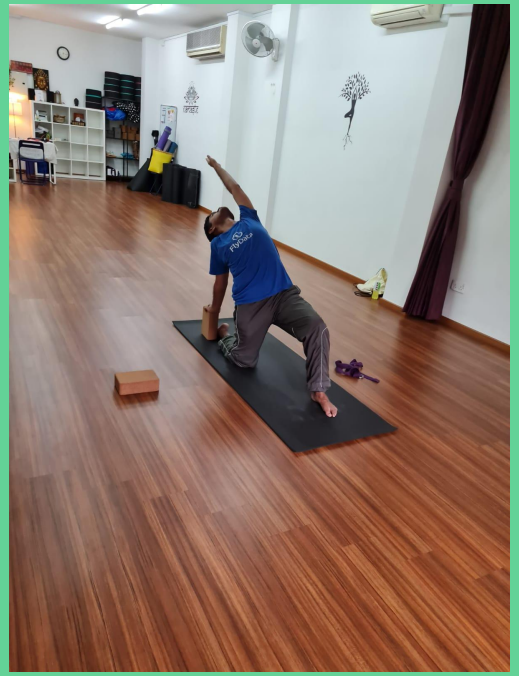
Spark Yoga - saving time & money with lean data pipelines

Vishnu Rao
Senior data engineer at Eyeota
(ex - Flipkart)

Running time is money

"some cloud provider"

Yoga Asanas



Yep thats me :)

What will we focus on today ?

What will we focus on today ?

1. Some Asanas for **Spark**

What will we focus on today ?

1. Some Asanas for **Spark**

2. And some simple actual Yoga Asanas too at the end :)

Spark Yoga Asanas

- Using Spark in a specific way to save time & money

Spark Yoga Asanas

- Using Spark in a specific way to save time & money
  - **Tuning / Using a specific Spark Configuration parameter**

Spark Yoga Asanas

- Using Spark in a specific way to save time & money
  - Tuning / Using a specific Spark Configuration parameter
  - **Choosing the kind of data lake it interacts with**

Spark Yoga Asanas

- Using Spark in a specific way to save time & money
  - Tuning / Using a specific Spark Configuration parameter
  - Choosing the kind of data lake it interacts with
  - **Running the spark job in a specific way**

Spark Yoga Asanas

- Using Spark in a specific way to save time & money
  - Tuning / Using a specific Spark Configuration parameter
  - Choosing the kind of data lake it interacts with
  - Running the spark job in a specific way
  - **Choosing the kind of framework it runs in**

Spark Yoga Asanas

- Using Spark in a specific way to save time & money
  - Tuning / Using a specific Spark Configuration parameter
  - Choosing the kind of data lake it interacts with
  - Running the spark job in a specific way
  - Choosing the kind of framework it runs in
  - **Balancing Time With Co$t**

Tuning / Using a specific Spark Configuration parameter (Asana 1)

Tuning / Using a specific Spark Configuration parameter (Asana 1)

Some of my frequently used configs are

Tuning / Using a specific Spark Configuration parameter (Asana 1)

Some of my frequently used configs are

1. Spark.hadoop.mapreduce.fileoutputcommitter.algorithm.version = 2 (saves time)

Tuning / Using a specific Spark Configuration parameter (Asana 1)

Some of my frequently used configs are

1. Spark.hadoop.mapreduce.fileoutputcommitter.algorithm.version = 2 (saves time)
    a. Or experiment with s3a committers

Tuning / Using a specific Spark Configuration parameter (Asana 1)

Some of my frequently used configs are

1. Spark.hadoop.mapreduce.fileoutputcommitter.algorithm.version = 2 (saves time)
    a. Or experiment with s3a committers

2. spark.serializer=org.apache.spark.serializer.KryoSerializer (better serialization)

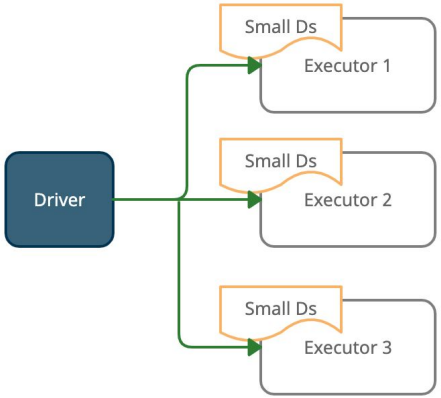
Tuning / Using a specific Spark Configuration parameter (Asana 1)

Some of my frequently used configs are

1. Spark.hadoop.mapreduce.fileoutputcommitter.algorithm.version = 2 (saves time)
   a. Or experiment with s3a committers

2. spark.serializer=org.apache.spark.serializer.KryoSerializer (better serialization)

3. ***spark.sql.autoBroadcastJoinThreshold=256MB (helps with the join)***



Tuning / Using a specific Spark Configuration parameter (Asana 1)

Some of my frequently used configs are

1. Spark.hadoop.mapreduce.fileoutputcommitter.algorithm.version = 2 (saves time)
    a. Or experiment with s3a committers

2. spark.serializer=org.apache.spark.serializer.KryoSerializer (better serialization)

3. spark.sql.autoBroadcastJoinThreshold=256MB (helps with the join)

4. ***spark.hadoop.fs.s3a.connection.ssl.enabled=false (saves time)***

Tuning / Using a specific Spark Configuration parameter (Asana 1)

Some of my frequently used configs are

1. Spark.hadoop.mapreduce.fileoutputcommitter.algorithm.version = 2 (saves time)
    a. Or experiment with s3a committers
2. spark.serializer=org.apache.spark.serializer.KryoSerializer (better serialization)
3. spark.sql.autoBroadcastJoinThreshold=256MB (helps with the join)
4. spark.hadoop.fs.s3a.connection.ssl.enabled=false (saves time)
5. ***If executor < 32G, Use* -XX:+UseCompressedOops (pointer size reduction)**

Choosing the kind of data lake it interacts with (Asana 2)

Choosing the kind of data lake it interacts with (Asana 2)

1. S3* or any other cloud object storage

Choosing the kind of data lake it interacts with (Asana 2)

1. S3* or any other cloud object storage
2. HDFS (On Premise / Cloud)

Choosing the kind of data lake it interacts with (Asana 2)

1. S3* or any other cloud object storage
2. HDFS (On Premise / Cloud)

The above are the more notable choices. Are they the only choices ?

Choosing the kind of data lake it interacts with (Asana 2)

1. S3* or any other cloud object storage
2. HDFS (On Premise / Cloud)

Evaluation Criteria:

Choosing the kind of data lake it interacts with (Asana 2)

1. S3* or any other cloud object storage
2. HDFS (On Premise / Cloud)

Evaluation Criteria:

1. Data access Time
2. Maintenance

Choosing the kind of data lake it interacts with (Asana 2)

	S3/Cloud Storage	HDFS (onprem / cloud)
Infra Maintenance	no 	yes 
Access times	Over network - https? 	Local access to node 
Compute	No, Need extra nodes 	Yes with hadoop 

Choosing the kind of data lake it interacts with (Asana 2)

	S3/Cloud Storage	HDFS (onprem / cloud)	Local File System + S3
Infra Maintenance	no ✓	yes ⓘ	Yes. local storage might be limited, extend via EBS ?
Access times	Over network - https? ↑	Local access to node ↓	Local access to node ↓
Compute	No, Need extra nodes ✗	Yes with hadoop ✓	Yes, you launch workers on the box ✓

How about Local File System ? 💡

Choosing the kind of data lake it interacts with (Asana 2)

	S3/Cloud Storage	HDFS (onprem / cloud)	Local File System + S3
Infra Maintenance	no 	yes 	Yes. local storage might be limited, extend via EBS ?
Access times	Over network - https? 	Local access to node 	Local access to node 
Compute	No, Need extra nodes 	Yes with hadoop 	Yes, you launch workers on the box 

Pro tip: ‘aws cp’ to local file system is faster than spark interacting with s3

Example of Orchestration

Note: while using local file system(file://), you need to download the entire dataset to each host

Choosing the kind of data lake it interacts with (Asana 2)

	S3/Cloud Storage	HDFS	Local File System + S3
Infra Maintenance	no 	yes 	Yes. local storage might be **limited**, extend via EBS ?
Access times	Over network - https? 	Local access to node 	Local access to node 
Compute	No, Need extra nodes 	Yes with hadoop 	Yes, you launch workers on the box 

Sometimes usage of Local File System along with S3,

- o- Can lead to less running times as its a local data access pattern
- o- Yes, we launch & maintain our own cluster, which might not be a burden as long as cost is low.

Running the spark job in a specific way (Asana 3)

Running the spark job in a specific way (Asana 3a)

We always tend to go the spark-client mode / run in a cluster .

Running the spark job in a specific way (Asana 3a)

We always tend to go the spark-client mode / run in a cluster.

How about the Spark - **Local** mode  ?

Spark.master = local[*]

Running the spark job in a specific way (Asana 3a)

We always tend to go the spark-client mode / run in a cluster.

How about the Spark - **Local** mode  ?

Spark.master = local[*]

- Valid way of running jobs. One might think it’s for only running tests.

Running the spark job in a specific way (Asana 3a)

We always tend to go the spark-client mode / run in a cluster.

How about the Spark - **Local** mode  ?

Spark.master = local[*]

- Valid way of running jobs. One might think it’s for only running tests.
- I say why not ? You get benefit of spark DSL while solving your problem.

Running the spark job in a specific way (Asana 3a)

We always tend to go the spark-client mode / run in a cluster.

How about the Spark - **Local** mode  ?

Spark.master = local[*]

- Valid way of running jobs. One might think it’s for only running tests.
- I say why not ? You get benefit of spark DSL while solving your problem.
- You can always scale horizontally by going to a cluster
    - Cluster is not always needed !

Running the spark job in a specific way (Asana 3a)

We always tend to go the spark-client mode / run in a cluster.

How about the Spark - **Local** mode  ?

Spark.master = local[*]

- Valid way of running jobs. One might think it’s for only running tests.
- I say why not ? You get benefit of spark DSL while solving your problem.
- You can always scale horizontally by going to a cluster
  - Cluster is not always needed !
- Also you have the option of running in a **kubernetes** pod !

Running the spark job in a specific way (Asana 3b)

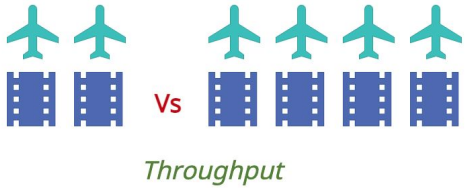
Data preparation

Running the spark job in a specific way (Asana 3b)

Data preparation

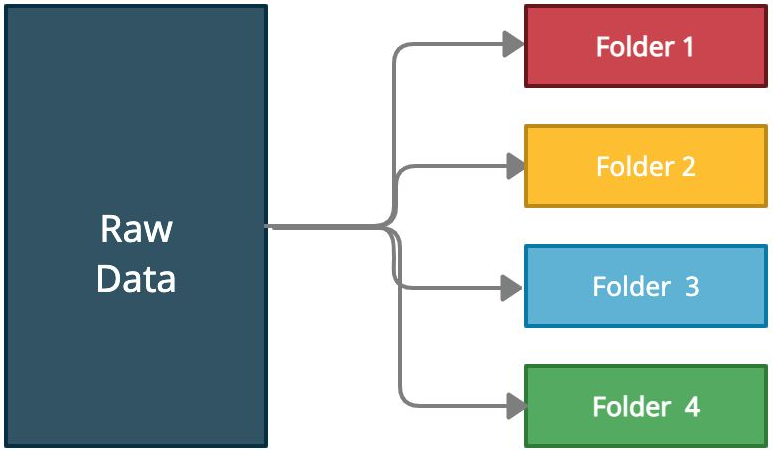
Repartition the data for better parallelization of work.

Ex: we wish to increase our write throughput to the db



Running the spark job in a specific way (Asana 3c)

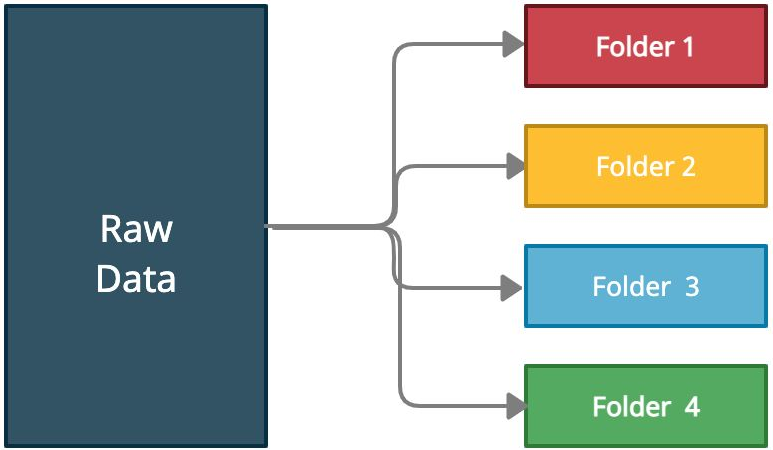
Data preparation - Sharding Data



Running the spark job in a specific way (Asana 3c)

Data preparation - Sharding Data

Goliath vs Mini Goliath's

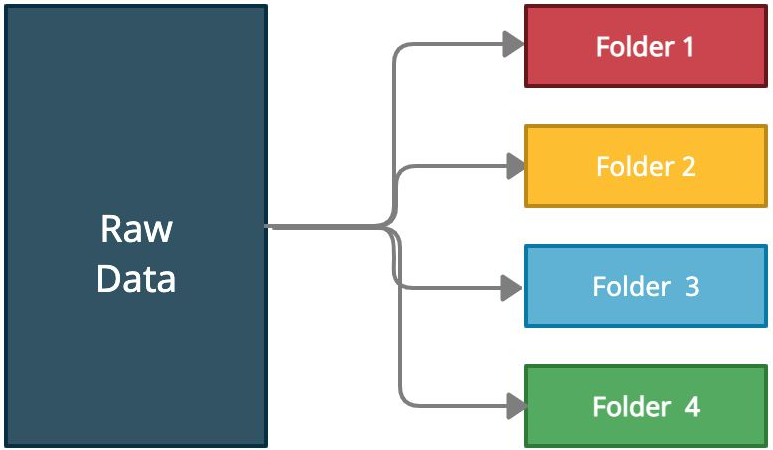


Running the spark job in a specific way (Asana 3c)

Data preparation - Sharding Data

Goliath vs Mini Goliath's

- **Divide & Conquer principle**

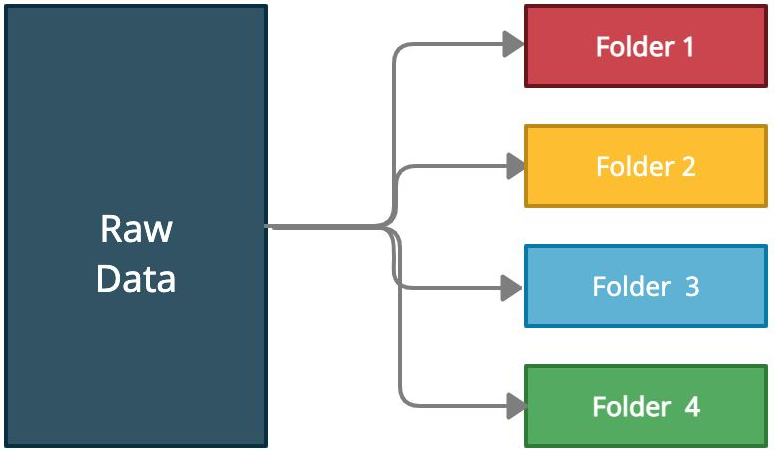


Running the spark job in a specific way (Asana 3c)

Data preparation - Sharding Data

- Divide & Conquer principle
  - **Say 100 shards/buckets/folders**

Goliath vs Mini Goliath's

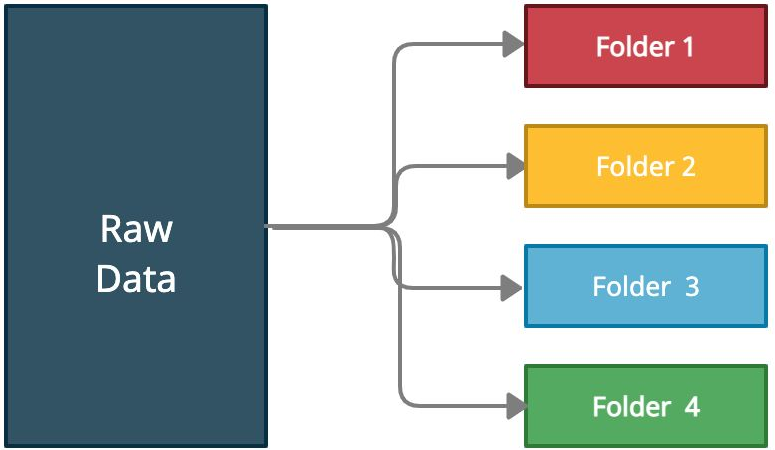


Running the spark job in a specific way (Asana 3c)

Data preparation - Sharding Data

- Divide & Conquer principle
  - Say 100 shards/buckets/folders
  - **100 spark jobs on 100 smaller datasets**

Goliath vs Mini Goliath's

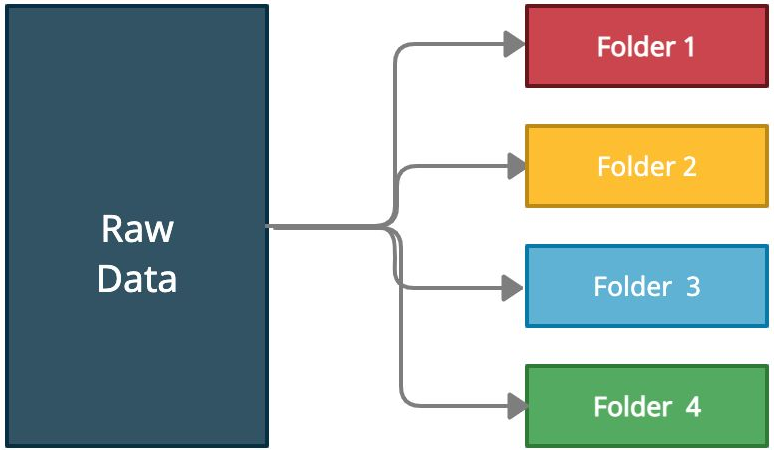


Running the spark job in a specific way (Asana 3c)

Data preparation - Sharding Data

- Divide & Conquer principle
  - Say 100 shards/buckets/folders
  - 100 spark jobs on 100 smaller datasets
- **Resource allocation now based on shard size , not large dataset => smaller boxes/cluster**

Goliath vs Mini Goliath's

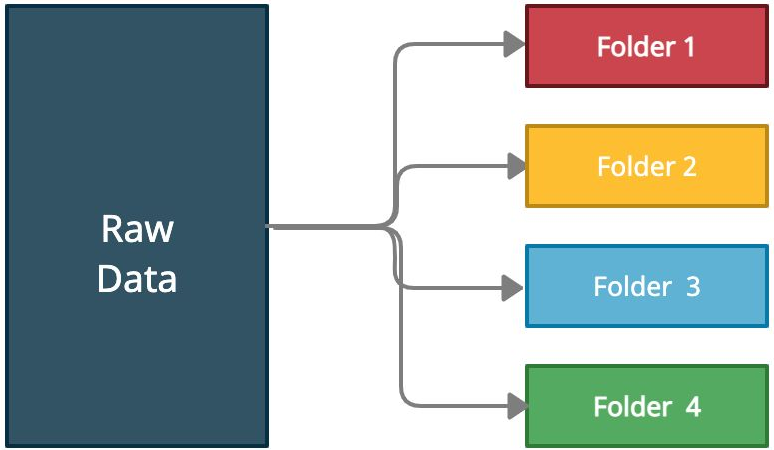


Running the spark job in a specific way (Asana 3c)

Data preparation - Sharding Data

- Divide & Conquer principle
  - Say 100 shards/buckets/folders
  - 100 spark jobs on 100 smaller datasets
- Resource allocation now based on shard size , not large dataset => smaller boxes/cluster
- **Isolated failures & ReRun - rerun jobs only for failed buckets**

Goliath vs Mini Goliath's

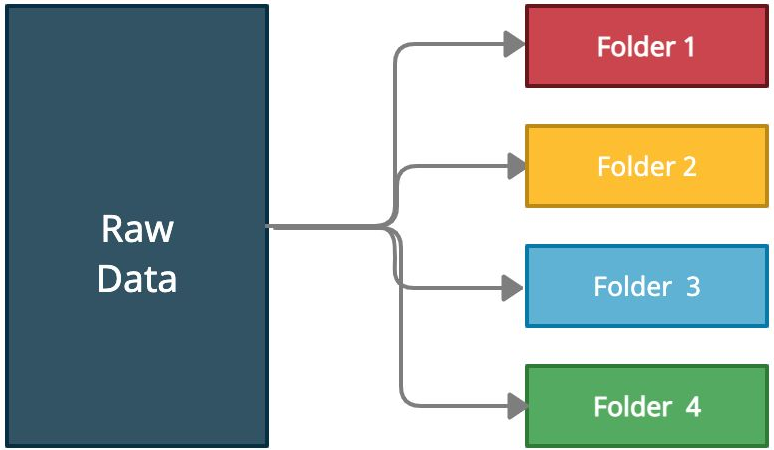


Running the spark job in a specific way (Asana 3c)

Data preparation - Sharding Data

- Divide & Conquer principle
  - Say 100 shards/buckets/folders
  - 100 spark jobs on 100 smaller datasets
- Resource allocation now based on shard size , not large dataset => smaller boxes/cluster
- Isolated failures & ReRun - rerun jobs only for failed buckets
- **Debugging failures / Improving performance focussed wrt small bucket**

Goliath vs Mini Goliath's



Running the spark job in a specific way (Asana 3c)

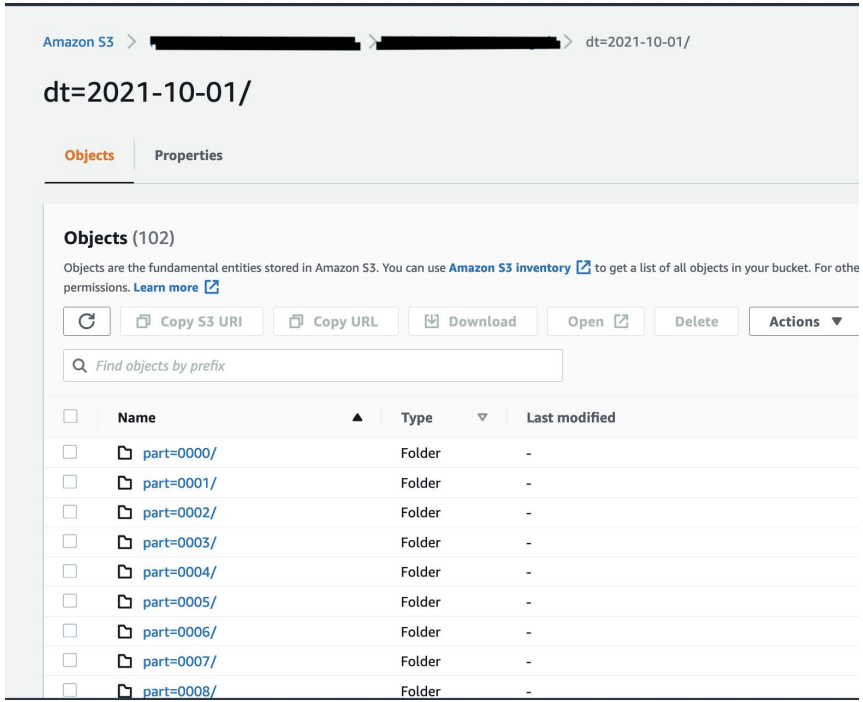
Data preparation - Sharding data

Does it help always ?

- No, Say you shard using user-id/order-id & a spark job needs to cut across all the shards (group by country rather than group by user - id)
- We might have to aggregate/merge the results obtained in each job to get the final result. This might be complex.

The Setup

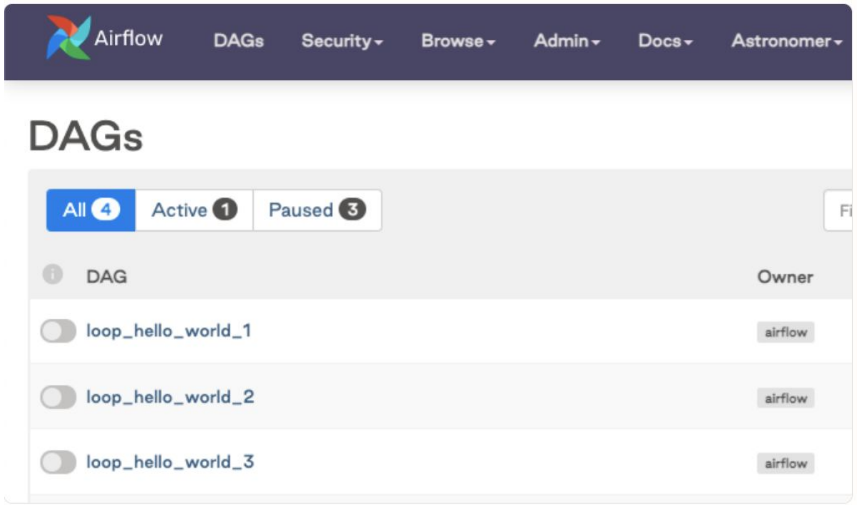
S3 Bucket for Raw data with 100 parts/shards/folders



The Setup

- Dynamically create ‘n’ Airflow dags.
- Run ‘n’ Airflow Dags for ‘n’ shards/folders

Sample source code taken from
https://www.astronomer.io/guides/dynamically-generating-dags

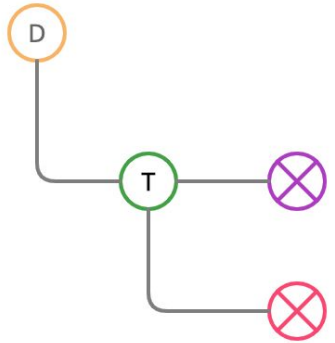



```python
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from datetime import datetime


def create_dag(dag_id,
               schedule,
               dag_number,
               default_args):

    def hello_world_py(*args):
        print('Hello World')
        print('This is DAG: {}'.format(str(dag_number)))

    dag = DAG(dag_id,
              schedule_interval=schedule,
              default_args=default_args)

    with dag:
        t1 = PythonOperator(
            task_id='hello_world',
            python_callable=hello_world_py)

    return dag


# build a dag for each number in range(10)
for n in range(1, 4):
    dag_id = 'loop_hello_world_{}'.format(str(n))

    default_args = {'owner': 'airflow',
                    'start_date': datetime(2021, 1, 1)
                    }

    schedule = '@daily'
    dag_number = n

    globals()[dag_id] = create_dag(dag_id,
                                   schedule,
                                   dag_number,
                                   default_args)
```

Running the spark job in a specific way (Asana 3d)

Reusable datasets

- Are there multiple actions on same dataset in the lineage ?



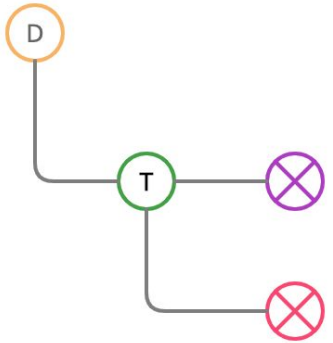
Running the spark job in a specific way (Asana 3d)

Reusable datasets

- Are there multiple actions on same dataset in the lineage ?

Dataset1 = spark.load(...)

*Dataset2 = **complex_transform_**(Dataset1); // takes lot of time.*



Dataset2.count() //action 1

Dataset2.save() // action 2

Running the spark job in a specific way (Asana 3d)

Reusable datasets

- Are there multiple actions on same dataset in the lineage ?

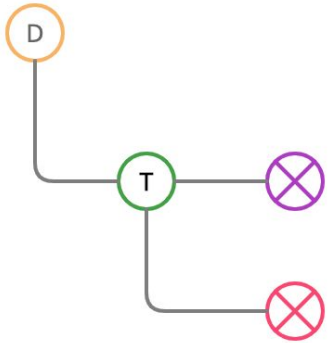
Dataset1 = spark.load(...)

*Dataset2 = **complex_transform_**(Dataset1); // takes lot of time.*

Dataset2.persist(DISK_ONLY); // try with memory if you have it

Dataset2.count() //action 1

Dataset2.save() // action 2



Choosing the framework it runs in (Asana 4)

Choosing the framework it runs in (Asana 4)

- EMR/Yarn - choice for most people

Choosing the framework it runs in (Asana 4)

- EMR/Yarn - choice for most people.
- How about Standalone Cluster ?

Choosing the framework it runs in (Asana 4)

- EMR/Yarn - choice for most people.
- How about Standalone Cluster ?
  - Yes you maintain the cluster
  - But sometimes you can be **smart** with it

Choosing the framework it runs in (Asana 4)

- EMR/Yarn - choice for most people.
- How about Standalone Cluster ?
  - Yes you maintain the cluster
  - But sometimes you can be **smart** with it
    - I/O intensive jobs - ex: writing to cassandra

Choosing the framework it runs in (Asana 4)

- EMR/Yarn - choice for most people.
- How about Standalone Cluster ?
  - Yes you maintain the cluster
  - But sometimes you can be **smart** with it
    - I/O intensive jobs - ex: writing to cassandra
      - => Less CPU used
      - => Not all cores used to full potential

Choosing the framework it runs in (Asana 4)

- EMR/Yarn - choice for most people.
- How about Standalone Cluster ?
  - Yes you maintain the cluster
  - But sometimes you can be **smart** with it
    - I/O intensive jobs - ex: writing to cassandra
      - => Less CPU used
      - => Not all cores used to full potential
      - Can we increase Threads -> **YES** !

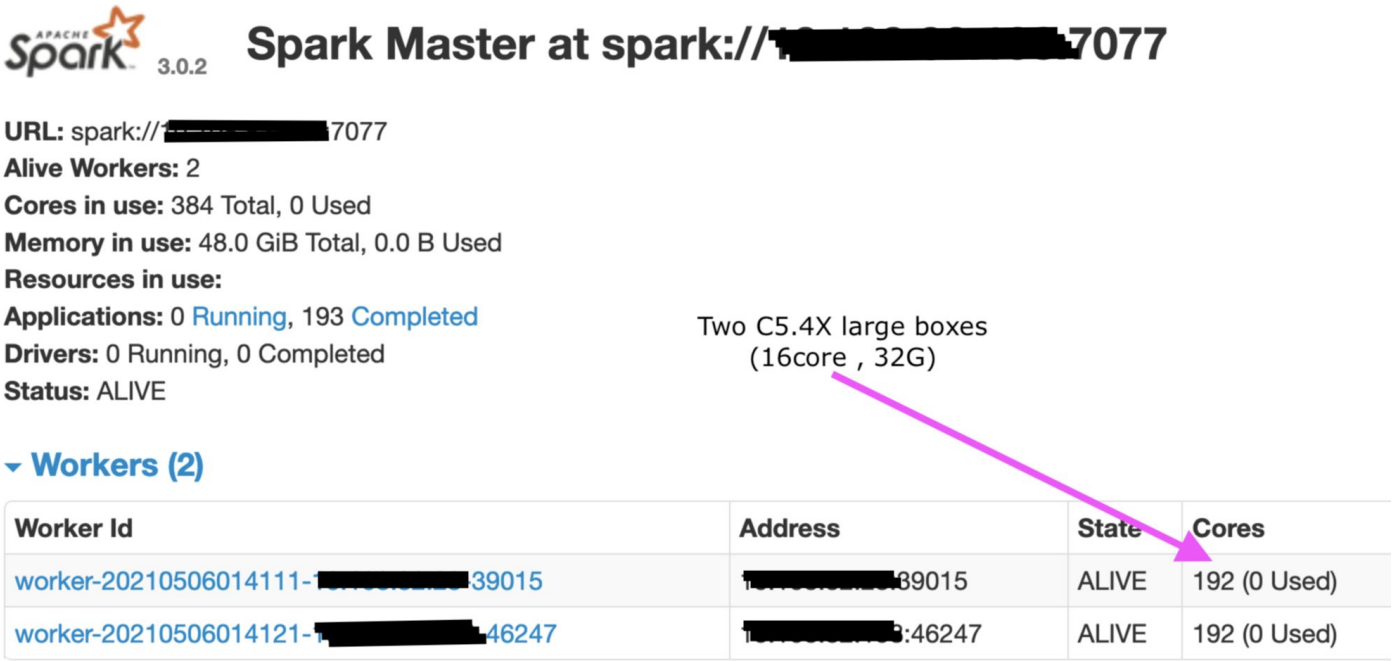
Choosing the framework it runs in (Asana 4)

- EMR/Yarn - choice for most people.
- How about Standalone Cluster ?
  - Yes you maintain the cluster
  - But sometimes you can be smart with it
    - I/O intensive jobs - ex: writing to cassandra
      - => Less CPU used
      - => Not all cores used to full potential
      - Can we increase Threads -> **YES** !
      - Box has N cores -> you configure SPARK_WORKER_CORES to M cores (M > N)

Choosing the framework it runs in (Asana 4)

- EMR/Yarn - choice for most people.
- How about Standalone Cluster ?
  - Yes you maintain the cluster
  - But sometimes you can be smart with it
    - I/O intensive jobs - ex: writing to cassandra
      - => Less CPU used
      - => Not all cores used to full potential
      - Can we increase Threads -> **YES** !
      - Box has N cores -> you configure SPARK_WORKER_CORES to M cores (M > N)
      - This is called **Core-Oversubscribing**
        - EMR might not give this option*. Only increase number of boxes => more$

Oversubscribing



Choosing the framework it runs in (Asana 4)

Spark on Kubernetes

Choosing the framework it runs in (Asana 4)

Spark on Kubernetes

+

Using Spot instances

Choosing the framework it runs in (Asana 4)

Spark on Kubernetes

+

Using Spot instances

=

Cheaper way to run workloads

Balancing Time vs Co$t (Asana 5)

- At the end of day $ does matter with an SLA (time) in mind.

Balancing Time vs Co$t (Asana 5)

- At the end of day $ does matter with an **acceptable** SLA (time) in mind.

Balancing Time vs Co$t (Asana 5)

- At the end of day $ does matter but with acceptable SLA (time) in mind.
- Cost

Balancing Time vs Co$t (Asana 5)

- At the end of day $ does matter but with acceptable SLA (time) in mind.
- Cost
  - Compute cluster cost i.e. how long boxes are used ?

Balancing Time vs Co$t (Asana 5)

- At the end of day $ does matter but with acceptable SLA (time) in mind.
- Cost
  - Compute cluster cost i.e. how long boxes are used ?
  - Storage Cost during computation

Balancing Time vs Co$t (Asana 5)

- At the end of day $ does matter but with acceptable SLA (time) in mind.
- Cost
  - Compute cluster cost i.e. how long boxes are used ?
  - Storage Cost during computation
- Sometimes we can decrease $ with acceptable increase in SLA (time)
  - Ex: Say running time

Balancing Time vs Co$t (Asana 5)

- At the end of day $ does matter but with acceptable SLA (time) in mind.
- Cost
  - Compute cluster cost i.e. how long boxes are used ?
  - Storage Cost during computation
- Sometimes we can decrease $ with acceptable increase in SLA (time)
  - Ex: Say running time
    - on Two c5.4x.large is 8 hours => X$ for 8 hour run time

Balancing Time vs Co$t (Asana 5)

- At the end of day $ does matter but with acceptable SLA (time) in mind.
- Cost
  - Compute cluster cost i.e. how long boxes are used ?
  - Storage Cost during computation
- Sometimes we can decrease $ with acceptable increase in SLA (time)
  - Ex: Say running time
    - on Two c5.4x.large is 8 hours => X$ for 8 hour run time
    - Theoretically on Two t3.2x.large is ~16 hours => Y$ for 16 hour run time (Y<X)

Balancing Time vs Co$t (Asana 5)

- At the end of day $ does matter but with acceptable SLA (time) in mind.
- Cost
  - Compute cluster cost i.e. how long boxes are used ?
  - Storage Cost during computation
- Sometimes we can decrease $ with acceptable increase in SLA (time)
  - Ex: Say running time
    - on Two c5.4x.large is 8 hours => X$ for 8 hour run time
    - Theoretically on Two t3.2x.large is ~16 hours => Y$ for 16 hour run time (Y<X)
    - If you are ok with the SLA of 16 hours, why not go for it ?

Exercise - *Box: t3.large, Xmx: 512m , Spark Local[4] v3.1.2*

Exercise - *Box: t3.large, Xmx: 512m , Spark Local[4] v3.1.2*

Dataset records = spark.session.orc.load("s3a://...").select(few columns); // 15gb in s3 folder

Dataset lowercase = records.map(.....transform to lowercase….);

lowercase.count(); //some action

Dataset Uppercase = records.map(.....transform to uppercase...);

Uppercase.count(); //some action

Running Time: **12.5 min**

Exercise - *Box: t3.large, Xmx: 512m , Spark Local[4] v3.1.2*

(***spark.hadoop.fs.s3a.connection.ssl.enabled=false***)

Dataset records = spark.session.orc.load("s3a://...").select(few columns); // 15gb in s3 folder

Dataset lowercase = records.map(.....transform to lowercase….);

lowercase.count(); //some action

Dataset Uppercase = records.map(.....transform to uppercase...);

Uppercase.count(); //some action

Running Time: **11.4min** 

Exercise - *Box: t3.large, Xmx: 512m , Spark Local[4] v3.1.2*

(***spark.hadoop.fs.s3a.connection.ssl.enabled=false***)

Dataset records = spark.session.orc.load("s3a://...").select(few columns); // 15gb in s3 folder

records.persist(DISK_ONLY); // if you have lot of free ram, try with memory

Dataset lowercase = records.map(.....transform to lowercase….);

lowercase.count(); //some action

Dataset Uppercase = records.map(.....transform to uppercase...);

Uppercase.count(); //some action

Running Time: **8.54 min** 

Exercise - *Box: t3.large, Xmx: 512m , Spark Local[4] v3.1.2*

(***aws s3 cp data to local disk & read from there- took 2min***)

Dataset records = spark.session.orc.load("file://...").select(few columns); // 15gb in s3 folder

Dataset lowercase = records.map(.....transform to lowercase….);

lowercase.count(); //some action

Dataset Uppercase = records.map(.....transform to uppercase...);

Uppercase.count(); //some action

Running Time: **5.9min**  *(including 2min for downloading data)*

Exercise - *Box: t3.large, Xmx: 512m , Spark Local[4] v3.1.2*

(***aws s3 cp data to local disk & read from there***)

Dataset records = spark.session.orc.load("file://...").select(few columns); // 15gb in s3 folder

records.persist(DISK_ONLY); // if you have lot of free ram, try with memory

Dataset lowercase = records.map(.....transform to lowercase….);

lowercase.count(); //some action

Dataset Uppercase = records.map(.....transform to uppercase...);

Uppercase.count(); //some action

Total Running Time: **6.8min**  (includes downloading data to disk. The best was 5.9min)

Exercise (Results)

Scenario	Minutes
Program as it is (from s3)	12.5
Program as it is (from s3) + ssl disable	11.4
Persisting disk (from s3)	8.6
Persist disk (from s3) + ssl disable	8.5
Aws cp to local	5.9
Aws cp to local + Persist (disk)	6.8
Aws cp to local + Persist (mem_ser) xms=6g	6.9
Aws cp to local + Persist (mem) xms=6g	6.7

**Aws s3 cp took 2 minutes*

Note: Please do run your own tests !

Explore your own code.

Finally, It’s about the Balance

So try out these
Asanas
for a cost-effective
Spark Life

Thank you

linkedin.com/in/213vishnu

twitter.com/sweetweet213

https://mash213.wordpress.com/