Big Data Analytics

Dr Sirintra Vaiwsri | Email: sirintra.v@itm.kmutnb.ac.th

Structured API Execution

Steps (Chambers and Zaharia, 2018)

1. Write DataFrame/SQL Code
2. Spark converts code to a Logical Plan if the code is valid.
3. Spark transforms the Logical Plan to a Physical Plan.
4. Spark executes the Physical Plan on a cluster.

Logical Plan (Chambers and Zaharia, 2018)

1. The code is first converted to an unresolved logical plan.
2. Spark then uses a “catalogue” to resolve tables and columns in the analyser.
3. The analyser rejects an unresolved logical plan if the referred tables do not exist, otherwise, it accepts which becomes a resolved logical plan.
4. Spark passes the resolved logical plan through the catalyst optimiser to optimise the resolved logical plan resulting in the “optimised logical plan”

Physical Plan (Chambers and Zaharia, 2018)

- Physical plan, also called Spark plan, generates different physical executions and compares them through a cost model.
- It can manage the execution of an optimised logical plan on a cluster.
- It results in RDDs and their transformations, in other words, the Physical Plan runs the code over the RDDs.

Dataframes

Dataframes

(Antolínez García, 2023; Chambers and Zaharia, 2018)

- Dataframes allow Spark to efficiently conduct shuffling by moving data across nodes.
- It represents a table of data containing rows and columns.
- Schema includes column name, data type, and nullable flag.

```
root
 |-- status_id: string (nullable = true)
 |-- status_type: string (nullable = true)
 |-- status_published: string (nullable = true)
 |-- num_reactions: string (nullable = true)
 |-- num_comments: string (nullable = true)
 |-- num_shares: string (nullable = true)
 |-- num_likes: string (nullable = true)
 |-- num_loves: string (nullable = true)
 |-- num_wows: string (nullable = true)
 |-- num_hahas: string (nullable = true)
 |-- num_sads: string (nullable = true)
 |-- num_angrys: string (nullable = true)
```

Dataframes (Antolínez García, 2023)

- Dataframes can be distributed across nodes to support distributed computing architecture.
- Users can query using SQL which is then sent to Dataframes and managed by the Catalyst Optimiser.
- Catalyst Optimiser is responsible for building query execution.
- Dataframes can be created from external data sources such as CSV and JSON files.

Dataframes

```
from pyspark.sql import SparkSession
# Create Spark Session
read_file = spark.read.format\
    (<FILE FORMAT>).option("header",<True/False>).\
    load(<FILE NAME>) # Read file
read_file.printSchema() # Print schema
```

```
root
 |-- status_id: string (nullable = true)
 |-- status_type: string (nullable = true)
 |-- status_published: string (nullable = true)
 |-- num_reactions: string (nullable = true)
 |-- num_comments: string (nullable = true)
 |-- num_shares: string (nullable = true)
 |-- num_likes: string (nullable = true)
 |-- num_loves: string (nullable = true)
 |-- num_wows: string (nullable = true)
 |-- num_hahas: string (nullable = true)
 |-- num_sads: string (nullable = true)
 |-- num_angrys: string (nullable = true)
```

Spark SQL (Antolínez García, 2023)

- Spark SQL can query data directly from file.

```
sqlDF = read_File.select(read_file.<column1>,\
    read_file.<column2>) # SQL query
sqlDF.show(<Row Numbers>) # Show result
```

Spark SQL (Antolínez García, 2023)

- Spark SQL query data directly from file with **where** clause.

```
from pyspark.sql.types import *
sqlDF = read_File.select(read_file.<column name1>,\
    read_file.<column name 2>).\
    where(read_file.num_reactions.\
    cast(IntegerType()) > <some numbers>).\
    withColumnRenamed(<old name>, \
    <new name>).\
    orderBy(readfile.<column name>) # SQL query
sqlDF.show(<Row Numbers>) # Show result
```

```
+--------------------+---------+
|           status_id|Reactions|
+--------------------+---------+
|246675545449582_7...|     3115|
|246675545449582_7...|     3190|
|246675545449582_7...|     3510|
+--------------------+---------+
only showing top 3 rows
```

Spark SQL (Antolínez García, 2023)

- Spark SQL can conduct directly over the file through temporary views by using `createOrReplaceTempView(<temp name>)`.

```
read_file.createOrReplaceTempView(<temp name>) # Create
                                               # temporary view
sqlDF = spark.sql("select * from <temp name>") # SQL query
sqlDF.show(<Row Numbers>) # Show result
```

status_id	status_type	status_published	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	num_angrys
246675545449582_1...	video	4/22/2018 6:00	529	512	262	432	92	3	1	1	0
246675545449582_1...	photo	4/21/2018 22:45	150	0	0	150	0	0	0	0	0
246675545449582_1...	video	4/21/2018 6:17	227	236	57	204	21	1	1	0	0

only showing top 3 rows

Save Modes (Antolínez García, 2023)

- Data can be saved into a file.
- Save Modes are:
  - errorifexists or error - exception is sent if data already exists
  - append - data is appended to the destination
  - overwrite - data is overwritten if data already exists
  - ignore - data will be ignored if data already exists

```
sqlDF.write.mode(“overwrite”).csv(“<folder>”) # Save data
```

Random Split (Chambers and Zaharia, 2018)

- randomSplit is used for splitting data in a Dataframe.
- It is useful when creating training and test sets are required such as when working with machine learning.

```
split = sqlDF.randomSplit([number1, number2]) # Split data where numbers 1 and 2 are sum to 1.0.
split[0].show() # Show data in the first set (number1)
split[1].show() # Show data in the second set (number2)
```

Aggregations

Aggregations (Chambers and Zaharia, 2018)

- Aggregation collects data together and produces one result for each group.
- Aggregations are available as functions such as count, first and last, min and max, sum, etc.

fb_live_thailand2 and fb_live_thailand3

```
read_file = spark.read.format("csv").\
    option("header", True).\
    load("data/*.csv") # Multiple files
# Create a temporary view
# Select all from the temporary view
# Show result
```

Aggregation Functions (Chambers and Zaharia, 2018)

- count is used for counting rows.

```
from pyspark.sql.functions import count
# Select all from the temporary view
sqlDF.select(count(“<column name>”)) # Select count

sqlDF = spark.sql(“select * from tempTable where \
       tempTable.status_type == ‘photo’”) # Select...where
sqlDF.select(count(“status_published”)) # Select count
```

```
+-----------------------+
|count(status_published)|
+-----------------------+
|                     22|
+-----------------------+
```

Aggregation Functions (Chambers and Zaharia, 2018)

- countDistinct is used for counting number of unique groups.

```
from pyspark.sql.functions import countDistinct
# Select all from the temporary view
sqlDF.select(countDistinct(“<column name>”)) # Select
                                             #countDistinct
```

```
+----------------------------+
|count(DISTINCT status_type)|
+----------------------------+
|                           2|
+----------------------------+
```

Aggregation Functions (Chambers and Zaharia, 2018)

- first and last are used to retrieve the first and last rows from a Dataframe.

```
from pyspark.sql.functions import first, last
# Select all from the temporary view
sqlDF.select(first(“<column name>”), \
       last(“<column name>”)) # Select first and last
```

```
+-----------------------+----------------------+
|first(status_published)|last(status_published)|
+-----------------------+----------------------+
|          4/22/2018 6:00|         3/23/2018 7:09|
+-----------------------+----------------------+
```

Aggregation Functions (Chambers and Zaharia, 2018)

- min and max are used for finding the minimum and maximum values from a Dataframe.

```
from pyspark.sql.functions import min, max
from pyspark.sql.types import *
# Select all from the temporary view
sqlDF = sqlDF.withColumn('<new column name>', \
        sqlDF['<old column name>']).\
        cast(IntegerType())) # Convert the column to integer
                                # then add as a new column name
sqlDF.select(min("<new column name>"), \
        max("<new column name>")) # Select min and max
```

min(reactions)	max(reactions)
18	529

Aggregation Functions (Chambers and Zaharia, 2018)

- sum and sumDistinct are used for summing a total where sumDistinct sums a distinct set of values.

```
from pyspark.sql.functions import sum, sumDistinct
# Select all from the temporary view
# Convert the column to an integer with the new column name
sqlDF.select(sum(“<new column name>”)) # Select sum
sqlDF.select(sumDistinct(“<new column name>”)) # Select sum distinct
```

```
+--------------+
|sum(reactions)|
+--------------+
|          8382|
+--------------+

+-----------------------+
|sum(DISTINCT reactions)|
+-----------------------+
|                   5557|
+-----------------------+
```


Dr Sirintra Vaiwsri

Aggregation Functions (Chambers and Zaharia, 2018)

- avg (average) is used for calculating an average value of a column in a Dataframe.

```
from pyspark.sql.functions import avg
# Select all from temporary view
# Convert the column to integer with the new column name
sqlDF.select(avg("<new column name>")) # Select average
```

Joins

Joins (Chambers and Zaharia, 2018)

- Join combines two sets of data based on the key of the left and right datasets (tables).
- Spark discards the unmatched rows and then returns matched rows.
- Join evaluates the result using a join expression.

```
join_column = sqlDF1[“<column name>”] == \
              sqlDF2[“<column name>”] # Identify column
                                      # for matching
```

Joins (Chambers and Zaharia, 2018)

- Inner join keeps rows with keys matched in the left and right datasets.

```
sqlDF1.join(sqlDF2, join_column).show() # Inner join and
                                        # show results


joinType = “inner”
sqlDF1.join(sqlDF2, join_column, joinType).show() # Inner
                                # join and show results
                                # with using joinType
```

Joins (Chambers and Zaharia, 2018)

- Inner join example result:

Table	status_id	status_type	status_published	num_reactions	num_comments	Table	status_id	status_type	status_published	num_reactions	num_shares
FB2	246675545449582_1...	video	4/22/2018 6:00	529	512	FB3	246675545449582_1...	video	4/22/2018 6:00	529	262
FB2	246675545449582_1...	photo	4/21/2018 22:45	150	0	FB3	246675545449582_1...	photo	4/21/2018 22:45	150	0
FB2	246675545449582_1...	video	4/21/2018 6:17	227	236	FB3	246675545449582_1...	video	4/21/2018 6:17	227	57
FB2	246675545449582_1...	photo	4/21/2018 2:29	111	0	FB3	246675545449582_1...	photo	4/21/2018 2:29	111	0
FB2	246675545449582_1...	photo	4/18/2018 3:22	213	0	FB3	246675545449582_1...	photo	4/18/2018 3:22	213	0
FB2	246675545449582_1...	photo	4/18/2018 2:14	217	6	FB3	246675545449582_1...	photo	4/18/2018 2:14	217	0
FB2	246675545449582_1...	video	4/18/2018 0:24	503	614	FB3	246675545449582_1...	video	4/18/2018 0:24	503	72
FB2	246675545449582_1...	video	4/17/2018 7:42	295	453	FB3	246675545449582_1...	video	4/17/2018 7:42	295	53
FB2	246675545449582_1...	photo	4/17/2018 3:33	203	1	FB3	246675545449582_1...	photo	4/17/2018 3:33	203	0

Joins (Chambers and Zaharia, 2018)

- Outer join keeps rows with keys in either the left or right datasets.

```
joinType = “outer”
sqlDF1.join(sqlDF2, join_column, joinType).show()
```

Table	status_id	status_type	status_published	num_reactions	num_comments	Table	status_id	status_type	status_published	num_reactions	num_shares
FB2	246675545449582_1...	photo	3/12/2018 5:51	145	9	null	null	null	null	null	null
FB2	246675545449582_1...	video	3/17/2018 7:47	90	78	null	null	null	null	null	null
FB2	246675545449582_1...	video	3/17/2018 8:07	75	36	null	null	null	null	null	null
FB2	246675545449582_1...	video	3/19/2018 22:34	37	0	null	null	null	null	null	null
FB2	246675545449582_1...	photo	3/20/2018 0:15	102	0	null	null	null	null	null	null
FB2	246675545449582_1...	video	3/20/2018 1:28	18	0	null	null	null	null	null	null
FB2	246675545449582_1...	photo	3/20/2018 1:54	98	0	null	null	null	null	null	null
FB2	246675545449582_1...	photo	3/21/2018 7:46	227	7	null	null	null	null	null	null
FB2	246675545449582_1...	photo	3/21/2018 8:40	234	15	null	null	null	null	null	null
FB2	246675545449582_1...	photo	3/22/2018 1:25	152	2	null	null	null	null	null	null
null	null	null	null	null	null	FB3	246675545449582_1...	video	3/23/2018 7:09	221	36
null	null	null	null	null	null	FB3	246675545449582_1...	video	3/26/2018 8:28	150	47
null	null	null	null	null	null	FB3	246675545449582_1...	video	3/30/2018 8:28	135	79
null	null	null	null	null	null	FB3	246675545449582_1...	video	4/1/2018 5:16	332	30
null	null	null	null	null	null	FB3	246675545449582_1...	photo	4/5/2018 9:23	346	0
null	null	null	null	null	null	FB3	246675545449582_1...	photo	4/8/2018 2:23	209	0
null	null	null	null	null	null	FB3	246675545449582_1...	photo	4/8/2018 5:10	313	2
null	null	null	null	null	null	FB3	246675545449582_1...	photo	4/9/2018 2:06	222	0
null	null	null	null	null	null	FB3	246675545449582_1...	photo	4/10/2018 1:01	210	3
null	null	null	null	null	null	FB3	246675545449582_1...	photo	4/11/2018 4:53	170	1

Joins (Chambers and Zaharia, 2018)

- Left outer join keeps rows with keys in the left dataset.
- Right outer join keeps rows with keys in the right dataset.

```
joinTypeLeft = “left_outer” # Left outer join
sqlDF1.join(sqlDF2, join_column, joinTypeLeft).show()
joinTypeRight = “right_outer” # Right outer join
sqlDF1.join(sqlDF2, join_column, joinTypeRight).show()
```

Left Outer Join Example Result

Table	status_id	status_type	status_published	num_reactions	num_comments	Table	status_id	status_type	status_published	num_reactions	num_shares
FB2	246675545449582_1...	video	4/22/2018 6:00	529	512	FB3	246675545449582_1...	video	4/22/2018 6:00	529	262
FB2	246675545449582_1...	photo	4/21/2018 22:45	150	0	FB3	246675545449582_1...	photo	4/21/2018 22:45	150	0
FB2	246675545449582_1...	video	4/21/2018 6:17	227	236	FB3	246675545449582_1...	video	4/21/2018 6:17	227	57
FB2	246675545449582_1...	photo	4/21/2018 2:29	111	0	FB3	246675545449582_1...	photo	4/21/2018 2:29	111	0
FB2	246675545449582_1...	photo	4/18/2018 3:22	213	0	FB3	246675545449582_1...	photo	4/18/2018 3:22	213	0
FB2	246675545449582_1...	photo	4/18/2018 2:14	217	6	FB3	246675545449582_1...	photo	4/18/2018 2:14	217	0
FB2	246675545449582_1...	video	4/18/2018 0:24	503	614	FB3	246675545449582_1...	video	4/18/2018 0:24	503	72
FB2	246675545449582_1...	video	4/17/2018 7:42	295	453	FB3	246675545449582_1...	video	4/17/2018 7:42	295	53
FB2	246675545449582_1...	photo	4/17/2018 3:33	203	1	FB3	246675545449582_1...	photo	4/17/2018 3:33	203	0
FB2	246675545449582_1...	photo	3/22/2018 1:25	152	2	null	null	null	null	null	null
FB2	246675545449582_1...	photo	3/21/2018 8:40	234	15	null	null	null	null	null	null
FB2	246675545449582_1...	photo	3/21/2018 7:46	227	7	null	null	null	null	null	null
FB2	246675545449582_1...	photo	3/20/2018 1:54	98	0	null	null	null	null	null	null
FB2	246675545449582_1...	video	3/20/2018 1:28	18	0	null	null	null	null	null	null
FB2	246675545449582_1...	photo	3/20/2018 0:15	102	0	null	null	null	null	null	null
FB2	246675545449582_1...	video	3/19/2018 22:34	37	0	null	null	null	null	null	null
FB2	246675545449582_1...	video	3/17/2018 8:07	75	36	null	null	null	null	null	null
FB2	246675545449582_1...	video	3/17/2018 7:47	90	78	null	null	null	null	null	null
FB2	246675545449582_1...	photo	3/12/2018 5:51	145	9	null	null	null	null	null	null

Right Outer Join Example Result

Table	status_id	status_type	status_published	num_reactions	num_comments	Table	status_id	status_type	status_published	num_reactions	num_shares
FB2	246675545449582_1...	video	4/22/2018 6:00	529	512	FB3	246675545449582_1...	video	4/22/2018 6:00	529	262
FB2	246675545449582_1...	photo	4/21/2018 22:45	150	0	FB3	246675545449582_1...	photo	4/21/2018 22:45	150	0
FB2	246675545449582_1...	video	4/21/2018 6:17	227	236	FB3	246675545449582_1...	video	4/21/2018 6:17	227	57
FB2	246675545449582_1...	photo	4/21/2018 2:29	111	0	FB3	246675545449582_1...	photo	4/21/2018 2:29	111	0
FB2	246675545449582_1...	photo	4/18/2018 3:22	213	0	FB3	246675545449582_1...	photo	4/18/2018 3:22	213	0
FB2	246675545449582_1...	photo	4/18/2018 2:14	217	6	FB3	246675545449582_1...	photo	4/18/2018 2:14	217	0
FB2	246675545449582_1...	video	4/18/2018 0:24	503	614	FB3	246675545449582_1...	video	4/18/2018 0:24	503	72
FB2	246675545449582_1...	video	4/17/2018 7:42	295	453	FB3	246675545449582_1...	video	4/17/2018 7:42	295	53
FB2	246675545449582_1...	photo	4/17/2018 3:33	203	1	FB3	246675545449582_1...	photo	4/17/2018 3:33	203	0
null	null	null	null	null	null	FB3	246675545449582_1...	photo	4/11/2018 4:53	170	1
null	null	null	null	null	null	FB3	246675545449582_1...	photo	4/10/2018 1:01	210	3
null	null	null	null	null	null	FB3	246675545449582_1...	photo	4/9/2018 2:06	222	0
null	null	null	null	null	null	FB3	246675545449582_1...	photo	4/8/2018 5:10	313	2
null	null	null	null	null	null	FB3	246675545449582_1...	photo	4/8/2018 2:23	209	0
null	null	null	null	null	null	FB3	246675545449582_1...	photo	4/5/2018 9:23	346	0
null	null	null	null	null	null	FB3	246675545449582_1...	video	4/1/2018 5:16	332	30
null	null	null	null	null	null	FB3	246675545449582_1...	video	3/30/2018 8:28	135	79
null	null	null	null	null	null	FB3	246675545449582_1...	video	3/26/2018 8:28	150	47
null	null	null	null	null	null	FB3	246675545449582_1...	video	3/23/2018 7:09	221	36

Assignment (1 point)

- Please implement the codes in slides 9 to 28 in one file and show the results to get 1 point.

References

- Chambers, B., & Zaharia, M. (2018). *Spark: The definitive guide: Big data processing made simple.* " O'Reilly Media, Inc.".
- Antolínez García, A. (2023). *Hands-on Guide to Apache Spark 3: Build Scalable Computing Engines for Batch and Stream Data Processing*. Berkeley, CA: Apress.