FIT5202 Data processing for Big data¶

Activity: Parallel Joins in Spark DataFrames¶

For this tutorial we will look at the high level join strategies used in Spark while performing join operation. We will try to understand the interal working of these strategies by examining the query execution plan and the graphical plan provided in Spark UI. We will then look into various other basic join algorithms used by Spark.

Let's get started.

Table of Contents¶

Import Spark classes and create Spark Context¶

TODO: In the cell block below, initialize the spark session named as **spark** and create the `SparkContext` names **sc** from that Spark Session.

Important: You cannot proceed to other steps without completing this.

In [1]:

```
# Import SparkConf class into program
from pyspark import SparkConf

# local[*]: run Spark in local mode with as many working processors as logical cores
on your machine
# If we want Spark to run locally with 'k' worker threads, we can specify as
"local[k]".
master = "local[*]"
# The `appName` field is a name to be shown on the Spark cluster UI page
app_name = "Parallel Join"
# Setup configuration parameters for Spark
spark_conf = SparkConf().setMaster(master).setAppName(app_name)

# Import SparkSession classes
from pyspark.sql import SparkSession # Spark SQL

#TODO : Initialize Spark Session and create a SparkContext Object
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-1-a999ef3f3d13> in <module>
      1 # Import SparkConf class into program
----> 2 from pyspark import SparkConf
      3
```

```
      4 # local[*]: run Spark in local mode with as many working processors as logical cores on your machine
      5 # If we want Spark to run locally with 'k' worker threads, we can specify as "local[k]".

ModuleNotFoundError: No module named 'pyspark'
```

Parallel Join Strategies¶

Creating datasets (i.e. dataframes)¶

In the code below, we are creating two datasets with a common key "id". When we are joining two tables, we need at least 1 common key.

In [ ]:

```
###### Setting dataset
import random
random.seed(0)

# List of tuples
tableA = [(i,'A'+str(i)) for i in range(100,110)]
tableB = [(i,'B'+str(i)) for i in range(10,1000)]
# Shuffle the lists to not have it ordered
random.shuffle(tableA)
random.shuffle(tableB)

# Converting to dataframe each list of tuples
df_A = spark.createDataFrame(tableA , ["id", "valueA"])
df_A.show()
df_B = spark.createDataFrame(tableB , ["id", "valueB"])
df_B.show()
```


1. Broadcast Hash Join¶

In this type of join, one dataset(the smaller one) is broadcasted (sent over) to each executor. By doing this, we can avoid the shuffle for the other larger dataset. Not doing the shuffle increase the speed of the join operation.

We need to use the broadcast function inside the join to broadcast the table

In [ ]:

```
from pyspark.sql.functions import broadcast

# Use broadcast function to specify the use of BroadcastHashJoin algorithm
df_joined_broadcast = df_B.join(broadcast(df_A),df_A.id==df_B.id,how='inner')
df_joined_broadcast.show()
```

Query execution plan¶

In [ ]:

```
## Show execution plan using function explain()
df_joined_broadcast.explain()
```

Explanation Query Plan with Broadcast Hash Join¶

The order of execution goes from top to bottom. The steps are:

1. Scan dataframe A (left side)
    - Filter id not null in dataframe A
2. Scan dataframe B (right side)
    - Filter id not null in dataframe B
3. Broadcast dataframe B: Send dataframe B to each each partition
4. BroadcastHashJoin: Perform join between each partition and the broadcasted dataframe B
5. Project: Select the attributes from both dataframes (df_A: id,valueA and df_b: id,valueB)

6. Collect all the results to the driver

Graphical Execution plan in Spark UI¶

Go to your **Spark UI** and Click on the **SQL** tab to view the graphical equivalent of the above physical plan.

2. Sort Merge Join¶

In this join approach, the datasets are sorted first and the second operation merges the sorted data in the partition. This is the **default** join algorithm used by spark.

In [ ]:

```
df_joined_sortmerge = df_A.join(df_B,df_A.id==df_B.id,how='inner')
df_joined_sortmerge.show()
```

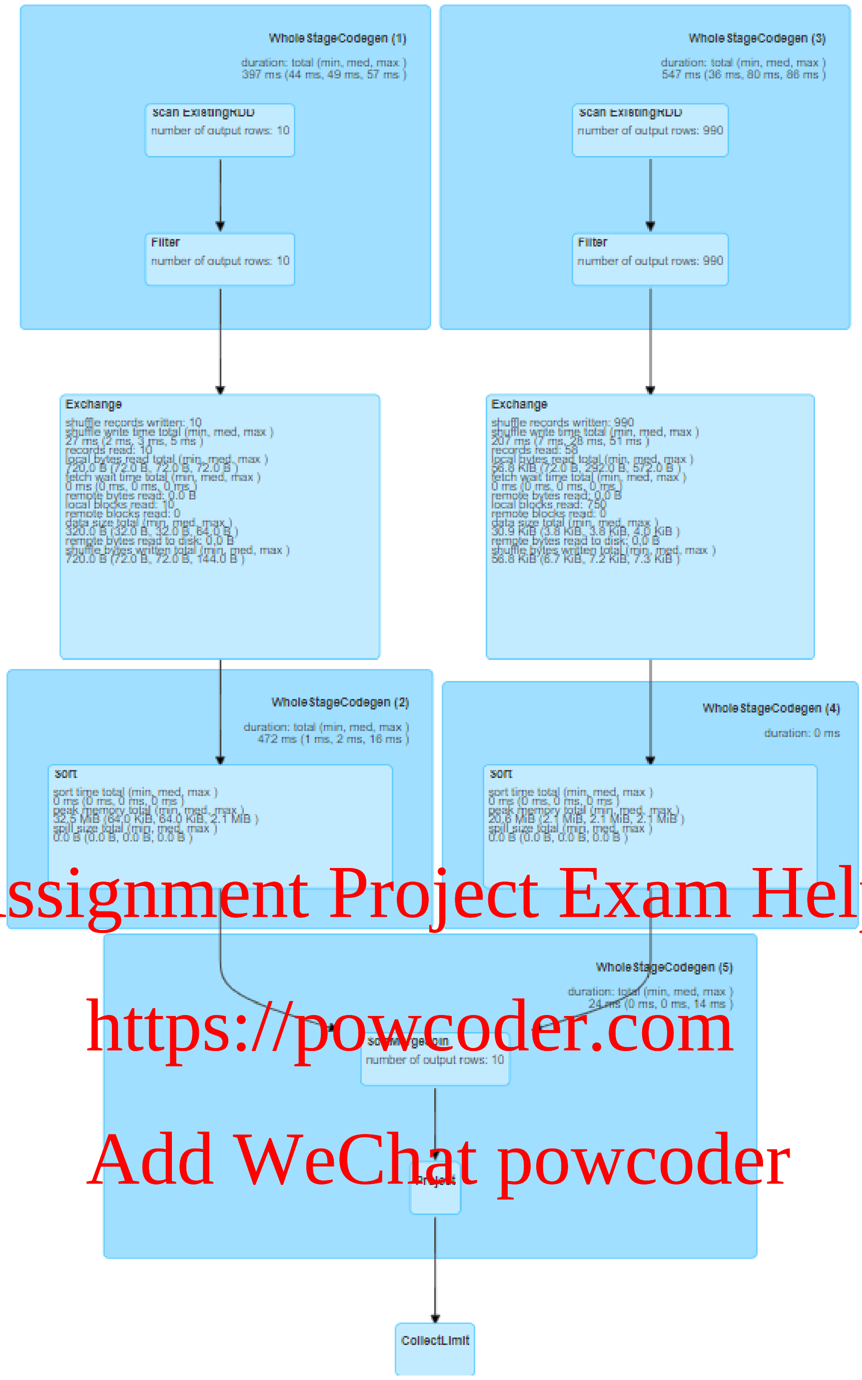
Physical Execution Plan¶

In [ ]:

```
df_joined_sortmerge.explain()
```

Graphical Execution Plan in Spark UI¶

Go to your **Spark UI** and Click on the **SQL** tab to view the graphical equivalent of the above



physical plan.

Explanation Query Plan with Sort Merge Join¶

The order of execution goes from top to bottom. The steps are:

1. Scan dataframe A (left side)
    - Filter id not null in dataframe A
2. Scan dataframe B (right side)
    - Filter id not null in dataframe B
3. Exchange dataframe A: Partition dataframe A with hash partitioning
4. Exchange dataframe B: Partition dataframe B with hash partitioning
5. Sort dataframe A: Sort data within each partition
6. Sort dataframe B: Sort data within each partition
7. Perform Sort Merge Join between both dataframes
8. Project: Select the attributes from both dataframes (df_A: id,valueA and df_b: id,valueB)
9. Collect all the results to the driver

Parallel Join¶

Now we will implement multiple join operations and visualise the parallelism embedded in Spark to perform these kind of queries. The join queries that we will perform are:

1. Inner Join
2. Left Join
3. Full Outer Join
4. Left Semi Join
5. Left Anti Join

All left operations have their right operations as well, but this is a commutative operation so we will focus only on left operations

In this tutorial, you will use three csv files as datasets which contains the information of the Summer Olympics (summer.csv) and Winter Olympics (winter.csv) plus the information of the list of countries (dictionary.csv).

In [ ]:

```
# Read files into dataframes
df_dictionary = spark.read.csv("dictionary.csv",header=True)
df_summer = spark.read.csv("summer.csv",header=True).repartition(4)
df_winter = spark.read.csv("winter.csv",header=True).repartition(4)

# Create Views from Dataframes
df_dictionary.createOrReplaceTempView("sql_dictionary")
df_summer.createOrReplaceTempView("sql_summer")
df_winter.createOrReplaceTempView("sql_winter")
```

1. Lab Task: In the following code block, display the number of partitions and the schema of the above three dataframes. **Examine how the method repartition is used here. What happens if we do not use repartition?**

In [ ]:

```
## Verifying the number of partitions for each dataframe
## You can explore the data of each csv file with the function printSchema()
print(f"####### DICTIONARY INFO:")
##TODO Print number of partitions and schema

print(f"####### SUMMER INFO:")
##TODO Print number of partitions and schema

print(f"####### WINTER INFO:")
##TODO Print number of partitions and schema
```


1. Inner Join¶

This join operation returns the result set that have matching values in both dataframes.

In [ ]:

```
#### Join summer and dictionary using Dataframes
df_dict_inner_summ =
df_dictionary.join(df_summer,df_dictionary.Code==df_summer.Country,how='inner')
print(df_dict_inner_summ.count())
df_dict_inner_summ.show()

## Join summer and dictionary using SQL
sql_dict_inner_summ = spark.sql('''
  SELECT d.*,w.*
  FROM sql_dictionary d JOIN sql_summer w
  ON d.Code=w.Country
''')
print(sql_dict_inner_summ.count())
sql_dict_inner_summ.show()
```

In [ ]:

```
# Now look at the exceution plan for the 2 previous objects
df_dict_inner_summ.explain()
sql_dict_inner_summ.explain()
```

TODO: By looking at the Physical Plan, try to understand the internal workings of joins in dataframe. Discuss this with your tutor.

2. Left Join¶

This join operation returns all records from the left dataframe and the matched records from the right dataframe.

In [ ]:

```
from pyspark.sql.functions import col

#### Join summer and dictionary using Dataframes
df_dict_left_summ =
df_dictionary.join(df_summer,df_dictionary.Code==df_summer.Country,how='left')
# df_dict_inner_summ = df_dict_inner_summ.filter(col('Discipline').isNull())
print(df_dict_left_summ.count())
df_dict_left_summ.show()

## Join summer and dictionary using SQL
sql_dict_left_summ = spark.sql('''
  SELECT d.*,w.*
  FROM sql_dictionary d LEFT JOIN sql_summer w
  ON d.Code=w.Country
''')
print(sql_dict_left_summ.count())
sql_dict_left_summ.show()
```


In [ ]:

```
# Now look at the exceution plan for the 2 previous objects
df_dict_left_summ.explain()
df_dict_left_summ.explain()
```

3. Full Outer Join¶

This join operation returns a result set that includes rows from both left and right dataframes.

In [ ]:

```
#### Join summer and dictionary using Dataframes
df_dict_outer_summ =
df_dictionary.join(df_summer,df_dictionary.Code==df_summer.Country,how='outer')
print(df_dict_outer_summ.count())
df_dict_outer_summ.show()

## Join summer and dictionary using SQL
sql_dict_outer_summ = spark.sql('''
  SELECT d.*,w.*
  FROM sql_dictionary d FULL OUTER JOIN sql_summer w
  ON d.Code=w.Country
''')
print(sql_dict_outer_summ.count())
sql_dict_outer_summ.show()
```

In [ ]:

```
# Now look at the exceution plan for the 2 previous objects
df_dict_outer_summ.explain()
sql_dict_outer_summ.explain()
```

TODO: Execution plan comparison Now dive into the execution plan of the previous 3 joins and their parallelism The objects that will be analysed and compared will be:

- df_dict_inner_summ
- df_dict_left_summ
- df_dict_outer_summ

The comparisons and analysis can be done using the Spark UI. Compare them after running the next code block. If preferred, you can run them one by one to see in the Jobs

In [ ]:

```
# These actions will execute the Query plan for each of the dataframes
df_dict_inner_summ.collect()
df_dict_left_summ.collect()
df_dict_outer_summ.collect()

## TODO: Look deep into what are the operations performed when an inner join
operation is executed.
## For this additional information and better visualisation, go to the Spark UI ->
SQL option
```

4. Left Semi Join¶

This join operation is like an inner join, but only the left dataframe columns and values are selected

2. Lab Task: Implement the **left_semi** join in **Spark SQL.** Ensure that the output from both the approaches is same.

In [ ]:


```
#### Join summer and dictionary using Dataframes
df_dict_semi_summ =
df_dictionary.join(df_summer,df_dictionary.Code==df_summer.Country,how='left_semi')
print(df_dict_semi_summ.count())
df_dict_semi_summ.show()

## TODO: Implement the SQL to perform left semi join between summer and dictionary
using SQL
```


5. Left Anti Join¶

This join operation is the difference of the left dataframe minus the right dataframe, as it selects all rows from df1 that are not present in df2

3. Lab Task: Implement the **left_anti** join in **Spark SQL.** Ensure that the output from both the approaches is same.

In [ ]:

```
#### Join summer and dictionary using Dataframes
df_dict_anti_summ =
df_dictionary.join(df_summer,df_dictionary.Code==df_summer.Country,how='left_anti')
print(df_dict_anti_summ.count())
df_dict_anti_summ.show()

## TODO: Implement the SQL to perform left anti join between summer and dictionary
using SQL
```

Congratulations on finishing this activity!

Having practiced today's activities, we're now ready to embark on a trip of the rest of exiciting FIT5202 activities! See you next week!