Understanding Apache Spark: A Unified Platform for Database Management

CSC-634 Summer 2021 Graduate Research Project

American University
Authors: Sihyuan Han, Yunting Chiu

Abstract

The new trend for equipping database management systems with analytics features for not only data storage and retrieval but also complex data analytics has gained its attention, especially after the pandemic. The need for acquiring data and processing data in a way that is for immediate interpretation is increasing. Studies that take years to finish are expected to be accompanied by a system where vital insights can be drawn as soon as the database is established due to the demand for fast response to an emergency like this. Therefore, an increasing number of data-related software have been brought out to engineers to reduce the time on data manipulation, modeling and prediction. Apache Spark is one of them in that it offers much tighter integration between relational and procedural processing and it also includes analytical features such as machine learning for in-depth data analysis.

Keywords: big data, relational database, Apache Spark, Spark SQL, Hadoop, MySQL, Data Warehouse

I. Introduction

We deal with data every day and it has already been embedded in our common sense that the world is more dependent on data than ever. People make decisions based on various data moment to moment. Last year has seen an unprecedented time where people track data of COVID-19 cases. While we tried to track how many new cases took place every day in our community, scholars and researchers were called to understand the meaning behind those numbers and to provide viable solutions to mitigate this pandemic. There is no doubt that Database Management Systems (DBMS), which is usually referred to as the technology solution used to optimize and manage the storage and retrieval of data from databases, play a significant role in events like this because it offers a systematic approach to manage databases. For example, Johns Hopkins University provided a database of live COVID-19 cases and even provided data maps for visualization purposes. However, little information can be drawn to understand those data. There has been a new trend for tuning DBMS automatically with computational and analytical features such as employing machine learning. That is to say, we are no longer living in the world of collecting data and doing reporting and visualizations. We try to understand stories behind data so that vital insights and predictions can be drawn to help solve problems.

In pursuit of this purpose, we found it useful to understand this trend by comparing MySQL and Apache Spark because there is a major distinction between the two — MySQL is not a machine learning application, it is simply a database. Apache Spark is an open-source data-processing engine for large data sets and it is designed to deliver the computational speed, scalability, and programmability required for big data — specifically for streaming data, graph data, machine learning, and artificial intelligence (AI) applications. In simple terms, Apache Spark offers the combination of relational processing (e.g. using queries) and complex analytics (e.g. machine learning).

In order to have a more comprehensive understanding of Apache Spark, the paper is organized as follows. We will first provide some basic concepts of Apache Spark and its language — Spark SQL. Then we will provide a detailed comparison between MySQL and Apache Spark and explain why Apache Spark is an improvement of MySQL. In the end, the paper concludes with practical example queries and wraps up with the recommendation of making a decision on choosing the suitable platform to manipulate specific data. Also, we will explain why big tech companies such as Google, Amazon and IBM are requiring their prospective data engineers to have a solid understanding of Apache Spark according to the industry trend.

II. Why Study Apache Spark?

Over the past few years, Spark has become a vital software for data analysts (Zhang et al., 2017). More and more businesses are looking for data scientists to strengthen their work teams and enhance product development. In order to increase the pace of work, big data-related firms such as Amazon are listing their job offers to look for engineers who are familiar with manipulating Spark. Hence, engineers who are interested in big data and especially focus on data related positions should understand the importance of Spark.

In most cases, people will notice that the first school or beginners who are exposed to database management and analysis usually start with MySQL. Nevertheless, MySQL is the basic database system while Spark is a framework that could implement multiple languages such as Python and Scala. The trend of Spark is ongoing with all the convenient features that provide the suitable integration with connecting and reading other databases with numeric structures.

In short, to acquire a thorough comprehension of Spark, it is necessary to have a basic concept of learning more detail about this system. Continuing with the next chapter, readers would have a general objective on Spark, and acquire a knowledge of its performance.

III. What is Apache Spark?

When it comes to Spark, it is important to learn about its history before digging deeper. Years ago, there were lots of data analysts who were not familiar with Java, which may cause difficulties doing MapReduce programming. In order to allow the programmer to easily operate SQL language on top of HDFS data, engineers develop a language called "Shark" (Chen, 2019). Due to security issues and limitations while working on Shark, in 2014, "Spark SQL" became one of the most popular data science languages.

Spark SQL supports several languages including Scala, Java and Python. Users could insert data formats such as Hive, HDFS, Cassandra or JSON into the Spark environment and further run queries for analyzing. Spark is convenient for data scientists who are already familiar with a data-related language. For instance, Spark could be utilized in R with library(sparklyr) or it could be built on Pycharm with PySpark. Spark handles not only SQL queries but also supports more directly intuitive language since SQL query has its restriction on complex tasks.

When Spark first came out, it was only allowed to insert RDD format data, which is non-structured data that brings obstacles when managing it. Then it had been upgraded to Spark 1.3, which is a distributed database based on columns, it became more simple to calculate or compress data. Finally there is Spark 2.0, which integrates dataframes and datasets. As an encoder distributed database, serialized structured data has compiled code into binary sequence to strengthen the efficiency. Spark 2.0 using the brand new interface SparkSession for inserting and transforming different structured data into dataframe. Furthermore, Spark keeps up turning dataframe into SQLContext and so engineers can still do basic SQL queries.

There is an interesting notion that readers should notice is that, unlike MySQL, Spark is not a database language that could store data. It is a manipulating data framework that is widely used for executing SQL languages. In addition, Spark can take advantage of JDBC to connect to the MySQL database system for further analysis.

In Sum, Spark is an integrated analytics engine for processing big data. It is a framework that is suitable for quickly performing very large tasks.

IV. SQL v.s. Spark SQL

Spark SQL is a new Apache Spark module that combines relational processing with the functional programming API of Spark. Spark SQL allows Spark programmers to take advantage of the benefits of relational processing while SQL users can call complicated analytics libraries in Spark (e.g., machine learning and artificial intelligence).

SQL is an acronym for Structured Query Language. It is a language for interacting with databases. For relational database management systems, SQL is the standard language. Spark SQL is a Spark module for dealing with structured data. It offers data frames as a programming abstraction and serves as a distributed SQL query engine. The merit of SparkSQL is that it has a wide range of applications. For instance, HiveMetastore can be used by SparkSQL to retrieve metadata from HDFS data. Summarizing, SparkSQL can use this metadata to improve the optimization of the queries it runs.

Spark SQL has two major differences from MySQL. Spark SQL, for starters, offers substantially tighter interaction between relational and procedural processing by employing a declarative DataFrame API that interfaces with procedural Spark code. Secondly, it comes with Catalyst, a highly extendable optimizer built using Scala capabilities that makes it simple to add composable rules, control code generation, and specify extension points. To be more specific, “Spark SQL is an evolution of both SQL-on-Spark and of Spark itself, offering richer APIs and optimizations while keeping the benefits of the Spark programming model (Armbrust et al., 2015)”. Because Spark SQL has API references, it can support the features of other programming languages such as Scala, Java, R and Python.

V. Interesting Features of Apache Spark

As a flexible computing framework, in comparison to other data-associated languages, Spark has numerous fascinating features that attract engineers. Spark has been known as a widely and extensively used platform for data manipulation, its feature not only includes speed but also compatibility while dealing with complicated tasks. Besides, Spark is also interchangeable on developing tools integration, the security of preventing data loss is also an enchanting function.

First, the most absorbing feature of Spark is speed. In comparison to MapReduce, no matter how big the data it calculates on computer memory or hard disk, it is quicker than other data manipulation languages. The maximum speed difference between Spark and Hadoop could be up to 100 times. In general, it's still 2-10 times quicker than Hadoop. Spark fulfills efficient DAG engine execution by handling data in memory space.

Next captivating function is the interchangeable purpose. For office or users who use plenty of different platforms at once, compatibility is extremely important, Spark R, GraphX, streaming or even MLlib are all compatible with the same Spark platform. To the office users, this feature can not only reduce labor cost but also lower down development time.

Another feature of Spark is compatibility. Spark does not need third-party system management, it can stand alone, which substantially decreases the entry barrier. Following the continuing update, Spark established its own “Date” type structure. Moreover, while reading data into Spark, if a string has been converted to invalid numeric, the error messages will pop up.

Last but not least, safety protection. Spark can operate isolated calculations on memory space, which is a convenient function but also may cause data loss issues when system errors occur. Thus, Spark developed a record system to store every running step. So even when the computer breaks, it can trace back every move and reckon the lost file.

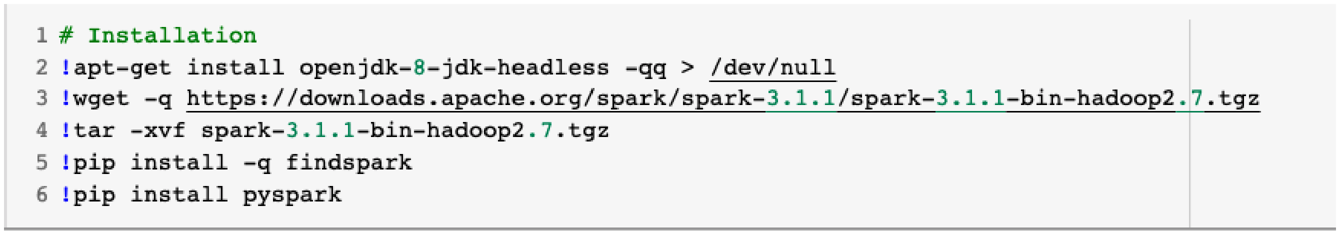
In short, there are many attractive features of Spark. However, the limitation of Spark may be that it is not a data management system that could store actual databases such as MySQL. Spark is a framework that could integrate with other languages and incorporate AI and machine learning. Therefore, engineers and data scientists still need to consider carefully while choosing the language they tend to operate depending on the purposes of data manipulation.

VI. Practical Queries

PySpark is a Python API for Apache Spark that allows us to tame Big Data by combining the simplicity of Python and the power of Apache Spark. To demonstrate the difference between MySQL and PySpark, we will use Google Colab to demonstrate insert, update, select, view, and delete queries in a Bank database using spark syntaxes. Please click here to run the entire code.

1. Start Spark in Google Colab
   Spark SQL is Apache Spark's module for working with structured data .PySpark is a Python interface to Apache Spark. It includes the PySpark shell for interactively examining data in a distributed environment, as well as the ability to develop Spark applications using Python APIs. Most Spark technologies, such as Spark SQL, DataFrame, Streaming, MLlib (Machine Learning), and Spark Core, are supported by PySpark.

2. Installing PySpark
   Firstly, the symbol(*!*) indicates that we want to execute a shell command on the computer. The second line uses the *wget* command to download PySpark. It's critical to double-check that the link is active and that we are utilizing the latest PySpark version. The last two lines use the *pip* command to install the *findspark* and *pyspark* modules.



```
# Installation
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://downloads.apache.org/spark/spark-3.1.1/spark-3.1.1-bin-hadoop2.7.tgz
!tar -xvf spark-3.1.1-bin-hadoop2.7.tgz
!pip install -q findspark
!pip install pyspark
```

So far, we have installed Spark and Java in Colab. To allow Pyspark to run in our workspace, we must configure the environment path. The following code sets the location of Java and Spark.

```python
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.1.1-bin-hadoop2.7"
```

The following code is testing the installation status.

```python
import findspark
findspark.init()
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()
```

Finally, we examine the PySpark version to verify if it is compatible with Colab.

```python
# Check the pyspark version
import pyspark
print(pyspark.__version__)
```

3. Create Bank Database and Insert Queries
   I will only create *account* and *loan* entities as an example. The original bank database is from here, where we practiced MySQL queries in CSC-634 class.

```python
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType,StructField, StringType, IntegerType
# create account table
account_attr = StructType([ \
    StructField("account_number",StringType(),True), \
    StructField("branch_name",StringType(),True), \
    StructField("balance",IntegerType(),True) \
  ])

# insert the values
account_val = [("A-101","Downtown", 500),
    ('A-102', 'Perryridge', 400),
    ('A-201', 'Brighton', 900),
    ('A-215', 'Mianus', 700),
    ('A-217', 'Brighton', 750),
    ('A-222', 'Redwood', 700),
    ('A-305', 'Round Hill', 350)
  ]

account = spark.createDataFrame(data = account_val, schema = account_attr)
# account.printSchema()
account.show()
```

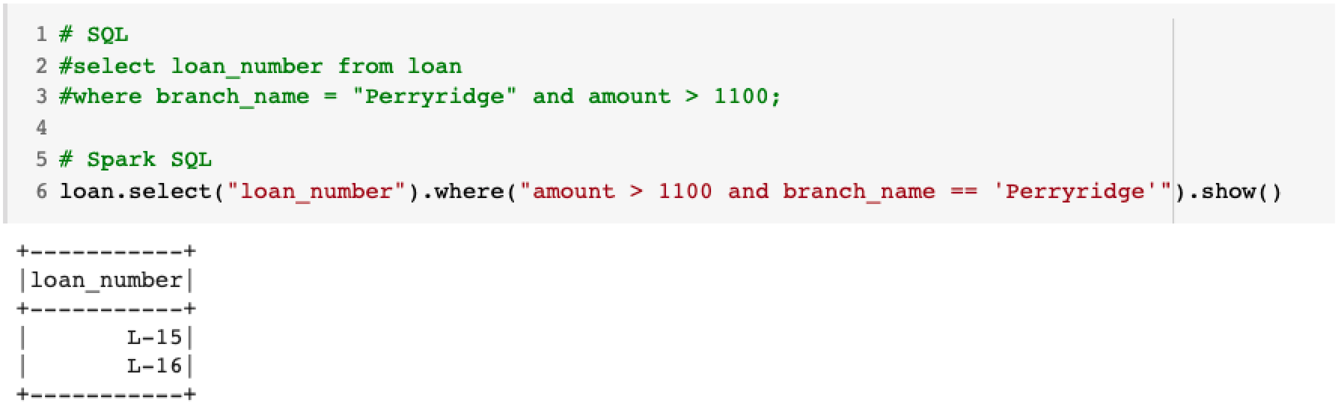
```
+--------------+-----------+-------+
|account_number|branch_name|balance|
+--------------+-----------+-------+
|         A-101|   Downtown|    500|
|         A-102| Perryridge|    400|
|         A-201|   Brighton|    900|
|         A-215|     Mianus|    700|
|         A-217|   Brighton|    750|
|         A-222|    Redwood|    700|
|         A-305| Round Hill|    350|
+--------------+-----------+-------+
# create loan table
loan_attr = StructType([ \
    StructField("loan_number",StringType(),True), \
    StructField("branch_name",StringType(),True), \
    StructField("amount",IntegerType(),True) \
  ])

# insert the values
loan_val = [("L-11","Round Hill", 900),
    ('L-14', 'Downtown', 1500),
    ('L-15', 'Perryridge', 1500),
    ('L-16', 'Perryridge', 1300),
    ('L-17', 'Downtown', 1000),
    ('L-23', 'Redwood', 2000),
    ('I-93', 'Mianus', 500)
  ]

loan = spark.createDataFrame(data = loan_val, schema = loan_attr)
# account.printSchema()
loan.show()
```

```
+-----------+-----------+------+
|loan_number|branch_name|amount|
+-----------+-----------+------+
|       L-11| Round Hill|   900|
|       L-14|   Downtown|  1500|
|       L-15| Perryridge|  1500|
|       L-16| Perryridge|  1300|
|       L-17|   Downtown|  1000|
|       L-23|    Redwood|  2000|
|       I-93|     Mianus|   500|
+-----------+-----------+------+
```

4. Retrieval Queries
   Find all loan numbers for loans made at the Perryridge branch with loan amounts greater than $1100.

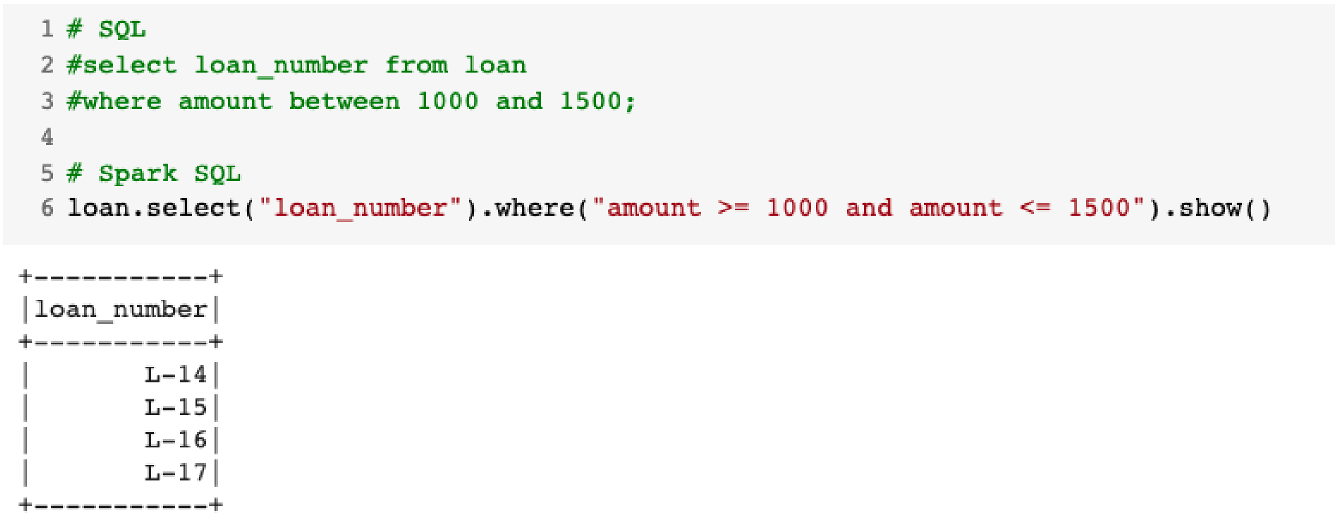


```
# SQL
#select loan_number from loan
#where branch_name = "Perryridge" and amount > 1100;

# Spark SQL
loan.select("loan_number").where("amount > 1100 and branch_name == 'Perryridge'").show()
```

```
+-----------+
|loan_number|
+-----------+
|       L-15|
|       L-16|
+-----------+
```

Find the loan number of those loans with loan amounts between $1,000 and $1,500. That is, >= $1,000 and <= $1,500.



```
# SQL
#select loan_number from loan
#where amount between 1000 and 1500;

# Spark SQL
loan.select("loan_number").where("amount >= 1000 and amount <= 1500").show()
```

```
+-----------+
|loan_number|
+-----------+
|       L-14|
|       L-15|
|       L-16|
|       L-17|
+-----------+
```

Using aggregate function: Find the *brance_name* in the *loan* table whose total amount is greater than 1000.

```
# SQL
# select branch_name , sum(amount) from loan
# group by branch_name
# having sum(amount) > 1000;

# Spark SQL
import pyspark.sql.functions as func
loan.groupBy("branch_name").agg(func.sum("amount")).where("sum(amount) > 1000").show()
```

```
+-----------+-----------+
|branch_name|sum(amount)|
+-----------+-----------+
| Perryridge|       2800|
|   Downtown|       2500|
|    Redwood|       2000|
+-----------+-----------+
```

PySpark inner join *account* and *loan* entities

```
# SQL
# from account inner join loan on account.branch_name = loan.branch_name;

# Spark SQL
account.join(loan, account.branch_name == loan.branch_name, "inner").show()
```

```
+--------------+-----------+-------+-----------+-----------+------+
|account_number|branch_name|balance|loan_number|branch_name|amount|
+--------------+-----------+-------+-----------+-----------+------+
|         A-102| Perryridge|    400|       L-15| Perryridge|  1500|
|         A-102| Perryridge|    400|       L-16| Perryridge|  1300|
|         A-215|     Mianus|    700|       I-93|     Mianus|   500|
|         A-305| Round Hill|    350|       L-11| Round Hill|   900|
|         A-101|   Downtown|    500|       L-14|   Downtown|  1500|
|         A-101|   Downtown|    500|       L-17|   Downtown|  1000|
|         A-222|    Redwood|    700|       L-23|    Redwood|  2000|
+--------------+-----------+-------+-----------+-----------+------+
```

5. Update Queries and View Queries
   We should know that Spark does not support *update* and *delete* queries on dataframes. For deletion, we must use Python's external API in the code.

   If the loan amount is greater than $1,000, the loan amount will be increased by 5%.

Before the query

```python
# Register the DataFrame as a global temporary view
loan.createGlobalTempView("loanTMP")

# Global temporary view is tied to a system preserved database `global_temp`
spark.sql("SELECT * FROM global_temp.loanTMP").show()
```

```
+-----------+-----------+------+
|loan_number|branch_name|amount|
+-----------+-----------+------+
|       L-11| Round Hill|   900|
|       L-14|   Downtown|  1500|
|       L-15| Perryridge|  1500|
|       L-16| Perryridge|  1300|
|       L-17|   Downtown|  1000|
|       L-23|    Redwood|  2000|
|       I-93|     Mianus|   500|
+-----------+-----------+------+
```

After the query

```python
from pyspark import SparkConf, SparkContext
from pyspark.sql import SQLContext, HiveContext
from pyspark.sql import functions as F

# SQL
# update loan
# set amount = amount + amount * 0.05 where amount > 1000;

# Spark SQL with external functions
loan = loan.withColumn("amount", F.when(F.col("amount") > 1000,
                      F.col("amount") + F.col("amount") * 0.05).otherwise(F.col("amount"))).show()
```

```
+-----------+-----------+------+
|loan_number|branch_name|amount|
+-----------+-----------+------+
|       L-11| Round Hill| 900.0|
|       L-14|   Downtown|1575.0|
|       L-15| Perryridge|1575.0|
|       L-16| Perryridge|1365.0|
|       L-17|   Downtown|1000.0|
|       L-23|    Redwood|2100.0|
|       I-93|     Mianus| 500.0|
+-----------+-----------+------+
```

6. Delete Queries

Before the query

```
account.show()
```

```
+--------------+-----------+-------+
|account_number|branch_name|balance|
+--------------+-----------+-------+
|         A-101|   Downtown|    500|
|         A-102| Perryridge|    400|
|         A-201|   Brighton|    900|
|         A-215|     Mianus|    700|
|         A-217|   Brighton|    750|
|         A-222|    Redwood|    700|
|         A-305| Round Hill|    350|
+--------------+-----------+-------+
```

After the query

```
# SQL
# delete from account where branch_name != "Downtown";

# Spark SQL with external functions
account_updated = account.filter(F.col("branch_name") == "Downtown")
account_updated.show()
```

```
+--------------+-----------+-------+
|account_number|branch_name|balance|
+--------------+-----------+-------+
|         A-101|   Downtown|    500|
+--------------+-----------+-------+
```

VII. Conclusion

In conclusion, Spark is a modern popular framework that database engineers are looking for. It's fascinating features including high efficiency and compatible function with Artificial Intelligence and machine learning starts a new chapter of data manipulating trend. This paper elaborates that Spark has been chosen to discuss because of its integrated and up-to-date features. With an introduction of Spark, researchers may have a general idea of Spark's history and its difference between each generation. A comparison between Spark SQL and MySQL would give current MySQL users a brief idea of what is the major difference between the two languages. Hands-on queries using bank data that we practiced in the CSC-634 course also helped us to understand Spark in-depth. After reading the paper, engineers or businesses may consider starting to learn or develop this new trendy platform.

References

Armbrust, M., Xin, R., Lian, C., Huai, Y., Liu, D., Bradley, J., Meng, X., Kaftan, T., Franklin, M., Ghodsi, A., & Zaharia, M. (2015). Spark SQL: Relational Data Processing in Spark. *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 1383–1394. https://doi.org/10.1145/2723372.2742797

Singh, P. (2019). Introduction to Spark. In Learn PySpark (pp. 1–16). Apress. https://doi.org/10.1007/978-1-4842-4961-1_1

Zhang, X., Khanal, U., Zhao, X., & Ficklin, S. (2018). Making sense of performance in in-memory computing frameworks for scientific data analysis: A case study of the spark system. *Journal of Parallel and Distributed Computing*, 120, 369–382. https://doi.org/10.1016/j.jpdc.2017.10.016