Apache Spark Monitoring and Debugging

Skills Network

Estimated time needed: 30 minutes

This lab will instruct you on how to monitor and debug a Spark application through the web UI.

Objectives

After completing this lab, you will be able to:

1. Start a Spark Standalone Cluster and connect with the PySpark shell.
2. Create a DataFrame and open the application web UI.
3. Debug a runtime error by locating the failed task in the web UI.
4. Run an SQL query to monitor, then scale up by adding another worker to the cluster.

Exercise 1 : Start a Spark Standalone Cluster

In this exercise, you will initialize a Spark Standalone Cluster with a Master and one Worker. Next, you will start a PySpark shell that connects to the cluster and open the Spark Application Web UI to monitor it. We will be using the Theia terminal to run commands and docker-based containers to launch the Spark processes.

Task A : Download Sample Data for Spark

1. Open a Theia terminal by clicking on the menu item `Terminal -> New Terminal`.
2. Use the following command to download the data set we will be using in this lab to the container running Spark.

```
1. 1

1. wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-BD0225EN-SkillsNetwork/labs/data/cars.csv
```

Copied! Executed!

Task B : Initialize the Cluster

1. Stop any previously running containers with the command:

```
1. 1

1. for i in `docker ps | awk '{print $1}' | grep -v CONTAINER`; do docker kill $i; done
```

Copied! Executed!

2. Remove any previously used containers:

   Ignore any errors that say "No such container"

```
1. 1

1. docker rm spark-master spark-worker-1 spark-worker-2
```

Copied! Executed!

3. Start the Spark Master server:

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8

1. docker run \
2.     --name spark-master \
3.     -h spark-master \
4.     -e ENABLE_INIT_DAEMON=false \
5.     -p 4040:4040 \
6.     -p 8080:8080 \
7.     -v `pwd`:/home/root \
8.     -d bde2020/spark-master:3.1.1-hadoop3.2
```

Copied! Executed!

4. Start a Spark Worker that will connect to the Master:

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7

1. docker run \
2.     --name spark-worker-1 \
3.     --link spark-master:spark-master \
4.     -e ENABLE_INIT_DAEMON=false \
5.     -p 8081:8081 \
6.     -v `pwd`:/home/root \
7.     -d bde2020/spark-worker:3.1.1-hadoop3.2
```

Copied! Executed!

Task C : Connect a PySpark Shell to the Cluster and Open the UI

1. Launch a PySpark shell in the running Spark Master container:

```
1. 1
2. 2
3. 3
4. 4

1. docker exec \
2.     -it `docker ps | grep spark-master | awk '{print $1}'` \
3.     /spark/bin/pyspark \
4.     --master spark://spark-master:7077
```

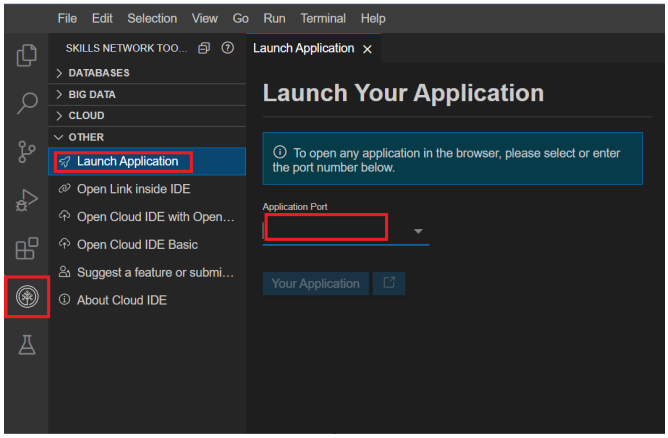
Copied! Executed!

2. Create a DataFrame in the shell with:

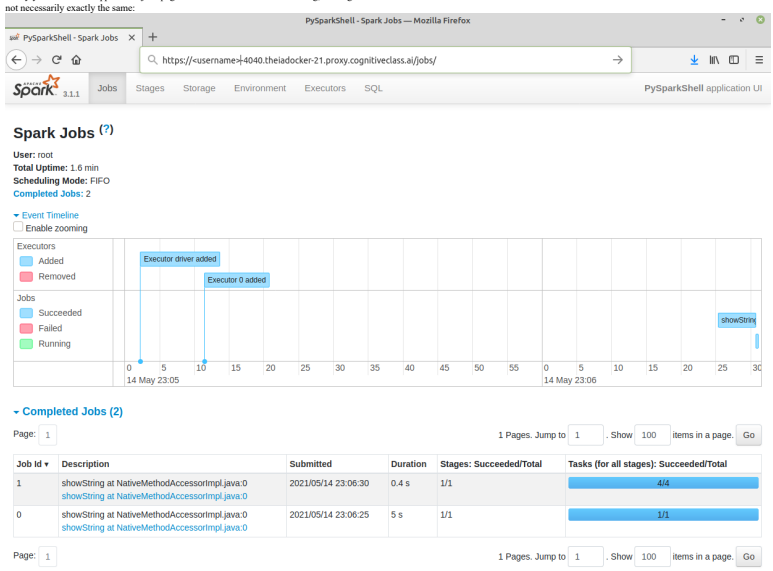
```
1. 1
2. 2
3. 3
4. 4

1. df = spark.read.csv("/home/root/cars.csv", header=True, inferSchema=True) \
2.     .repartition(32) \
3.     .cache()
4. df.show()
```

Copied!

3. Click on the Skills Network button on the left, it will open the "Skills Network Toolbox". Then click `OTHER` then `Launch Application`. From there you should be able to enter the port number as `4040` and launch the Spark Application UI in your browser.



4. Verify you can see the application jobs page that should look like the following, although not necessarily exactly the same:



Exercise 2 : Run an SQL Query and Debug in the Application UI

In this exercise, you will define a user-defined function (UDF) and run a query that results in an error. We will locate that error in the application UI and find the root cause. Finally, we will correct the error and re-run the query.

Task A : Run an SQL Query

1. Define a UDF to show engine type. Copy and paste the code and click `Enter`.

```
1
2
3
4
5
6
7
8
```

```
from pyspark.sql.functions import udf
import time

@udf("string")
def engine(cylinders):
    time.sleep(0.2)  # Intentionally delay task
    eng = {6: "V6", 8: "V8"}
    return eng[cylinders]
```

Copied!

2. Add the UDF as a column in the DataFrame

```
1
```

```
df = df.withColumn("engine", engine("cylinders"))
```

Copied!

3. Group the DataFrame by "cylinders" and aggregate other columns

```
1
```

```
dfg = df.groupby("cylinders")
```

Copied!

```
1
```

```
dfa = dfg.agg({"mpg": "avg", "engine": "first"})
```

Copied!

```
1
```

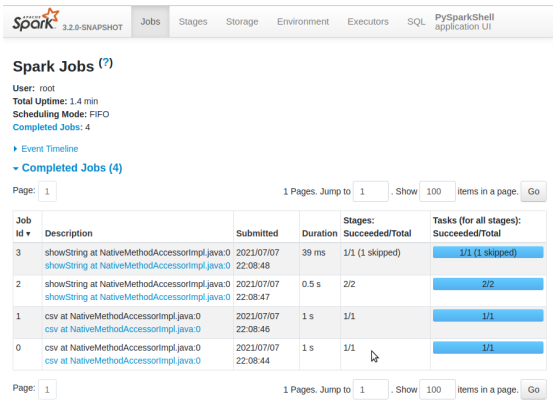
```
dfa.show()
```

Copied!

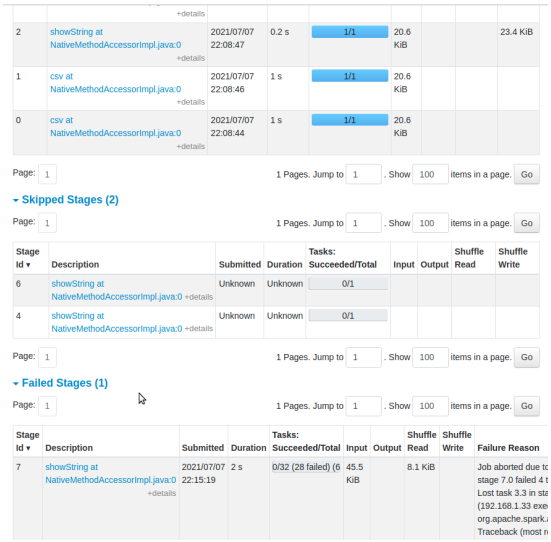
4. The query will have failed and you should see lots of messages and outputs in the console. The next task will be to locate the error in the Application UI and determine the root cause.

Task B : Debug the error in the Application UI

1. Find the error in the Application UI
Open UI to the Jobs, look at list of Failed Jobs, click on first job.



2. This will bring up the Job details with a list of stages for that job. In the list of Failed Stages, click on the first failed stage to show the stage details with a list of tasks for that stage.



	+details							
2	showString at NativeMethodAccessorImpl.java:0 +details	2021/07/07 22:08:47	0.2 s	1/1	20.6 KiB			23.4 KiB
1	csv at NativeMethodAccessorImpl.java:0 +details	2021/07/07 22:08:46	1 s	1/1	20.6 KiB			
0	csv at NativeMethodAccessorImpl.java:0 +details	2021/07/07 22:08:44	1 s	1/1	20.6 KiB			

Page: 1 1 Pages. Jump to 1 . Show 100 items in a page. Go

Skipped Stages (2)

Page: 1 1 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
6	showString at NativeMethodAccessorImpl.java:0 +details	Unknown	Unknown	0/1				
4	showString at NativeMethodAccessorImpl.java:0 +details	Unknown	Unknown	0/1				

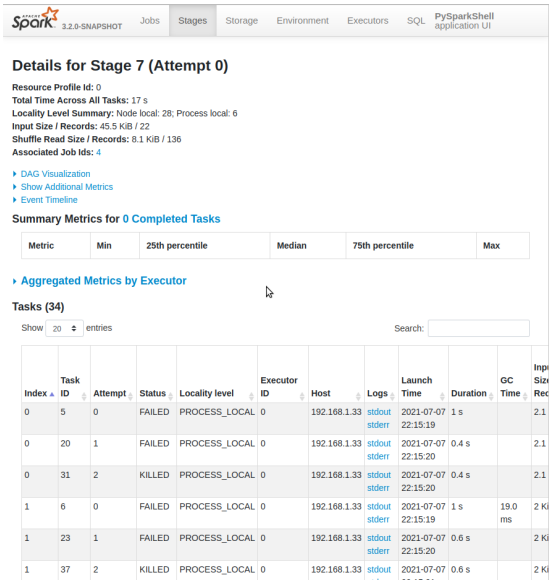
Page: 1 1 Pages. Jump to 1 . Show 100 items in a page. Go

Failed Stages (1)

Page: 1 1 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write	Failure Reason
7	showString at NativeMethodAccessorImpl.java:0 +details	2021/07/07 22:15:19	2 s	0/32 (28 failed) (6	45.5 KiB		8.1 KiB		Job aborted due to stage 7.0 failed 4 t Lost task 3.3 in sta (192.168.1.33 exe org.apache.spark. Traceback (most r

3. Here we see lots of failed tasks. Looking at the first one, the far right column shows details of the failure.



Spark 3.2.0-SNAPSHOT Jobs Stages Storage Environment Executors SQL PySparkShell application UI

Details for Stage 7 (Attempt 0)

Resource Profile Id: 0
Total Time Across All Tasks: 17 s
Locality Level Summary: Node local: 28; Process local: 6
Input Size / Records: 45.5 KiB / 22
Shuffle Read Size / Records: 8.1 KiB / 136
Associated Job Ids: 4

DAG Visualization
Show Additional Metrics
Event Timeline

Summary Metrics for 0 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max

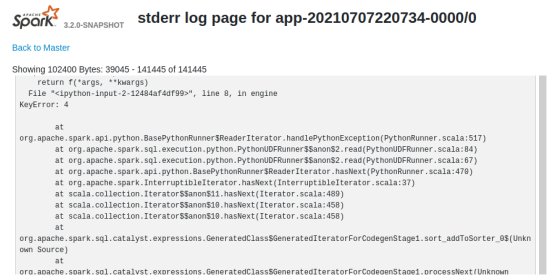
Aggregated Metrics by Executor

Tasks (34)

Show 20 entries Search:

Index	Task ID	Attempt	Status	Locality level	Executor ID	Host	Logs	Launch Time	Duration	GC Time	Inp Size Rec
0	5	0	FAILED	PROCESS_LOCAL	0	192.168.1.33	stdout stderr	2021-07-07 22:15:19	1 s		2.1
0	20	1	FAILED	PROCESS_LOCAL	0	192.168.1.33	stdout stderr	2021-07-07 22:15:20	0.4 s		2.1
0	31	2	KILLED	PROCESS_LOCAL	0	192.168.1.33	stdout stderr	2021-07-07 22:15:20	0.4 s		2.1
1	6	0	FAILED	PROCESS_LOCAL	0	192.168.1.33	stdout stderr	2021-07-07 22:15:19	1 s	19.0 ms	2 Ki
1	23	1	FAILED	PROCESS_LOCAL	0	192.168.1.33	stdout stderr	2021-07-07 22:15:20	0.6 s		2 Ki
1	37	2	KILLED	PROCESS_LOCAL	0	192.168.1.33	stdout	2021-07-07	0.6 s		2 Ki

Click to expand the details.



Spark 3.2.0-SNAPSHOT **stderr log page for app-20210707220734-0000/0**

Back to Master

Showing 102400 Bytes: 39045 - 141445 of 141445

```
    return f(*args, **kwargs)
  File "<ipython-input-2-12484af4df99>", line 8, in engine
KeyError: 4

        at
org.apache.spark.api.python.BasePythonRunner$ReaderIterator.handlePythonException(PythonRunner.scala:517)
        at org.apache.spark.sql.execution.python.PythonUDFRunner$$anon$2.read(PythonUDFRunner.scala:84)
        at org.apache.spark.sql.execution.python.PythonUDFRunner$$anon$2.read(PythonUDFRunner.scala:67)
        at org.apache.spark.api.python.BasePythonRunner$ReaderIterator.hasNext(PythonRunner.scala:470)
        at org.apache.spark.InterruptibleIterator.hasNext(InterruptibleIterator.scala:37)
        at scala.collection.Iterator$$anon$11.hasNext(Iterator.scala:489)
        at scala.collection.Iterator$$anon$10.hasNext(Iterator.scala:458)
        at scala.collection.Iterator$$anon$10.hasNext(Iterator.scala:458)
        at
org.apache.spark.sql.catalyst.expressions.GeneratedClass$GeneratedIteratorForCodegenStage1.sort_addToSorter_0$(Unkn
own Source)
        at
org.apache.spark.sql.catalyst.expressions.GeneratedClass$GeneratedIteratorForCodegenStage1.processNext(Unknown
```

Scroll down a little until you can see the last part of the Python error that shows the cause. You should be able to see this was caused by a KeyError in our UDF `engine()`.

You could also view these errors by looking at the column that has links to the logs and click on "std err" to show the standard error log.

Close the PySpark browser tab.

4. In the Theia terminal, fix the UDF by adding an entry to the dictionary of engine types and provide a default for all other types. Copy and paste this code and click `Enter`.

```
1
2
3
4
5
```

```
@udf("string")
def engine(cylinders):
    time.sleep(0.2)  # Intentionally delay task
    eng = {4: "inline-four", 6: "V6", 8: "V8"}
    return eng.get(cylinders, "other")
```

Copied!

5. Re-run the query. You will have to add the "engine" column again and enter the query since we changed the UDF.

```
1
```

```
df = df.withColumn("engine", engine("cylinders"))
```

Copied!

```
1
```

```
dfg = df.groupby("cylinders")
```

Copied!

```
1
```

```
dfa = dfg.agg({"mpg": "avg", "engine": "first"})
```

Copied!

```
1
```

```
dfa.show()
```

Copied!

Once the query completes without errors, you should see output similar to this.

```
1
2
3
4
5
6
7
8
9
```

```
+---------+-----------------+-------------+
|cylinders|         avg(mpg)|first(engine)|
+---------+-----------------+-------------+
|        6|19.985714285714288|           V6|
|        3|            20.55|        other|
|        5|27.366666666666664|        other|
|        4|29.286764705882348|  inline-four|
|        8|14.963107961165506|           V8|
+---------+-----------------+-------------+
```

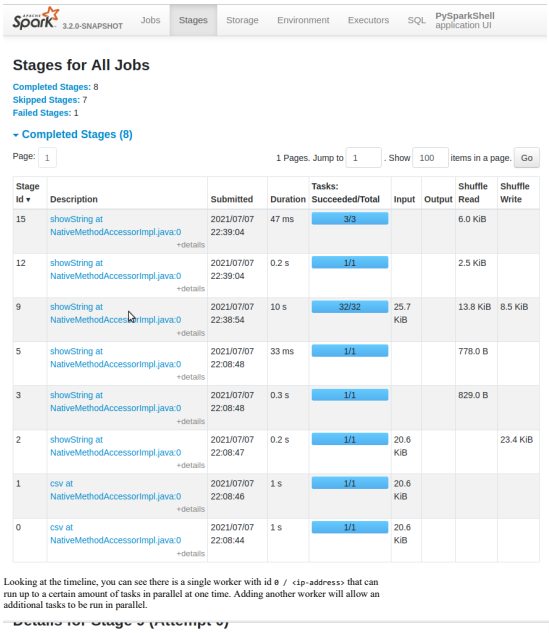
Copied!

Exercise 3 : Monitor Application Performance with the UI

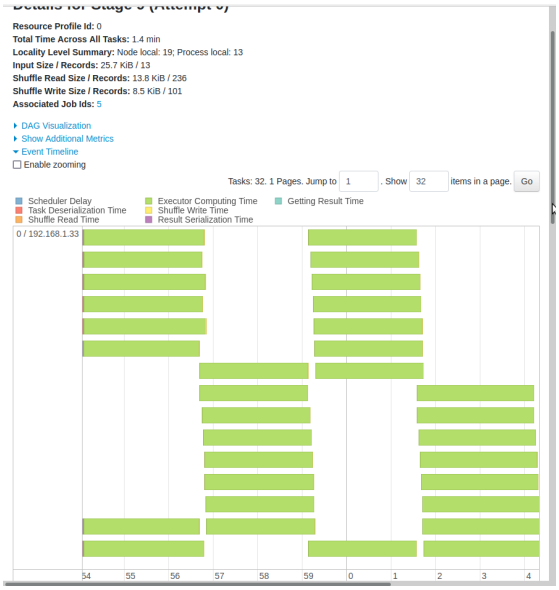
Now that we have run our query successfully, we will scale up our application by adding a worker to the cluster. This will allow the cluster to run more tasks in parallel and improve the overall performance.

Task A : Add a Worker to the Cluster

1. View the Stages tab, then click on the stage with 32 tasks. In that stage our UDF is being applied to each partition of the DataFrame.



Looking at the timeline, you can see there is a single worker with id `0 / <ip-address>` that can run up to a certain amount of tasks in parallel at one time. Adding another worker will allow an additional tasks to be run in parallel.



2. Open a new Theia terminal by clicking on the menu item `Terminal -> New Terminal`.

3. Add a second worker to the cluster with the command in the new terminal:

```
1
2
3
4
5
6
```

```
docker run \
    --name spark-worker-2 \
    --link spark-master:spark-master \
    -e ENABLE_INIT_DAEMON=false \
    -p 8082:8082 \
    -d bde2020/spark-worker:3.1.1-hadoop3.2
```

Copied! Executed!

4. If the command is successful, there will be a single output showing the container id:

```
1
2
3
```

```
4. 4
5. 5
6. 6
7. 7
8. 8
```

```
1. theia@theiadocker-user:/home/project$ docker run \
2. >     --name spark-worker-2 \
3. >     --link spark-master:spark-master \
4. >     -e ENABLE_INIT_DAEMON=false \
5. >     -p 8082:8082 \
6. >     -d bde2020/spark-worker:3.1.1-hadoop3.2
7. 1935a71827668ae3476e6a16f0bebcd4c2a342a21271dc22be487aa1b1731708
8. theia@theiadocker-user:/home/project$
```

Copied!

5. Click back to the first terminal that has the PySpark shell open to continue.

Task B : Re-run the query and check performance

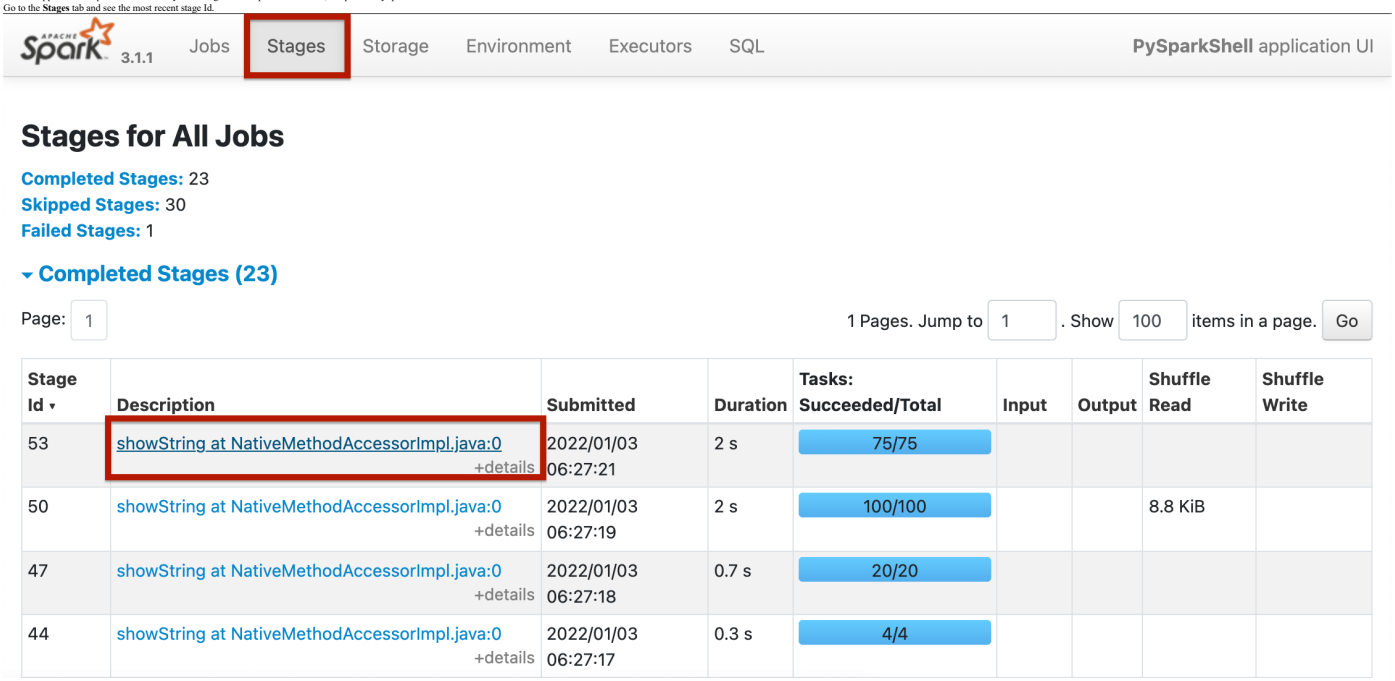
1. Re-run the query, this time we can simply call `show()` again:

```
1. 1
```

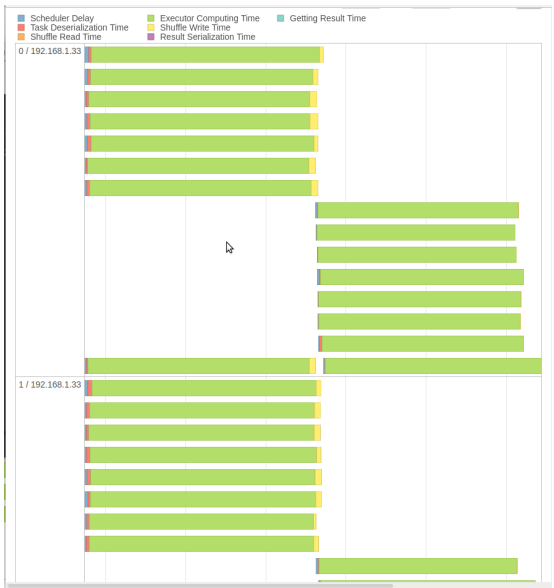
```
1. dfa.show()
```

Copied!

2. Launch Application on port number `4040` by following the same process as above, to open the PySpark browser.
   Go to the **Stages** tab and see the most recent stage Id.



3. You will see that the additional worker with id `1 / <ip-address>` is listed and now allows more tasks
   to be run in parallel. The task timeline should look similar to the following.



Author(s)

Aije

Other Contributor(s)

Lavanya

Changelog

Date	Version	Changed by	Change Description
2021-07-16	0.1	Aije	Initial version created
2022-01-03	0.2	Lavanya	Changed the instructions for second node
2022-09-01	0.3	K Sundararajan	Updated instructions for `Launch Application` as per new Theia IDE