We use the HDFS command get to copy my_result from HDFS to our local file system. Figure 3.11 shows it.
- (06) hdfs dfs -rm -r /user/my_HDFS/my_dataset/
- (07) hdfs dfs -rm -r /user/my_HDFS/my_result/

We use the HDFS command rm to remove the folders my_dataset and my_result from HDFS. By doing so, we restore the content of HDFS to the one presented in Figure 3.7, i.e., ready to run the script again with a new data analysis application.

	tab-overview						
Hadoop Overview Datar	odes Datanode Volume Fa	ilures Snaps	shot Startup Progress	s Utilities –			
Overview 'locall	nost:9000' (active)						
Started:	Mon Jul 13 14:56:19 IST	2020					
Version:	2.7.1, r15ecc87ccf4a022	8f35af08fc56de	536e6ce657a				
Compiled:	2015-06-29T06:04Z by j	enkins from (det	tached from 15ecc87)				
Cluster ID:	CID-4ba043aa-b6fd-41f0)-8d50-81fed803	321c7				
Block Pool ID:	BP-1629936368-127.0.1	.1-15946485716	660				
FileJournalManager(root=/tmp/ha	.doop-alejandro/dfs/name)	EditLogFileOut	iputstream(/tmp/nauoop-	alejanaroransmann	oround no conto_mprog	_	
FileJournalManager(root=/tmp/ha	doop-alejandro/dfs/name)	EditLogFileOut	ipursu eant/unpriaucop-		oron on cons_mprov	_	
FileJournalManager(root=/tmp/ha	doop-alejandro/dfs/name)	EditLogFileOut	Туре		eroen en eens_mprog	State	
FileJournalManager(root=/tmp/ha	doop-alejandro/dfs/name)	EditLogFileOut	Type	ND_EDITS		State	
FileJournalManager(root=/tmp/ha NameNode S Storage Directory /tmp/hadoop-alejandro/dfs/name Hadoop Overview Datance	doop-alejandro/dfs/name) torage odes Datanode Volume Fai	EditLogFileOut	Type IMAGE_A shot Startup Progress	ND_EDITS Utilities -		State Active	
FileJournalManager(root=/tmp/ha NameNode S Storage Directory /tmp/hadoop-alejandro/dfs/name ladoop Overview Datance Datanode Info n operation	doop-alejandro/dfs/name) torage des Datanode Volume Fai	EditLogFileOut	Type IMAGE_A	ND_EDITS Utilities ~		State	
FileJournalManager(root=/tmp/ha NameNode S Storage Directory /tmp/hadoop-alejandro/dfs/name -Iadoop Overview Datand Datanode Info n operation Node	doop-alejandro/dfs/name) torage des Datanode Volume Fait symmation	e Capacity	Type IMAGE_A shot Startup Progress	ND_EDITS Utilities -	Blocks Block pool u	sed Failed Volumes	Version
FileJournalManager(root=/tmp/ha NameNode S Storage Directory /tmp/hadoop-alejandro/dfs/name Hadoop Overview Datance Datanode Info n operation Node Smartebuses:50010 (127.0.0.1.50010)	doop-alejandro/dfs/name) torage dods Datanode Volume Fai trimation Last contact Admin Stat 1 In Service	e Capacity 1.83 TB	Used Non DFS Used 24 KB 237.44 GB	ND_EDITS Utilities - Remaining I 1.6 TB (Blocks Block pool u 0 24 KB (0%)	sed Failed Volumes	Version 2.7.1
FileJournalManager(root=/tmp/ha NameNode S Storage Directory /tmp/hadoop-alejandro/dfs/name Hadoop Overview Datance Datanode Info n operation Node smartebuses:50010 (127.0.0.1:50010) Decomissioning	doop-alejandro/dfs/name) torage Datanode Volume Fai rimation	e Capacity 1.83 TB	Type IMAGE_A shot Startup Progress Used Non DFS Used 24 KB 237.44 GB	ND_EDITS Utilities - Remaining I 1.6 TB (Blocks Block pool u D 24 KB (0%)	sed Falled Volumes 0	Version 2.7.1
FileJournalManager(root=/tmp/ha NameNode S storage Directory /tmp/hadoop-alejandro/dfs/name adoop overview Datance Datanode Info n operation vode smartebuses:50010 (127.0.0.1:50010) Decomissioning	doop-alejandro/dfs/name) torage des Datanode Volume Fai firmation Last contact Admin Stat 1 In Service	e Capacity 1.83 TB	Type IMAGE_A shot Startup Progress used Non DFS Used 24 KB 237.44 GB	ND_EDITS Utilities Remaining I 1.6 TB (Blocks Block pool u 0 24 KB (0%)	sed Failed Volumes 0	Version 2.7.1

Figure 3.6: Cluster Overview

Permission

Owner

Group

Size

Hadoop	Overview	Datanodes	Snapshot	Startup Progress	Utilities -				
Brow		irooton							
DIOW	SE D	rectory	ý						
/user									Go!
Permissio	n c	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
drwxr-xr-x	8	lejandro	supergroup	0 B	7/13/2020, 3:03:03 PM	0	0 B	my_HDFS	
Hadoop	Overview	Datanodes	Snapshot	Startup Progress	Utilities 👻				
Hadoop	Overview	Datanodes	Snapshot	Startup Progress	Utilities -				
Hadoop Brow	overview	Datanodes	Snapshot	Startup Progress	Utilities -				
Hadoop Brow	overview	Datanodes	Snapshot	Startup Progress	Utilities -				

Figure 3.7: HDFS

Replication

Block Size

Name

Last Modified

my_data	aset								
<	> • tables	my_dataset →		٩	:= =	008			
\odot	Recent								
企	Home	file 1.txt	file 2.txt	file 3.txt	file 4.txt				
Ē.	Desktop								
D	Documents								
⇒	Downloads								
99	Music								
۵	Pictures								
•	Videos			"File 1 by	t" colocted (r	02 hutos)			
	Trash	_		Inte_1.cx	t selected (s	82 Dytes)			
Ор	en▼ 🖪			f 1ARTeBuses/Deli	ile_1.txt verables/WP2/0			Save	
Lore habi veli ac v curs comm	em ipsum dolor itant morbi tr it pulvinar vo vehicula liber sus turpis. Qu nodo eget maur:	sit amet, co istique seneo lutpat. Integ o, quis tinc: isque mattis is ut molest:	onsectetur ctus et ne ger lorem idunt erat augue vit ie. Phasel	r adipisci etus et ma sapien, p . Pellent tae ligula llus purus	ng elit. lesuada osuere u esque nu ultrició enim, to	Cras et : ames ac f augue eg la metus es, eu bil mpus non	iaculis nibh. turpis egesta get, tincidun , porta sit a bendum eros r elit ut, rut	Pellentesque s. Quisque te t lacinia qua met odio in, utrum. Vivamu rum tristique	mpus a mm. Etiam lacinia s tellus.
-						Plain Text 🔻	Tab Width: 8 🔻	Ln 1, Col 582	▼ INS

Figure 3.8: Dataset Folder: my_dataset

	aleja	ndro@smartebuses: ~/Files	
File Edit View S	Search Terminal Help		
alejandro@smar	tebuses:~/Files\$ clea	г	
alejandro@smart alejandro@smart	tebuses:~/Files\$ hdfs tebuses:~/Files\$	dfs -put ./my_dataset/ /user/my_H	IDFS/my_dataset

Browse Directory

er/my_HDFS/my	_dataset						
ermission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
w-rr	alejandro	supergroup	7 B	7/13/2020, 3:08:18 PM	1	128 MB	file_1.txt
w-rr	alejandro	supergroup	7 B	7/13/2020, 3:08:18 PM	1	128 MB	file_2.txt
w-rr	alejandro	supergroup	7 B	7/13/2020, 3:08:18 PM	1	128 MB	file_3.txt
/W-rr	alejandro	supergroup	7 B	7/13/2020, 3:08:18 PM	1	128 MB	file_4.txt
	odes	Snanshot Start	un Progress	l Itilities 👻			
	oues	File informatio	n - file 1.b	xt	×		
	ton	Download					
	lory	/					
		Block informat	ion Block	0 🔻			
		Block ID: 1073	741825 BP-16299363	68-127 0 1 1-1594648571660		Block Size	
		Generation Sta	amp: 1001			128 MB	
		Size: 7				128 MB	
		Availability:				128 MB	
		 smarteb 	uses			128 MB	
					Close		
					Close		

Figure 3.9: Command put: my_dataset from local file system to HDFS

Hadoop	Overview	Datanodes	Snapshot	Startup Progress	Utilities -				
Brow	se Di	irecto	У						
/user/my_HDI	-5/								G0!
Permission	0	wner	Group	Size	Last Modified	Replication	Block Size	Name	
drwxr-xr-x	al	ejandro	supergroup	0 B	13/07/2020, 15:30:11	0	0 B	my_dataset	
drwxr-xr-x	al	ejandro	supergroup	0 B	13/07/2020, 18:19:44	0	0 B	my_result	
Hadoop	Overview	Datanodes	Snapshot	Startup Progres	s Utilities -				

Browse Directory

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
rw-rr	alejandro	supergroup	0 B	13/07/2020, 18:19:44	1	128 MB	_SUCCESS	
w-rr	alejandro	supergroup	1.46 KB	13/07/2020, 18:19:44	1	128 MB	part-00000	
		dee Oessekst						
	ano	File info	mation - part	-00000		×		
	_							
	` †	Orv Download						
		Block in	nformation E	Block 0 🔻				
		Plack	0: 1072741840					
		Block F	ool ID: BP-1629	936368-127.0.1.1-1594648571660		Block Siz		
	2	s Genera	tion Stamp: 102	5		128 MB		
	c	s Size: 14	1			128 MB		
	_	Availab	ility:					
		• 5	martebuses					
	_							
	_							

Figure 3.10: HDFS With New Folder my_result

	<pre>~/CIT/2_Research/2_SMARTeBuses/Deliverables/WP2/Code/Setu</pre>			
part-00000 _SUCCES	s			
OpenEnam(2)nec(6)neque(1)nibk(5)nisi(3)nisi(1)non(6)nulla(4)nunc(1)odio(3)orci(2)pellentesquephasellusplaceratporta(1)porttitorposuere(3)	part-00000 -/CIT/2_Research/2_SMARTeBuses/Deliverables/WP2/Code/Setup/3_Hadoop/4_MapReduce/my_result (7) (1) (3) (1) (2)	Save	= (

Figure 3.11: Command get: my_result from HDFS to local file system

3.4.5 MapReduce

 \sim

Once the cluster is started, we can check the status of the Yarn ResourceManager and NodeManager at http://localhost:8088/

Figure 3.12 shows it. As we can see, there is no application being run so far.

							Abo	out th	e Clu	istei	r				Logg	ed in as: dr.who
- Cluster	Cluster Met	rics														
About Nodes	Apps Submitted	Apps Pending	Apps Running	Apps Complete	Containers ed Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
Node Labels	0	0	0	0	0	0 B	8 GB	0 B	0	8	0	1	0	0	0	0
Applications	Scheduler M	Metrics														
NEW		Scheduler	Type		Sched	uling Resourc	e Tyne			Minimum	Allocation			Maximum	Allocation	
NEW SAVING SUBMITTED	Capacity Sc	heduler	1900	IME	MORYI	uning ricoouro	0 1300	<m< th=""><th>emory:1024.</th><th>vCores:1></th><th>, aloouton</th><th></th><th><memory:8192.< th=""><th>vCores:8></th><th>17 diooddon</th><th></th></memory:8192.<></th></m<>	emory:1024.	vCores:1>	, aloouton		<memory:8192.< th=""><th>vCores:8></th><th>17 diooddon</th><th></th></memory:8192.<>	vCores:8>	17 diooddon	
ACCEPTED									,						01	
RUNNING			Cluste	× ID: 150	4660054999											uster overview
FAILED		Pasaura	oManagar	toto: ST/	4000234000											
KILLED	F	ResourceMa	anager HA s	tate: acti												
Cabadular	Besou	rceManage	r RMStateS	tore: ora	apache.hadoop.y	arn.server.res	ourcemana	ger.recovery.Ni	IIRMStateSt	ore						
Scheduler	Re	sourceMan	ager started	don: Mor	Jul 13 18:10:54	+0100 2020		5								
		ResourceN	lanager vers	sion: 2.7.	1 from 15ecc87c	f4a0228f35af	08fc56de53	6e6ce657a by	enkins sour	ce checksu	m 1042198b3cf	b903a5d8d	e2fdcd09218 on 201	5-06-29T0	6:12Z	
10015			Hadoop vers	sion: 2.7.	1 from 15ecc87c	f4a0228f35af	08fc56de53	6e6ce657a by	enkins sour	ce checksu	m fc0a1a23fc18	868e4d5ee7	fa2b28a58a on 2015	5-06-29T06	5:04Z	
															Logg	ed in as: dr.who

Nodes of the cluster

- Cluster	Cluster Met	rics															
About Nodes	Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decom	missioned odes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
Node Labels	0	0	0	0	0	0 B	8 GB	0 B	0	8	0	1	<u>0</u>		0	<u>0</u>	0
Applications	Scheduler I	Metrics															
NEW SAVING		Scheduler 1	Гуре		Sched	luling Resource	Туре			Minimum	Allocation				Maximum	Allocation	
SUBMITTED	Capacity Sc	heduler		[MEMC	PRY]			<11	emory:1024,	vCores:1>			<merr< td=""><td>ory:8192, \</td><td>/Cores:8></td><td></td><td></td></merr<>	ory:8192, \	/Cores:8>		
RUNNING	Show 20 🗸	entries													Sea	irch:	
FINISHED FAILED	Node Labels ^	Rack \$	Node State ≎	Node Add	dress ≎	Node HTTP Address	≎ La	ast health-upda	te ≎	Health-rep	oort \diamond C	Containers	Mem Used ≎	Mem Avail ≎	VCore Used	s VCore ≎ Avail	s Version ≎ ≎
Scheduler		/default- rack	RUNNING	smartebuse	es:38381 <u>sm</u>	nartebuses:804	2 Mon Ju 2020	ul 13 18:12:56	+0100		0		0 B	8 GB	0	8	2.7.1
> Tools	Showing 1 to	o 1 of 1 entri	ÐS												First	Previous 1	Next Last
							ΔΙ	Annl	icati	ons						Logge	d in as: dr.who:
								· Abbi	Guu	0113							

		-														
- Cluster	Cluster Met	rics														
About Nodes	Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	/ Rebooted Nodes
Node Labels Applications	0 Scheduler M	0 Metrics	0	0	0	0 B	8 GB	0 B	0	8	0	<u>1</u>	<u>0</u>	<u>0</u>	0	<u>0</u>
<u>NEW</u> NEW_SAVING SUBMITTED	Capacity Sc	Scheduler	Гуре	[MEM	Schedu ORY]	ling Resourc	е Туре	<	emory:1024,	Minimum vCores:1>	Allocation		<memory:8192,< td=""><td>Maximum vCores:8></td><td>Allocation</td><td></td></memory:8192,<>	Maximum vCores:8>	Allocation	
ACCEPTED RUNNING	Show 20 🗸	entries												Sea	arch:	
FINISHED FAILED KILLED	ID - Us	ser ≎ I	Name 💠	Applic	ation Type	≎ Que	eue ¢	StartTime No data av	♦ Fin railable in tab	nishTime Ile	≎ State	\$ F	inalStatus 🗘	Progres	is ¢	Tracking UI 💲
Scheduler	Showing 0 to	o 0 of 0 entri	es											F	rst Previou	is Next Last
> Tools																

Figure 3.12: ResourceManager Overview

							A	ll Appl	icati	ons					Logo	ed in as: dr.who
	Cluster Met	rics														
About	Apps	Apps	Apps	Apps	Containers	Memory	Memo	ry Memory Beserved	VCores	VCores	VCores	Active	Decommissioned	Lost	Unhealthy	Rebooted
Node Labels	1	1	0	0	0	0 B	8 GB	0 B	0	8	0 .	140000	0	0	0	0
Applications	Scheduler N	Metrics											<u>^</u>	<u>^</u>	<u>^</u>	
NEW SAVING		Scheduler -	Туре		Schedu	ing Resourc	ce Type			Minimum /	Allocation			Maximun	n Allocation	
SUBMITTED	Capacity Sc	heduler		[MEM	ORY]			<me< td=""><td>emory:1024,</td><td>vCores:1></td><td></td><td colspan="5"><memory:8192, vcores:8=""></memory:8192,></td></me<>	emory:1024,	vCores:1>		<memory:8192, vcores:8=""></memory:8192,>				
ACCEPTED RUNNING	Show 20 ¥	entries												Se	arch:	
FINISHED FAILED		ID	*	User ≎	Na	ne	\$	Application Type	Queue \$	StartTime	FinishTime	State	≎ FinalStatus	Prog	gress ≎ T	racking UI ≎
Scheduler	application	1594660254	888 0001	alejandro s	streamjob455462	3490106481	1759.jar	MAPREDUCE	default	Mon Jul 13 18:19:31 +0100 2020	N/A	ACCEPT	ED UNDEFINED		Ap	olicationMaster
→ Tools	Showing 1 to	o 1 of 1 entri	es											First	Previous 1	Next Last
Ene				Арр	olicatio	on ap	oplie	cation_	_159	4660)2548	88_	0001		Logg	ed in as: dr.who
- Cluster	Kill Applicati	on														
About					Heen eleiend										Applic	ation Overview
Nodes					Name: stream	ro ob45546234	49010648	1759 jar								
Applications				Applicatio	n Type: MAPRI	EDUCE	10010040	1700.jul								

Node Labels	Name:	streamjob4554	4623490106481759.jar						
Applications	Application Type:	MAPREDUCE							
NEW	Application Tags:								
NEW SAVING	YarnApplicationState:	ACCEPTED: v	waiting for AM container to be allocat	ed, launched and reg	ister with RM.				
SUBMITTED	FinalStatus Reported by AM:	Application ha	s not completed yet.						
ACCEPTED	Started:	Mon Jul 13 18	:19:31 +0100 2020						
EINICHED	Elapsed:	1sec							
FAILED	Tracking URL:	ApplicationMa	ster						
KILLED	Diagnostics:								
Cabadular									
Scheduler									
> Tools									Application Metrics
10013				Total Resource	Preempted:	<memory:0, vcores:0=""></memory:0,>			
			Total Number of	Non-AM Containers	Preempted:	0			
			Total Numb	er of AM Containers	Preempted:	0			
			Resource P	reempted from Curr	ent Attempt:	<memory:0, vcores:0=""></memory:0,>			
			Number of Non-AM Containers P	reempted from Curr	ent Attempt:	0			
				Aggregate Resource	e Allocation:	0 MB-seconds, 0 vcore-s	seconds		
								0 1	
	Show 20 V entries							Search:	
	Attempt ID	~	Started	\$		Node	\$	Logs	s \$
	appattempt 1594660254888 0001 000001		Mon Jul 13 18:19:31 +0100 2020	1	http://smartebus	es:8042	Logs		
	Showing 1 to 1 of 1 entries							First Previou	us 1 Next Last

Figure 3.13: ResourceManager: MapReduce Application in Progress

												Application Overview
					User: alejan	dro						application overheit
els				Applicati	Name: stream	njob4554623	49010648175	9.jar				
				Applicati	on Tags:	EDOOL						
D			Fin	YarnApplicati alStatus Reporte	onState: FINIS d by AM: SUCC	HED						
					Started: Mon J	ul 13 18:19:3	31 +0100 2020	1				
				Track	Elapsed: 12sec ing URL: Histor	v						
				Diag	nostics:							
												Application Metrics
						Total Reso	urce Preemp	ted: <memory:0< td=""><td>vCores:0></td><td></td><td></td><td>Application Method</td></memory:0<>	vCores:0>			Application Method
				Tota	al Number of Nor Total Number of	n-AM Contai	ners Preemp	ted: 0				
					Resource Pree	npted from	Current Atter	npt: <memory:0< td=""><td>vCores:0></td><td></td><td></td><td></td></memory:0<>	vCores:0>			
			Nu	nber of Non-AM	Containers Pree Age	npted from pregate Reso	Current Atter ource Allocat	npt: 0 ion: 55727 MB-s	econds, 33 vco	re-seconds		
	Show 20 v entries										Search:	
	Atte	mpt ID		v	Started		\$	No	de	\$	Lo	gs ≎
	appattempt 1594660254888 000	1_000001		Mon Jul 13 1	8:19:31 +0100 20	20	h	tp://smartebuses:	3042	Logs	Eirot Broui	oue 1 Next Last
	Chowing 1 to 1 of 1 childs										11131 1104	ous i nort Last
	_											Logged in as: dr.who
_				Α	I Appli	icatio	ons					
					• •							
	Cluster Metrics											
	Cluster Metrics	Apps	Containers M	lemory Memor	y Memory Reserved	VCores	VCores Total	VCores Ac	ive Decomn	nissioned L	.ost Unhe	althy Rebooted
	Cluster Metrics Apps Apps Apps Submitted Pending Running 1 0 0	Apps Completed	Containers M Running 0	lemory Memor Used Total 3 8 GB	y Memory Reserved 0 B	VCores Used 0	VCores Total 8 0	VCores Ac Reserved No 1	ive Decomn des No <u>0</u>	nissioned L des No <u>0</u>	ost Unhe odes No	althy Rebooted les Nodes 0
=[d	Cluster Metrics Apps Apps Apps Submitted Pending Running 1 0 0 Scheduler Metrics	Apps Completed	Containers M Running 0	lemory Memor Used Total 3 8 GB	y Memory Reserved 0 B	VCores Used 0	VCores Total 8 0	VCores Ac Reserved No 1	ive Decomn des No <u>0</u>	nissioned L des No Q	odes Unhe Dodes Nor	althy Rebooted les Nodes 0
	Cluster Metrics Apps Submitted Pending Apps 1 0 0 Scheduler Metrics Scheduler Type	Apps Completed	Containers M Running 0 0 Scheduling I	lemory Memor Used Total B 8 GB Resource Type	y Memory Reserved 0 B	VCores Used 0	VCores Total 8 0 Minimum Alle	VCores Ac Reserved No 1	ive Decomn No <u>0</u>	nissioned L des No 0 Ma	ximum Allocat	althy Rebooted Nodes <u>0</u>
NG 2	Cluster Metrics Apps Apps Apps Submitted Pending Running 1 0 0 Scheduler Metrics Scheduler Type Capacity Scheduler	Apps Completed 1 [MEMOF	Containers M Running 0 5 0 Scheduling I YY]	lemory Mernor Used Total 3 8 GB Resource Type	y Memory Reserved 0 B	VCores Used 0	VCores Total 8 0 Minimum Alle vCores:1>	VCores Ac Reserved No 1 Docation	ive Decomn les No 0_ <mem< td=""><td>nissioned L des Ni Q Ma pry:8192, vCor</td><td>ximum Allocat</td><td>althy Rebooted Nodes 0</td></mem<>	nissioned L des Ni Q Ma pry:8192, vCor	ximum Allocat	althy Rebooted Nodes 0
<u> </u>	Cluster Metrics Apps Apps Apps Submitted Pending Apps 1 0 0 Scheduler Metrics Scheduler Type Capacity Scheduler Show 20 v entries	Apps Completed 1 [MEMOF	Containers M Running 0 5 0 Scheduling I YY]	lemory Memor Used Total 3 8 GB Resource Type	y Memory Reserved 0 B	VCores Used 0	VCores Total 8 0 Minimum Alle vCores:1>	VCores Ac Reserved No 1	ive Decomn No 0 <mem< td=""><td>nissioned L des N 0 Ma ory:8192, vCor</td><td>ximum Allocat Search:</td><td>althy Rebooted Nodes Q</td></mem<>	nissioned L des N 0 Ma ory:8192, vCor	ximum Allocat Search:	althy Rebooted Nodes Q
LG LG	Cluster Metrics Apps Apps Apps Submitted Pending Running 1 0 0 Scheduler Metrics Scheduler Type Capacity Scheduler Show 20 ~ entries	Apps Completed 1 [MEMOF	Containers M Running 0 0 Scheduling I RY] Name	Iemory Used Total B 8 GB Resource Type	y Memory Reserved 0 B <me Application Type</me 	VCores Used 0 mmory:1024, v ≎ Queue ≎	VCores Total 8 0 Minimum Alk vCores:1> StartTime	VCores Acc Reserved No 1 cation ♦ FinishTime <	ive Decomm No 0 <mem State ≎</mem 	hissioned L des N 0 Ma ory:8192, vCor FinalStatus	ximum Allocations Search: ≎ Progre	althy Rebooted Nodes Q ion ss \diamond Tracking UI \diamond
NG 2	Cluster Metrics Apps Apps Apps Submitted Pending Running 1 0 0 Scheduler Metrics Capacity Scheduler Show 20 entries ID application 1594660254888 0001	Apps Completed 1 (MEMOP User alejandro stre	Containers Running D 0 0 Scheduling I IY] Name amjob4554623490	lemory Memor Used Total 3 8 GB Resource Type \$ 106481759.jar	y Memory Reserved 0 B <pre></pre>	VCores Used 0 emory:1024, v ≎ Queue ¢ default	VCores Total 8 0 Minimum Allk vCores:1> StartTime Mon Jul 13 18:19:31	VCores Ac Reserved No 1 ccation	ive Decomm No 0 <mem State ≎ FINISHED</mem 	hissioned L des N Ma ory:8192, vCor FinalStatus SUCCEEDEI	ximum Allocat search: ≎ Progre	althy Rebooted Nodes Q Ion ss ≎ Tracking UI History

Figure 3.14: ResourceManager: MapReduce Application Finished

Hadoop	Overview Datar	nodes Snapshot	Startup Progres	s Utilities -				
Brows	se Direc	ctory						
/user/my_HDFS	5/						Go	!
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
drwxr-xr-x	alejandro	supergroup	0 B	13/07/2020, 15:30:11	0	0 B	my_dataset	
drwxr-xr-x	alejandro	supergroup	0 B	13/07/2020, 18:19:44	0	0 B	my_result	
Hadoop	Overview Data	anodes Snapshot	Startup Progre	ess Utilities –				

Browse Directory

/user/my_HDFS/my	_result							Go!
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
-rw-rr	alejandro	supergroup	0 B	13/07/2020, 18:19:44	1	128 MB	_SUCCESS	
-rw-rr	alejandro	supergroup	1.46 KB	13/07/2020, 18:19:44	1	128 MB	part-00000	

Figure 3.15: HDFS: MapReduce Result in my_result



Figure 3.16: HDFS: MapReduce Result Brought Back to Local File System

We edit the command (04) # MapReduce or Spark Job Command from the script data_analysis.sh(cf. Figure 3.5) to run our introductory MapReduce application. The MapReduce command is presented below:

On it, we use the library Hadoop Streaming [15] to program a MapReduce application where the Map and Reduce stages are specified in the Python files my_mapper.py and my_reducer.py, resp. Both my_mapper.py and my_reducer.py are executable scripts, configured to read from the standard input and write to the standard output. The MapReduce application itself is quite simple: it produces as output the new folder my_result, containing the word count for the dataset provided in the input folder my_dataset.

Figures 3.17, 3.18 and 3.19 present the file my_mapper.py.

```
(01) # -----
(02) # IMPORTS
(03) # -----
(04) import sys
(05) import re
```

Figure 3.17: my_mapper.py: Import Section

We present more detailed instructions about the steps followed in the script:

- Lines (01)–(05) import the libraries sys and re. The former is used to redirect the input and output streams to stdin and stdout, resp. The latter is used to compile regular expressions for processing the content being read.
- Lines (06)–(27) define the function my_map. It receives as parameters an input and an output stream. The function reads the whole content provided by the input stream, producing as result a number of text lines to be written to the output stream. In concrete, for each different word being read by the standard input, the function produces a key-value line with the format word\t (num_appearances) \n, where num_appearances is the amount of times the word word has been read.

The function uses a dictionary my_dict to collect as keys all words being read, with each word having as its associated key the number of appearances. my_dict is initialised in line (11) and it is populated in the for loop of lines (13)–(23). Line (14) ensures we read one line from the standard input at a time. Line (16) splits the text line to the list of words it contains. The loop of lines (18)–(23) processes each word separately. On line (19) we use re.sub and lower to remove any non-alphabetic character and to lower any upper case letter appearing in the word,

```
(06) # ------
(07) # FUNCTION my map
(08) # ------
(09) def my_map(my_input_stream,
               my_output_stream
              ):
        # 1. We create a dictionary with all
(10)
             the different words in the file
        my dict = \{\}
(11)
(12)
        num_appearances = 1
(13)
        # 2. We traverse the file content, to populate my_dict
(14)
        for line in my_input_stream:
            # 2.1. We process the line
(15)
            word_list = line.split(" ")
(16)
(17)
            # 2.2. We populate the dictionary
                   with the words of the sentence
(18)
            for w in word list:
                my_word = re.sub(r"[^a-zA-Z]", "", w).lower()
(19)
                if (my_word in my_dict):
(20)
(21)
                    my_dict[my_word] =
                     my_dict[my_word] + num_appearances
(22)
                else:
(23)
                   my_dict[my_word] = num_appearances
        # 3. We write the content of the dict
(24)
(25)
        for key in my_dict:
            my_str = key + " \setminus t (" + str(my_dict[key]) + ") \setminus n"
(26)
(27)
            my_output_stream.write(my_str)
```

Figure 3.18: my_mapper.py: my_map Function

resp. Finally, lines (20)–(23) check if the word has already been registered previously. If so, it increments its number of appearances by one. Otherwise, it enters the word in the dictionary with a single appearance.

Once the entire content of the standard input has been read and processed, lines (24)–(27) produce the lines to be written by the output stream. Line (25) traverses the words stored in the dictionary. Line (26) produces the String with the key-value pair associated to the word. Finally, line (27) writes this String to the standard output.

• Lines (31)–(38) define the main entry point for the program. Lines (35)–(36) redirect the input and output streams to stdin and stdout, resp. Line (38) calls to the aforementioned function my_map.

```
(28) # -----
(29) # MAIN
(30) # ------
(31) if _____ == '___main__':
       # 1. We use as many input arguments as needed
(32)
(33)
       pass
(34)
       # 2. We set the input and output streams
       my_input_stream = sys.stdin
(35)
(36)
       my_output_stream = sys.stdout
(37)
       # 3. We launch the Map program
(38)
       my_map(my_input_stream,
           my_output_stream
          )
```

Figure 3.19: my_mapper.py: Main Entry Point

A copy of the file my_mapper.py is to be placed on each DataNode of the cluster involved in the Map stage, so as to process the subset of my_dataset associated to it.

Figures 3.20, 3.21 and 3.22 present the file my_reducer.py.

(01) # ----(02) # IMPORTS
(03) # ----(04) import sys

Figure 3.20: my_mapper.py: Import Section

The program is very similar to my_mapper.py, so we only higlight the differences:

- Lines (01)–(04) do no longer need to import the library re.
- Lines (05)-(26) define the function my_reduce. The function has the same responsability as my_map. However, instead of reading from my_dataset, the function reads the sorted key-value pairs word\t (num_appearances) \n produced by the stage my_map. Lines (14)-(17) parse each entry to get its associated word and number of appearances.
- Lines (31)–(38) define the main entry point for the program, calling to the aforementioned function my_reduce.

A copy of the file my_reducer.py is to be placed on each node of the cluster involved in the Reduce stage, so as to process the subset of the sorted key-value entries associated to it.

Figure 3.13 presents the status of the ResourceManager once the command (04) of the script data_analysis.sh is launched. As we can see, the MapReduce application is considered to

```
(05) # ------
(06) # FUNCTION my reduce
(08) def my_reduce(my_input_stream,
                  my_output_stream
                 ):
(09)
        # 1. We create a dictionary with all
             the different words in the file
        my dict = \{\}
(10)
(11)
        # 2. We traverse the file content, to populate my_dict
(12)
        for line in my_input_stream:
            # 2.1. We get the info from the line
(13)
            line = line.replace("\n", "")
(14)
            info = line.split("\t")
(15)
(16)
            my_word = info[0]
(17)
            num_appearances = int(info[1][1:-1])
            # 2.2. We populate the dictionary
(18)
                   with the words of the sentence
            if (my_word in my_dict):
(19)
(20)
               my_dict[my_word] =
                 my_dict[my_word] + num_appearances
(21)
            else:
(22)
               my_dict[my_word] = num_appearances
        # 3. We write the content of the dict
(23)
(24)
        for key in my_dict:
            my_str = key + " \setminus t (" + str(my_dict[key]) + ") \setminus n"
(25)
(26)
            my_output_stream.write(my_str)
```

Figure 3.21: my_mapper.py: my_map Function

be in progress. Figure 3.14 presents the status once the application finishes. As the execution is successful, the new folder my_result is available now in HDFS, with Figure 3.15 showing it. While the content of the files is not directly accessible in HDFS, we can execute the command get to bring the folder back to our local file system, so as to explore it (Figure 3.16 shows it). All in all, the MapReduce application finds 135 different words with their associated number of appearances in $my_dataset$.

```
(27) # -----
(28) # MAIN
(29) # ------
(30) if _____ == '____main___':
      # 1. We use as many input arguments as needed
(31)
(32)
       pass
(33)
       # 2. We set the input and output streams
       my_input_stream = sys.stdin
(34)
(35)
       my_output_stream = sys.stdout
(36)
       # 3. We launch the Map program
(37)
       my_map(my_input_stream,
           my_output_stream
          )
```

Figure 3.22: my_mapper.py: Main Entry Point

3.4.6 Stop a Hadoop Cluster

Once we have run our data analysis application, we stop the Hadoop Single Node Cluster with Pseudo-Distributed Operation.

Figure 3.23 presents the script to stop the cluster. It can be run from a terminal in our local server.

```
(01) #!/bin/bash
(02) ssh localhost
(03) hdfs dfs -rm -r /user
(04) stop-yarn.sh
(05) stop-dfs.sh
```

Figure 3.23: Script Stop_Hadoop_Cluster.sh

Next, we present more detailed instructions about the steps followed in the script:

• (01) #!/bin/bash

We indicate bin bash as the interpreter being used.

• (02) ssh localhost

We connect by ssh to the localhost without a passphrase.

- (03) hdfs dfs -rm -r /user We delete our HDFS folder.
- (04) stop-yarn.sh

We stop the YARN Job Scheduler, specifically its daemons ResourceManager and NodeManager. If the command is successfully executed, then we should get the following:

```
stopping yarn daemons
stopping resourcemanager
localhost: stopping nodemanager
localhost: nodemanager did not stop gracefully after 5 seconds: killing
no proxyserver to stop
```

• (05) stop-dfs.sh

We stop the HDFS daemons NameNode and DataNode. If the command is successfully executed, then we should get the following:

```
Stopping namenodes on [localhost]
localhost: stopping namenode
localhost: stopping datanode
Stopping secondary namenodes [0.0.0.0]
0.0.0.0: stopping secondarynamenode
```

3.5 Spark

In this section we discuss how to install and configure Spark. We also present a Spark Core and a Spark SQL application running on top of the Hadoop cluster described in Section 3.4.

3.5.1 Installing Spark

We install and configure Spark 2.4.5 [27]. While there is a more recent version available (Spark 3.0.0 [28]), this version is still in preview mode and thus it is not stable.

Figure 3.24 presents the script to install and configure Spark 2.4.5. It can be run from a terminal in our local server.

```
(01) #!/bin/bash
(02) wget https://downloads.apache.org/spark/spark-2.4.5/
spark-2.4.5-bin-hadoop2.7.tgz
(03) tar -xzvf spark-2.4.5-bin-hadoop2.7.tgz
(04) sudo mv spark-2.4.5-bin-hadoop2.7 /usr/local/spark
(05) python3.7 -m pip install pyspark
(06) sudo gedit ~/.bashrc
(07) spark-submit --version
```

Figure 3.24: Script Spark_2_4_5.sh

Next, we present more detailed instructions about the steps followed in the script:

• (01) #!/bin/bash

We indicate bin bash as the interpreter being used.

• (02) wget https://downloads.apache.org/spark/spark-2.4.5/ spark-2.4.5-bin-hadoop2.7.tgz

We download Spark 2.4.5 as a Gzipped source tarball. Among the different versions, we choose the one that is pre-built for Hadoop 2.7.

- (03) tar -xzvf spark-2.4.5-bin-hadoop2.7.tgz We extract it.
- (04) sudo mv spark-2.4.5-bin-hadoop2.7 /usr/local/spark We move the extracted folder to /usr/local/spark.
- (05) python3.7 -m pip install pyspark

We use pip, the package installer for Python, to download pyspark 2.4.5 [29]. This package is useful as it allows us to develop, debug and run a Spark application locally, using the Python 3.7.7 interpreter and the Python IDE PyCharm [30]. Once the application has been tested, it can be submitted for running in the Hadoop Single Node Cluster with Pseudo-Distributed Operation described in Section 3.4.

• (06) sudo gedit ~/.bashrc

We use super user privileges with the text editor gedit so as to modify the content of the configuration file bashrc. In our case we add:

```
export PATH=$PATH:/usr/local/spark/bin/
export PYSPARK_PYTHON=python3.7
```

• (07) spark-submit --version

We launch Spark 2.4.5 to ensure it has been installed. In this case we should get the following: Welcome to

Using Scala version 2.11.12, OpenJDK 64-Bit Server VM, 1.8.0_252 Branch HEAD

Compiled by user centos on 2020-02-02T19:38:06Z Revision cee4ecbb16917fa85f02c635925e2687400aa56b Url https://gitbox.apache.org/repos/asf/spark.git Type --help for more information.

3.5.2 Spark Core

In Section 3.4 we started the cluster and run a MapReduce application on it. Figure 3.14 showed the status of the Yarn ResourceManager and NodeManager (available at http://localhost: 8088/) including such MapReduce application.

We edit now the command (04) # MapReduce or Spark Job Command from the script data_analysis.sh (cf. Figure 3.5) to run our introductory Spark Core application. The Spark Core command is presented below:

(04) spark-submit \setminus

```
--master yarn --deploy-mode cluster \
./my_Spark_Core_example.py \
/user/my_HDFS/my_dataset \
/user/my_HDFS/my_result
```

On it, we use <code>spark-submit</code> to launch the Spark Core application in the Hadoop Single Node Cluster ter with Pseudo-Distributed Operation. The line <code>--master yarn --deploy-mode cluster</code> specifies Yarn to be the ResourceManager handling the application. The line <code>./my_Spark_Core_example.py</code> specifies the Python file containing the Spark Core application. Finally, the lines <code>/user/my_HDFS/my_dataset</code> and <code>/user/my_HDFS/my_result</code> are the first and second parameter of the Python program, resp.

The functionality of the Spark Core application is equivalent to the one of the MapReduce application presented in Section 3.4: it produces as output the new folder my_result, containing the word count for the dataset provided in the input folder my_dataset.

Figures 3.25, 3.26 and 3.27 present the file my_Spark_Core_example.py.

```
(01) # -----
(02) # IMPORTS
(03) # -----
(04) import pyspark
(05) import sys
(06) import re
```

Figure 3.25: my_Spark_Core_example.py: Import Section

We present more detailed instructions about the steps followed in the script:

- Lines (01)-(06) import the libraries pyspark, sys and re. The library pyspark allows us to use the Spark Core API. The library sys is used to pass the input and output directories to the program. The library re is used to compile regular expressions for processing the content being read.
- Lines (07)-(20) define the function my_spark_core_job. It receives as parameters the SparkContext (sc) and the aforementioned input and output directories (my_dataset_dir and my_result_dir, resp). The function processes the dataset in my_dataset_dir to produce the new folder my_result_dir with its word count.

Line (12) applies the creator operation textFile (with the folder my_dataset_dir as parameter) to load the dataset into inputRDD, an RDD of String items. In particular, each line of text of the dataset file turns into one item of such RDD. Figure 3.8 showed the content of file_1.txt of my_dataset. Figure 3.28 presents how the first 2 lines of the file translate into 2 items of inputRDD.

Line (14) applies the transformation operation flatMap (with the lambda function lambda line: line.split(" ") as parameter) on inputRDD to produce the new RDD all_wordsRDD of String items. In particular, each item (text line) of inputRDD is to be split into a list with the words it contains, and then this word list is exploded, producing

```
(07) # -----
(08) # FUNCTION my spark core job
(09) # ------
(10) def my_spark_core_job(sc, my_dataset_dir, my_result_dir):
        # 1. Operation C1: textFile
(11)
        inputRDD = sc.textFile(my_dataset_dir)
(12)
        # 2. Operation T1: flatMap
(13)
(14)
        all wordsRDD =
                 inputRDD.flatMap(lambda line: line.split(" "))
(15)
        # 3. Operation T2: map
(16)
        clean_wordsRDD =
            all_wordsRDD.map(lambda w:
                  (re.sub(r"[^a-zA-Z]", "", w).lower(),
                   1
                  )
                           )
       # 4. Operation T3: reduceByKey
(17)
       solutionRDD =
(18)
                  clean_wordsRDD.reduceByKey(lambda x, y: x + y)
(19)
       # 5. Operation A1: saveAsTextFile
(20)
       solutionRDD.saveAsTextFile(my_result_dir)
```

Figure 3.26: my_Spark_Core_example.py: my_spark_core_job Function

one item per word. Figure 3.29 presents the items produced in all_wordsRDD for the first item of inputRDD showed in Figure 3.28.

Line (16) applies the transformation operation map (with the lambda function lambda w: (re.sub(r"[^a-zA-Z]", "", w).lower(), 1) as parameter) on all_wordsRDD to produce the new RDD clean_wordsRDD of tuple (String, int) items. In particular, each item (word) of all_wordsRDD is to see removed any non-alphanumeric character; besides that, any upper-case letter is turned into its equivalent lower-case letter. The function returns the tuple (word, 1), registering one appearance of the word. Figure 3.30 presents the items produced in clean_wordsRDD for the subset of items of all_wordsRDD showed in Figure 3.29.

Line (18) applies the transformation operation reduceByKey (with the lambda function lambda x, y: x + y as parameter) on clean_wordsRDD to produce the new RDD solutionRDD of tuple (String, int) items. In particular, all items containing the same key (word) are aggregated (adding the number of appearances). Figure 3.31 presents the content of solutionRDD for the words showed in Figure 3.30.

Line (20) applies the action <code>operation</code> <code>saveAsTextFile</code> (with the folder <code>my_result_dir</code>

```
(21)
    #
(22) # MAIN
(23) # ---
(24)
         # 1. We use as many input arguments as needed
         my_dataset_dir = "/FileStore/tables/my_dataset/"
(25)
         my_result_dir = "/FileStore/tables/my_result/"
(26)
(27)
         if (len(sys.argv) > 1):
            my_dataset_dir = sys.argv[1]
(28)
(29)
            my_result_dir = sys.argv[2]
         # 2. We configure the Spark Context
(30)
(31)
         sc = pyspark.SparkContext.getOrCreate()
         sc.setLoqLevel('WARN')
(32)
         print ("\n\n')
(33)
(34)
         # 3. We call to our main function
(35)
         my_spark_core_job(sc, my_dataset_dir, my_result_dir)
```

Figure 3.27: my_Spark_Core_example.py: Main Entry Point

as parameter) to store the content of solutionRDD into it. The result folder my_result_dir contains as many files as partitions are in solutionRDD. In particular, each item of the RDD is stored as a text line. Figures 3.34 and 3.35 present the files and content produced by solutionRDD when the program is executed in the Hadoop cluster.

• Lines (21)–(35) define the main entry point for the program. Lines (25)–(26) specify the input and output directories. When testing the program locally, we run the program in PyCharm and assign the dataset folder my_dataset of the local file system. When running the program in the Hadoop cluster we use lines (27)–(29) to set the input and output directories to the HDFS folders /user/my_HDFS/my_dataset and /user/my_HDFS/my_result (as described in the command (04) spark submit of the script data_analysis.sh for launching the program). Lines (31)–(33) create the SparkContext, configuring how verbose it should be on reporting the status when running the application. Finally, line (35) calls to the aforementioned function my_spark_core_job.

A copy of the file my_Spark_Core_example.py is to be placed on the DataNode the cluster running the Spark Driver process.

Figure 3.32 presents the status of the ResourceManager once the command (04) of the script data_analysis.sh is launched. As we can see, the Spark Core application is considered to be in progress. Figure 3.33 presents the status once the application finishes. As the execution is successful, the new folder my_result is available now in HDFS, with Figure 3.34 showing it. While the content of the files is not directly accessible in HDFS, we can execute the command get to bring the folder back to our local file system, so as to explore it (Figure 3.35 shows it). All in all, the Spark Core application finds 135 different words with their associated number of appearances in $my_dataset$.

Lorem ipsum dolor sit amet, elit. Cras et nibh. Pellentesque\n habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Quisque tempus a\n

• • •

Figure 3.28: inputRDD Content

Lorem ipsum dolor sit amet, elit. Cras et nibh. Pellentesque\n ...

Figure 3.29: all_wordsRDD Content

('lorem, 1)
('ipsum, 1)
('dolor, 1)
('sit, 1)
('amet, 1)
('elit, 1)
('cras, 1)
('et, 1)
('nibh, 1)
('pellentesque, 1)
...

Figure 3.30: clean_wordsRDD Content

```
('lorem', 4)
('ipsum', 2)
('dolor', 4)
('sit', 6)
('amet', 6)
('elit', 2)
('cras', 3)
('et', 4)
('nibh', 5)
('pellentesque', 7)
...
```

Figure 3.31: RDDs Content



About				Application Overview
Nodes	User	alejandro		
Node Labels	Name	my_Spark_Core_example.py		
Applications	Application Type	SPARK		
NEW	Application Tags			
NEW SAVING	YarnApplicationState	RUNNING: AM has registered with RM and started running.		
SUBMITTED	FinalStatus Reported by AM	Application has not completed yet.		
ACCEPTED	Started	Mon Jul 13 17:29:54 +0100 2020		
EINTCHED	Elapsed	7sec		
FAILED	Tracking URL	ApplicationMaster		
KILLED	Diagnostics			
Echodulor				
scheduler				
> Tools				Application Metrics
10010		Total Resource Preempted:	<memory:0, vcores:0=""></memory:0,>	
		Total Number of Non-AM Containers Preempted:	0	
		Total Number of AM Containers Preempted:	0	
		Resource Preempted from Current Attempt:	<memory:0, vcores:0=""></memory:0,>	
	Num	ber of Non-AM Containers Preempted from Current Attempt:	0	
		Aggregate Resource Allocation:	12874 MB-seconds, 6 vcore-seconds	
	Show 20 × entries			Search:
	Attempt ID	Obstad	Mada a	Law A
	Attempt ID *	Starteo 0	NODE 0	Logs 🗘
	appattempt 1594657479419 0001 000001	Mon Jul 13 17:29:54 +0100 2020 http://sr	nartebuses:8042 Logs	
	Showing 1 to 1 of 1 entries			First Previous 1 Next Last

Figure 3.32: ResourceManager: Spark Core Application in Progress

luster	run Application										
About										Applicat	ion Overvie
Nodes Node Labels				User: Name:	alejandro mv Spark Core	example.pv					
Applications			Appl	lication Type:	SPARK	,					
NEW SAVING			Appl YarnApp	lication Tags:	FINISHED						
SUBMITTED			FinalStatus Rep	orted by AM:	SUCCEEDED						
RUNNING				Started:	Mon Jul 13 17:2	9:54 +0100 2020)				
FINISHED FAILED			т	racking URL:	History						
KILLED				Diagnostics:							
cheduler											
ools								-		Applic	ation Metric
			Tota	I Number of No	n-AM Containe	ce Preempted: rs Preempted:	<memory:0, vcore<br="">0</memory:0,>	es:0>			
				Total Number	of AM Containe	rs Preempted:	0				
			Number of Non-AM (Resource Pree	mpted from Cu	Irrent Attempt:	<memory:0, td="" vcor<=""><td>es:0></td><td></td><td></td><td></td></memory:0,>	es:0>			
				Ag	gregate Resou	rce Allocation:	76312 MB-second	is, 37 vcore-seconds			
	Show 20 🖌 entries								Sear	ch:	
	Atterr	ipt ID	Ŧ	Started		\$	Node	\$		Logs	<
	appattempt 1594657479419 0001	000001	Mon Jul 13 17	7:29:54 +0100 2	020	http://s	martebuses:8042	Lc	gs		
									First	Previous 1	Next Last
	Showing 1 to 1 of 1 entries										
	Showing 1 to 1 of 1 entries									Logge	d in as: dr.w
	Showing 1 to 1 of 1 entries		Δ		icatio	ns				Logge	d in as: dr.w
She e	Showing 1 to 1 of 1 entries		A	ll Appl	licatio	ns				Logge	d in as: dr.w
	Showing 1 to 1 of 1 entries		A	ll Appl	icatio	ns				Logge	d in as: dr.w
	Showing 1 to 1 of 1 entries	Apps Containe	Al		icatio	NS	ores Active	Decommissioned	Lost	Logge Unhealthy	d in as: dr.w Rebooted
luster Hout	Showing 1 to 1 of 1 entries	Apps Containe Completed Running	RI rs Memory Memory Used Total	y Memory Reserved	VCores Used	NCores VCo Total Rese	ores Active rved Nodes	Decommissioned Nodes	Lost Nodes	Logge Unhealthy Nodes	d in as: dr.w Rebooted Nodes
Cluster About Vodes Vode Labels Applications	Showing 1 to 1 of 1 entries	Apps Containe Completed Running 1 0	A rs Memory Memory Used Total 0 B 8 GB	y Memory Reserved 0 B	VCores Used 0 8	NCores VCo Total Rese	rved Active Nodes	Decommissioned Nodes Q	Lost Nodes 0 9	Logge Unhealthy Nodes	d in as: dr.w Rebooted Nodes Q
Cluster About Vodes Vode Labels Applications NEW	Showing 1 to 1 of 1 entries	Apps Containe Completed Running 1 0	rs Memory Memory Used Total 0 B 8 GB	y Memory Reserved 0 B	VCores Used 0 8	NS VCores VCC Total Rese	rrved Nodes	Decommissioned Nodes 0	Lost Nodes 0 0	Logge Unhealthy Nodes 2	d in as: dr.w Rebooted Nodes <u>0</u>
luster dods ludes lode Labels pplications NEW SAVING SUBMITED	Showing 1 to 1 of 1 entries Cluster Metrics Apps Apps Apps 1 0 0 0 Scheduler Metrics Scheduler Type Capacity Scheduler	Apps Containe Completed Running 1 0 Schw (MEMORY)	rs Memory Veed 0 B B 6 GB eduling Resource Type	II Appl y Memory Reserved 0 B	VCores Used 0 8	VCores VCc Total Rese o Vinimum Allocatie Sores:1>	ores Active rved Nodes <u>1</u> on	Decommissioned Nodes 2 <memory:8192, td="" v<=""><td>Lost Nodes 0 9 Maximum A Cores:8></td><td>Logge Unhealthy Nodes 2</td><td>d in as: dr.w Rebooted Nodes <u>0</u></td></memory:8192,>	Lost Nodes 0 9 Maximum A Cores:8>	Logge Unhealthy Nodes 2	d in as: dr.w Rebooted Nodes <u>0</u>
Iuster bout todes tode Labels polications NEW SAVING SUCOPTED SUCOPTED SUCOPTED SUNNING	Showing 1 to 1 of 1 entries	Apps Completed 1 0 Schu [MEMORY]	rs Memory Veed 0 B 8 8 GB aduling Resource Type	II Appl y Memory Reserved 0 B <π	VCores Used 0 8	VCores VCC Total Rese 0 Vinimum Allocatio cores:1>	ores Active Nodes 1	Decommissioned Nodes Q <memory:8192, td="" v<=""><td>Lost Nodes 0 9 Maximum A Cores:8> Sear</td><td>Logge Unhealthy Nodes 2 Allocation ch:</td><td>d in as: dr.w Rebooted Nodes Q</td></memory:8192,>	Lost Nodes 0 9 Maximum A Cores:8> Sear	Logge Unhealthy Nodes 2 Allocation ch:	d in as: dr.w Rebooted Nodes Q
Juster Vbout vodes vode Labels vpplications NEW SAVING SUBMITTED ACCEPTIC ENTRY ENTRY ACCEPTIC	Showing 1 to 1 of 1 entries Cluster Metrics Apps Apps Apps Apps Submitted Pending Running 1 0 0 Scheduler Metrics Scheduler Type Capacity Scheduler Show 20 v entries ID v	Apps Completed T (MEMORY) User © Nat	rs Memory Memory Used Total 0 B 8 6 GB aduling Resource Type me • Applica	II Appl y Memory Reserved 0 B <pre> </pre>	VCores Used 0 8 hemory:1024, vC	NS VCores VCC Total VCC Rese 0 Vinimum Allocati Cores:1> Time \diamond Fini	ores Active Nodes 1 on	Decommissioned Nodes Q ⊲memory.8192, v e ≎ FinalStatus	Lost Nodes Q g Maximum A Cores:8> Sear ≎ Pr	Logge Unhealthy Nodes 2 Allocation ch:	d in as: dr.w Rebooted Nodes Q
Cluster About Vodes Vode Labels Applications NEW NEW SUBMITTED SUBMITTED RUNNING FINISHED FINISHED FINISHED FAILED	Showing 1 to 1 of 1 entries	Apps Completed Running 1 0 Scht [MEMORY] User \diamond Nat alejandro, my Spark Gr	rs Memory Memory Jused Total 0 B B GB aduling Resource Type me c Applica	y Memory Reserved 0 B	VCores Used 0 8 hermory:1024, vC	NS VCores VCc Total Resc 0 Vinimum Allocatie ores:1> VCmme < Finis Iul 13. Mon.	rres Active rved Nodes 1 on hTime ≎ Stat	Decommissioned Nodes omemory.8192, v e < FinalStatus HED. SLICCEEDFI	Lost Nodes Q g Maximum A Cores:8> Sear ≎ Pr	Logge Unhealthy Nodes 2 Allocation cch: cogress \$	d in as: dr.w Rebooted Nodes Q Tracking U

Figure 3.33: ResourceManager: Spark Core Application Finished

Hadoop Overview Datanodes Snapshot Startup Progress Utilities

Browse Directory

/user/my_HDFS/my_	user/my_HDFS/my_result C													
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name							
-174/-11	aleiandro	supergroup	0.8	13/07/2020 17:49:09	1	128 MB	SUCCESS							
	alejandro	supergroup	0 D	10/07/2020, 17:40:00		120 MD	_0000200							
-rw-rr	aiejandro	supergroup	606 B	13/07/2020, 17:49:09	I	128 MB	pan-00000							
-rw-rr	alejandro	supergroup	427 B	13/07/2020, 17:49:09	1	128 MB	part-00001							
-rw-rr	alejandro	supergroup	362 B	13/07/2020, 17:49:09	1	128 MB	part-00002							
-rw-rr	aleiandro	supergroup	500 B	13/07/2020. 17:49:09	1	128 MB	part-00003							

Figure 3.34: HDFS: Spark Core Result in my_result

my_d	ataset my_result	
<	> < 2_Spark_Co	ore my_result > Q = = = • • •
0	Recent	
ଳ ଅ	Home	part-00000 part-00001 part-00002 part-00003
	Desktop	
	Downloads	_SUCCESS
ř	Music	
~ ~	Nusic	Open ▼ Parc-00000 Save ≡ © © ⊗
	Pictures	('nibh', 5) ('finibus', 1)
		('pellentesque', 7)
(())	Irash	('varius', 3)
		('metus', 4) ('duis', 2)
		('malesuada', 4)
		('suscipit', 2)
		Plain Text ▼ Tab Width: 8 ▼ Ln 1, Col 1 ▼ INS

Figure 3.35: HDFS: Spark Core Result Brought Back to Local File System

3.5.3 Spark SQL

We edit now the command (04) # MapReduce or Spark Job Command from the script data_analysis.sh (cf. Figure 3.5) to run our introductory Spark SQL application. The Spark SQL command is presented below:

```
(04) spark-submit \
    --master yarn --deploy-mode cluster \
    ./my_Spark_SQL_example.py \
    /user/my_HDFS/my_dataset \
    /user/my_HDFS/my_result
```

As we can see, the command is the same as for the Spark Core application, it only changes the name of the Python program. And so it is the functionality of the Spark SQL application equivalent to the ones of the MapReduce and the Spark Core applications: it produces as output the new folder my_result, containing the word count for the dataset provided in the input folder my_dataset. Figures 3.36, 3.37, 3.38 and 3.39 present the file my_Spark_SQL_example.py.

```
(01) # -----
(02) # IMPORTS
(03) # -----
(04) import pyspark
(05) import pyspark.sql.functions
(06) import pyspark.sql.types
(07) import sys
```

Figure 3.36: my_Spark_SQL_example.py: Import Section

The program is very similar to my_spark_core_example.py, so we only higlight the differences:

- Lines (01)-(07) import the additional libraries pyspark.sql.functions and pyspark.sql.types to use the Spark SQL API.
- Lines (08)-(36) define the function my_spark_sql_job. It receives as parameters the SparkSession (spark) and the input and output directories (my_dataset_dir and my_result_dir, resp). The function processes the dataset in my_dataset_dir to produce the new folder my_result_dir with its word count.

Line (12) defines the schema my_schema for the dataset, matching each line of text of the dataset to a Row object with a single column line, of type String.

Line (14) applies the creator operation read (with the folder my_dataset_dir and my_schema as parameters) to load the dataset into inputDF, a DataFrame of Row objects with one column line of type String. Figure 3.8 showed the content of file_1.txt of my_dataset. Figure 3.40 presents how the first 2 lines of the file translate into 2 Row items of inputDF.

Lines (16)–(18) apply the transformation operations withColumn and drop on inputDF to produce the new DF sentenceDF of Row objects with one column words_list of type

list of String. In particular, in line (17) the operation withColumn creates the new column words_list by splitting the String of line into its words. Then, in line (18) the operation drop removes the column line, for it to not appear in the generated sentenceDF. Figure 3.41 presents the Row produced in sentenceDF for the first item of inputDF showed in Figure 3.40.

Lines (20)–(30) repeat the application of withColumn and drop over 3 consecutive DataFrames, to shape the content of its unique column to the desired format by: exploding the list of words into a single Row per word (line 21), removing any non-alphanumerical character (line 25) and turning any upper-case character into its equivalent lower-case one (line 29). Likewise, lines (22), (26) and (30) remove any intermediate column being produced. Figures 3.42, 3.43 and 3.44 show the content of these 3 DataFrames, which end up producing the new DataFrame lowerDF.

Lines (32)–(33) apply the transformation operation groupBy on lowerDF to produce the new DF solutionDF of Row objects with two column: word (of type String) and count (word) (of type Integer). In particular, in line (33) the operation groupBy specifies the aggregation of Column word, counting the appearances for each group. Figure 3.45 shows the content of solutionDF for the Rows showed in Figure 3.44.

Lines (35)–(36) apply the action operation write (with the folder my_result_dir as parameter) to store the content of solutionDF into it. Figures 3.48 and 3.49 present the files and content produced by solutionDF when the program is executed in the Hadoop cluster.

• Lines (37)–(51) define the main entry point for the program. In particular, line (47) creates the SparkSession. Finally, line (51) calls to the aforementioned function my_spark_sql_job.

A copy of the file my_Spark_SQL_example.py is to be placed on the DataNode the cluster running the Spark Driver process.

Figure 3.46 presents the status of the ResourceManager once the command (04) of the script data_analysis.sh is launched. As we can see, the Spark SQL application is considered to be in progress. Figure 3.47 presents the status once the application finishes. As the execution is successful, the new folder my_result is available now in HDFS, with Figure 3.48 showing it. While the content of the files is not directly accessible in HDFS, we can execute the command get to bring the folder back to our local file system, so as to explore it (Figure 3.49 shows it). All in all, the Spark SQL application finds 135 different words with their associated number of appearances in my_dataset.

```
(08) # -----
(09) # FUNCTION my_spark_sql_job
(10) # ------
                            _____
(11) def my_spark_sql_job(spark, my_dataset_dir, my_result_dir):
(11) # 1. We define the Schema of our DF.
        my_schema = pyspark.sql.types.StructType(
(12)
                       [pyspark.sql.types.StructField("line",
                       pyspark.sql.types.StringType(), True)
                       ]
                                                 )
        # 2. Operation C1: Load DataFrame
(13)
        inputDF = spark.read.format("csv") \
(14)
                             .option("delimiter", ";") \
                             .option("quote", "") \
                             .option("header", "false") \
                             .schema(my_schema) \
                             .load(my_dataset_dir)
        # 3. Operation T1: split
(15)
        sentenceDF = inputDF \
(16)
(17)
                  .withColumn("words_list",
                        pyspark.sql.functions.split(
                          pyspark.sql.functions.col("line"),
                           )
                        ) \
                  .drop("line")
(18)
        # 4. Operation T2: explode
(19)
        wordsDF = sentenceDF \setminus
(20)
                  .withColumn("draft_word",
(21)
                             pyspark.sql.functions.explode(
                       pyspark.sql.functions.col("words_list")
                                                           )
                              )
                  .drop("words_list")
(22)
```

Figure 3.37: my_Spark_SQL_example.py: my_spark_sql_job Function

```
(23)
         # 5. Operation T3: regexp_replace
(24)
         cleanDF = wordsDF \
(25)
              .withColumn("clean_word",
                           pyspark.sql.functions.regexp_replace(
                               "draft_word",
   r"[^a-zA-Z]",
                               .....
                                                                  )
                          ) \
(26)
              .drop("draft_word")
         # 6. Operation T4: lower
(27)
         lowerDF = cleanDF \setminus
(28)
              .withColumn("word",
(29)
                           pyspark.sql.functions.lower(
                      pyspark.sql.functions.col("clean_word")
                                                        )
                          ) \
(30)
              .drop("clean_word")
(31)
         # 7. Operation T5: GroupBy
         solutionDF = lowerDF \
(32)
                .groupBy(["word"]).agg({"word": "count"})
(33)
         # 8. Operation A1: We save the results
(34)
         solutionDF.write.format("csv") \
(35)
                          .save(my_result_dir)
(36)
```

Figure 3.38: my_Spark_SQL_example.py: my_spark_sql_job Function

```
(37) # -----
(38) # MAIN
(39) # ---
                 _____
(40)
        # 1. We use as many input arguments as needed
        my_dataset_dir = "/FileStore/tables/my_dataset/"
(41)
(42)
        my_result_dir = "/FileStore/tables/my_result/"
(43)
        if (len(sys.argv) > 1):
(44)
           my_dataset_dir = sys.argv[1]
           my_result_dir = sys.argv[2]
(45)
        # 2. We configure the Spark Session
(46)
(47)
        spark = pyspark.sql.SparkSession.builder.getOrCreate()
(48)
        spark.sparkContext.setLogLevel('WARN')
(49)
        print("\n\n\n")
        # 3. We call to our main function
(50)
        my_spark_sql_job(spark, my_dataset_dir, my_result_dir)
(51)
```

Figure 3.39: my_Spark_SQL_example.py: Main Entry Point

++
line
++
Lorem ipsum dolor
habitant morbi tr
++

Figure 3.40: inputDF Content

+		+
	words_	_list
+		+
[Lorem,	ipsum.,	d
+		+

Figure 3.41: sentenceDF Content

+	+
	draft_word
+	+
Lorem	
ipsum.	
dolor	
+	+

Figure 3.42: wordsDF Content

+	+
	clean_word
+	+
Lorem	
ipsum	
dolor	
+	+

Figure 3.43: cleanDF Content

+	+
	word
+	+
lorem	
ipsum	
dolor	
+	+

Figure 3.44: lowerDF Content

+		++
	word	count (word)
+		++
llorem		4
ipsum		2
dolor		4
+		++

Figure 3.45: solutionDF Content

She e					All	Appl	icat	tions	5				Log	gged in as: dr.wh
- Cluster	Cluster Metrics		0.11				1/0	110	10	A				
About Nodes	Submitted Pending Ru	Apps Apps Inning Comple	ted Running	Used	Total	Reserved	Use	d Total	Reserved	Nodes	Nodes	Nodes	Nodes	Nodes
Node Labels	3 0 1	2	1	2 GB	8 GB	0 B	1	8	0	1	0	0	<u>0</u>	<u>0</u>
NEW	Scheduler Metrics		Ontended	Deces				h finimum	Alle and an	Mavimum Allandian				
NEW SAVING SUBMITTED	Capacity Scheduler	[M	EMORY]	ing nesour	ce Type	<m< td=""><td colspan="7">Minimum Allocation Maximum Allocation Amemory:1024, vCores:1> Amemory:8192, vCores:8></td><td></td></m<>	Minimum Allocation Maximum Allocation Amemory:1024, vCores:1> Amemory:8192, vCores:8>							
ACCEPTED	Show 20 v entries										Search:			
FINISHED FAILED	ID	- User ≎	Name	ô	Application Type Queue			tartTime ≎	FinishTime	State	FinalStatus	Progre	ess û	Tracking UI
KILLED	application 1594657479419	0003 aleiandro	my Spark SQL e	example pv	SPARK	¢ defa	ult M	on Jul 13	N/A	ACCEPTE			A	pplicationMaster
Scheduler	Sec. 201101 100-100147.0410	<u>alojandio</u>	,_opun_out_t		5.74.44	0010	1	7:55:57		AUGULI TEL		L		rependention invitability
Tools	application 1594657479419	0002 alejandro	my_Spark_Core_	example.py	SPARK	defa	ult M	on Jul 13 7:48:56	Mon Jul 13 17:49:10	FINISHED	SUCCEEDED		H	<u>istory</u>
	application 1594657479419	0001 alejandro	my_Spark_Core_e	example.py	SPARK	defa	ult M 17 +(on Jul 13 7:29:54 0100 2020	Mon Jul 13 17:30:09 +0100 2020	FINISHED	SUCCEEDED		H	istory
	Showing 1 to 3 of 3 entries											First	Previous	1 Next Last
		Ар	plicatio	on ap	oplica	ation_	_15	9465	57479	419_	0003			
													Applic	cation Overview
Nodes				User: ale	andro									
Node Labels Applications		Type: SP	ARK_SQL	_example.py										
NEW		Tags:												
NEW SAVING SUBMITTED		yAM: Ap	RUNNING: AM has registered with RM and started running. Application has not completed vet.											
ACCEPTED RUNNING		arted: Mo	Mon Jul 13 17:55:57 +0100 2020											
FINISHED			Tracking	URL: Ap	ec plicationMas	er								
KILLED			Diagno	stics:										
Scheduler														
> Tools							Tota	Resource	Preempted:	memon/:0 vCr	noe:0~		App	plication Metrics
					Tota	I Number of I	Ion-AM (Containers	Preempted: 0	nemory.o, voc	هرل. 50 ۱۱			
						Total Numbe	r of AM	Containers	Preempted: 0					
				Number	of Non-AM	Containers Pr	empted	from Curre	ent Attempt: <	memory:0, vCo	res:0>			
						4	agregat	e Resource	Allocation: 5	373 MB-secon	ds. 2 vcore-seconds			

Figure 3.46: ResourceManager: Spark SQL Application in Progress

Started Mon Jul 13 17:55:57 +0100 2020

SMARTeBuses

Show 20 🗸 entries

wing 1 to 1 of 1 entri

Attempt ID appattempt 1594657479419 0003 000001 Search: Logs

≎ Logs

Node http://smartebuses:8042

Submitted Pending Running Competed Running Used Total Reserved Used Total Reserved Nodes N		Apps	Apps	Apps	Containers	Memory	Memory Mem	ory VCo	res VCores	VCores	Active De	commissioned	Lost	Unhealthy	Rebooted
Scheduler Metrics Scheduler Metrics Cupacity Scheduler MEMORY Metrics Maximum Allocation Maximum Allocation Maximum Allocation Maximum Allocation Metrics	Submitted P 3 0	'ending	Running 0	Completee 3	d Running 0	0 B	Total Reser 8 GB 0 B	ved Use 0	ed Total 8	Reserved 1	Nodes 0	Nodes	Nodes 0	s Nodes 0	0 Nodes
Scheduler Type Maximum Allocation Maximum Allocation Capacity Scheduler (pacely Scheduler)	Scheduler Met	trics													
Show 20 ventries Search: ID User o Name Application Type o Queue o StarTime o FinishTime o State o FinishTime o History History <td colspan="9">Scheduler Type Scheduling Resource Type Minimum Allo</td> <td>Allocation</td> <td><</td> <td>memory:8192. v</td> <td>Maximi Cores:8</td> <td>um Allocation 8></td> <td></td>	Scheduler Type Scheduling Resource Type Minimum Allo									Allocation	<	memory:8192. v	Maximi Cores:8	um Allocation 8>	
ID • User o Name o Application Type o Queue o StartTime o FinishTime o State o FinishTime o Progress o Tracking application 1594657479419 0003 alejandro my_Spark_SQL_example.py SPARK default Mon Jul 13 (1755;77) Mon Jul 13 (1755;77) FinishTime o State o Finishtin finishtim Finishtin finis	Show 20 v en	ntries		[,,	5	Search:	
application 1594657479419 0003 alejandro my_Spark_Core_example.py SPARK default Mon.Jul 13 1755:57 40100 2020 2020 application 1594657479419 0002 alejandro my_Spark_Core_example.py SPARK default Mon.Jul 13 17555 1774519.40100 40100 2020 2020 40100 2020 2020 application 1594657479419 0001 alejandro my_Spark_Core_example.py SPARK default Mon.Jul 13 172556 1774519.40100 40100 2020 2020 40100 2020 2020 Showing 1 to 3 of 3 entries First Previous 1 Next Core_example.py SPARK default Mon.Jul 13 177254 40100 2020 2020 2020 First Previous 1 Next Core example.py SPARK default Mon.Jul 13 177254 40100 2020 2020 2020 First Previous 1 Next Core example.py SPARK default Mon.Jul 13 177254 40100 2020 2020 2020 First Previous 1 Next Core example.py SPARK default Mon.Jul 13 177254 40100 2020 2020 2020 First Previous 1 Next Core example.py SPARK default Mon.Jul 13 17735794192_00022 KII Application Cve Vaer: alejandro Name: my_Spark_Core_example.py SPARK Application Cve Started Mon.Jul 13 17748:56 +0100 2020 Eigend Mission Cve Eigend Mission Started Mission Prevention Core example.py SPARK Application Cve Started Mon.Jul 13 17748:56 +0100 2020 Eigend Mission Cve Eigend Missi	ľ	ID	-	User ≎	Name	\$	Application Type \$	Queue ≎	StartTime ≎	FinishTime ≎	State 0	FinalStatus	•	Progress \$	Tracking U
application 1594657479419 0002 alejandro my_Spark_Core_example.py SPARK default Mon_ki 13 1748.56 19 0100 17480.500 192020 2020 1990 100 1990 100 1990 199	pplication 159	346574794	19 0003	alejandro	my_Spark_SQL_6	example.py	SPARK	default	Mon Jul 13 17:55:57	Mon Jul 13 17:56:19 +0100 2020	FINISHED	SUCCEEDE)		History
application 1594657479419 0001 alejandro my_Spark_Core_example.py SPARK default Mon_Jul 13 Mon_Jul 13 Mon_Jul 13 1730.09 +0100 2020 2020 2020 First Previous 1 Next Logged In as: Application application_1594657479419_0002 Application Application Application Curves alejandro User: alejandro User: alejandro User: alejandro User: alejandro Mame: my_Spark_Core_example.py Application Tags: VarnApplication Tags: VarnApplication Tags: VarnApplication Tags: VarnApplication Tags: VarnApplication Tags: VarnApplication State: FinalStatus Reported by AM: SUCCEEDED FinalStatus Reported by AM: SUCCEEDED Elapaed: 13ee Tracking URL: History Diagnostics:	application 159	46574794	19 0002	alejandro	my_Spark_Core_	example.py	SPARK	default	Mon Jul 13 17:48:56 +0100 2020	Mon Jul 13 17:49:10 +0100 2020	FINISHED	SUCCEEDE)		History
Showing 1 to 3 of 3 entries Eric Previous 1 Next First Previous 1 N	application 159	46574794	<u>19 0001</u>	alejandro	my_Spark_Core_	example.py	SPARK	default	Mon Jul 13 17:29:54 +0100 2020	Mon Jul 13 17:30:09 +0100 2020	FINISHED	SUCCEEDE)		History
Application application_1594657479419_0002 I Application User: alejandro User: alejandro Name: rny_Spark_Core_example.py Application Ove Application Type: SPARK Application Type: SPARK Application Type: SPARK Application Type: SPARK Application State: FinalStatus Reported by Atu 13 17:48:56 +0100 2020 Elapset: 13acc Tracking URL: History Diagnostics: Total Resource Preempted: -memory:0, vCores:0> Total Mesource Preempted: -0	nowing 1 to 3 r	of 3 entrie	IS										Fil	rst Previous 1	Next Last
User: aligianfo Name: my.Spark_Core_example.py Application Type: SPArk Application Tags: YarnApplicationState: FiloHSHED FinalStatus Reported by AMI: SUCCEEDED Started: Mon.Jul 13 17:48:56 +0100 2020 Elapsed: 13sec Tracking URL: History Diagnostics:				Арј	plicatio	on ap	plicatio	n_15	9465	74794:	L9_0	002		Logg	ed in as: dr.w
Name: my spark_Core_example.py Application Type: SPARK Application Tags: YarnApplicationState: YarnApplicationState: FINISHED FinalStatus Reported by AM: SUCCEEDED Started: Mon Jul 13 17:48:56 +0100 2020 Elapsed: 13sec Tracking URL: History Diagnostics: Application M Total Resource Preempted:	Kill Application			Арј	plicatio	on ap	plicatio	n_15	9465	74794:	L9_0	002		Logg	ed in as: dr.w
Application Tags: YamApplicationState: FINISHED FinalStatus Reported by AM: SUCCEEDED Started: Mon Jul 13 17:48:56 +0100 2020 Elapsed: 13sec Tracking URL: <u>History</u> Diagnostics: Total Resource Preempted: mailto:scattarows.com Application M Total Resource Preempted: o	Kill Application			Арј	plicatio	on ap	oplicatio	n_15	9465	74794:	L9_0	002		Applica	ed in as: dr.w
YarnApplicationState: FiNSHED FinalStatus Reported by AM: SUCCEDED Started: Mon Jul 13 17:48:56 +0100 2020 Elapsed: 13sec Tracking URL: <u>History</u> Diagnostics: Total Resource Preempted: <memory:0, vcores:0=""> Total Number of Non-AM Containers Preempted: 0 Total Number of Non-AM Containers Preempted: 0</memory:0,>	Kill Application			Арј	plicatio	on ap	oplicatio	r: alejandro e: my_Spar e: SPARK	9465	74794: ^{в.ру}	L9_0	002		Applica	ed in as: dr.w
Total Resource Preempted: cmemory:0, vCores:0> Total Number of Non-AM Containers Preempted: 0	Kill Application			Арј	plicatio	on ap	plicatio Use Nam Application Typ Application Tag	r: alejandro e: my_Spar e: SPARK s:	9465	74794: ^{в.ру}	L9_0	002		Applica	ed in as: dr.w
Elapsed: 13sec Tracking URL: <u>History</u> Diagnostics: Total Resource Preempted: <pre>cmemory:0, vCores:0></pre> Total Number of Non-AM Containers Preempted: 0	Kill Application			Арј	plicatio	on ap	Use Nam Application Tay YarnApplication Tata YarnApplication Tata	r: alejandro e: my_Spar e: SPARK s: e: FINISHE e: FINISHE	b b c c c c c c c c c c c c c c c c c c	74794: ^{в.ру}	L9_0	002		Applica	ed in as: dr.w
Tacking URL: <u>History</u> Diagnostics: Total Resource Preempted: <pre>screemory:0, vCores:0></pre> Application M Total Number of Non-AM Containers Preempted: 0	Kill Application			Арј	plicatio	FinalS	Use Deplication Nam Application Tag YarnApplicationStat tatus Reported by Al Starte	r: alejandro e: my_Spar e: SPARK s: e: FINISHE M: SUCCEE d: MO Jul	9465 k_Core_exampl D :DED 13 17:48:56 +01	74794: •.py	L9_0	002		Applica	ed in as: dr.w
Application M Total Resource Preempted: <memory:0, vcores:0=""> Total Number of Non-AM Containers Preempted: 0</memory:0,>	Kill Application			Ар	plicatio	FinalS	Uss Nam Application Typ Application Typ YarnApplicationStat tatus Reported by A Starte Elapse	n_15 r: alejandro : my_Spal : SPARK : : FINISHE M: SUCCEB : SUCEB : SUCEB : SUCCEB : SUCCEB : SUCEB :	9465 k_Core_exampl D DEDED 13 17:48:56 +01	74794: ^{8.} py	19_0	002		Applica	ed in as: dr.w
Total Resource Preempted: <pre>cmemory:0, vCores:0></pre> Total Number of Non-AM Containers Preempted: 0	Kill Application			Ар	plicatio	FinalS	Usa Nam Application Application Tag YarnApplication Stat tatus Reported by Al Starte Elapse Tracking UR Diagnostic	n_15 r: alejandro e: my_Spal e: SPARK s: e: FINISHE M: SUCCEE M: SUCCEE d: Mon Jul d: 13sec L: History s:	9465 k_Core_exampl D DEDED 13 17:48:56 +01	74794: ^{в.ру}	19_0	002		Applica	ed in as: dr.w
Total Resource Preempted: <pre><pre><pre><pre><pre><pre><pre><pre></pre></pre></pre></pre></pre></pre></pre></pre>	Kill Application			Арј	plicatio	FinalS	Usa Nam Application Tay YarnApplication Tag YarnApplication Tata YarnApplication Tata YarnApplication Tata YarnApplication Tata Starte Elapses Tracking UR Diagnostic	n_15 r: alejandrr e: my_Spai e: SPARK s: FINISHE d: SUCCE d: Mon Jul d: SUCCE d: Mon Jul d: SuCCE s: s:	9465 k_Core_exampl D EDED 13 17:48:56 +01	74794: B.py 00 2020	19_0	002		Applic	ed in as: dr.w
Iotal Number of Non-AM Containers Preempted: 0	Kill Application			Арј	plicatio	FinalS	Ust Nam Application Tay YamApplication Tata YamApplication Tata YamApplication Tata YamApplication Tata YamApplication Tata YamApplication Tata YamApplication Tata Starfee Elapse Tiracking UR Diagnostic	n_15 r: alejandro e: my_Spai e: FINISHE e: FINISHE e: FINISHE d: SUCCEE d: Mon Jul d: 13sec L: History s:	9465 k_Core_exampl D EDED 13 17:48:56 +01	74794 : e.py 00 2020	L9_0	002		Applic	ed in as: dr.w ation Overviev
Iotal Number of AM Containers Preembted: 0	Kill Application			Арј	plicatio	FinalS	Usc Nam Application Typ Application Tgp Application Tgp Applic	n_15 r: alejandro r: my_Span s: SPARK s: r: FINISHE d: SUCCEE d: Mon Jul d: 13sec L: History s: Total F	b c.Core_exampl b cDED 13 17:48:56 +01	74794 : e.py 00 2020	L9_0	002		Applica	ed in as: dr.w
Resource Preempted from Current Attempt: <pre>cmemory:0, vCores:0></pre>	Kill Application			Ар	plicatio	FinalS	Usa Nam Application Tag YarnApplication Tag YarnApplication Tag YarnApplication Tatu Application Tag YarnApplication Tatu Starte Elapse Tracking UR Diagnostic	r: alejandro e: my_Spane e: SPARK s: e: FINISHE d: SUCCE d: Mon Jul d: 13sec L: <u>History</u> s: Total F Non-AM Co care of AM CC	esource Preen nutainers Preen nutainers Preen	74794 : 	L9_0	002		Applic: Applic	ed in as: dr.w ation Overview ication Metric:
Number of Non-AM Containers Preempled from Current Attempt: 0	Kill Application			Ар	plicatio	FinalS	Usa Nam Application Tay YamApplication Tay YamApplication Tata YamApplication Tata YamApplication Tata YamApplication Tata Starte Etapse Tracking UR Diagnostic	r: alejandro: my_Spare s: SPARK s: c: FINISHE d: SUCCEE d: Mon Jul d: 13sec L: History s: Total F Non-AM Co or of AM Co Preempted T	k_Core_exampl k_Core_exampl D DEDED 13 17:48:56 +01 kesource Preen ntainers Preen on Current At	74794: e.py 00 2020 ppted: o ppted: 0 ppted: 0 ppted: 0	L9_0	002		Logg Applicu Appl	ed in as: dr.w
Aggregate resource Antocation. Tosto improventius, of volie-seconds	COD Kill Application			Ар	plicatio	FinalS	Usc Nam Application Typ Application Tgp JarnApplication Tgp JarnApplication Tgp JarnApplication Tgp JarnApplication Tgp JarnApplication Tgp Jagnostic Diagnostic Total Number of Total Number of Total Number of Total Number of Total Number of Non-AM Contailarers	r: alejandri : my_Spake : SPARK : e: FINISHE : SUCCEE d: Mon Jul d: 13sec : History : : Total F Non-AM Cc per of AM Cc Preempted ff Arconexio	b k_Core_exampl D DDD DDD DDD DDD DDD DDD DDD DDD DDD	9,97	L9_0	002		Logg Applici Appl	ed in as: dr.w
Show 20 v entries Search:				Ар	plicatio	FinalS	Usa Nam Application Ty Application Ty Application Tak Starte Elapse Tracking UR Diagnostic Diagnostic Total Number o Total Number o Total Number o Notal Number o Total Number o Notal Number o Total Number o Notal Number o Total Number o Total Number o Notal Number o Total Number o Notal Number o Total Number o Total Number o Notal Number o Total Number o Notal Number o Total Num	r: alejandrc e: my_Spate e: SPARK s: e: FINISHE d: Mon Jul d: 13sec 1: History s: Total F Non-AM Cc Preempted fi Aggregate	esource Preen on Current At Resource Alloc	apted: ⊲mermory pipted: 0 pipted: 0 pipted: 0 manture sempt: 0 attion: 75915 ME	L9_0	002		Applici Applici	ed in as: dr.w
	Kill Application	1		Ар	plicatio	FinalS	Usa Nam Application Tag YarnApplication Tag YarnApplication Tag YarnApplication Tatus YarnApplication Tatus State Elapse Tracking UR Diagnostic Diagnostic Total Number o Total Number o Total Number o Total Number o Non-AM Containers	n_15 r: alejandrc r: my_Spare s: r: FINISHE r: MonJAM Non-AM rotal F Non-AM correspited fi Aggregate	k_Core_exampl k_Core_exampl D D D D D D D D D D D D D D D D D D D	74794: B. py 00 2020 hpted: 0 pited: 0 empt: 0 empt: 0 ation: 75915 ME	L9_0	002	5	Applic: Applic Appl	ed in as: dr.w

Figure 3.47: ResourceManager: Spark SQL Application Finished

Hadoop	Overview	Datanodes	Snapsh	not Startup Progress	Utilities 👻			
Brow	vse Di	rector	У					
/user/my_HDI	FS/my_result							Go!
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
-rw-rr	alejandro	supergroup	0 B	13/07/2020, 18:04:22	1	128 MB	_SUCCESS	
-rw-rr	alejandro	supergroup	0 B	13/07/2020, 18:04:21	1	128 MB	part-00000-5e3032c0-95cd-4dcb-af64-f46bc5a3b213-c000.csv	
-rw-rr	alejandro	supergroup	12 B	13/07/2020, 18:04:15	1	128 MB	part-00003-5e3032c0-95cd-4dcb-af64-f46bc5a3b213-c000.csv	
-rw-rr	alejandro	supergroup	7 B	13/07/2020, 18:04:15	1	128 MB	part-00004-5e3032c0-95cd-4dcb-af64-f46bc5a3b213-c000.csv	
-rw-rr	alejandro	supergroup	11 B	13/07/2020, 18:04:15	1	128 MB	part-00005-5e3032c0-95cd-4dcb-af64-f46bc5a3b213-c000.csv	
-rw-rr	alejandro	supergroup	11 B	13/07/2020, 18:04:15	1	128 MB	part-00006-5e3032c0-95cd-4dcb-af64-f46bc5a3b213-c000.csv	
-rw-rr	alejandro	supergroup	23 B	13/07/2020, 18:04:15	1	128 MB	part-00011-5e3032c0-95cd-4dcb-af64-146bc5a3b213-c000.csv	

Figure 3.48: HDFS: Spark SQL Result in my_result



Figure 3.49: HDFS: Spark SQL Result Brought Back to Local File System

4 Conclusions and Future Work

In this deliverable we have introduced the Big Data ecosystem of tools we are going to use for storing and analysis bus and wind power-related large-datasets.

This ecosystem includes HDFS as a distributed file system (designed to efficiently allocate data across the multiple nodes of the cluster), Yarn as a resource manager (responsible for schedule and monitor the execution of our data analysis applications) and MapReduce, Spark Core and Spark SQL as frameworks for easily writing applications processing large-scale datasets across a cluster in a reliable, fault-tolerant manner. While Python is selected as the programming language of choice (with MapReduce, Spark Core and Spark SQL providing an API for it), the data analytics applications run on top of the Java Runtime Environment.

We have presented detailed scripts for installing, configuring and applying Java OpenJDK 8, Python 3.7.7, Hadoop 2.7.1 and Spark 2.4.5. In the case of Hadoop, the scripts include how to start and stop a Single Node Cluster with Pseudo-Distributed Operation. In the case of MapReduce, Spark Core and Spark SQL, a detailed explanation of an introductory example has been presented, together with a detailed explanation of its execution in the aforementioned cluster.

Bibliography

- [1] V. Mayer-Schonberger and K. Cukier, Big Data. Hmhbooks, 2013.
- [2] Apache Hadoop, https://hadoop.apache.org/.
- [3] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in 26th Symposium on Mass Storage Systems and Technologies (MSST'10), 1–10. IEEE Computer Society, 2010.
- [4] Hadoop Distributed File System 3.2.1, https://hadoop.apache.org/docs/r3.2.1/ hadoop-project-dist/hadoop-hdfs/HdfsDesign.html.
- [5] V. Vavilapalli, A. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. A. Curino, O. O'Malley, S. R. Radia, B. C. Reed, and E. Baldeschwieler, "Apache hadoop yarn: yet another resource negotiator," in *4th Symposium on Cloud Computing (SOCC'13)*, 1–16. ACM, 2013.
- [6] Apache Hadoop Yarn, https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/ YARN.html/.
- [7] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in 6th Symposium on Operating System Design and Implementation (OSDI'04), 137–150, 2004.
- [8] Hadoop MapReduce 3.2.1, https://hadoop.apache.org/docs/r3.2.1/hadoop-mapreduce-client/ hadoop-mapreduce-client-core/MapReduceTutorial.html.
- [9] Apache Spark, https://spark.apache.org/.
- [10] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in 9th Symposium on Networked Systems Design and Implementation (NSDI'12), 15–28. USENIX, 2012.
- [11] Apache Spark Core, https://spark.apache.org/docs/2.4.5/rdd-programming-guide.html.
- [12] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, and M. Zaharia, "Spark sql: Relational data processing in spark," in 15th International Conference on Management of Data (SIGMOD'15), 1383–1394. ACM Press, 2015.
- [13] Apache Spark SQL, https://spark.apache.org/docs/2.4.5/sql-programming-guide.html.
- [14] Java OpenJDK, https://openjdk.java.net/.
- [15] Hadoop Streaming 2.7.1, https://hadoop.apache.org/docs/r2.7.1/hadoop-streaming/ HadoopStreaming.html.
- [16] The Scala Programming Language, https://www.scala-lang.org/.
- [17] Ubuntu 20.04 LTS, https://ubuntu.com/.
- [18] Java OpenJDK 8, https://openjdk.java.net/projects/jdk8u/.
- [19] Java OpenJDK 11, https://openjdk.java.net/projects/jdk/11/.
- [20] Python 3.7.7, https://www.python.org/downloads/release/python-377/.
- [21] Python 3.8.2, https://www.python.org/downloads/release/python-382/.

- [22] pip 20.1, https://pypi.org/project/pip/.
- [23] Hadoop 2.7.1, https://hadoop.apache.org/docs/r2.7.1/.
- [24] Hadoop 3.2.1, https://hadoop.apache.org/docs/r3.2.1/.
- [25] OpenSSH, https://www.openssh.com/.
- [26] Lorem Ipsum Generator, https://www.lipsum.com/.
- [27] Apache Spark 2.4.5, https://spark.apache.org/docs/2.4.5/.
- [28] Apache Spark 3.0.0, https://spark.apache.org/docs/3.0.0-preview/.
- [29] pyspark 2.4.5, https://pypi.org/project/pyspark/.
- [30] PyCharm 2020.1, https://www.jetbrains.com/pycharm/.

SMARTeBuses

SMART electric Buses

August 19, 2020