

Something More

Optimization used in Project :

Spark :

- 1.Partition and Coalesce
- 2.Broadcast variable
- 3.Broadcast Join
- 4.Tune below parameter
- 5.Error mode optimization
- 6.Avoid to send data to driver
- 7.Compression
- 8.Cache and persist
- 9.File format selection
- 10.fetch and batch size
- 11.Parallism
- 12. Avoid Groupbykey

<https://www.analyticsvidhya.com/blog/2020/11/8-must-know-spark-optimization-tips-for-data-engineering-beginners/>

<https://spark.apache.org/docs/latest/sql-performance-tuning.html>

Hive :

- 1.Partition
- 2.Bucketing
- 3.Execution engine
- 4.Map join
- 5.SMB join
- 6.vectorization

Error Faced in Project :

1.Driver Memory Exception

When the Spark driver runs out of memory, exceptions similar to the following exception occur.

Exception in thread "broadcast-exchange-0" java.lang.**OutOfMemoryError: Not enough memory to build and broadcast the table**

to all worker nodes. **As** a workaround, you can either disable broadcast by setting `spark.sql.autoBroadcastJoinThreshold` to `-1` or increase the spark driver memory by setting `spark.driver.memory` to a higher value

Resolution :

```
--driver-memory <XX>G
```

2. A Spark job may fail when the Application Master (AM) that launches the driver exceeds the memory limit and is eventually terminated by YARN. The following error occurs:

Diagnostics: Container [`pid=<XXXXX>`,`containerID=container_<XXXXXXXXXX>_<XXXX>_<XX>_<XXXXXX>`] is running beyond physical memory limits.
Current usage: `<XX>` GB of `<XX>` GB physical memory used; `<XX>` GB of `<XX>` GB virtual memory used. Killing container

Resolution :

```
--driver-memory <XX>G
```

3. Executor Memory error :

When the executor runs out of memory, the following exception might occur.

Executor task launch worker **for** task XXXXXX ERROR Executor: Exception in task XX.X in stage X.X (TID XXXXXX)
java.lang.OutOfMemoryError: GC overhead limit exceeded

Resolution : `--executor-memory <XX>G`

4. Executor container killed by YARN for exceeding memory limits

When the container hosting the executor needs more memory for overhead tasks or executor tasks, the following error occurs.

org.apache.spark.SparkException: Job aborted due to stage failure: Task X in stage X.X failed X times,
most recent failure: Lost task X.X in stage X.X (TID XX, XX.XX.X.XXX, executor X):
ExecutorLostFailure
(executor X exited caused by one of the running tasks) Reason: Container killed by YARN **for** exceeding memory limits. XX.X GB
of XX.X GB physical memory used. Consider boosting `spark.yarn.executor.memoryOverhead`

Resolution :

Set a higher value for `spark.yarn.executor.memoryOverhead` based on the requirements of the job. The executor memory overhead value increases with the executor size (approximately by 6-10%). As a best practice, modify the executor memory value accordingly.

5. Spark job repeatedly fails

When the cluster is fully scaled and the cluster is not able to manage the job size, the Spark job may fail repeatedly.

Resolution: Run the Sparklens tool to analyze the job execution and optimize the configuration accordingly.

6. The FileAlreadyExistsException error occurs in the following scenarios:

Failure of the previous task might leave some files that trigger the FileAlreadyExistsException errors as shown below.

```
org.apache.spark.SparkException: Task failed while writing rows
at
org.apache.spark.sql.execution.datasources.FileFormatWriter$.org$apache$spark$sql$execution$da
tasources$FileFormatWriter$$executeTask(FileFormatWriter.scala:272)
at
org.apache.spark.sql.execution.datasources.FileFormatWriter$$anonfun$write$1$$anonfun$apply$mc
V$sp$1$.apply(FileFormatWriter.scala:191)
at
org.apache.spark.sql.execution.datasources.FileFormatWriter$$anonfun$write$1$$anonfun$apply$mc
V$sp$1$.apply(FileFormatWriter.scala:190)
at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:87)
at org.apache.spark.scheduler.Task.run(Task.scala:108)
at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:335)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
... 1 more
Caused by: org.apache.hadoop.fs.FileAlreadyExistsException:
s3://xxxxxx/xxxxxxx/xxxxxx/analysis-
results/datasets/model=361/dataset=encoded_unified/dataset_version=vvvvv.snappy.parquet
already exists
at org.apache.hadoop.fs.s3a.S3AFileSystem.create(S3AFileSystem.java:806)
at org.apache.hadoop.fs.FileSystem.create(FileSystem.java:914)
```

Resolution :

1. Identify the original executor failure reason that causes the FileAlreadyExistsException error.
2. Verify size of the nodes in the clusters.
3. Upgrade them to the next tier to increase the Spark executor's memory overhead.

7. Error when the total size of results is greater than the Spark Driver Max Result Size value

Description: When the total size of results is greater than the Spark Driver Max Result Size value, the following error occurs.

```
org.apache.spark.SparkException: Job aborted due to stage failure: Total size of serialized results of x tasks (y MB) is bigger than spark.driver.maxResultSize (z MB)
```

Resolution: Increase the Spark Drive Max Result Size value by modifying the value of --conf spark.driver.maxResultSize in the Spark Submit Command Line Options on the Analyze page.

8. Too Large Frame error

Description: When the size of the shuffle data blocks exceeds the limit of 2 GB, which spark can handle, the following error occurs.

```
org.apache.spark.shuffle.FetchFailedException: Too large frame: XXXXXXXXXX
at
org.apache.spark.storage.ShuffleBlockFetcherIterator.throwFetchFailedException(ShuffleBlockFetcherIterator.scala:513)
at
org.apache.spark.storage.ShuffleBlockFetcherIterator.next(ShuffleBlockFetcherIterator.scala:444)
```

```
Caused by: java.lang.IllegalArgumentException: Too large frame: XXXXXXXXXX
at org.spark_project.guava.base.Preconditions.checkArgument(Preconditions.java:119)
at
org.apache.spark.network.util.TransportFrameDecoder.decodeNext(TransportFrameDecoder.java:133)
```

Resolution: Perform one of the following steps to resolve this error:

Solution 1:

1. Run the job on Spark 2.2 or higher version because Spark 2.2 or higher handles this issue in a better way when compared to other lower versions of Spark.

Use the following Spark configuration:

Modify the value of spark.sql.shuffle.partitions from the default 200 to a value greater than 2001.

Set the value of spark.default.parallelism to the same value as spark.sql.shuffle.partitions.

Solution 2:

Identify the DataFrame that is causing the issue.

Add a Spark action(for instance, df.count()) after creating a new DataFrame.

Print anything to check the DataFrame.

If the print statement is not executed for a DataFrame, then the issue is with that DataFrame.

After the DataFrame is identified, repartition the DataFrame by using df.repartition() and then cache it by using df.cache().

If there is skewness in the data and you are using Spark version earlier than 2.2, then modify the code.

9. Scenario: Java heap space error when trying to open Apache Spark history server

Issue

You receive the following error when opening events in Spark History server:

```
scala.MatchError: java.lang.OutOfMemoryError: Java heap space (of class java.lang.OutOfMemoryError)
```

Resolution :

You can increase the Spark History Server memory by editing the SPARK_DAEMON_MEMORY property in the Spark configuration and restarting all the services.

Sample Interview Questions on project :

Refer : Project_Interview_Sample_Questions.txt

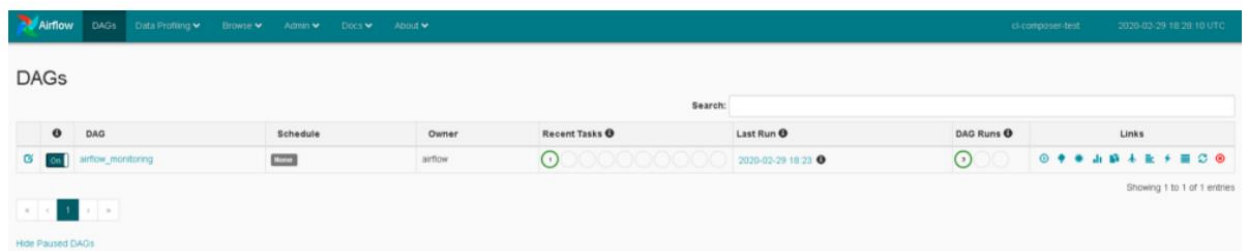
Data Reconciliation :




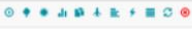
- 1.Count
- 2.Null
- 3.RI
- 4.Duplicate
- 5.Detailed Validation

Job Scheduling in project :

- 1.Historical Data Load Jobs (One time)
- 2.Delta Processing Jobs (Daily)
- 3.Deduplication Jobs (Daily)
- 4.Transformation Jobs (Daily)
- 5.Report Generation jobs (Daily)

Sample Airflow dag :



DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
 airflow_monitoring	Hourly	airflow		2020-02-29 18:23		

Showing 1 to 1 of 1 entries

Sample Dag

STEP 1: Libraries needed

from datetime import timedelta, datetime

from airflow import models

from airflow.operators.bash_operator import BashOperator

STEP 2: Define a start date

In this case yesterday

```
yesterday = datetime(2020, 2, 28)
```

```
# STEP 3: Set default arguments for the DAG
```

```
default_dag_args = {  
    'start_date': yesterday,  
    'depends_on_past': False,  
    'email_on_failure': False,  
    'email_on_retry': False,  
    'retries': 1,  
    'retry_delay': timedelta(minutes=5)  
}
```

```
# STEP 4: Define DAG
```

```
# set the DAG name, add a DAG description, define the schedule interval and pass the default arguments  
defined before
```

```
with models.DAG(  
    'simple_workflow',  
    description='Hello World =)',  
    schedule_interval=timedelta(days=1),  
    default_args=default_dag_args) as dag:
```

```
# STEP 5: Set Operators
```

```
# BashOperator
```

```
# Every operator has at least a task_id and the other parameters are particular for each one, in this case, is  
a simple BashOperator this operator will execute a simple echo "Hello World!"
```

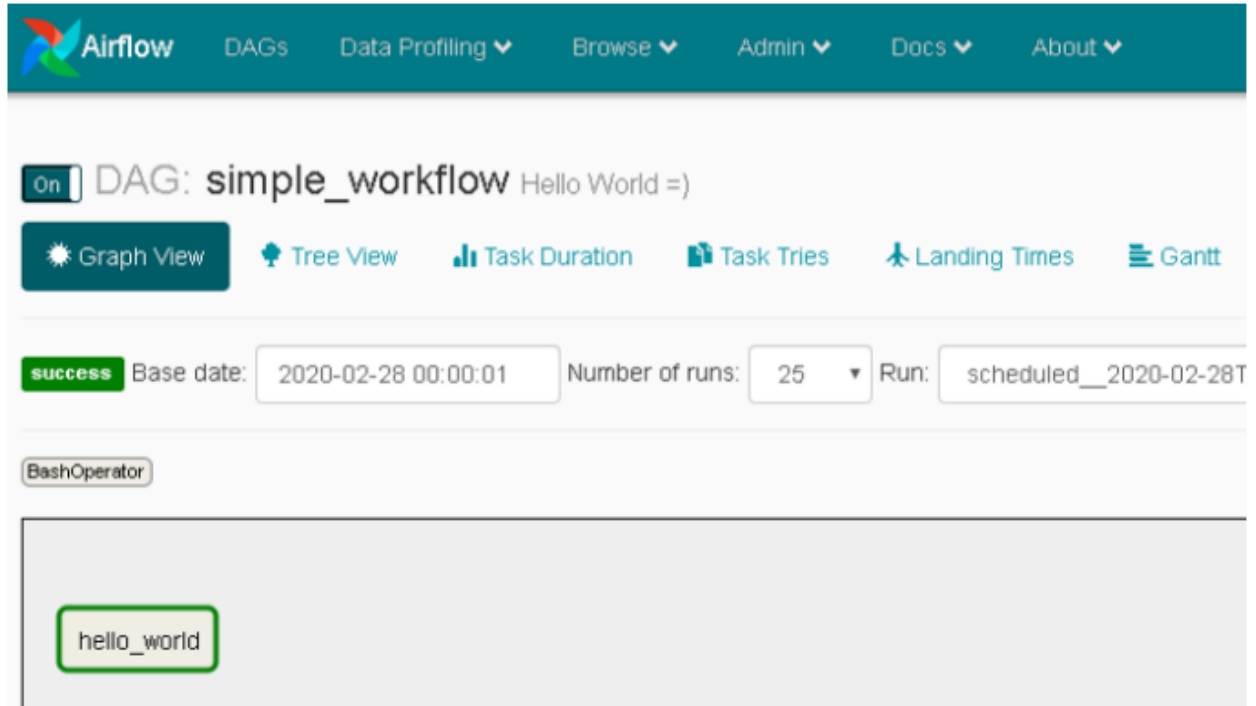
```
helloOp = BashOperator(  
    task_id='hello_world',  
    bash_command='echo "Hello Worl!"'  
)
```

```
# STEP 6: Set DAGs dependencies
```

```
# Since we have only one, we just write the operator
```

```
helloOp
```

Graph View :

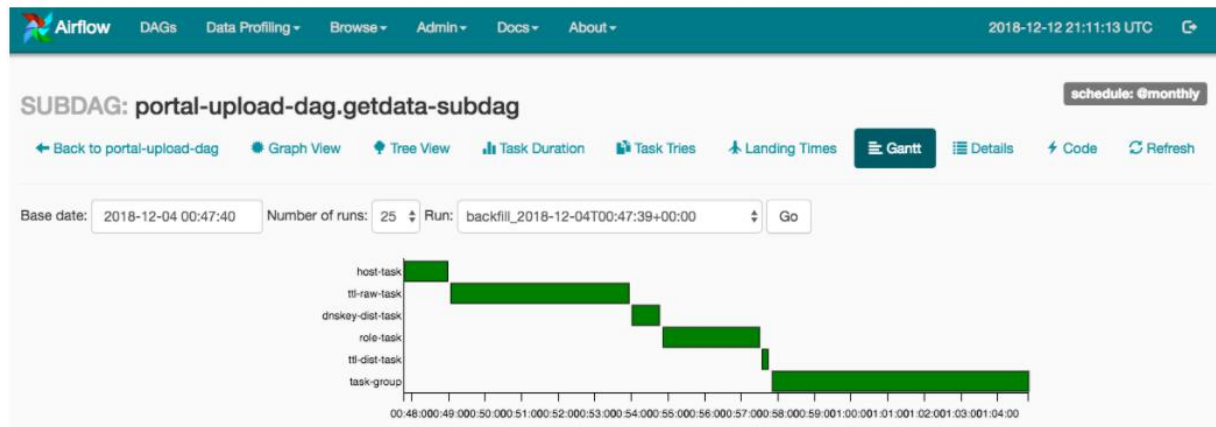


The screenshot shows the Airflow web interface for a DAG named 'simple_workflow' with the description 'Hello World =)'. The 'Graph View' tab is selected, showing a single task node 'hello_world' within a 'BashOperator' group. The status is 'success'. The interface includes a top navigation bar with links to DAGs, Data Profiling, Browse, Admin, Docs, and About. Below the DAG name, there are tabs for Graph View, Tree View, Task Duration, Task Tries, Landing Times, and Gantt. A filter section shows 'Base date: 2020-02-28 00:00:01', 'Number of runs: 25', and 'Run: scheduled__2020-02-28T'. The task node 'hello_world' is highlighted with a green border.

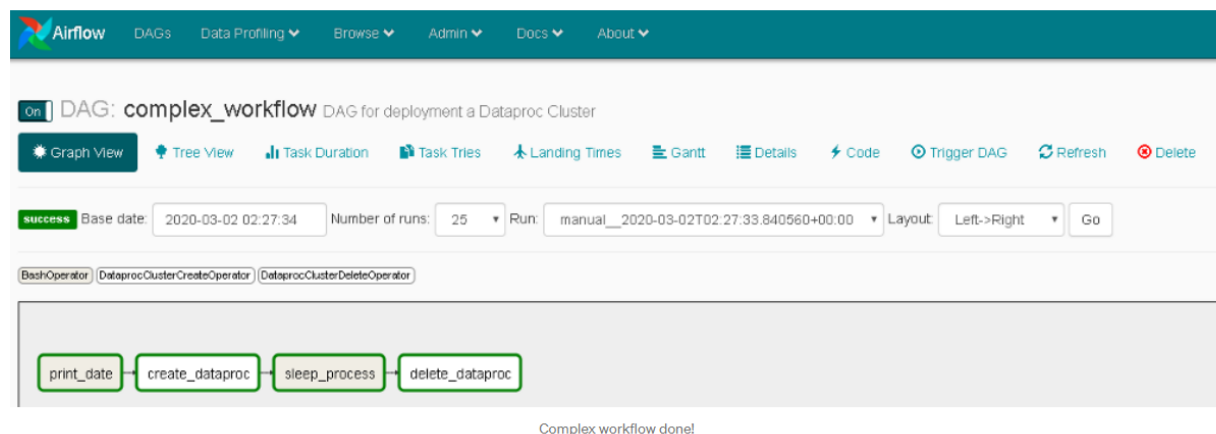
Click hellor_world task

View Log :

SAYU

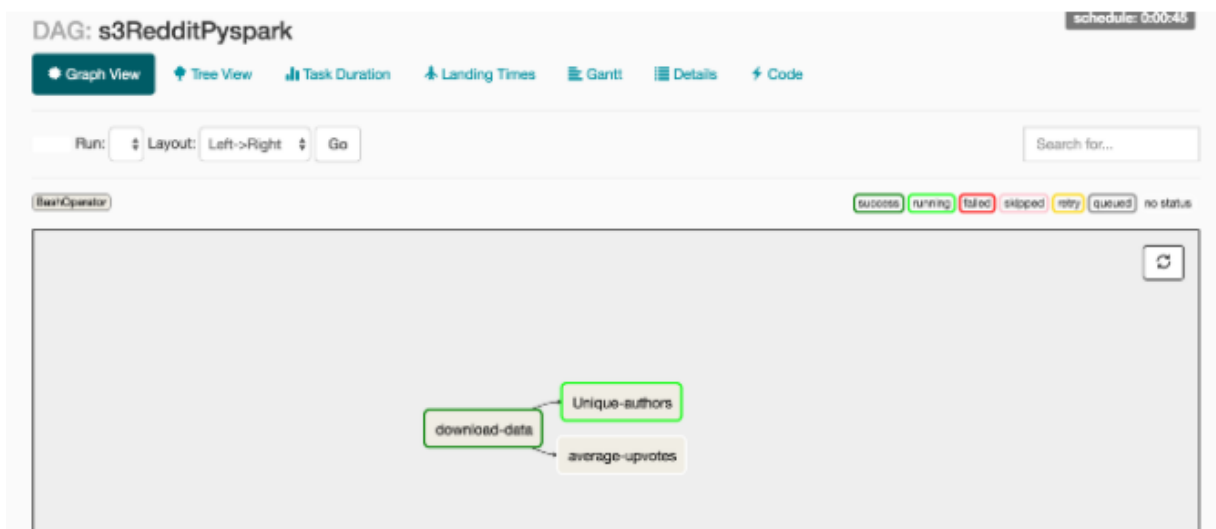


Complex Workflow :



Complex workflow done!

Running Job :



```
#Spark dag
```

```
from airflow.models import DAG
```

```
from airflow.providers.apache.spark.operators.spark_jdbc import SparkJDBCOperator
```

```
from airflow.providers.apache.spark.operators.spark_sql import SparkSqlOperator
```

```
from airflow.providers.apache.spark.operators.spark_submit import SparkSubmitOperator
```

```
from airflow.utils.dates import days_ago
```

```
with DAG(
```

```
    dag_id='example_spark_operator',
```

```
    schedule_interval=None,
```

```
    start_date=days_ago(2),
```

```
    tags=['example'],
```

```
) as dag:
```

```
    # [START howto_operator_spark_submit]
```

```
    submit_job = SparkSubmitOperator(
```

```
        application="${SPARK_HOME}/examples/src/main/python/pi.py", task_id="submit_job"
```

```
    )
```

```
    # [END howto_operator_spark_submit]
```

```
    # [START howto_operator_spark_jdbc]
```

```
    jdbc_to_spark_job = SparkJDBCOperator(
```

```
        cmd_type='jdbc_to_spark',
```

```
        jdbc_table="foo",
```

```
        spark_jars="${SPARK_HOME}/jars/postgresql-42.2.12.jar",
```

```
        jdbc_driver="org.postgresql.Driver",
```

```
        metastore_table="bar",
```

```
        save_mode="overwrite",
```

```
        save_format="JSON",
```

```
        task_id="jdbc_to_spark_job",
```

```
    )
```

```
    spark_to_jdbc_job = SparkJDBCOperator(
```

```
        cmd_type='spark_to_jdbc',
```

```
        jdbc_table="foo",
```

```
        spark_jars="${SPARK_HOME}/jars/postgresql-42.2.12.jar",
```

```
        jdbc_driver="org.postgresql.Driver",
```

```
    metastore_table="bar",  
    save_mode="append",  
    task_id="spark_to_jdbc_job",  
)  
# [END howto_operator_spark_jdbc]
```

```
# [START howto_operator_spark_sql]  
sql_job = SparkSqlOperator(sql="SELECT * FROM bar", master="local", task_id="sql_job")  
# [END howto_operator_spark_sql]
```

```
*****END*****
```

Sayu Software