Big Data Research 19–20 (2020) 100135

Contents lists available at ScienceDirect

Big Data Research

www.elsevier.com/locate/bdr

Automated Deployment of a Spark Cluster with Machine Learning Algorithm Integration

A.M. Fernández, D. Gutiérrez-Avilés, A. Troncoso, F. Martínez–Álvarez *

Data Science and Big Data Lab, Pablo de Olavide University, ES-41013 Seville, Spain

ARTICLE INFO

Article history:
Received 19 November 2019
Received in revised form 5 April 2020
Accepted 26 April 2020
Available online 8 May 2020

Keywords:
Big data analytics
Apache Spark
Machine learning
Cluster deployment

ABSTRACT

The vast amount of data stored nowadays has turned big data analytics into a very trendy research field. The Spark distributed computing platform has emerged as a dominant and widely used paradigm for cluster deployment and big data analytics. However, to get started up is still a task that may take much time when manually done, due to the requisites that all nodes must fulfill. This work introduces LadonSpark, an open-source and non-commercial solution to configure and deploy a Spark cluster automatically. It has been specially designed for easy and efficient management of a Spark cluster with a friendly graphical user interface to automate the deployment of a cluster and to start up the distributed file system of Hadoop quickly. Moreover, LadonSpark includes the functionality of integrating any algorithm into the system. That is, the user only needs to provide the executable file and the number of required inputs for proper parametrization. Source codes developed in Scala, R, Python, or Java can be supported on LadonSpark. Besides, clustering, regression, classification, and association rules algorithms are already integrated so that users can test its usability from its initial installation.

1. Introduction

The era of Big Data [1] has changed the way that data are stored and processed. The need for systems able to efficiently perform both actions has dramatically increased recently [2–4].

Although Spark is an open-source framework under the Apache 2.0 license, it was initially created and developed by the University of California [5]. It provides an interface to deploy fault-tolerant clusters for distributed computing based on the parallelization of data and to develop software under the MapReduce paradigm [6]. It offers a new programming framework providing us two main tools: on the one hand, a high level of abstraction of the MapReduce paradigm allowing an easier way to develop distributed and concurrent applications and, on the other hand, an interface to deploy fault-tolerant clusters for distributed computing based on the partition of data. The MapReduce paradigm [6], as mentioned above, refers to two differentiated tasks: map and reduce. Mapper tasks consist in the transformation of a dataset into another one composed of tuples (pairs of key/value). Reducer tasks take the output of previous mapper tasks and combine tuples to obtain a smaller set of tuples.

Spark programming is focused on the use of a data structure called Resilient Distributed Dataset (RDD) [7], which allows data distribution across the nodes of a cluster. The primary programming language supported by Spark is Scala, but it also supports Java, R, or Python. Moreover, it can be used under different operating systems, such as Linux, MAC OS, or Windows.

For proper cluster management, Spark can make use of Apache's managers like YARN [8], Mesos [9], or even it can make use of the native Spark manager (Standalone). As for the distributed data storage, several implementations can be used as NoSQL databases (Cassandra, MongoDB, or HBase, for example) or a cloud storage service (Amazon S3 or Microsoft Azure, among other). Another well-known and a *de facto* standard for distributed data storage is the Hadoop Distributed File System (HDFS). HDFS is a distributed, scalable and portable file system that may store huge files, typically in ranges of GB to TB (even PB), across multiple machines. It can achieve reliability by replicating the cross multiple hosts, and therefore, does not require any range storage on hosts.

However, to the author's knowledge, there is no friendly application able to effortlessly deploy and parametrize a Spark cluster as well as a distributed file system for free and providing open source. Thus, the main goal is the development of an application that, by just a few clicks and a graphical user interface, fully deploys and configures a Spark cluster with HDFS. That is, it aims at automating the cluster deployment, thus avoiding a complicated and tedious manual configuration. As we can see in Section 5, only

* Corresponding author.
E-mail addresses: amfergom@alu.upo.es (A.M. Fernández), dgutavi@upo.es (D. Gutiérrez-Avilés), atrolor@upo.es (A. Troncoso), fmaralv@upo.es (F. Martínez–Álvarez).

https://doi.org/10.1016/j.bdr.2020.100135
2214-5796/

the Databricks private company has a similar framework to our proposal in this work. However, although it allows the cluster management with different settings (https://databricks.com), the users cannot control physical resources. Moreover, Databricks offers a commercial license (pay per use), whereas LadonSpark provides an open-source license free-to-use.

The LadonSpark tool offers an open-source and non-commercial solution to automatically configure and deploy a Spark cluster. Besides, the main advantage that a potential user acquires when he/she installs this system is to avoid the need to use an administrator role. Therefore, any user that have several machines connected by a network can configure and deploy a Spark cluster in a user-friendly, and free of charge way, and without any system administrator skills. Note that this fact means a great advantage, for instance, for small-medium data science research groups, as well as for other type of users. The application has also been designed to easily integrate new algorithms by just uploading executable files and configuring the inputs. As a sample usage, the tool incorporates some algorithms of the machine learning library (MLlib) of Spark, in particular, Kmeans (clustering), Generalized linear models (regression), and FP-Growth (pattern extraction).

LadonSpark is available at https://github.com/datascienceresearchlab/LadonSpark. In this GitHub repository, you can find a complete manual (with an installation guide and a user guide), a video with a demonstration of use, the source code, and the releases of the system.

The rest of the paper is structured as follows. Section 2 provides a general overview of the state-of-the-art. Section 3 describes the proposed approach. In Section 4 an algorithm deployment analysis is presented. Section 5 introduces a comparative study of the different open-source solutions. Finally, the conclusions drawn have been summarized in Section 6.

2. Related work

Cloud computing is an emerging technology particularly suitable for the execution of distributed algorithms for big data analysis. This technology allows big data processing and management without requiring physical computers in the workplace. In the last years, many works have been published about cloud infrastructures for real-world applications. Next, those directly related to big data will be described.

One of the most relevant cluster deployment applications is the Databricks platform [10]. This platform was developed to create and manage Spark clusters to facilitate the workflow of a data scientist in big data environments. Another application following the same model is Spark Notebook [11], which provides an interactive web-based editor that can combine Scala code, SQL queries, Markup, and JavaScript collaboratively in order to explore, analyze and learn from massive data sets. From scientific and educational environments, there is a lack of proposals implementing the functions LadonSpark offers, but there are approximations that are analyzed below.

In [12], the Plug and Play Bench (PAPB) application was presented, and it offers an abstraction layer over the infrastructure that integrates and simplifies the deployment of big data benchmarking tools on clusters of machines. The PAPB architecture is based on three parts: a container layer, a middleware layer, and a cluster layer wherein using Docker containers [13] as one of its main characteristics. The MLI software was presented in [14]. This application is a programming interface implemented using Spark and designed for building machine learning algorithms in distributing environments. Its primary goal is to simplify the development of high-performance, scalable, distributed algorithms. The authors in [15] proposed a new framework called GeoSpark to execute data analysis algorithms taking into consideration the geolocation of the data. The approach was designed by making use of three layers: the Apache Spark layer, Spatial RDD Layer, and Spatial Query Processing Layer. Finally, they concluded that GeoSpark has a better runtime performance than Hadoop-based counterparts. An elastic resource manager was introduced in [16] to make better use of hardware resources, and thus, improve cluster efficiency. The proposed approach can dynamically shrink or expand the size of the container depending on the actual resource needs of the tasks, which are being executed. Reported results showed that the CPU performance was improved up to 1.5 times when the resources were adjusted to the computing needs. The architectural components of a framework, so-called SmartHealth, proposed to provide services of big data analytics was described in [17]. It focuses on several applications in the healthcare domain. As the primary use cases, the authors listed patient profile analytics, effective public health strategies and, improved remote patient monitoring.

Regarding machine learning algorithms, we can find different frameworks, as Weka [18] or KNime [19]. These platforms provide the possibility of submitting a machine learning task to an existing Spark cluster. This feature is different from LadonSpark, which allows us to configure and deploy a Spark cluster. We can use LadonSpark to configure and deploy the Spark cluster first and, then, submit our machine learning workflows to it, using Weka or KNime. A great variety of works discussing the execution of algorithms across different types of clusters using Apache Spark in the cloud or smart grids can also be found in the literature [20–26].

A platform to unify different frameworks existing in big data was developed in [27]. Although some issues regarding the integration of Spark were reported, they eventually got to develop a platform for the analysis of data stored in relational databases.

However, after a thorough analysis of all these works, it can be concluded that most of them are new algorithmic proposals or even modifications to improve their efficiency to analyze big data. Additionally, most of them require to pay for technological platforms used to deploy clusters. Some others simply launches algorithms over already deployed clusters. LadonSpark, on the contrary, represents a free tool to quickly deploy and manage clusters in users' networks and provides a repository in which users can share their algorithms with the Spark community.

3. The LadonSpark application

Spark does not offer a complex administration of the nodes that form part of the cluster, but several tools to manage a Spark cluster can be found nowadays.

The Spark management system is mainly based on three configuration files, which are created to deploy the cluster. These files are stored as templates in the directory "$SPARK_HOME/conf". A proper configuration of such files allows launching a Spark cluster. An explanation of the functionality of each file can be found below:

1. File *spark-default.conf*. This file defines the default configuration of different parameters for the cluster so that it can be deployed.
2. File *slaves*. This file contains the list of the worker nodes that form the cluster. It contains the IP addresses for all these nodes.
3. File *spark-env.sh*. This file contains the main parameters of the configuration of a Spark cluster, e.g., the RAM that can be used, the number of cores and, the number of instances. It is worth noting that there exist many more parameters, but those mentioned above are the ones with the highest impact in the configuration.

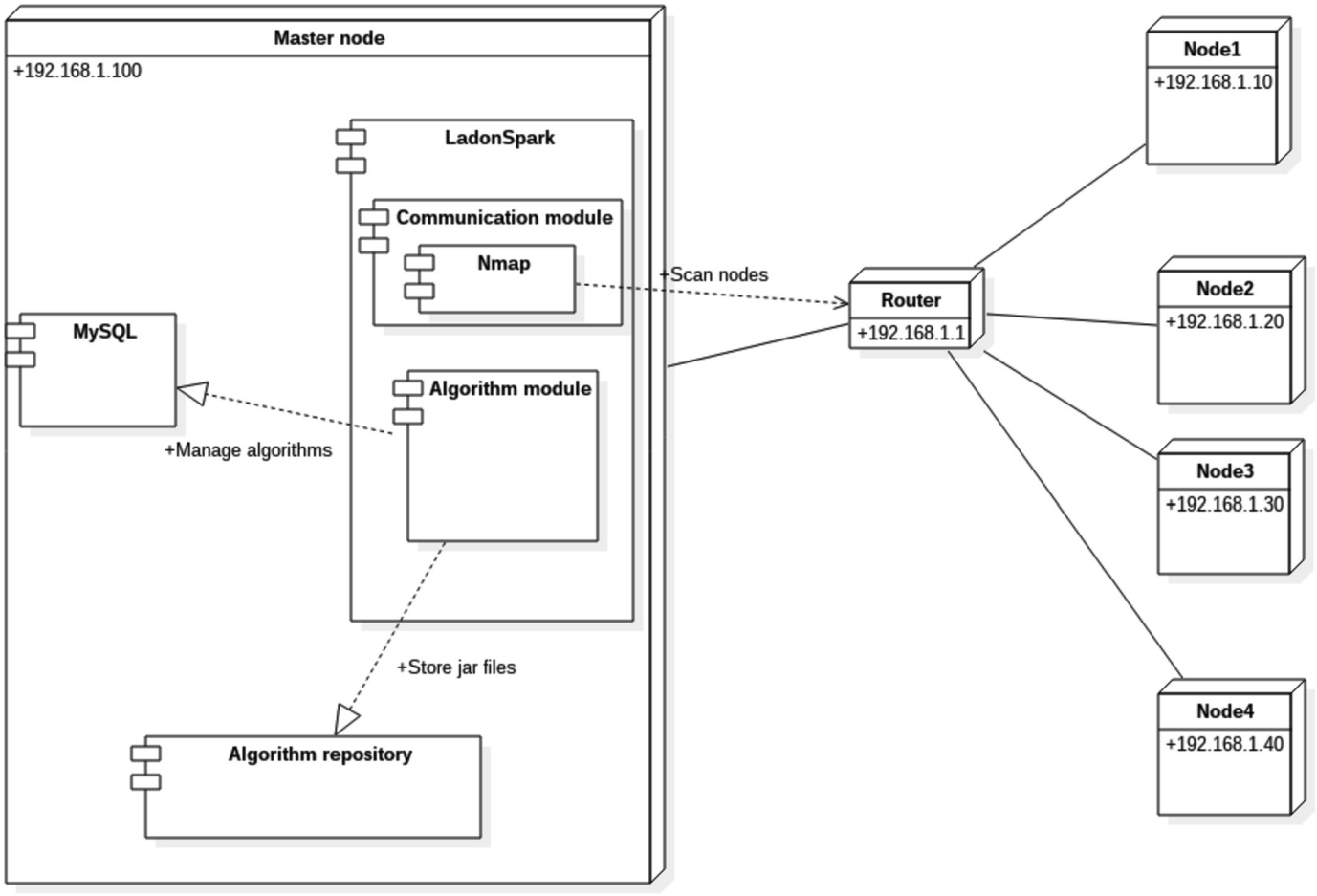

Fig. 1. A general illustration of the LadonSpark application.

This process previously described has to be repeated for each node that forms the cluster. Once the configuration files have been modified for all nodes, the cluster can be initialized by native scripts of Spark. These scripts can be found in the directory "$SPARK_HOME/bin" and can be any of these: *start-slaves.sh*, *start-master.sh* or *start-all.sh.*

Thus, the configuration of a Spark cluster is a repetitive and humdrum process, and it can be extremely difficult if the nodes are spread in different physical locations. Although there are several Spark cluster managers to deploy a cluster such as Apache Mesos or Apache Yarn, the nodes in the cluster must be individually configured for both cases.

Given that a cluster cannot be currently deployed without any initial configuration steps, the LadonSpark web application has been developed to automatize the entire configuration process. Thus, LadonSpark automatically sends the configuration files to all nodes. The application has been divided, mainly, in two parts: algorithms and communications. The first one manages both algorithms execution and parameterizations, whereas the second one allows automatized setting and launching the Spark cluster.

Fig. 1 depicts a diagram of how the cluster is deployed by the web application here proposed. The application is launched from the master node, which contains a database and a repository of algorithms. Namely, the database stores parameters needed for the algorithms and the repository stores .jar files. The diagram also shows the slave nodes as well as the communications with external resources. These modules, along with the general characteristics and prerequisites, are described in subsequent subsections.

3.1. General characteristics

The LadonSpark software is an open-source application designed to offer to the scientific community an easy-to-use user interface to deploy a Spark cluster. LadonSpark can use any type of machine (newer or older one) for connecting them and for providing the hardware support to the deployed Spark cluster. LadonSpark provides service-level authentication in its distributed file system, as well as an authentication layer for its Http web consoles. Every data transfer through the distributed file system is encrypted. The authentication and encryption functions are implemented by Hadoop-HDFS being the distributed file system of LadonSpark; thus, data security and confidentiality are guaranteed. On the other hand, it must be taken into account that LadonSpark is installed by the user in its own local infrastructure. A summary of these characteristics of the LadonSpark software, classified into advantages and weaknesses, is shown in Table 1.

3.2. Prerequisites

The prerequisites described below are necessary for the correct functioning of the LadonSpark approach:

1. Shared dataset. The dataset to be processed by an algorithm has to be shared by all the cluster nodes. Currently, there are two different ways to share it:
   (a) HDFS System. This system distributes a dataset in all nodes of the cluster. The LadonSpark application integrates the

Table 1
Advantages and weaknesses of LadonSpark.

Advantages	Weaknesses
GPL v3 license	Real machines are required
Easy installation and use	Lack of virtualization approach
Designed for the scientific community	Software systems knowledge needed for installation
The use of any machine is available, even older ones	Not possible to scale to public networks

HDFS, which can be launched using a script that has been developed to install it across the cluster easily.
   (b) File repository. The dataset is replicated in every node at a specific folder. That way, Spark can access to the required specific data blocks in every node. This option reduces the computational time, but it requires much space in memory for each node.
2. RSA ring. RSA keys are necessary for the exchange of information between nodes without having to enter credentials for each connection.
3. Global user. It is necessary to facilitate the RSA ring. Hence, the access to the path of the files is greatly simplified through the same user and password for all nodes.
4. Nmap. It is a critical prerequisite because this application sniffs the network and creates the nodes list. Nmap must be installed in the master node, enabling it to discover new potential nodes to be part of the cluster.
5. Spark package. This package must be downloaded and unzipped in the specific path "/home/username".
6. Scala package. As happens with the Spark package, the Scala package must be downloaded and unzipped for the proper execution of an algorithm, which has been developed in the Scala programming language.

Finally, two new libraries have been included in the last update of the LadonSpark, and therefore, their installation is required to execute an algorithm in both R or Python languages supported by Spark.

1. R-base. This library allows executing R code from Spark. This language has been included because it is one of the most used languages for data analysis currently.
2. Python. This language is a pervasive and popular programming language nowadays. For that reason, this library has been included in developing algorithms using Python from Spark.

3.3. System architecture

For the development of the LadonSpark application, the Model-View-Controller architectural pattern has been used to improve the implementation and to make the application more scalable. For each layer, the following technologies have been used:

1. View. The goal of this layer is the interaction between the user and the system. The main web technology that was used for its development is JSP. For style, CSS, Ajax, JQuery, and JavaScript were used. JSP provides an optimized iteration with the final user through dynamic forms.
2. Controller. This layer is responsible for managing all the requests from the users. The Servlets class was used to build this layer. This layer gets the entities of the model and sends them to the view layer for its presentation.
3. Model. The model layer manages access and modifies the entities. Moreover, the purpose of this layer is to improve the transparency, and for this reason, the Hibernate framework was used along with the Data Access Object pattern to manage entities as different objects.

3.4. Communications

In this section, the module related to communications is described. This module can be considered critical since the cluster must be deployed before any other further action is taken. Transparent and efficient management of the cluster is the main objective of this module.

Three steps must be followed to deploy a cluster. The description of each step is shown below:

1. Network scanning. First, Nmap technology is used to find and identify all nodes in the network. Next, the LadonSpark application builds a list containing all the information for each node from a .xml file generated by the Nmap. Afterwards, this information is shown in a specific user interface.
2. IP filtering. Once the result of the scanning is shown in the web interface, the user must select the slaves that are going to be part of the cluster.
3. Configuration files and sending. After selecting the nodes, the configuration files can be either parametrized as aforementioned in Section 3 or the default configuration can be selected. The configuration by default can be summarized as one core, one instance, and one GB of RAM. Finally, these files are sent to each node of the cluster.
4. Cluster starting. This final step consists in the invocation of an internal Spark script to deploy the cluster. Note that the configuration of the nodes was made in the previous step.

Fig. 2 shows the steps followed by the LadonSpark application in the communication module from the beginning at the communication web page to the deploying of the cluster.

3.5. Algorithms

This module responds to the need of applying some machine learning algorithms to process and analyze big datasets once the cluster is deployed. Following the idea of simplifying the set of steps to facilitate the use of the LadonSpark application, this module has been designed with two differentiated parts: the first one to execute algorithms, and the second one to provide the possibility that the user executes their own algorithms by adding them to the internal repository.

1. Algorithm execution. A friendly user interface displays the parameters of the algorithm selected from a drop-down list. Also, the user can add the URL from which the dataset can be uploaded. Finally, after running the algorithm, the results are shown.
2. Adding an algorithm. An important functionality of the LadonSpark application is to allow the integration of new algorithms. An algorithm can be included in the internal repository by uploading a .jar file and by defining the parameters of the algorithm from a friendly web form. Once the form has been sent, the new algorithm to be executed is inserted in the above drop-down list.

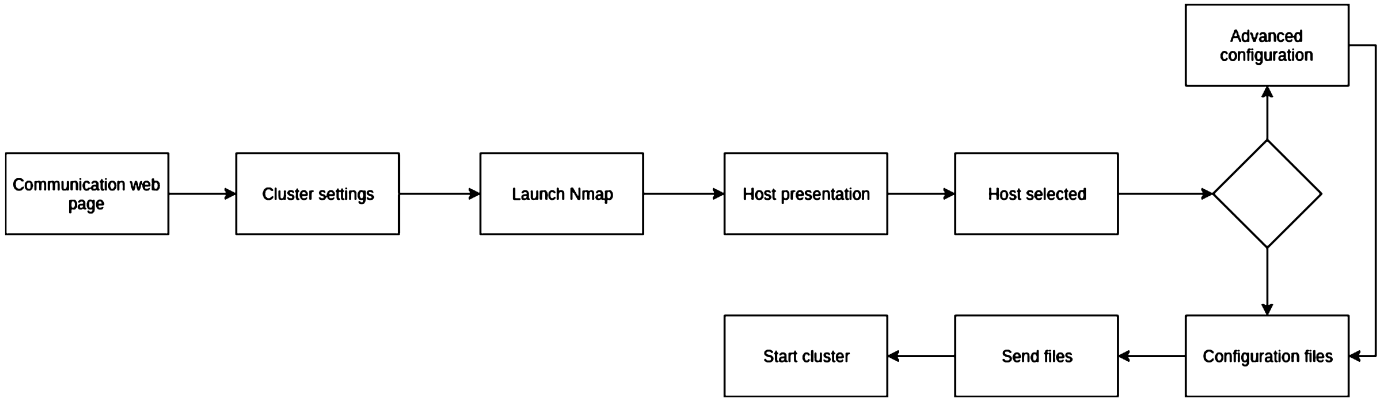

Fig. 2. Scheme of the communication module of the LadonSpark.

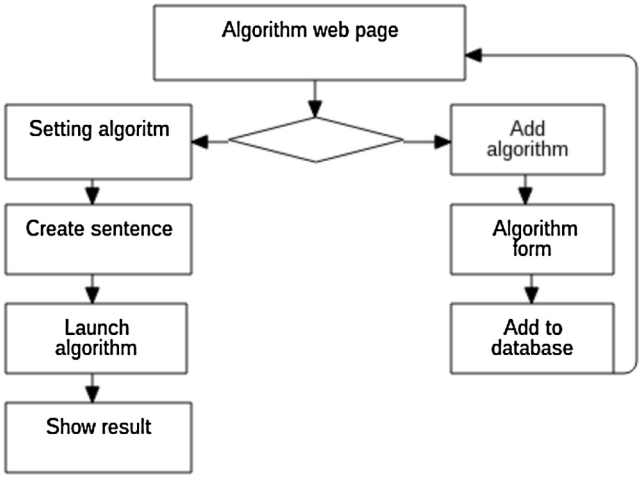

Fig. 3. Scheme of the algorithm module of the LadonSpark.

Finally, Fig. 3 presents the necessary steps for both adding and executing algorithms in the Spark cluster from the algorithm module of the LadonSpark.

4. Algorithm deployment analysis

LadonSpark provides, as well as the automatic deployment of a Spark cluster, the function of submitting a machine learning algorithm to the cluster and run it through a graphical interface. Therefore, this section aims to show the usefulness of the LadonSpark and, to prove the effectiveness (in terms of execution time) of deploying machine learning applications in a Spark cluster.

The following study case is based on several well-known machine learning algorithms. These algorithms have been chosen from the MLlib machine learning library from Spark to cover the most common data mining tasks. Namely, the Kmeans algorithm has been selected as a reference clustering method, a regression technique to obtain a generalized linear model (GLM), and the FP-Growth approach for association rules. The primary purpose of this study is to analyze how the configuration of a Spark cluster has a notable influence on the runtime of each one of these algorithms, depending on the size of the dataset to be processed. Therefore, the hyperparameter configuration of the algorithms has been set to the Spark MLlib default [28–30]. Hence, for the Kmeans algorithm, the number of initial clusters is set to 2 and, the iterations to 20; for the GLM algorithm the regularization parameter is set to 1.0 and the iterations to 100; and, finally, for the FP-Growth algorithm, the minimum support parameter is set to 0.3. The Apache Hadoop HDFS configuration has also been set to default.

Table 2
Different configuration settings for the Spark cluster.

Number of nodes	Number of cores	RAM
2	24	32
4	48	64
6	72	96
12	144	192

Up to twelve computers connected through a private 10 Gigabit Ethernet network with Intel Core i7-5820K processor at 3.3 GHz with 15 MB of cache, 12 cores and 64 GB of RAM and, 1.2 TB SSD hard disks have been used for this purpose. Each computer worked under an Ubuntu 18.04 operating system. Each dataset consists of a group of instances composed of eleven attributes. Therefore, the dimension is fixed to eleven attributes, and the number of rows (instances) vary depending on the different file sizes of the experimentation (from 7 MB to 70 GB). The values of each instance have been randomly generated.

Table 2 shows the different configurations of the Spark cluster used in the experiments to assess the computing time of Kmeans, GLM, and FP-Growth algorithms. Namely, the number of nodes belonging to the cluster, the total number of cores and the total RAM are specified. The LadonSpark has been used to manage the Spark cluster with the configurations from Table 2 and to apply the previously mentioned algorithms to datasets with sizes of different magnitude order. In particular, two experiments have been conducted regarding the type of data storage. First, the algorithms have been applied from the LadonSpark with the datasets in local storage; that is, the data are replicated in each slave node and the master. Alternatively, the HDFS has been used since the LadonSpark also supports it.

Tables 3, 4 and 5 show the runtimes (expressed in seconds) for the Kmeans, GLM and FP-Growth, respectively, when data are stored in the local file system. It can be appreciated that execution times are of same magnitude order independently of the number of the nodes in the Spark cluster, except for the two largest datasets (7 GB and 70 GB), in which the reduction of runtime is remarkable for all algorithms when increasing the number of slaves in the cluster.

Tables 6, 7 and 8 summarize execution times (expressed in seconds) for all algorithms with datasets of different sizes and different cluster configurations, when the HDFS was used. By contrast to the previous tables, decrements in execution times are not appreciated in this situation. On the contrary, there is an increment in runtimes when the number of nodes is increased.

This fact is due to HDFS needs computing time for collecting the distributed blocks of data and building the dataset before providing it to an algorithm as the input dataset. However, in the local

Table 3
Kmeans execution times in local data storage (in seconds).

Data size	2 nodes	4 nodes	6 nodes	12 nodes
7 MB	21	26	28	34
70 MB	34	34	37	48
700 MB	144	90	72	64
7 GB	1500	900	460	133
70 GB	17505	9822	5030	1523

Table 4
GLM execution times in local data storage (in seconds).

Data size	2 nodes	4 nodes	6 nodes	12 nodes
7 MB	19	24	32	53
70 MB	19	26	30	49
700 MB	43	37	34	28
7 GB	318	216	126	97
70 GB	5243	3550	2915	1417

Table 5
FP-Growth execution times in local data storage (in seconds).

Data size	2 nodes	4 nodes	6 nodes	12 nodes
7 MB	12	14	16	20
70 MB	15	19	22	25
700 MB	36	35	41	49
7 GB	246	138	120	78
70 GB	2820	2120	1735	819

Table 6
Kmeans execution times in HDFS data storage (in seconds).

Data size	2 nodes	4 nodes	6 nodes	12 nodes
7 MB	28	37	53	98
70 MB	36	44	46	57
700 MB	180	210	222	312
7 GB	1680	1740	1980	2230
70 GB	21340	23550	25929	34057

Table 7
GLM execution times in HDFS data storage (in seconds).

Data size	2 nodes	4 nodes	6 nodes	12 nodes
7 MB	19	27	39	62
70 MB	21	21	30	59
700 MB	51	96	72	103
7 GB	354	420	474	631
70 GB	4215	4480	4529	7357

storage case, the hard disc provides the data directly without any preprocessing. Therefore, we can conclude that HDFS is inefficient compared to the local storage and replication strategy in terms of running times.

However, HDFS is much more efficient than the local storage strategy in terms of data uploading times, being an essential tool in Big data environments. This fact can be observed in Fig. 4; the depicted lines represent how much time is spent to upload (y axis, expressed in seconds) a file of a particular size (x axis, expressed in gigabytes) for each specific strategy (HDFS or local) using n nodes (2, 4, 6 and, 12). We can observe how, for all number of nodes considered, HDFS is more efficient than the local strategy, and the differences between the uploading times increase with the size of the file.

At a new stage, a new parameter has been explored according to the research published in [31]. The authors showed that the number of blocks in which data are divided to be distributed across the Spark cluster is a parameter necessary to improve the performance of the algorithms when processing big data. This parameter indicates to Spark the number of partitions that should create from the input file to obtain the best computing times. Note that, by default, Spark splits data into blocks of 64 MB.

Table 8
FP-Growth execution times in HDFS data storage (in seconds).

Data size	2 nodes	4 nodes	6 nodes	12 nodes
7 MB	11	15	18	24
70 MB	16	19	22	43
700 MB	84	87	90	103
7 GB	360	276	294	478
70 GB	3971	2849	2432	4023

Table 9
Kmeans algorithm execution times with the optimal number of partitions in local data storage (in seconds).

Data size	2 nodes	4 nodes	6 nodes	12 nodes
7 MB	17 (32)	22 (64)	29 (64)	35 (32)
70 MB	25 (64)	35 (64)	33 (128)	38 (64)
700 MB	92 (64)	142 (64)	78 (32)	89 (32)
7 GB	1320 (32)	960 (32)	568 (32)	145 (128)
70 GB	15465 (32)	11340 (32)	9935 (128)	2845 (128)

Table 10
GLM algorithm execution times with the optimal number of partitions in local data storage (in seconds).

Data size	2 nodes	4 nodes	6 nodes	12 nodes
7 MB	17 (64)	26 (64)	32 (64)	45 (64)
70 MB	22 (64)	24 (64)	27 (128)	63 (128)
700 MB	57 (64)	46 (64)	47 (32)	83 (32)
7 GB	516 (64)	324 (128)	259 (32)	108 (32)
70 GB	7450 (64)	4871 (128)	3720 (128)	1124 (128)

Table 11
FP-Growth execution times with the optimal number of partitions in local data storage (in seconds).

Data size	2 nodes	4 nodes	6 nodes	12 nodes
7 MB	10 (64)	13 (64)	14 (64)	22 (64)
70 MB	18 (64)	21 (64)	20 (64)	39 (64)
700 MB	48 (64)	52 (128)	56 (32)	101 (64)
7 GB	390 (128)	278 (128)	220 (128)	153 (64)
70 GB	5220 (128)	3989 (128)	3420 (128)	1020 (128)

Tables 9, 10 and 11 present, expressed in seconds, the runtimes for each algorithm respectively, when the number of data blocks has been specified. In this work, 32, 64 and 128 partitions of data have been tested, being the optimal number of blocks for each configuration of the Spark cluster and for each dataset shown in parentheses. As expected, a reduction of execution times are not appreciated in comparison with the runtimes obtained when storing data in a local file system without providing the number of partitions to be considered for the processing of data. This is mainly due to data are not distributed as they are stored in all nodes of the cluster, and therefore, the number of partitions has no effect on the computing time.

Tables 12, 13 and 14 show the computing times (expressed in seconds) for each algorithm respectively, when using the HDFS and the number of data blocks is established as input parameter. It can be observed that computing times decrease when using an optimal distribution of data for the datasets with sizes from 700 MB to 70 GB for a cluster with 4 to 12 nodes. Therefore, the number of partitions has a great influence when a distributed computing is carried out, that is, by using a high number of nodes and a large dataset.

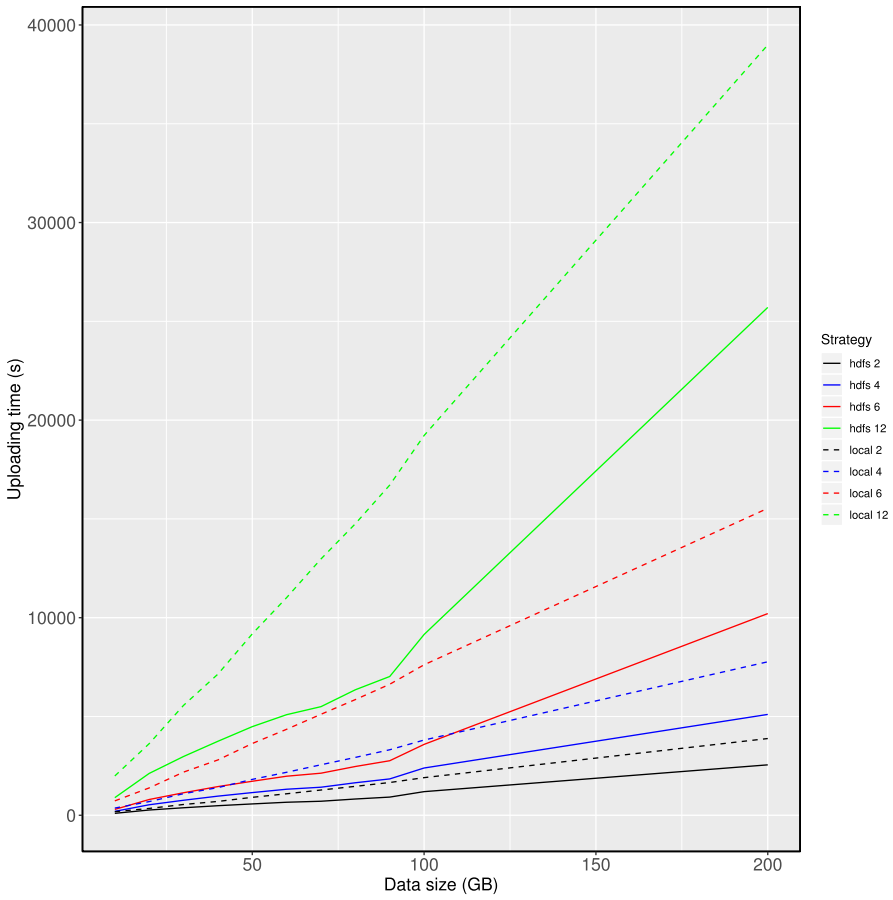

Fig. 4. Data uploading time evolution for local and HDFS strategies. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Table 12
Kmeans execution times with the optimal number of partitions in HDFS data storage (in seconds).

Data size	2 nodes	4 nodes	6 nodes	12 nodes
7 MB	18 (32)	28 (32)	31 (64)	42 (64)
70 MB	33 (64)	38 (128)	43 (64)	53 (32)
700 MB	144 (32)	138 (64)	158 (64)	189 (32)
7 GB	1220 (32)	1020 (128)	900 (32)	678 (128)
70 GB	9832 (32)	9002 (32)	8814 (64)	5735 (128)

Table 13
GLM execution times with the optimal number of partitions in HDFS data storage (in seconds).

Data size	2 nodes	4 nodes	6 nodes	12 nodes
7 MB	18 (128)	28 (64)	30 (128)	52 (128)
70 MB	23 (32)	27 (128)	30 (64)	61 (32)
700 MB	66 (64)	72 (64)	84 (32)	134 (32)
7 GB	528 (64)	480 (32)	397 (128)	201 (128)
70 GB	1634 (128)	1583 (128)	1255 (128)	725 (128)

Table 14
FP-Growth execution times with the optimal number of partitions in HDFS data storage (in seconds).

Data size	2 nodes	4 nodes	6 nodes	12 nodes
7 MB	12 (64)	13 (32)	15 (32)	28 (64)
70 MB	18 (128)	26 (64)	24 (64)	41 (32)
700 MB	84 (64)	72 (128)	84 (64)	101 (128)
7 GB	360 (128)	258 (128)	150 (32)	123 (64)
70 GB	3840 (128)	3126 (128)	2831 (128)	1340 (128)

5. Comparative study

In this section, we have carried out a comparative analysis between the existing solutions of Spark cluster deployment tools and our proposal.

After an exhaustive searching task, it can be confirmed that there are not many alternative tools that offer the specific functionalities and capabilities that LadonSpark does. Nevertheless, a set of four representative tools has been selected for comparative purposes.

First, Databricks [32] is a widely used software for automatic deployment of Spark clusters. This tool provides a graphical user interface and requires a low-level administration role. Furthermore, it offers the possibility of virtualizing the system. In absolute terms, this software would be a perfect alternative to LadonSpark with the exception of a few difficulties. Mainly, Databricks provides all its functionalities over commercial license (pay-per-use) and its non-commercial version limits the characteristics of the potential Spark cluster deployments. On the contrary, LadonSpark provides all its functionalities with no restriction under a non-commercial license.

Regarding Kubernetes [33], it is a container (like Docker) management and deployment tool. Its philosophy consists in the deployment of microservices in a high-availability environment based on load balancers through Kubernetes services and monitorization. The installation and deployment of a Kubernetes system in a local cluster requires medium-high administration system knowledge. With regard to this issue, LadonSpark provides an advantage over Kubernetes. Furthermore, Kubernetes clusters use a virtualized network to support container connections whereas LadonSpark re-

Table 15
Comparison between LadonSpark and other tools.

Tool	GUI	Administration level	Virtualization	Automatic deployment
LadonSpark	Yes	Low	No	Yes
Databricks	Yes	Low	Yes	Yes
Kubernetes	No	Low	Yes	No
Spark-cluster-deployment	No	High	No	Yes
Automate-Spark	No	High	No	Yes

quires a private network with real nodes since it is developed to take advantage of real resources.

Finally, we can find two other alternatives within the academic community. Concerning the spark-cluster-deployment alternative in [34], we can find that a potential user must have high system administrator skills to install and maintain the tool since there is no graphical user interface that facilitates that task. The same objections can be found for the Automate-Spark [35]. About these points, LadonSpark offers a graphical user interface and an automatic installation that does not require high system administration skills.

As summarized in Table 15, the comparative analysis has been performed attending to four main characteristics: the offer of a graphical user interface (column GUI), the system administration level required to install the tool and deploy a Spark cluster (column Administration level), the possibility of system virtualization (column Virtualization) and if the system can perform an automatic deployment of the Spark cluster with minimum intervention of the user (column Automatic deployment).

6. Conclusions

This work provides an overview of a new application, called LadonSpark, for fast and easy management of a Spark cluster. The main objective of LadonSpark is to turn the configuration and deployment of a Spark cluster into a simple task by using a graphical user interface. Furthermore, the LadonSpark has been created so that any algorithm can be integrated by merely indicating its parameters and uploading *.jar* files from a dynamic form. Thus, the user should determine the path of the dataset as well as the algorithm that the user would like to apply. The proposed tool has been compared with other alternatives, and its usefulness probed with an exhaustive machine learning running tests. As a result, we can conclude that LadonSpark is a valuable tool for the quick deployment of a Spark cluster. Future works are directed towards the development of an application able to remotely control and manage all nodes in a cluster in real-time.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors would like to thank the Spanish Ministry of Science, Innovation and Universities for the support under the project TIN2017-88209-C2-1-R.

References

[1] N. Marz, J. Warren, Big Data: Principles and Best Practices of Scalable Realtime Data Systems, Manning Publications Co., 2015.
[2] A. Galicia, J.F. Torres, F. Martínez-Álvarez, A. Troncoso, A novel Spark-based multi-step forecasting algorithm for big data time series, Inf. Sci. 467 (2018) 800–818.
[3] R. Talavera-Llames, R. Pérez-Chacón, A. Troncoso, F. Martínez-Álvarez, Big data time series forecasting based on nearest neighbors distributed computing with Spark, Knowl.-Based Syst. 161 (1) (2018) 12–25.
[4] J.F. Torres, A. Galicia, A. Troncoso, F. Martínez-Álvarez, A scalable approach based on deep learning for big data time series forecasting, Integr. Comput.-Aided Eng. 25 (4) (2018) 335–348.
[5] M. Zaharia, R.S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M.J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, I. Stoica, Apache Spark: a unified engine for big data processing, Commun. ACM 59 (11) (2016) 56–65.
[6] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.
[7] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, I. Stoica, Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing, in: Proceedings of the USENIX Conference on Networked Systems Design and Implementation, 2012, p. 2.
[8] V.K. Vavilapalli, A.C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, E. Baldeschwieler, Apache Hadoop YARN: Yet Another Resource Negotiator, in: Proceedings of the Annual Symposium on Cloud Computing, 2013, pp. 5:1–5:16.
[9] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A.D. Joseph, R.H. Katz, S. Shenker, I. Stoica, Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center, Technical Report UCB/EECS-2010-87, Electrical Engineering and Computer Sciences, University of California at Berkeley, May 2010.
[10] Databricks Inc. Databricks platform, https://databricks.com/.
[11] Spark notebook, http://spark-notebook.io/.
[12] S. Ceesay, A. Barker, B. Varghese, Plug and play bench: simplifying big data benchmarking using containers, in: Proceedings of the IEEE International Conference on Big Data, 2017, pp. 2821–2828.
[13] Docker, https://www.docker.com/.
[14] E.R. Sparks, A. Talwalkar, V. Smith, J. Kottalam, X. Pan, J. Gonzalez, M.J. Franklin, M.I. Jordan, T. Kraska, MLI: an API for distributed machine learning, in: Proceedings of the IEEE International Conference on Data Mining, 2013, pp. 1187–1192.
[15] J. Yu, J. Wu, M. Sarwat, Geospark: a cluster computing framework for processing large-scale spatial data, in: Proceedings of the 23rd International Conference on Advances in Geographic Information Systems (SIGSPATIAL), USA, 2015, pp. 70:1–70:4.
[16] Y. Zhao, G. Wu, Yadoop: an elastic resource management solution of yarn, in: Proceedings of the IEEE Symposium on Service-Oriented System Engineering (SOSE), March 2015, pp. 276–283.
[17] J. Chen, K. Li, Z. Tang, K. Bilal, K. Li, A parallel patient treatment time prediction algorithm and its applications in hospital queuing-recommendation in a big data environment, IEEE Access 4 (2016) 1767–1783.
[18] Weka, https://www.cs.waikato.ac.nz/ml/weka/.
[19] Knime, https://www.knime.com/.
[20] R.L. Talavera, R. Pérez-Chacón, M. Martínez-Ballesteros, A. Troncoso, F. Martínez-Álvarez, A nearest neighbours-based algorithm for big time series data forecasting, Lect. Notes Comput. Sci. 5391 (2016) 674–679.
[21] B.R. Chang, H.F. Tsai, Y.A. Wang, C.F. Huang, Resilient distributed computing platforms for big data analysis using spark and hadoop, in: Proceedings of the International Conference on Applied System Innovation (ICASI), May 2016, pp. 1–4.
[22] J.F. Torres, A. Troncoso, I. Koprinska, Z. Wang, F. Martínez-Álvarez, Big data solar power forecasting based on deep learning and multiple data sources, Expert Syst. 36 (4) (2019) e12394.
[23] R. Pérez-Chacón, R.L. Talavera-Llames, A. Troncoso, F. Martínez-Álvarez, Finding electric energy consumption patterns in big time series data, in: Proceedings of the International Conference on Distributed Computing and Artificial Intelligence, 2016, pp. 231–238.
[24] R. Shyam, H.B. Bharathi Ganesh, S. Sachin Kumar, Prabaharan Poornachandran, K.P. Soman, Apache spark a big data analytics platform for smart grid, Proc. Technol. 21 (2015) 171–178.
[25] A. Galicia, R. Talavera-Llames, A. Troncoso, I. Koprinska, F. Martínez-Álvarez, Multi-step forecasting for big data time series forecasting based on ensemble learning, Knowl.-Based Syst. 163 (2018) 830–841.
[26] R. Talavera-Llames, R. Pérez-Chacón, A. Troncoso, F. Martínez-Álvarez, MV-kWNN: a novel multivariate and multi-output weighted nearest neighbors

algorithm for big data time series forecasting, Neurocomputing 353 (2019) 56–73.
[27] B. Chang, H. Tsai, Y. Tsai, C. Kuo, C. Chen, Integration and optimization of multiple big data processing platforms, Eng. Comput. 33 (6) (2016) 1680–1704.
[28] Kmeans Spark Sacaladoc, https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.clustering.KMeans.
[29] GLM Spark Sacaladoc, https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.regression.LinearRegressionWithSGD.
[30] FPGrowth Spark Sacaladoc, https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.fpm.FPGrowth.
[31] J. Maillo, S. Ramírez, I. Triguero, F. Herrera, kNN-IS: an iterative spark-based design of the k-nearest neighbors classifier for big data, Knowl.-Based Syst. 117 (2017) 3–15.
[32] Databricks community, https://community.cloud.databricks.com/login.html.
[33] Kubernetes, https://spark.apache.org/docs/latest/running-on-kubernetes.html.
[34] Spark Cluster Deployment, https://github.com/adobe-research/spark-cluster-deployment.
[35] Automating SPARK-YARN-HADOOP-HDFS Cluster Deployment using Ansible, https://github.com/rshad/Automate-Spark-Hadoop-HDFS-Configuration-Ansible.