Optimizations of Distributed Computing Processes on Apache Spark Platform

Tarik Hajji, *Member, IAENG,* Riad Loukili, Ibtissam El Hassani, and Tawfik Masrour, *Member, IAENG*

***Abstract*—The frequently difficult process of examining large and diverse amounts of information is known as "big data analysis." The goal is to find insights, such as hidden patterns, unexpected correlations, or market trends like consumer preferences. Thus, these insights can help companies make quick and informed business decisions. First, we suggest a manual database schema creation technique to reduce the response time of the data analysis process in the context of big data under the Spark platform. Then, in order to distribute the activities to be performed on the Spark slots more evenly, we suggest a data allocation technique. The data is then saved in RAM, using the cache for faster reading. Finally, to solve the problems of loading data into the cache, we suggest using a different file format called "Parquet." We evaluated our strategy using the "fire calls for service" (CFS) data set, which led to a 97.77% faster response time. This demonstrates how the suggested optimization options considerably reduce the time needed to import the cache and read the database.**

***Index Terms*—Distributed computing, Big data, Hadoop, Spark, Optimization, Machine learning, Random forest**

I. Introduction

APPLICATIONS for artificial intelligence [1], [2] are increasingly diverse, and many of them are based on a large amount of data (big data), especially in an industrial environment [3], [4], [5], [6]. Big data (BD) refers to extremely large data sets that can be analyzed by computers to reveal patterns, trends, and associations, particularly with respect to human behavior and interactions [7]. The explosion of data availability, increased storage capacity, and analysis capability have given birth to the notion of BD [8]. Every second, 29,000 "gigabytes" of information are published in the world, which is 2.5 "exabytes" per day or 912.5 "exabytes" per year [9]. This volume of BD is growing at a dizzying pace and is giving rise to new types of statistics [10]. This gigantic amount of data is characterized by the following problems (5V's): Volume, Velocity, Variety, Value, and Veracity [11].

BD analytic functions include statistics [12], spatial analysis [13], semantics [14], interactive discovery and visualization [15]. Analytical models are used to correlate different types and sources of data to create associations and make relevant discoveries [16]. Unstructured and semi-structured data types in Fig. 1 are generally not suitable for traditional relational databases (RD) [17]. This is because they are based on structured data sets. Moreover, RD cannot always handle large data packages with frequent or even continuous updates [18]. This is the case, for example, for stock market transactions, the online activities of Internet users, or the performance of mobile applications. So many organizations that collect, process, and analyze large amounts of data are turning to "NoSQL" databases [19]. The best solution of BD problems is to use the distributed system techniques [20] which is a model whose components located on networked computers communicate and coordinate their actions by transmitting messages on a computer cluster like in Fig. 2 such as Hadoop Eco-system in Fig. 3 and its complementary tools [21], [22], [23], including : "YARN" [24], "Map Reduce" [25], Spark [26], HBase [27], Hive [28], Kafka [29] and Pig [30]. These technologies form the basis of an open source software infrastructure that supports the processing of large and heterogeneous data sets on clustered systems. Hadoop, displayed in Fig. 4, is an open source framework in Java that enables distributed processing of large data sets on clusters of commodity computers using simple programming models [31]. In the Hadoop implementation, the program is sent to the data, unlike traditional systems where the data is sent to the program. In the latest version 3.3.x of Hadoop, we find three essential components: Hadoop's distributed file system (HDFS) to manage data storage problems in Hadoop clusters, "YARN" for managing cluster resources, and "Map Reduce" as a data processing framework [32].

A. HDFS Architecture and Components

Hadoop clusters can read and write more than a "terabyte" of data per second, and HDFS, a distributed file system, eliminates all the major drawbacks of traditional file systems in terms of cost (no licensing and support fees), speed, and reliability (HDFS copies data multiple times). HDFS in Fig. 5 allows user data to be stored in files, follows a hierarchical file system with directories or files, and supports operations such as create, remove, move, and rename.

HDFS provides the following access mechanisms: a Java API for applications, Python access, a C language wrapper for "non-Java" applications, "Web GUI" (content management system) utilized through an "HTTP" browser, and a file system shell for executing HDFS commands.

Manuscript received May 16, 2022; revised January 01, 2023. This work was supported in part by Moulay Ismail University, Meknes, Morroco.

Tarik Hajji is Professor at Laboratory of Mathematical Modeling, Simulation and Smart Systems (L2M3S), Artificial Intelligence for Engineering Sciences Team (IASI), Department of Mathematics and Computer Science, 15290 ENSAM, Moulay Ismail University, 50500 Meknes, Morocco, e-mail: t.hajji@umi.ac.ma

Riad Loukili is PhD student at Laboratory of Mathematical Modeling, Simulation and Smart Systems (L2M3S), Artificial Intelligence for Engineering Sciences Team (IASI), Doctoral Studies Center, 15290 ENSAM, Moulay Ismail University, 50500 Meknes, Morocco, e-mail: r.loukili@edu.umi.ac.ma

Tawfik Masrour is Professor at Laboratory of Mathematical Modeling, Simulation and Smart Systems (L2M3S), Artificial Intelligence for Engineering Sciences Team (IASI), Department of Mathematics and Computer Science, 15290 ENSAM, Moulay Ismail University, 50500 Meknes, Morocco, e-mail: t.masrour@ensam.umi.ac.ma

Ibtissam EL Hassani is Professor at Laboratory of Mathematical Modeling, Simulation and Smart Systems (L2M3S), Artificial Intelligence for Engineering Sciences Team (IASI), Department of Mathematics and Computer Science, 15290 ENSAM, Moulay Ismail University, 50500 Meknes, Morocco, e-mail: i.elhassani@ensam.umi.ac.ma

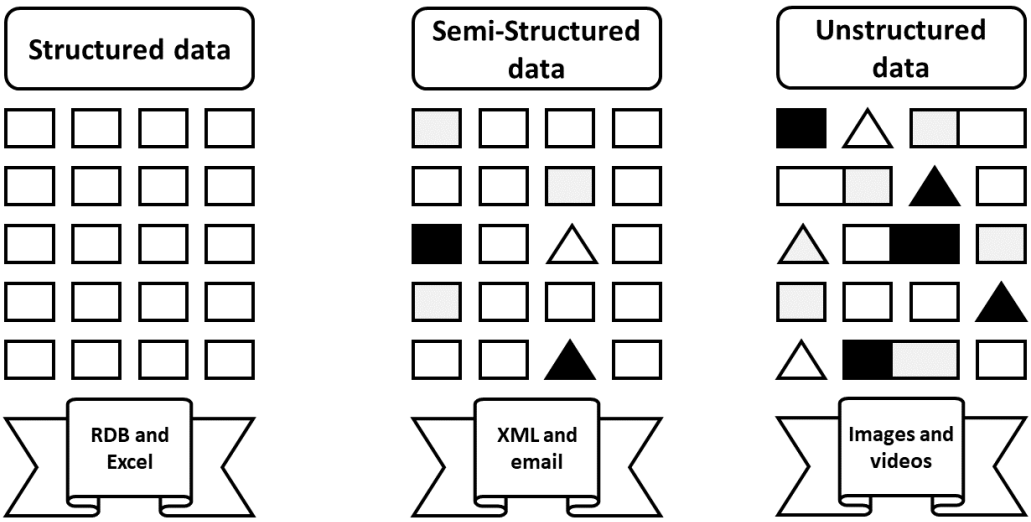

Fig. 1. Unstructured data is a collection of many different forms of data that are saved in their native formats, as opposed to structured data, which is very particular and is stored in a preset manner.

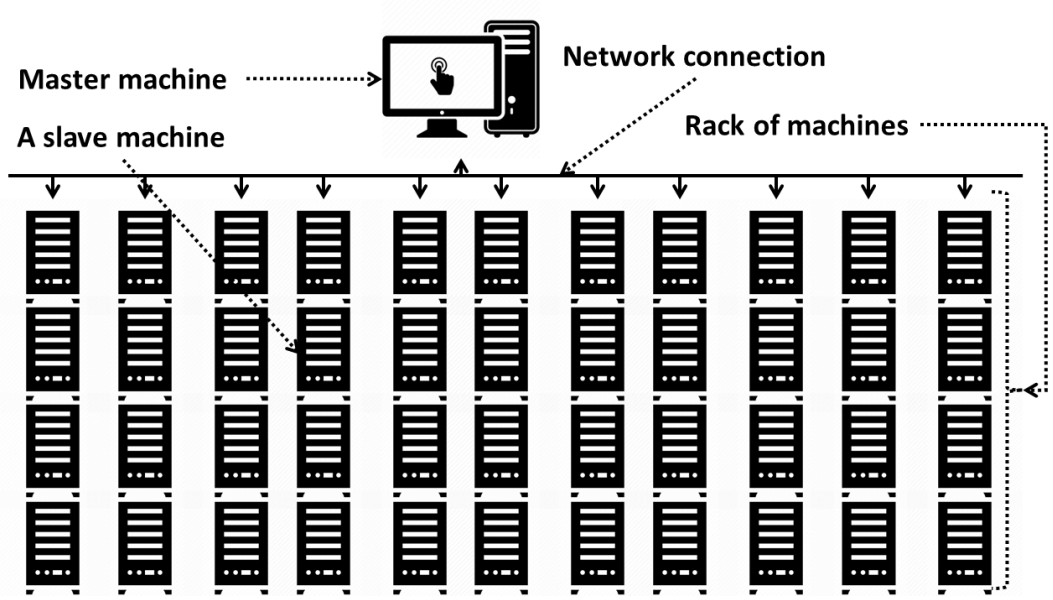

Fig. 2. A group of computers that cooperate so that they can be perceived as a single system is known as a computer cluster.

B. YARN Architecture and Components

”YARN” (Yet Another Resource Negotiator) is a resource manager designed to monitor and manage workloads, maintain a multi-tenant environment, manage Hadoop’s high availability features, and implement security controls. It was created by separating the processing engine and management functions from ”Map Reduce”. Reduced data movement (there is no need to move data between Hadoop and systems running on different clusters), increased cluster utilization (resources unused by one can be consumed by another), and lower operational costs are just a few of the benefits offered by ”YARN,” which takes care of providing computational resources for application execution (there is only one ”do-it-all” cluster to manage). The three important elements of the ”YARN” architecture in Fig. 6 are the resource manager (RM), the application master (AM), and the node manager (NM).

The RM is the master, it runs several services, including the resource scheduler. The AM negotiates resources so that a single application runs in the first container allocated to it. A container is a fraction of the NM capacity and is used by the client for running a program. NM is the slave when it starts; it announces itself to the RM and offers the working containers requested by each AM to the RM. Each NM takes instructions from the RM reports and handles containers on a single node.

C. Distributed Processing in ”Map Reduce” (MR)

MR is a programming model in Fig. 7 that allows huge data sets to be processed and analyzed simultaneously in a logical manner in distinct clusters. While sorting the data,

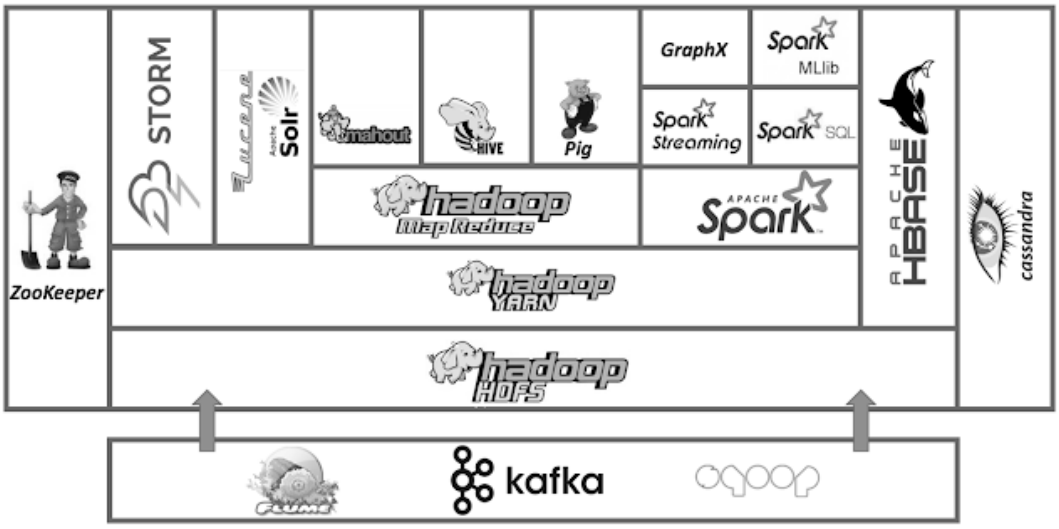

Fig. 3. The many parts of the Apache Hadoop software library are collectively referred to as the "Apache Hadoop ecosystem," which also contains a wide variety of auxiliary tools and open source projects. The Hadoop ecosystem includes a number of well-known tools, such as HDFS, Hive, Pig, "YARN", Map Reduce, Spark, HBase, Oozie, Sqoop, Zookeeper, etc.

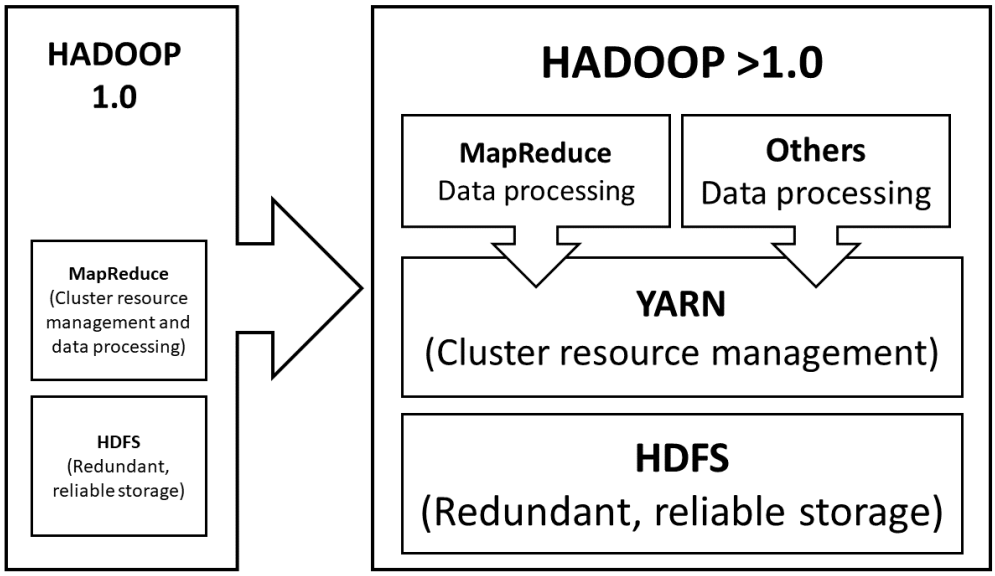

Fig. 4. This is a Hadoop ecosystem. Before 2012, users could write "Map Reduce" programs using scripting languages. Since 2012 users could work on multiple processing models in addition to "Map Reduce" like in Fig. 3.

the map condenses it into logical clusters, removing useless information while keeping the relevant one.

The MR execution process starts with the map phase (reads assigned input split from HDFS, parses the input into records as key-value pairs, applies the map function to each record, and informs the master node of its completion). Then, the partition phase begins (each mapper must determine which reducer will receive each of the outputs, for any key, the destination partition is the same as the number of partitions, which determine the number of reducers). After that, the shuffle phase (which fetches input data from all map tasks for the portion corresponding to the reduce tasks bucket) Then, the sort phase (merge and sort all map outputs into a single run). Finally, the reduce phase applies the user-defined reduce function to the merged data.

The essentials of each MR phase are as follows: the number of reduce tasks can be defined by the users. Each reduce task is assigned a set of record groups, that is, intermediate records corresponding to a group of keys. For each group, a user-defined reduce function is applied to the recorded values. The reduce tasks are read from every map task, and each read returns the record groups for that reduce task. It is not possible to begin the reduction phase until all mappers have finished processing.

D. MR Limits and Paper Architecture

We started our paper by introducing the notion of BD and its different problems. Then, we presented Hadoop and its major elements as well as a proposed solution. But in reality, the MR model of Hadoop has several disadvantages [33]. For example, the reduce phase cannot start until all mappers have finished processing, and MR does not allow writing complex processes consisting of several map and

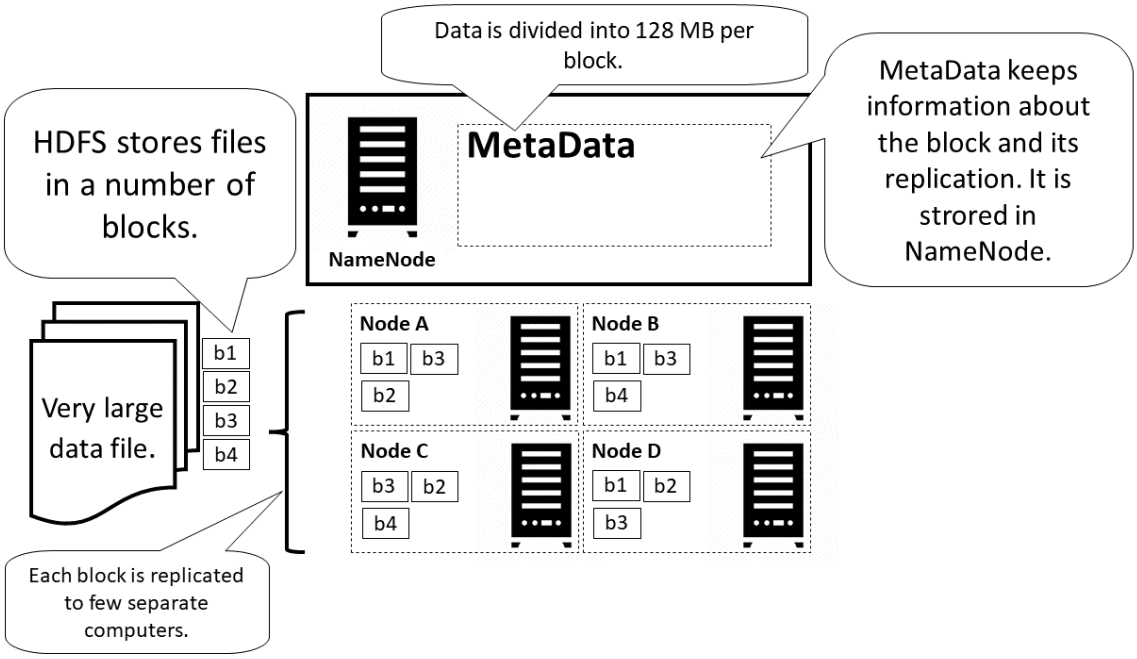

Fig. 5. HDFS architecture and components of Hadoop: Each file is split into a sequence of blocks. The metadata are kept in the NameNode and the data are stored on the different DataNode.

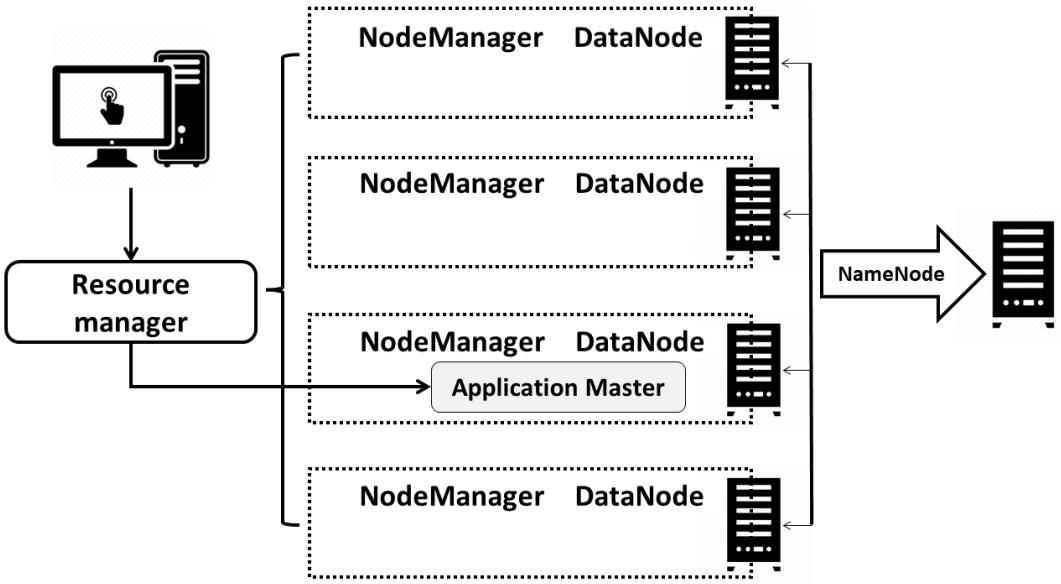

Fig. 6. There are five steps involved in running an application by "YARN": the client submits an application to the RM, the RM allocates a container, the AM contacts the related NM, the NM launches the container and the container executes the AM.

reduce phases because the data of each phase must be stored in HDFS to be reused immediately afterwards in the next phase. This takes a lot of time and space, as "YARN" jobs take a long time to start and run. What does it mean that there is considerable latency.

To remedy this problem, Apache Spark (AS) has been proposed as an alternative to Hadoop MR because it makes much better use of the central memory of the machines in the cluster and manages the chaining of tasks itself. In order to further improve the efficiency of using AS, we present in this article an optimization approach to be made in the process of BD analyzing with AS. These optimizations lead to a 97.77% gain in exclusion time, which proves the efficiency of the proposed approach.

The rest of this paper is organized as follows: The related works section is composed of a presentation of AS and some related works. The methodology section is composed of different stages of our approach. We first proposed a manual approach to define the database schema. Then, we proposed a data allocation algorithm to better distribute the tasks to be executed on the AS slots. Then, we used the cache to save the data in the central memory for a higher reading speed. Finally, we proposed to use a specific file format called "Parquet" to address the problems of loading data into the cache. At the end of this paper, the results and discussion section are presented.

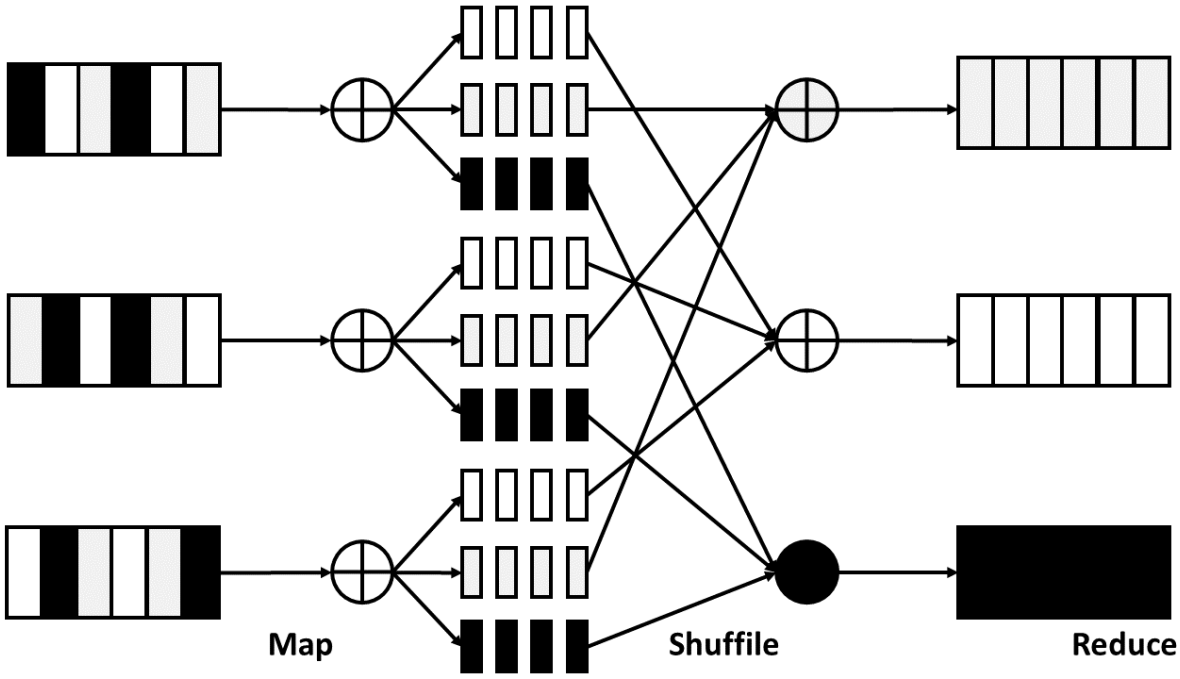

Fig. 7. Hadoop Map Reduce uses data when it works with user provided mappers and reducers. The data is read from files into mappers and emitted by mappers to the reducers. The processed data is sent back by the reducers. Data emitted by reducers goes into output files.

II. BD Analysis Process with AS and Related Work

Typically, BD analysts adopt the concept of a Hadoop data pool [34]. In such architectures, data can be analyzed directly in a Hadoop cluster so that it can be executed using AS. Stored data must be properly organized, configured, and partitioned [35], [36]. This way, good performance of "ETL" (Extract, Transform, and Load) tasks and analytical queries can be achieved [37]. Once the data is ready, it can be analyzed. This can be done using software with data mining tools [38] or predictive analysis tools [39]. But also machine learning [40] which uses algorithms to analyze large data sets, as well as deep learning [41], a more advanced branch of machine learning [42].

However, integrating BD tools into a coherent architecture remains a challenge for many "IT" and analytic teams. That's because they need to identify the right combination of technologies. Then put the pieces together to meet their data analysis needs.

A. Spark Resilient Distributed Data Set (RDD) vs Spark-SQL

Spark RDD is an immutable collection of objects that defines the data structure of Spark. Which is characterized by the following features: lazy evaluation, coarse grained operation, in memory computation, fault tolerant, partitioning, persistent, immutable and location-stickiness [43]. The RDD in Spark allows us to perform a lot of operations, such as iterative algorithms, interactive data mining tools, inefficient implementation of distributed shared memory (DSM), and slower computation when distributed computing systems store data in HDFS or Amazon S3 [44]. However, Spark RDD has no optimization engine, which represents our problems that we try to treat in this article.

Spark RDD supports many types of data: primitive (Integer, Character, Boolean), sequence (Strings, Lists, Arrays, Tuples, Dicts, Nested), Java and Scala objects, and mixed data. And the RDD can be created by three different ways [45]:

1) Paralleled collections (RDDs are created by taking an existing collection and passing it to the Spark context parallelize method).
2) Existing RDDs (RDDs can be created from existing RDDs by transforming one RDD into another one).
3) External Data (the external data set supported by Hadoop, including the local file system, HDFS, Cassandra, HBase, and many more (CSV, JSON, and text files) can be loaded from an external storage system to create RDDs).
4) Using the "Parquet" format for data materialization in the cache.

RDD supports two types of operations: transformation and action.

1) Transformation allows us to create a new data set using the existing one, and there are two types: narrow transformation (is self sufficient, it is the result of map and filter, such that the data is from a single partition only) and wide transformation (is not self sufficient, it is the result of group by key and reduce by key like functions, such that the data can be from multiple partitions).
2) Action allows us to return a value to the driver program, after running a computation on the data set. Actions are the RDD operations that produce non RDD values [46].

To store the result of RDD evaluation, we can use two techniques (which offer benefits in terms of cost and execution time): caching and persistence (see [47]). These RDDs are connected together in the form of a graph named a

directed acyclic graph (DAG), which groups the operations to be performed on RDDs. To get around the drawbacks of the DAG perspective, computation in MR is done as follows: Data is read from HDFS, map and reduce operations are applied to it, and the result is written back to HDFS. The requirement for DAG in Spark was established [48].

The data in an RDD is split into various configurable segments called "partitions" (while running on a cluster, the number of partitions by default is 2) of three types based on size: hash partitioning, range partitioning, and custom partitioning.

In this paper, we propose an approach for the optimal configuration of the number of partitions to minimize the execution time, which is related to the two types of scheduling in Spark (scheduling across applications and scheduling within applications).

Spark offers Spark-SQL a very clever (is a module for structured data processing that is built on top of Spark's core) mechanism that allows users to use the power of "SQL" to query data [49]. Spark-SQL provides the following capabilities for using structured and semi structured data: it acts as a distributed SQL query engine, provides data frames for programming abstraction, and can be used with platforms such as Scala, Java, R, and Python. It allows users to query structured data in Spark programs.

To process data using Spark-SQL, we must convert our RDD to a Spark-SQL data frame. For example, we can use the reflection based approach to automatically convert an RDD with case classes to a data frame using a relational schema. We show in this paper that this approach is relatively too slow compared to the programmatic approach, which is used when you cannot define case classes ahead of time.

B. Related Work

There are many works in the literature that adapt Spark to the problem at hand in order to obtain a faster computational speed when processing large data compared to the standalone method. [50], [51], [52], [53], [54].

Yinan and all authors in [50] worked on wind speed time series prediction to promote the use of wind energy. However, traditional standalone methods are not able to meet the requirements of BD environments. For this, they proposed a hybrid distributed computing framework on AS that divides wind speed BD into RDD groups and operates them in parallel. The proposed module can accurately predict the wind speed in several steps. In addition, the efficiency of different components of the framework is verified. It is also proven that the proposed distributed computing framework has a faster computational speed when processing large data sets.

Population-based "meta-heuristic" algorithms are also used to provide optimizations in the computational process in AS. The authors of [51] used the whale optimization algorithm (WOA), which is a recent artificial intelligence meta-heuristic algorithm based on the feeding behavior of the humpback whale bubble network, to improve performance and reduce computational complexity in the AS context. Similarly, the experimental results demonstrated the superiority of the proposed implementation in terms of speed and scalability.

In [52], the authors presented a distributed computing method for accelerating the space-time linear "K-function" in AS, proposing four strategies for simplifying procedures and accelerating distributed computing.

Our optimization approach for distributed computing processes on the AS platform differs from others in that it is universal, as it can be adapted to most big data analysis problems, is simple to implement, and offers a faster computing speed.

III. Methodology

Our optimization methodology in Fig. 8 is composed of the following four interventions:

1) Data importation process.
2) Data distribution algorithm to better exploit resources.
3) Cache configuration in the RAM to reuse the data.
4) Using the "Parquet" format for data materialization in the cache.

With a manual schema design technique, we start our procedure by loading data as a Spark-SQL data frame from multiple file systems. Then, a data frame partitioning mechanism is used to bypass Spark's default settings, which frequently cause problems. The partitions must then be converted to "Parquet" format and loaded into memory as RDDs. Finally, configure the services for the in-memory cache.

A. Data Importation Process

Spark performs all data analysis operations in memory and in real time. It only calls on disks when its memory is no longer sufficient. This work in memory reduces the latency between treatments, which explains this velocity. However, Spark does not have its own file management system. It needs to be provided with one, for example: HDFS, Informix, Cassandra, Open Swift, or Amazon S3. It is advisable to use it with Hadoop, which currently remains the best global storage solution due to its more advanced administration, security, and monitoring tools.

There are two ways to import data into Spark. The first one, we use in the auto-detection of the schema. The second one is that we indicate the database schema manually. We performed a comparative study (to be detailed later) using a database to select the appropriate data import approach. We noticed that when we specify the database schema manually, the execution time is improved by 97.7%.

Technically, Spark-SQL provides functions to read and write a file or directory of files in "CSV" format into a Spark data frame. The option function can be used to customize the read/write behavior, such as controlling the behavior of the header, delimiter character, character set, etc.

The Spark-SQL API includes several classes to write "SQL" queries using chained method calls. Data frame is the main class; It defines the methods to be applied to the tables. A data frame can be created either directly when loading data or via an RDD, but a schema must be defined first. It should be noted that a schema is a list of "StructFields", each of which is a pair (name, type).

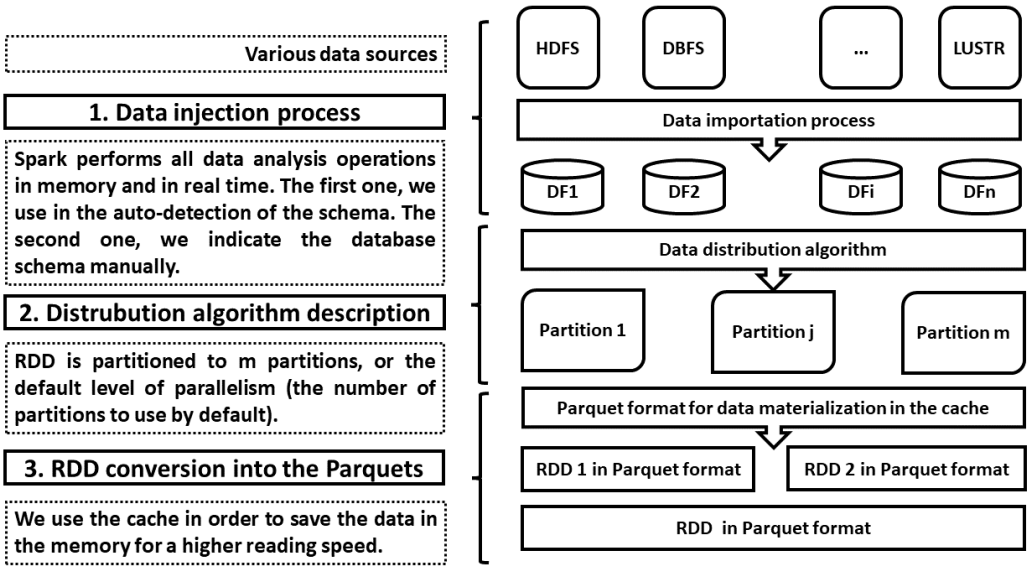

Fig. 8. Description of the methodology: we start with the data import process, then the description of the data distribution algorithm, then the maturity of the data in the cache, and finally the conversion of the RDD into ”Parquet” format.

B. Data Distribution Algorithm

An immutable, partitioned collection of objects that may be processed in parallel is represented by a resilient distributed data set (RDD), which is the fundamental data structure of Spark. RDD is partitioned on a fixed number of partitions, or an default level of parallelism is used (the number of partitions to use by default). The number of partitions can increase or decrease the level of parallelism in this RDD. Internally, this uses a shuffle to redistribute the data.

To define an appropriate number of partitions, we propose (Fig. 9) : to obtain information about the resources assigned to the task from a resource manager such as ”YARN” and to calculate the maximum number of cores for the container. Then we set the new partition number to this maximum.

To get information about a specific container for an application attempt we can use RM REST API’s (Allow the user to get information about the cluster, metrics on the cluster, scheduler information, information about nodes in the cluster, and information about applications on the cluster.) For example, the following ”GET-HTTP” request ”http://rm-http-address:port/ /{appid}/appattempts/{appAttemptId}/containers/{contId}” returns the maximum number of cores allocated to the job for the container as an integer.

C. Configuration of the Cache in the RAM to Reuse the Data

For some workloads, it is possible to improve performance either by caching data in memory or by enabling some experimental options. To cache tables in-memory using Spark-SQL in columnar format we can call the Spark catalog cache table or data frame cache. But to be more efficient, we need to configure some parameters in Tab. I correctly.

D. Parquet Format for Data Materialization in the Cache

We use the cache in order to save the data in the memory (RAM) for a higher reading speed. But this operation takes relatively more time because we copy a file to RAM, which is not an easy task. To remedy this problem, we propose to use ”Parquet” file format. Which is an open source file format available to any project in the Hadoop ecosystem. Apache ”Parquet” is designed for efficient as well as performing flat columnar storage format of data compared to row based files like ”CSV” or ”TSV” files. Our data frames can be saved as ”Parquet” files, maintaining the schema information and the result is also a data frame.

IV. Case Study

In this case study, we developed an ML model to distinguish between genuine and fake emergency calls using the default mode and the suggested method. We processed this study using two data sets: fire incidents (FI) and San Francisco open data on public safety (SFODPS). SFODPS includes all fire unit responses to calls. Each record includes the call number, incident number, address, unit identifier, call type, and disposition. All relevant time intervals are also included. Since this data set is response-based and most calls involve multiple units, there are multiple records for each call number. Addresses are associated with a block number, intersection, or call box, not a specific address.

A. Data Set Description

FI in Fig. 10 includes a summary of each incident (non medical) that the San Francisco fire department responded to. The call number, incident number, address, number, and type of each responding unit, call category (as determined by dispatch), primary circumstance (field observation), actions taken, and property damages are all included in each incident report. This data set satisfies the 5 requirements (5Vs) of BD. In a public safety framework for BD use, the importance of the 5V can be illustrated as follows:

1) Volume: In the field of public safety, and especially for firefighters, the volumes of data to be collected and analyzed are considerable and constantly increasing.

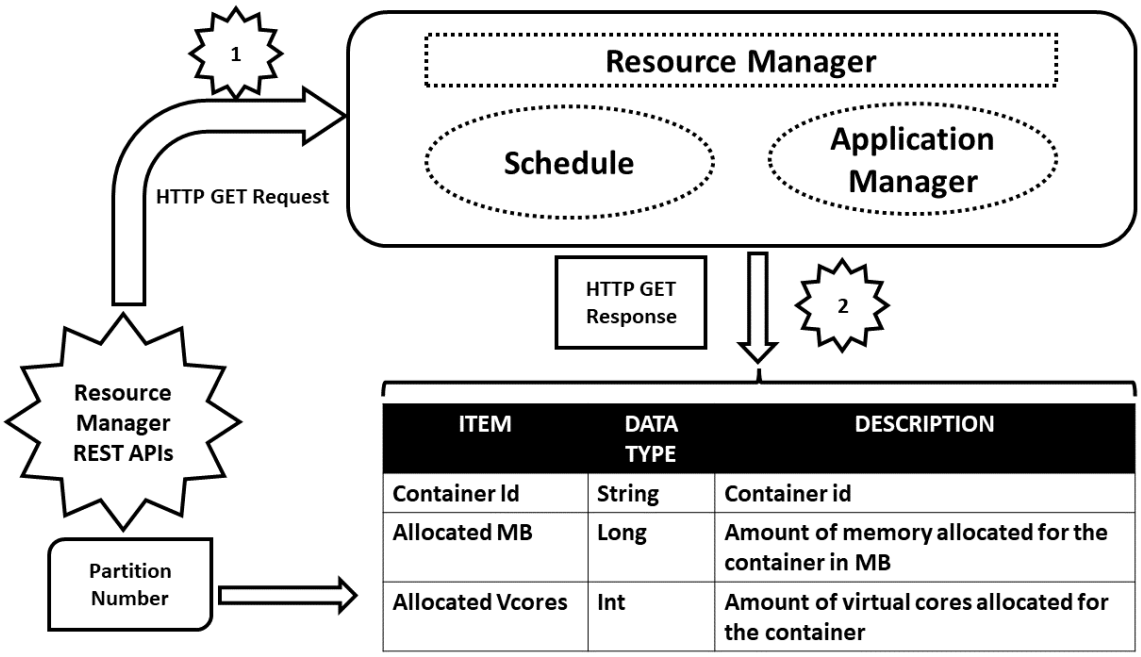

Fig. 9. This figure shows how to get information about the resources of a specific container by using the resource manager REST API to determine the proper number of data partitions.

TABLE I
CONFIGURATION OF CACHE PARAMETERS RAM TO USE SPARK CATALOG CACHE TABLE

Parameters+	Default	Meaning
Compression+	True	When set to true Spark-SQL will automatically select a compression codec for each column based on statistics of the data.
Batch Size+	10000	Controls the size of batches for columnar caching. Larger batch sizes can improve memory utilization and compression, but risk OOMs when caching data.
Max Partition Bytes +	128 MB	The maximum number of bytes to pack into a single partition when reading files.
Open Cost In Bytes+	4 MB	The estimated cost to open a file, measured by the number of bytes could be scanned in the same time. This is used when putting multiple files into a partition. It is better to over estimated, then the partitions with small files will be faster than partitions with bigger files (which is scheduled first).
Shuffle Partitions+	200	Configures the number of partitions to use when shuffling data for joins or aggregations.

Analytics and geolocation practices, for example, are areas that contribute to the explosion of data volume, which needs to be augmented by data from connected objects.

2) Velocity: More and more often, data must be collected and processed in real time, such as for some uses of predictive emergency calls on a website.
3) Variety: data can take many different and heterogeneous forms (voice, web analytics, text, images, etc.).
4) Veracity: The veracity or reliability of data is threatened by declarative behavior (on forms), by the diversity of collection points, by the multiplication of data formats, and by false alerts.
5) Value: in the context of infrared obesity, it is a matter of being able to focus on data that has real value and is actionable.

B. Development Environment

We used the Data-bricks Community Edition [55], which is the free version of the Data-bricks cloud-based BD platform. Its users can access a microcluster as well as a cluster manager and a notebook environment. Data-bricks is an American enterprise software company founded by the creators of Spark. The company also created ”Delta Lake,” ”MLflow” and ”Koalas,” open source projects that span data engineering, data science, and machine learning. Data-bricks develops a web based platform for working with Spark that provides automated cluster management and Python-style notebooks.

	CalNumber	UnitId	IncidentNumber	CallType	CallDate	WatchDate
1	219690030	T03	21030278	Alarms	03/10/2021	03/10/2021
2	203421272	BLS841	20139667	Medical Incident	12/07/2020	12/07/2020
3	213602525	T10	21160001	Structure Fire	12/26/2021	12/26/2021
4	210683285	58	21030264	Structure Fire	03/29/2021	03/29/2021
5	201354667	VAN1	20136406	Medical Incident	11/29/2020	11/29/2020

Fig. 10. Extract from data set used.

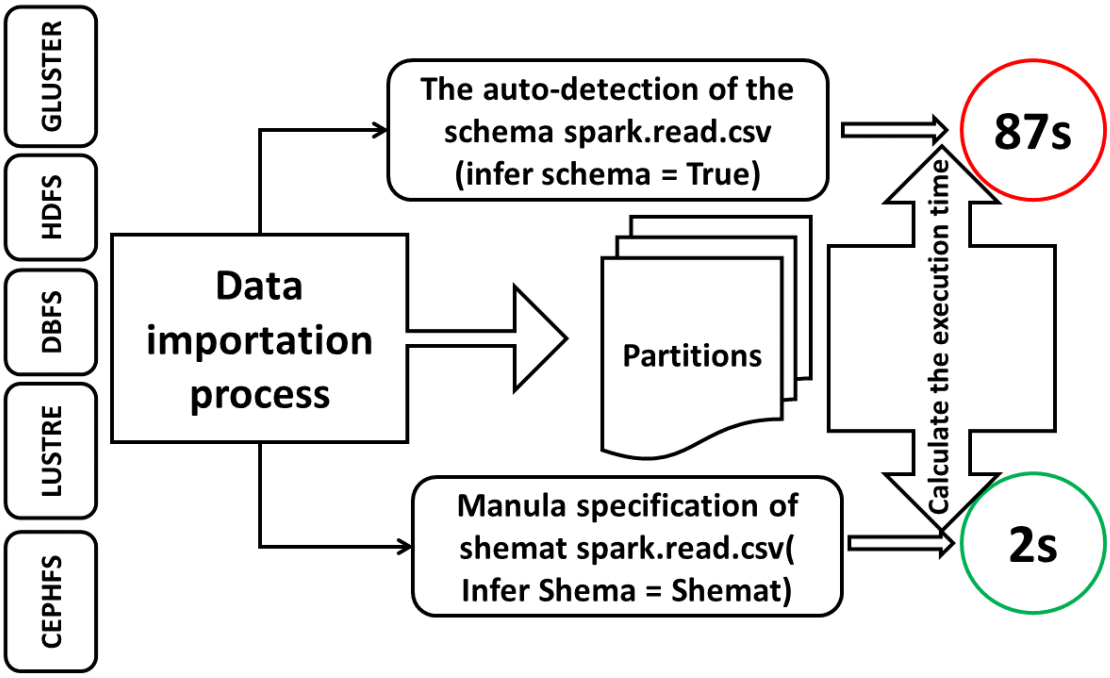

Fig. 11. Automatic and manual data import modes: we can see that the automatic mode lasted 87 seconds, while the manual mode lasted only 2 seconds

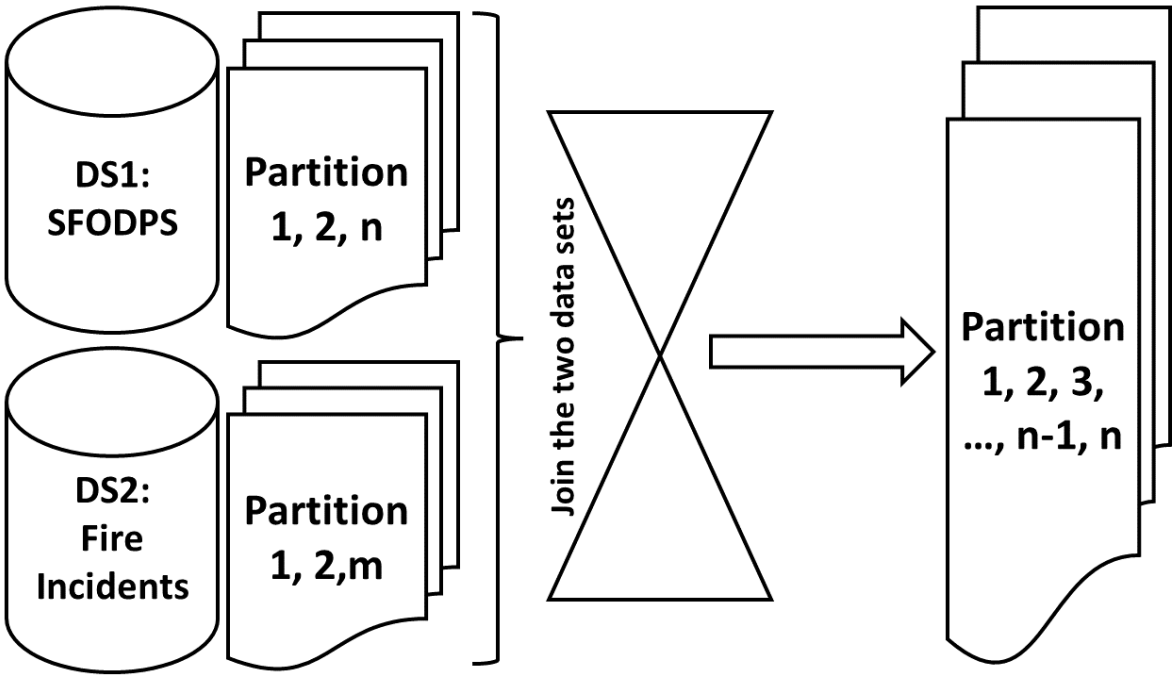

Fig. 12. Concatenation of data sources to create a homogeneous data set.

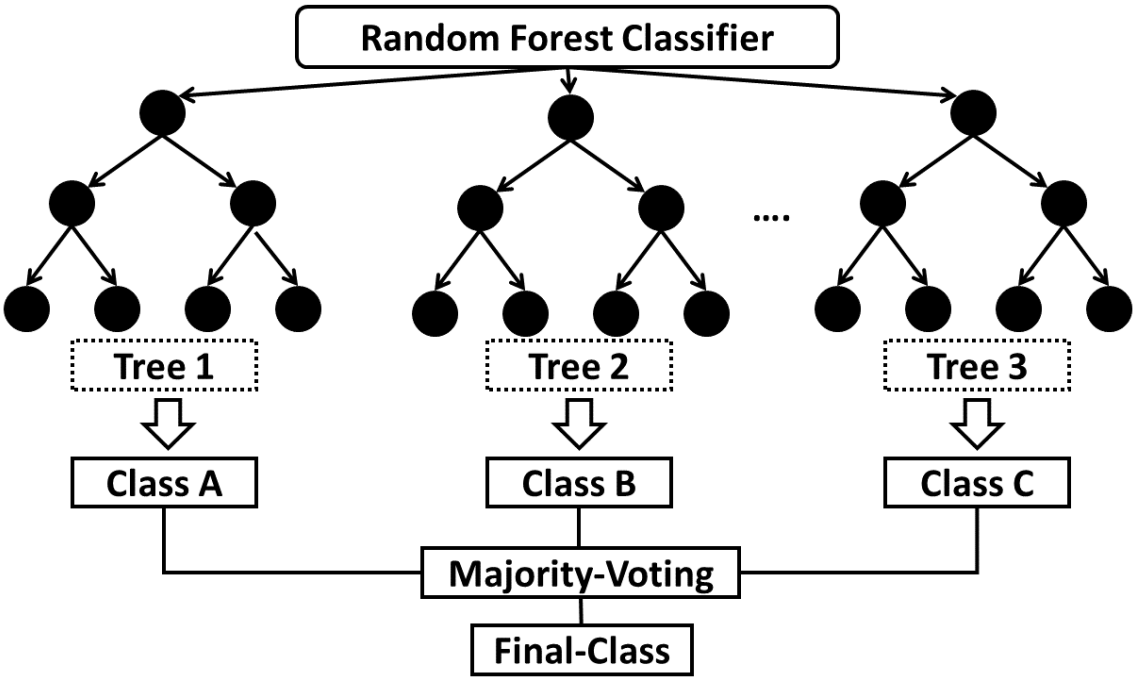

Fig. 13. A well-known machine learning method from the supervised learning approach is Random Forest. It can be applied to classification and regression issues in machine learning. It is based on the idea of ensemble learning, which is the practice of integrating various classifiers to address a complicated issue and enhance the model's performance.

C. Database Import

As illustrated in Fig. 11, there are two methods for importing data into Spark: The first one uses the auto-detection of the schema. The second one is that we specify the database schema manually. This allows us to reduce the execution time from 1 minute and 27 seconds to only 2 seconds.

D. Full Data Scan

A complete scan of all the data allows us to know the total number of records. This operation took us 35 seconds. In order to minimize this execution time, we combine several techniques. By default, Spark subdivides the database into 17 partitions. In our case, we will spread our data over 6 partitions in order to better distribute the tasks to be executed on the Spark slots (3 slots). This allows us to exploit our resources in a better way. Then we use the cache in order to save the data in the RAM for a higher reading speed.

E. Materialize the Data in the Cache

This operation took much longer than before since we were copying a "CSV" file to RAM, which is not a smooth task (6 minutes). To solve this problem, we have to use the "Parquet" file format. Our final execution time was only 0.43 seconds, demonstrating that these strategies do shorten the time spent reading from the database and importing the cache. Hence, an improvement of 4400% and a 97.77% of time saving.

F. Machine Learning and Prediction

First, we need to join the two databases by adding a column for false calls. The figures in Fig. 12 illustrate this process. Then we prepare the processing pipeline and the random forest learning algorithm for classification, like in Fig. 13. It supports both binary and multiclass labels, as well as both continuous and categorical features. The training did not take much time since we used the "SparkML" library, which uses paralyzed machine learning algorithms. This operation took 3 minutes for the whole data set.

V. Conclusion

Users want to use stream processing engines like Spark, Flink, and Storm to feed data into Hadoop ecosystems and execute real-time analytics on it. In addition, the development of AI technologies has made BD intelligence software simpler to use.

For many "IT" and analytic teams, integrating BD tools into a coherent architecture is still difficult. They must determine the best technological fusion. They then assembled the parts to satisfy their requirements for data analysis.

We have suggested a set of practices in this post that have proven to be incredibly successful. We initially suggested defining the database schema manually. Then, we propose a data allocation algorithm to better distribute the tasks to be executed on the Spark slots. Then, for faster reading, we employ the cache to store the data in the memory (RAM). Finally, we suggest using the "Parquet" file format to address the issue of loading data into the cache.

It should be underlined at the conclusion that despite the BD mode's significant expansion, the subject is still open and has a number of untapped potentials.

References

[1] H. Canot, P. Durand, E. Frenod, B. Hassoune-Rhabbour, and V. Nassiet, "Regularized artificial neural networks for predicting the strain of traction-aged polymer systems part i," in *International Conference of Applied Ingineering Mathematics*, 2022.

[2] K. Ota and H. Katagiri, "Demand forecast for bento by machine learning using product popularity based on rating systems."

[3] Z. Hu, L. Wang, Y. Luo, Y. Xia, and H. Xiao, "Speech emotion recognition model based on attention cnn bi-gru fusing visual information." *Engineering Letters*, vol. 30, no. 2, pp. 427–434, 2022.

[4] I. E. Hassani, C. E. Mazgualdi, and T. Masrour, "Artificial intelligence and machine learning to predict and improve efficiency in manufacturing industry," *arXiv preprint arXiv:1901.02256*, 2019.

[5] M. Rhazzaf and T. Masrour, "Smart autonomous vehicles in high dimensional warehouses using deep reinforcement learning approach." *Engineering Letters*, vol. 29, no. 1, pp. 244–252, 2021.

[6] C. El Mazgualdi, T. Masrour, I. El Hassani, and A. Khdoudi, "Machine learning for kpis prediction: a case study of the overall equipment effectiveness within the automotive industry," *Soft Computing*, vol. 25, no. 4, pp. 2891–2909, 2021.

[7] S. Yin and O. Kaynak, "Big data for modern industry: challenges and trends [point of view]," *Proceedings of the IEEE*, vol. 103, no. 2, pp. 143–146, 2015.

[8] S. Kaisler, F. Armour, J. A. Espinosa, and W. Money, "Big data: Issues and challenges moving forward," in *2013 46th Hawaii International Conference on System Sciences*. IEEE, 2013, pp. 995–1004.

[9] Y. Khourdifi and M. Bahaj, "Analyzing social media opinions using hybrid machine learning model based on artificial neural network optimized by particle swarm optimization," in *International Conference on Advanced Intelligent Systems for Sustainable Development*. Springer, 2019, pp. 123–131.

[10] D. A. McFarland, K. Lewis, and A. Goldberg, "Sociology in the era of big data: The ascent of forensic social science," *The American Sociologist*, vol. 47, no. 1, pp. 12–35, 2016.

[11] M. S. Mahdavinejad, M. Rezvan, M. Barekatain, P. Adibi, P. Barnaghi, and A. P. Sheth, "Machine learning for internet of things data analysis: A survey," *Digital Communications and Networks*, vol. 4, no. 3, pp. 161–175, 2018.

[12] P. Galeano and D. Peña, "Data science, big data and statistics," *Test*, vol. 28, no. 2, pp. 289–329, 2019.

[13] T. J. Barnes and M. W. Wilson, "Big data, social physics, and spatial analysis: The early years," *Big Data & Society*, vol. 1, no. 1, 2014.

[14] P. Ceravolo, A. Azzini, M. Angelini, T. Catarci, P. Cudré-Mauroux, E. Damiani, A. Mazak, M. Van Keulen, M. Jarrar, G. Santucci *et al.*, "Big data semantics," *Journal on Data Semantics*, vol. 7, no. 2, pp. 65–85, 2018.

[15] Q. Liu, M. Vorvoreanu, K. P. Madhavan, and A. F. McKenna, "Designing discovery experience for big data interaction: a case of web-based knowledge mining and interactive visualization platform," in *International Conference of Design, User Experience, and Usability*. Springer, 2013, pp. 543–552.

[16] J. Ming, L. Zhang, J. Sun, and Y. Zhang, "Analysis models of technical and economic data of mining enterprises based on big data analysis," in *2018 IEEE 3rd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*. IEEE, 2018, pp. 224–227.

[17] R. Sint, S. Schaffert, S. Stroka, and R. Ferstl, "Combining unstructured, fully structured and semi-structured information in semantic wikis," in *CEUR Workshop Proceedings*, vol. 464. Heraklion Crete, Greece, 2009, pp. 73–87.

[18] H. Mohanty, P. Bhuyan, and D. Chenthati, *Big data: A primer*. Springer, 2015, vol. 11.

[19] R. Addo-Tenkorang and P. T. Helo, "Big data applications in operations/supply-chain management: A literature review," *Computers & Industrial Engineering*, vol. 101, pp. 528–543, 2016.

[20] I. Gorton and J. Klein, "Distribution, data, deployment: Software architecture convergence in big data systems," *IEEE Software*, vol. 32, no. 3, pp. 78–85, 2014.

[21] S. Landset, T. M. Khoshgoftaar, A. N. Richter, and T. Hasanin, "A survey of open source tools for machine learning with big data in the hadoop ecosystem," *Journal of Big Data*, vol. 2, no. 1, pp. 1–36, 2015.

[22] M. M. Rathore, H. Son, A. Ahmad, A. Paul, and G. Jeon, "Real-time big data stream processing using gpu with spark over hadoop ecosystem," *International Journal of Parallel Programming*, vol. 46, no. 3, pp. 630–646, 2018.

[23] S. Mazumder and S. Dhar, "Hadoop ecosystem as enterprise big data platform: perspectives and practices," *International Journal of Information Technology and Management*, vol. 17, no. 4, pp. 334–348, 2018.

[24] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*, 2013, pp. 1–16.

[25] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "Mapreduce online." in *Nsdi*, vol. 10, no. 4, 2010.

[26] S. Salloum, R. Dautov, X. Chen, P. X. Peng, and J. Z. Huang, "Big data analytics on apache spark," *International Journal of Data Science and Analytics*, vol. 1, no. 3, pp. 145–164, 2016.

[27] R. C. Taylor, "An overview of the hadoop/mapreduce/hbase framework and its current applications in bioinformatics," *BMC Bioinformatics*, vol. 11, no. 12, pp. 1–6, 2010.

[28] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, "Hive: a warehousing solution over a map-reduce framework," *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1626–1629, 2009.

[29] P. Le Noac'H, A. Costan, and L. Bougé, "A performance evaluation of apache kafka in support of big data streaming applications," in *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 2017, pp. 4803–4806.

[30] A. Jain and V. Bhatnagar, "Crime data analysis using pig with hadoop," *Procedia Computer Science*, vol. 78, pp. 571–578, 2016.

[31] V. B. Bobade, "Survey paper on big data and hadoop," *Int. Res. J. Eng. Technol*, vol. 3, no. 1, pp. 861–863, 2016.

[32] A. Alam and J. Ahmed, "Hadoop architecture and its issues," in *2014 International Conference on Computational Science and Computational Intelligence*, vol. 2. IEEE, 2014, pp. 288–291.

[33] L. Song, H. Zhang, and D. Feng, "Design of sprint parallelization of data mining algorithms based on cloud computing." *Engineering Letters*, vol. 30, no. 2, pp. 399–405, 2022.

[34] A. Schätzle, M. Przyjaciel-Zablocki, A. Neu, and G. Lausen, "Sempala: Interactive sparql query processing on hadoop," in *International Semantic Web Conference*. Springer, 2014, pp. 164–179.

[35] J. Yu, J. Wu, and M. Sarwat, "Geospark: A cluster computing framework for processing large-scale spatial data," in *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2015, pp. 1–4.

[36] Y. Arfat, R. Mehmood, and A. Albeshri, "Parallel shortest path graph computations of united states road network data on apache spark," in *International Conference on Smart Cities, Infrastructure, Technologies and Applications*. Springer, 2017, pp. 323–336.

[37] N. Biswas and K. C. Mondal, "Integration of etl in cloud using spark for streaming data," in *International Conference on Emerging Applications of Information Technology*. Springer, 2021, pp. 172–182.

[38] S. Salloum, R. Dautov, X. Chen, P. X. Peng, and J. Z. Huang, "Big data analytics on apache spark," *International Journal of Data Science and Analytics*, vol. 1, no. 3, pp. 145–164, 2016.

[39] T. Daghistani, H. AlGhamdi, R. Alshammari, and R. H. AlHazme, "Predictors of outpatients' no-show: big data analytics using apache spark," *Journal of Big Data*, vol. 7, no. 1, pp. 1–15, 2020.

[40] H. Tarik and O. J. Mohammed, "Big data analytics and artificial intelligence serving agriculture," in *International Conference on Advanced Intelligent Systems for Sustainable Development*. Springer, 2019, pp. 57–65.

[41] H. Tarik, M. Tawfik, D. Youssef, S. Simohammed, O. J. Mohammed, and J. E. Miloud, "Towards an improved cnn architecture for brain tumor classification," in *International Conference Europe Middle East & North Africa Information Systems and Technologies to Support Learning*. Springer, 2019, pp. 224–234.

[42] Y. Douzi, T. Hajji, M. Benabdellah, A. Azizi, and T. Masrour, "Classification and watermarking of brain tumor using artificial and convolutional neural networks," in *International Conference on Artificial Intelligence & Industrial Applications*. Springer, 2020, pp. 61–77.

[43] R. K. Mishra, "Spark architecture and the resilient distributed dataset," in *PySpark Recipes*. Springer, 2018, pp. 85–114.

[44] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10)*, 2010.

[45] J. Yu, Z. Zhang, and M. Sarwat, "Spatial data management in apache spark: the geospark perspective and beyond," *GeoInformatica*, vol. 23, no. 1, pp. 37–78, 2019.

[46] S. Chellappan and D. Ganesan, "Introduction to apache spark and spark core," in *Practical Apache Spark*. Springer, 2018, pp. 79–113.

[47] Z. Yang, D. Jia, S. Ioannidis, N. Mi, and B. Sheng, "Intermediate data caching optimization for multi-stage and parallel big data frameworks," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 2018, pp. 277–284.

[48] H. Sreeyuktha and J. Geetha Reddy, "Partitioning in apache spark," in *Innovations in Computer Science and Engineering*. Springer, 2019, pp. 493–498.

[49] T. Ivanov and M.-G. Beer, "Performance evaluation of spark sql using bigbench," in *Big Data Benchmarking*. Springer, 2015, pp. 96–116.

[50] Y. Xu, H. Liu, and Z. Long, "A distributed computing framework for wind speed big data forecasting on apache spark," *Sustainable Energy Technologies and Assessments*, vol. 37, 2020.

[51] M. AlJame, I. Ahmad, and M. Alfailakawi, "Apache spark implementation of whale optimization algorithm," *Cluster Computing*, vol. 23, no. 3, pp. 2021–2034, 2020.

[52] Y. Wang, Z. Gui, H. Wu, D. Peng, J. Wu, and Z. Cui, "Optimizing and accelerating space–time ripley's k function based on apache spark for distributed spatiotemporal point pattern analysis," *Future Generation Computer Systems*, vol. 105, pp. 96–118, 2020.

[53] Z. Wu, Y. Li, A. Plaza, J. Li, F. Xiao, and Z. Wei, "Parallel and distributed dimensionality reduction of hyperspectral data on cloud computing architectures," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 9, no. 6, pp. 2270–2278, 2016.

[54] I. Chebbi, W. Boulila, N. Mellouli, M. Lamolle, and I. R. Farah, "A comparison of big remote sensing data processing with hadoop mapreduce and spark," in *2018 4th International Conference on Advanced Technologies for Signal and Image Processing (ATSIP)*. IEEE, 2018, pp. 1–4.

[55] R. Ilijason, *Beginning Apache Spark Using Azure Databricks: Unleashing Large Cluster Analytics in the Cloud*. Springer, 2020.