Distributed Computing with Spark SQL

Sizhe Liu

Version 1.0

Contents

1 Introduction

Δ Question
Amdahl's Law

```
// the amount of acceleration we would see from parallelizing a task is a function of
   what portion of the task can be completed in parallel.
```

Δ Question
How to create a table using CSV files?

```
CREATE TABLE IF NOT EXISTS fireCalls
USING csv
OPTIONS (
  header "true",
  path "/mnt/davis/fire-calls/fire-calls-truncated-comma.csv",
  inferSchema "true"
)
```

Δ Question
How to select a column with spaces in its name?

```
//Use `xxxx sss`
```

2 Spark Core Concepts

Δ Question
Basic construction of spark? And what is its local mode

```
//Spark is made of one driver and many executors.
//In the local mode, the driver and executor are on the same physical machine.
```

Δ Question
What is the unit of parallelism within a Spark cluster

```
//it is called slot. A slot can be either a core or a thread in a core. It can be
   calculated as "machines*cores*threads".
```

Δ Question
What is the unit of parallelism for data?

```
//It is called partitions. Partitions are part of a large distributed dataset.
```

Δ Question
How to determine the number of partitions?

```
//size of dataset, underlying partitioning of data by some other feature, and cluster
   configurations
```

Δ Question
what happens if I increase number of partitions arbitrarily?

```
It increases the cost of communications between different units.
```

2.1 Caching

Δ Question
What would happen when you cache your data?

```
//Spark will load our data from online server onto the memory of our executors.
CACHE TABLE XXX
```

Δ Question
How to check storage level information after each SQL command?

```
//Use the Spark UI by clicking the "view" option after each command.
//In the storage tab, we can tell how the data is stored
```

Δ Question
What does the storage level "Memory Deserialized 1x Replicated " mean

```
//The data is stored as deserialized data, which takes less time to make them paralleled.
    But it takes more space to store them than the serialized data.
```

Δ Question
What is Tungsten?

```
//Tungsten optimized the storage space for data.
```

Δ Question
How to uncache data?

```
UNCACHE TABLE xxxx
```

Δ Question
What does "CACHE LAZY"do?

```
CACHE LAZY TABLE XXXX
//It will only cache data as it is required/queried.
//It will not materialize all the entries in the database until your query need to be
   performed all the entries.
```

```
//Make sure clear cache before close a section on spark
CLEAR CACHE
```

2.2 Shuffle Partitions

Δ Question
What is Narrow/Wide Transformation?

```
//Any transformation that can be done on each partition in parallel.
//On the other hand, wide transformation requires information from other partitions
//The number of post shuffle partitions is set to 200. But it can be changed.
```

Δ Question
What is shuffle read/write?

```
//With the shuffle write, they add up their count or they do their GROUP BY locally,
   write it out to their local disk, then they go ahead and read around those different
   files from the other executors. So that's what we have with the shuffle read.

//by default in Spark, it uses 200 tasks for the shuffle read. So our resulting partition
    number is 200.
```

Δ Question
How to change the number of tasks used for shuffle read?

```
SET spark.sql.shuffle.partitions=x
//For small dataset, use small x, use larger x for large dataset to speed up your queries
   .
//Some number of shuffle partitions to try:8,64,100,400...
```

2.3 Analyze Spark UI

Δ Question
What is A stage boundary?

```
//A stage boundary occurs when all of your slots are available units of processing have
   to sync up with each other. So Spark is also known as a bulk synchronous processing
   engine.

//For example, When we perform a count, each executor has to sum up their count locally.
   Only once all of them finish, then one slot is tasked with adding up all of the counts
    from the other executors.
```

When you use spark UI to analyze read and write, make sure each partition has relatively equal amount of date to work on.

2.4 Broadcast Joins

In a standard join, ALL the data is shuffled.This can be really expensive.

An alternate way of joining your data sets is doing something called a broadcast join. **If you have two datasets and one is significantly smaller than the other, instead of partitioning it up across your three workers, you instead broadcast an entire copy of that data set to every worker**. So you can do those joins locally. So this will be a lot more efficient.

2.4.1 Automatic and Manual broadcasting

Depending on size of the data that is being loaded into Spark, Spark uses internal heuristics to decide how to join that data to other data.

- Automatic broadcast depends on "spark.sql.autoBroadcastJoinThreshold" The setting configures the maximum size in bytes for a table that will be broadcast to all worker nodes when performing a join. Default is 10MB
- A broadcast function can be used in Spark to instruct Catalyst that it should probably broadcast one of the tables that is being joined.
- If the broadcast hint isn't used, but one side of the join is small enough (i.e., its size is below the threshold), that data source will be read into the Driver and broadcast to all Executors.

Δ Question
How to check the autoBroadcastJoinThreshold?

```
//Use Python:
%python
spark.conf.get("spark.sql.autoBroadcastJoinThreshold")
```

Δ Question
How to broadJoin manually?

```
EXPLAIN
  SELECT /*+ BROADCAST(db1) */ *
  FROM db1
  JOIN db2 on db1s.`xxx` = db2.`xxx`
//The result shows the physical plan
== Physical Plan ==
*(2) BroadcastHashJoin [Call Number#3109], [Call_Number#3038],
```

3 Engineering Data Pipelines

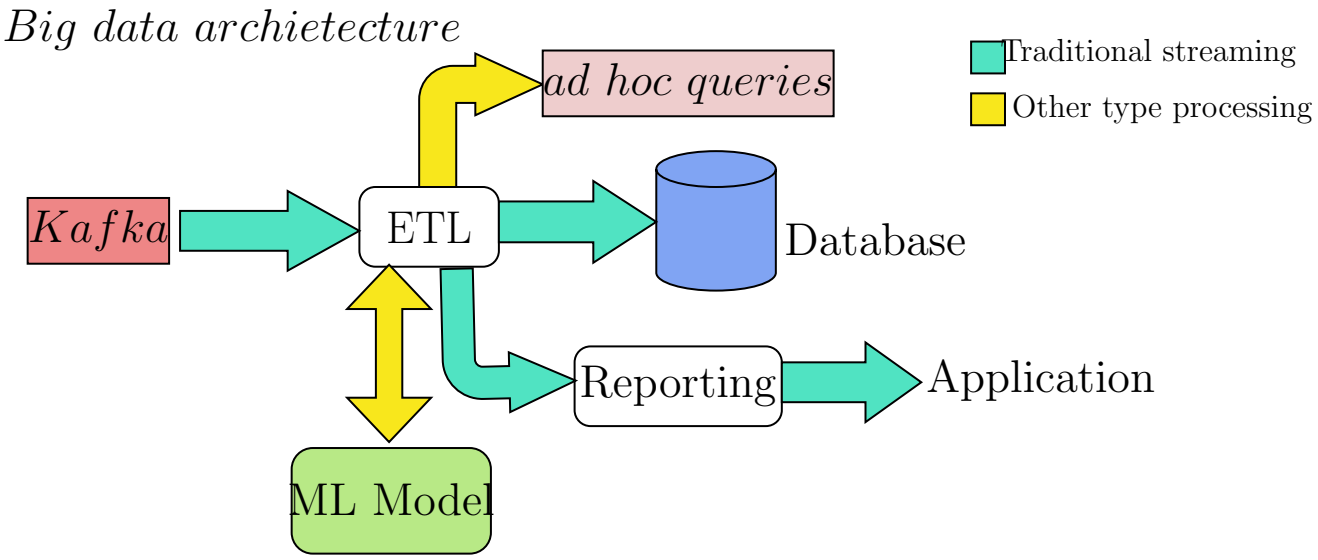

Figure 1: Big Data Archietecture

A data application needs a scalable way to receive high volume, velocity and variety of data. This means that when data arrives in the system, it won't overload other aspects of our infrastructure. This piece of our infrastructure operates like a queue. The most common solution for this queue is the technology Apache Kafka, another open source project.

In the paradigm above, producers of data published to topics. Maybe those producers are a number of weather sensors that are all writing raw weather data to the same weather topic. There can be an arbitrary number of producers and an arbitrary number of topics. Consumers then read data from the topic in order to make use of it. This makes sure that data can arrive in the system when we need it to, and when we need to take in different data we can just add more topics.

In ETL, we might take that raw weather data, pull out the fields we need and insert it into a database. That database will then serve the data maybe to visitors of a website. We'd also want to store the raw data in case we ever need to return to it. This can be done using a data lake normally backed by a blob store such as Amazon S3 or the Azure Blob.

Spark offers an optimized Compute Engine that connects the data where it lifts.By spinning up a Spark cluster when it's needed, connecting to data where it lives and shutting it down afterwards, not only can you save resources but you don't have to worry about updating your cluster to the latest version of Spark.

Generally speaking, data transfer is the bottleneck for most tasks in a big data environment. These IO bound problems demand attention to how the data is stored and read into the cluster.

Finally, there are number of different file types.Most of these cases you've seen in this series of courses have pertained to the domain of OLAP or Online Analytical Processing.OLAP workloads focus on reporting and Ad hoc analysis or data exploration where an analyst is exploring the data that's available to him or her. OLTP is online transaction processing. This involves some sort of transaction executed against a database. For instance, a visitor to a website might update their profile which would trigger a transaction made against the database. The main difference between OLAP and OLTP workloads is whether the work needs to occur in real time. While most of an analyst's responsibilities fall under the category of OLAP, work with Spark is used for OLTP as well.

3.1 Accessing Data

BLOB stores are a common way of storing large amounts of data. JDBC is an application programming interface or API for Java environments. This allows us to query most databases. Spark works well with JDBC, including a functionality called predicate push down. We're aspects of SQL queries such as filter operations, can be done by the database itself transferring that data into the Spark cluster.

△ Question
How to mount data bucket from AWS?

```
//First, define the credentials:
%python
ACCESS_KEY = "AKIAJBRYNXGHORDHZB4A"
# Encode the Secret Key to remove any "/" characters
SECRET_KEY = "a0BzE1bSegfydr3%2FGE3LSPM6uIV5A4hOUfpH8aFF".replace("/", "%2F")
AWS_BUCKET_NAME = "davis-dsv1071/data/"
MOUNT_NAME = "/mnt/davis-tmp"

//we then use the following command to mount the data in the workspace.
%python
try:
  dbutils.fs.mount("s3a://{}:{}@{}".format(ACCESS_KEY, SECRET_KEY, AWS_BUCKET_NAME),
      MOUNT_NAME)
except:
  print("""{} already mounted. Unmount using 'dbutils.fs.unmount("{}")' to unmount first"
      "".format(MOUNT_NAME, MOUNT_NAME))
```

After a cell used to mount a bucket is run, you can access the data in this mount point in any notebook or any cluster in your service provider, and share the mount between colleagues.

△ Question
How to get the name of mountpoint?

```
%fs mounts
```

△ Question
How to unmount a mountpoint?

```
%python
dbutils.fs.unmount("/mount/point/path")
```

3.1.1 Serial JDBC Reads

To leverage the inherent efficiencies of database engines, Spark uses an optimization called predicate push down. Predicate push down uses the database itself to handle certain parts of a query (the predicates). In mathematics and functional programming, a predicate is anything that returns a Boolean. In SQL terms, this often refers to the WHERE clause.

The following query creates a db called "twitterJDBC" from a table called "Account"

```
CREATE TABLE IF NOT EXISTS twitterJDBC
USING org.apache.spark.sql.jdbc
OPTIONS (
  driver "org.postgresql.Driver",
  url "jdbc:postgresql://server1.databricks.training:5432/training",
  user "readonly",
  password "readonly",
  dbtable "Account"
)
```

Now, Add a subquery to dbtable, which pushes the predicate to JDBC to process before transferring the data to your Spark cluster.

```
CREATE TABLE IF NOT EXISTS twitterPhilippinesJDBC
USING org.apache.spark.sql.jdbc
OPTIONS (
  driver "org.postgresql.Driver",
  url "jdbc:postgresql://server1.databricks.training:5432/training",
  user "readonly",
  password "readonly",
  dbtable "(SELECT * FROM Account WHERE location = 'Philippines') as subq"
)
```

3.1.2 Parallel JDBC Reads

The following commands partition the column "userID" into 25 parts, and read data in a parallel manner.

```
DROP TABLE IF EXISTS twitterParallelJDBC;

CREATE TABLE IF NOT EXISTS twitterParallelJDBC
USING org.apache.spark.sql.jdbc
OPTIONS (
  driver "org.postgresql.Driver",
```

```
  url "jdbc:postgresql://server1.databricks.training:5432/training",
  user "readonly",
  password "readonly",
  dbtable "Account",
  partitionColumn '"userID"',
  lowerBound 2591,
  upperBound 951253910555168768,
  numPartitions 25
)
```

Note taht the lowerbound and upperbount of "userID" are also specified. Remember to use '"user ID"' to make accessing process case sensitive.

Now, the following time counting tells that the parralell reading is indeed faster:

```
%python
%timeit sql("SELECT * from twitterJDBC").describe()
%python
%timeit sql("SELECT * from twitterParallelJDBC").describe()
```

3.2 File Formats

Parquet format is a column-based format to store data, which has performance boosted when they are read on Spark. This is for a number of reasons. One is because it's able to compress our data so we have less data actually moving across the network. It also stores enough of the metadata so that we don't have to worry about the schema inference which was part of the big cost associated with reading this data.

3.3 Schemas and Types

△ Question
How to create table using user-defined schema from JSON file?

```
CREATE OR REPLACE TEMPORARY VIEW fireCallsJSON (
  `Call Number` INT,
  `Unit ID` STRING,
  `Incident Number` INT,
  `Call Type` STRING,
  `Call Date` STRING,
  `Watch Date` STRING,
  `Received DtTm` STRING,
  `Entry DtTm` STRING,
  `Dispatch DtTm` STRING,
  `Response DtTm` STRING,
  `On Scene DtTm` STRING,
  `Transport DtTm` STRING,
  `Hospital DtTm` STRING,
  `Call Final Disposition` STRING,
  `Available DtTm` STRING,
```

```
  `Address` STRING,
  `City` STRING,
  `Zipcode of Incident` INT,
  `Battalion` STRING,
  `Station Area` STRING,
  `Box` STRING,
  `Original Priority` STRING,
  `Priority` STRING,
  `Final Priority` INT,
  `ALS Unit` BOOLEAN,
  `Call Type Group` STRING,
  `Number of Alarms` INT,
  `Unit Type` STRING,
  `Unit sequence in call dispatch` INT,
  `Fire Prevention District` STRING,
  `Supervisor District` STRING,
  `Neighborhooods - Analysis Boundaries` STRING,
  `Location` STRING,
  `RowID` STRING
)
USING JSON
OPTIONS (
    path "/mnt/davis/fire-calls/fire-calls-truncated.json"
)
//Notice that by using user-defined schema, we can build table faster.
```

3.4 Writing data

Two core concepts are: A partition is a portion of your total data set, which is divided into many of these portions so Spark can distribute your work across a cluster.

The other concept needed to understand Spark's computation is a slot (also known as a core). A slot/core is a resource available for the execution of computation in parallel. **In brief, a partition refers to the distribution of data while a slot refers to the distribution of computation.**

> **△ Question**
> How to write dataframe in Python?

```
%python
df.write.mode("OVERWRITE").csv(username + "/filename.csv")
```

> **△ Question**
> Check the file you already write?

```
%python
dbutils.fs.ls(username + "/filename.csv")
//It can be seen that the data is stored in different parts, one for each partition.
```

Δ Question
How to check the number of partitions you just write?

```
//In python
%python
df.rdd.getNumPartitions()
```

3.4.1 Change number of partitions when write data

Hint	Transformation type	Use	Evenly distributes data across partitions?
SELECT /*+ COALESCE(n) */	narrow (does not shuffle data)	reduce the number of partitions	No
SELECT /*+ REPARTITION(n) */	wide (includes a shuffle operation)	increase the number of partitions	Yes

Table 1: Controlling concurrency

Δ Question
Examples of using REPARTITION/COALESCE?

```
CREATE OR REPLACE TEMPORARY VIEW fireCallsCSV1p
  AS
SELECT /*+ COALESCE(1) */ *
FROM fireCallsCSV
//If we run sql("SELECT * FROM fireCallsCSV8p").rdd.getNumPartitions(), the result should
    be 1

CREATE OR REPLACE TEMPORARY VIEW fireCallsCSV8p
  AS
SELECT /*+ REPARTITION(12) */ *
FROM fireCallsCSV

//If we run sql("SELECT * FROM fireCallsCSV8p").rdd.getNumPartitions(), the result should
    be 12
```

3.5 Managed and Unmanaged(external) Table

Data on the Spark cluster disappears when the cluster is terminated, unless it saved elsewhere.

Managed table manages metadata and tables, if we drop that table, we'll remove both the metadata for the table as well as the data itself.

Unmanaged tables perform a little bit differently. Unmanaged tables manage the metadata, but the data itself is sitting in a different location, often backed by blob stores like the Azure Blob or S3. In this case, Spark is not going to delete that table when we perform a drop table operation.

Δ Question
How to create managed/unmanaged table?

```
USE default;

CREATE TABLE IF NOT EXISTS tableManaged (
  var1 INT,
  var2 INT
);

INSERT INTO tableManaged
  VALUES (1, 1), (2, 2)

//Now, if we run the following command:
DESCRIBE EXTENDED tableManaged
//we found that the location of the table is: dbfs:/user/hive/warehouse/tablemanaged, and
    its type is "EXTERNAL".

//Now, we create an unmanaged table:
CREATE EXTERNAL TABLE IF NOT EXISTS tableUnmanaged (
  var1 INT,
  var2 INT
)
LOCATION '/tmp/unmanagedTable'
```

If we drop managed table by using "DROP TABLE", we can no longer access the information of the table. On the other hand, the data from unmanagedtable will still be in the warehouse.

4 Machine Learning Applications of Spark

```
//Objectives: Preprocess data for use in a machine learning model
//Step through creating a sklearn logistic regression model for classification
//Predict the Call Type Group for incidents in a SQL table

//Load the mnt/davis/fire-calls/fire-calls-clean.parquet data as fireCallsClean table.
USE DATABRICKS;
CREATE TABLE IF NOT EXISTS fireCallsClean
USING parquet
OPTIONS (
  path "mnt/davis/fire-calls/fire-calls-clean.parquett"
)
//How many calls of Call_Type_Group "Fire"?
SELECT COUNT(`Call_Type_Group`) AS Group_Count, `Call_Type_Group`
FROM fireCallsClean
GROUP BY `Call_Type_Group`
ORDER BY Group_Count DESC

//Let's drop all the rows where Call_Type_Group = null. Since we don't have a lot of
   Call_Type_Group with the value Alarm and Fire, we will also drop these calls from the
```

```
   table. Call this new table fireCallsGroupCleaned.

CREATE TABLE IF NOT EXISTS fireCallsGroupCleaned AS (
  SELECT *
  FROM fireCallsClean
  WHERE `Call_Type_Group` IN ("Potentially Life-Threatening", "Non Life-threatening")
)

//We probably don't need all the columns of fireCallsGroupCleaned to make our prediction.
    Select the following columns from fireCallsGroupCleaned and create a view called
   fireCallsDF so we can access this table in Python

DROP VIEW IF EXISTS fireCallsDF;
CREATE OR REPLACE TEMPORARY VIEW fireCallsDF AS (
SELECT `Call_Type`,`Fire_Prevention_District`, `Neighborhooods_-_Analysis_Boundaries`,`
   Number_of_Alarms`,`Original_Priority`,
`Unit_Type`,`Battalion`, `Call_Type_Group`
FROM fireCallsGroupCleaned
)

//Fill in the string SQL statement to load the fireCallsDF table you just created into
   python.

%python
# # TODO
df = sql("""SELECT * FROM fireCallsDF""")
display(df)

//Creating a Logistic Regression Model in Sklearn
%python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

pdDF = df.toPandas()
le = LabelEncoder()
numerical_pdDF = pdDF.apply(le.fit_transform)

X = numerical_pdDF.drop("Call_Type_Group", axis=1)
y = numerical_pdDF["Call_Type_Group"].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

%python
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
```

```
ohe = ("ohe", OneHotEncoder(handle_unknown="ignore"))
lr = ("lr", LogisticRegression())

pipeline = Pipeline(steps = [ohe, lr]).fit(X_train, y_train)
y_pred = pipeline.predict(X_test)

%python
from sklearn.metrics import accuracy_score
print(f"Accuracy of model: {accuracy_score(y_pred, y_test)}")
%python
import mlflow
from mlflow.sklearn import save_model

model_path = "/dbfs/" + username + "/Call_Type_Group_lr"
dbutils.fs.rm(username + "/Call_Type_Group_lr", recurse=True)
save_model(pipeline, model_path)

//Now that we have created and trained a machine learning pipeline, we will use MLflow to
    register the .predict function of the sklearn pipeline as a UDF which we can use
   later to apply in parallel. Now we can refer to this with the name predictUDF in SQL.

%python
import mlflow
from mlflow.pyfunc import spark_udf

predict = spark_udf(spark, model_path, result_type="int")
spark.udf.register("predictUDF", predict)

//Create a view called testTable of our test data X_test so that we can see this table in
    SQL.

%python
X_test.to_csv("/dbfs/" + username + "/Call_Type_Group_lr"+ "/modeltest.csv")
dbutils.fs.ls("/liu_si_zhe@outlook.com/Call_Type_Group_lr/")

CREATE OR REPLACE TEMPORARY VIEW testTable
USING csv
OPTIONS (
  header "true",
  path "/liu_si_zhe@outlook.com/Call_Type_Group_lr/modeltest.csv",
  inferSchema "true"
)

//Create a table called predictions using the predictUDF function we registered
   beforehand. Apply the predictUDF to every row of testTable in parallel so that each
```

```
   row of testTable has a Call_Type_Group prediction.

USE DATABRICKS;
DROP TABLE IF EXISTS predictions;

CREATE TEMPORARY VIEW predictions AS (
  SELECT cast(predictUDF(Call_Type,Fire_Prevention_District, `Neighborhooods_-
     _Analysis_Boundaries`,Number_of_Alarms,Original_Priority,
  Unit_Type,Battalion) as double) as Call_Type_Group, *
  FROM testTable
)
```

5 Useful link at Databricks

Utilities
Python with Spark