BIG DATA PROCESSING WITH APACHE SPARK

Quang-Quy Tran¹*, Duc-Binh Nguyen², Thi-Thuy Linh Nguyen³, Thi-Thu Oanh Nguyen⁴

Abstract – With exponential information growth, it is no surprise that we are in a period of history as the Information Age. The rapid growth of data has presented challenges in terms of storage and processing technology. This article refers to Apache Spark, an eco-system that provides many integrated technologies in Big Data processing with a variety of machine learning libraries and data storage platforms. Apache Spark provides distributed data processing for open-source applications, loading data in-memory and making operations for analyzing data of any size, and efficient support for popular programming languages like Java, Scala, R, and Python. The article aims to compare the superior computing power of Spark to Hadoop and how to connect Spark with today’s popular data processing tools like the R language.

Keywords: Apache Spark, Big Data, distributed-computing, R language.

I. INTRODUCTION

A. Distributed data storage

The rapid proliferation of data at the beginning of the 21st century led to the challenge of finding new data storage and processing technologies. The introduction of the Google File System (GFS) in 2003 was announced by Google as a technology capable of splitting data into smaller pieces and storing them in many different computers [1]. A year later, Google published a paper describing the technology for processing data on the GFS file system, and became known as MapReduce. This processing technology consists of two different tasks: map and reduce. The map task provides the ability to split a large task into small tasks, while the reduce task combines the tasks after being processed from the map task. The word count problem in a document is considered a typical example of describing the performance of MapReduce [2].

In 2006, a group of Yahoo technology engineers researched and improved MapReduce and GFS in an independent research project, which led to the foundation of HDFS (Hadoop Distributed File System). This system has helped to process big data on a distributed platform and become today’s web data processing technology. It is rapidly developing and used by most companies and organizations today in big data storage. Figure 1 illustrates the operation of MapReduce through a simple word-counting problem.

![Example of word counting problem using MapReduce](image)

B. Apache Spark

In 2009, Apache Spark was developed and is a product of a research project at UC Berkeley’s AMPLab aiming to improve MapReduce. It provides superior capabilities for optimizing command-line data processing on multiple machines. Additionally, it can load data directly

¹²⁴Thai Nguyen University, Vietnam
³Lao Cai College, Vietnam
⁴Corresponding author: tqquy@ictu.edu.vn
Received date: 05th July 2023; Revised date: 15th August 2023; Accepted date: 29th August 2023
Quang-Quy Tran, Duc-Binh Nguyen, Thi-Thuy Linh Nguyen et al.

Information Technology

into memory, making data processing much faster than Hadoop (a commonly used big data storage and processing system based on HDFS and MapReduce) several times. An example of Spark’s ability to process large data compared to Hadoop through the logistic regression machine learning algorithm is shown in Figure 2, where with an increasing number of iterations up to 30 times, Spark’s processing time is much faster than Hadoop [4], described in Figure 2.

Fig. 2: Executing logistic regression with Hadoop and Spark [3]

Apache Spark has several advantages in processing big data, which include:
- Providing multiple data processing libraries, clustering technology, and system storage capabilities.
- Effective support for popular data science programming languages such as Java, Scala, R, and Python.
- Provides a variety of machine learning and data processing libraries like Spark SQL, Mlib, GraphX..., the structured, unstructured, and semi-structured data storage platforms like HDFS, Cassandra, Hbase. Figure 3 shows an overview of the integration technologies in Apache Spark.

II. STRUCTURE OF APACHE SPARK
A. Spark Core

Spark Core is the foundation of Apache Spark. It provides a simple programming interface for big data processing. Spark Core is written by Scala but integrated with APIs in Python and R, which are today’s popular open-source platforms for data processing. Many libraries are built for Spark Core with different tasks, like Mlib for machine learning, GraphX for graphics, Spark Streaming for streaming data in real-time, and Spark SQL for structured data processing. Cluster manager used for posts mathematics for processing clustered data through the service is executed, the Spark Core processes the cluster data across platforms like Hadoop YARN, Apache Mesos, Amazon EC2, and Spark Standalon [4]. Figure 4 shows Spark’s core platforms and multi-layer architecture Apache Spark.

Fig. 3: Apache Spark’s integrated Big Data technology platforms [3]

Fig. 4: Spark Core and Apache Spark multi-layer architecture [4]

B. Executing jobs on Apache Spark

An executed Spark job consisting of 5 entities executing concurrent tasks like in Hadoop data processing in Spark will follow the cluster model, which includes a main management server (Driver program) and the physical machines that process the data (Worker Node). The server has the role of writing and reading data from metadata (a form of data that backs up other data’s
information), then uses Spark as a library and defines the specific data computation addresses and workflows of physical clients. Physical sub-machines will include CPU, memory and storage resources, and a physical client - the machine that executes data processing tasks [5].

C. Sparklyr

In most data analysis projects, the main goal is to understand the insights that the data can provide. The main steps in data analysis can be identified as shown in Figure 6.

According to the steps outlined in Figure 6, the data is first brought into the analytics system, where the data analyst performs data wrangling and data visualization on the data after it has been cleaned to discover trends and relationships within the data. To explore further, after the manipulation and visualization data can be fed into a statistical model to uncover hidden patterns.

One issue that arises when working with small data sets and data that can be stored well in memory is that these steps can be performed well on data analytics platforms such as R or Python. However, if the data volume becomes large and the computational memory is insufficient, or the computation methods are too slow, the data will become Big Data. In this case, is it feasible to combine data analytics platforms such as R and Python with Spark? Figure 7 provides the answer to this question.

The idea is to use R to tell Spark what data operations to run, and then only bring the results into R. As illustrated in Figure 7, the ideal method pushes compute to the Spark cluster and then collects results into R.

III. CONNECTING R AND SPARK WITH BIG DATA

A. Connect Spark and R

Spark is built on the Scala programming language platform, so to run Scala, we need to install Java on the system [9]. To check the version of Java installed on a computer, in the Rstudio environment, run the command:

```
System("java -version")  # Check the Java version
```

In Figure 7, the Big Data analyst can easily perform tasks such as data input, visualization, preprocessing, and data modeling on Spark through the R language platform. The idea of Figure 6 shows that we can use R to communicate with Spark to analyze the data and return the results to R for display to the user. This method is called “Push Compute” - pushing the computation up to Spark, and “Collect Results” - collecting the results back to R [7].

Sparklyr is an open-source package that provides an interface between R and Apache Spark. Most of the functions in this package support simplifying data processing on Spark. For example, when we need to build a linear regression model, instead of using common commands like `lm()` in R, we can use the
ml_linear_regression() command in Spark when calling from the R language. This allows R users to quickly call functions in Spark, as shown in Figure 8.

![Fig. 8: Functions in R that invoke methods in Spark](image)

Spark can be installed in Rstudio via the sparklyr package:

```
1. install.packages("sparklyr")
2. library(sparklyr)
3. spark_install()
```

In line 5, we put data into Spark via the above sc connection setup and then show the first 4 data:

```
5. cars = copy_to(sc, mtcars)
6. head(cars, n = 4)
```

In line 5, we put data into Spark via the above sc connection setup and then show the first 4 data:

```
7. library(dplyr)
5. mtcars %>% select(mpg, hp, vs) %>%
   sample_n(100) %>%
   collect() %>%
   plot()
```

We download the sparklyr package from CRAN (The Comprehensive R Archive Network), followed by calling the library to use it, and finally installing Spark.

```
Spark operates on the cluster initialization model. To initialize a cluster with one main server and multiple worker nodes, use the following commands:
```

```
4. sc = spark_connect(master = "local")
```

In the above statement the master parameter defines which machine is specified as a server in the Spark cluster, usually this machine called driver node.

### B. Data analysis and visualization

When the connection between R and Spark is successful, the data can be uploaded to the Spark systems and analyzed. The following example analyzes a data set called mtcars [8] which describes the characteristics of cars:

```
Table 1: Show first 4 data
```

```
<table>
<thead>
<tr>
<th></th>
<th>mpg</th>
<th>cyl</th>
<th>disp</th>
<th>hp</th>
<th>wt</th>
<th>qsec</th>
<th>vs</th>
<th>am</th>
<th>gear</th>
<th>carb</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21</td>
<td>6</td>
<td>160</td>
<td>110</td>
<td>3.9</td>
<td>2.62</td>
<td>16.5</td>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>21</td>
<td>6</td>
<td>160</td>
<td>110</td>
<td>3.9</td>
<td>2.88</td>
<td>17.0</td>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>22.8</td>
<td>4</td>
<td>108</td>
<td>93</td>
<td>3.85</td>
<td>2.32</td>
<td>18.6</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>21.4</td>
<td>6</td>
<td>258</td>
<td>110</td>
<td>3.08</td>
<td>3.22</td>
<td>19.4</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>
```

```
Fig. 10: The scatter plot between miles per gallon and horse power
```

### C. Building models from data

Typically, data is often fed into some statistical machine learning model to uncover correlations between variables in a data set. Correlation analysis is a popular form when considering the relationship between continuous variables, linear regression models are often used to detect correlations. For example: We want to build a linear regression model to find out the correlation between the fuel consumption variable (mpg) and
the pulling power of each car (hp) in the mtcars dataset.

The results from the model show that the intercept of the regression model is 30.09 and the weights value for the independent variable hp is -0.06. The regression equation form:

\[ mpg = 30.09 - 0.06 \times hp \]

This simple model can be used for predicting values outside of the mtcars dataset, such as adding data values with a pull (hp) greater than 250 and visualizing the data the values are calculated.

![Graph showing the difference between actual and predicted values](image)

**Fig. 11: Comparison of predicted and actual values through model**

The red points represent the original values in the data set while the blue points for the predicted values with the with a pull (hp) greater than 250. The results show that the majority of cars that are predicted to have a towing capacity of over 250 consumes a lot of fuel and the distance traveled is also less than that of vehicles with less traction.

**IV. CONCLUSION**

In the content of this article, we have presented the foundation of Apache Spark and compared it with the previous Hadoop distributed data processing system, the advantage of Apache Spark is the ability to compute and process data in large-scale with fast speed, handle stream data well, and support many machine learning libraries. We also introduce in the article about sparklyr library that allows to connect Spark and R language, simplifying statements in big data analysis with R and Spark makes big data analysis simple and more coherent.

**REFERENCES**


