
Big Data Infrastructure and Cloud Computing

Evaluation Practical Work Report

Professor - Erwan Rouzel

Poorva Vivek MORVEKAR

Master of Science in Artificial Intelligent Systems

Spring 2020

Introduction

The aim of the project is to perform Spark operations on the complete works of Shakespeare.

The document is available here: <https://ocw.mit.edu/ans7870/6/6.006/s08/lecturenotes/files/t8.shakespeare.txt>

We will be using the below tools:

Recommended IDE: PyCharm Community Edition

Amazon EMR Cluster:

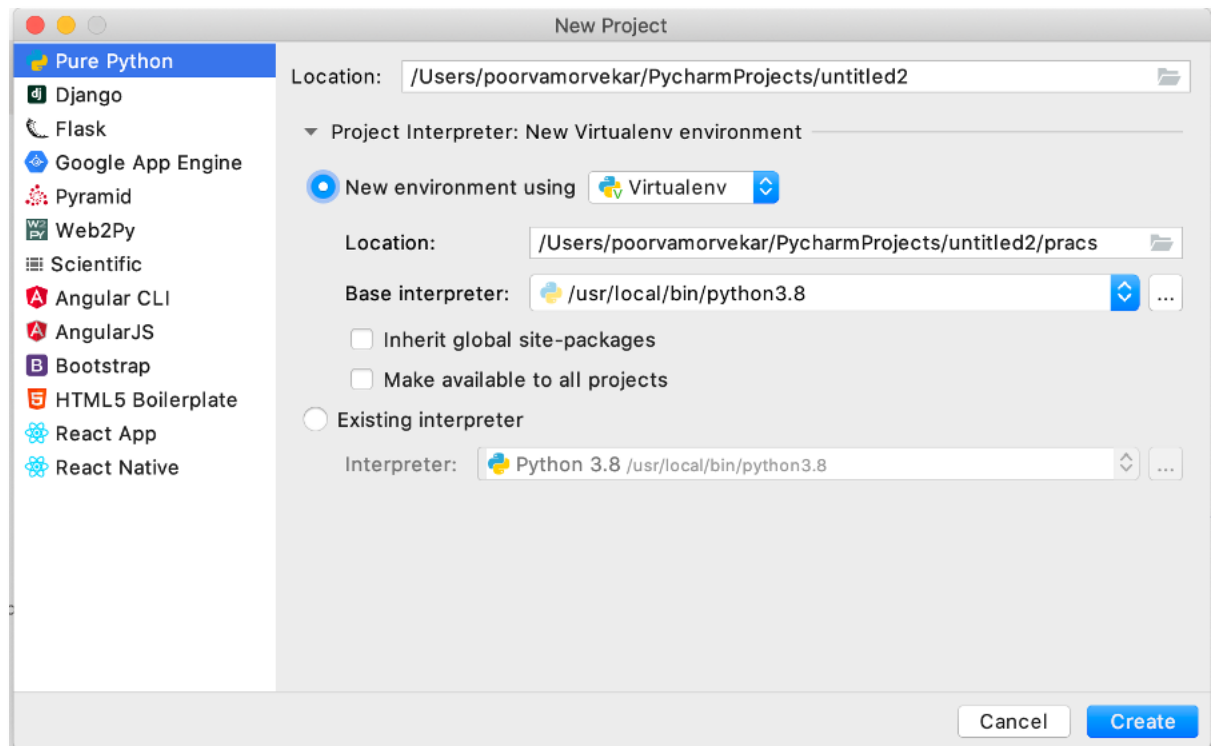
It is a managed cluster platform that simplifies running big data frameworks such as Apache Hadoop and Apache Spark on AWS to process and analyze vast amount of data.

Amazon S3:

Simple Storage Service (S3) is a cloud storage resource available on AWS. It offers S3 buckets which are similar to file folders, stores objects which consists of data and descriptive metadata.

Create a virtual environment in PyCharm and set it up to use Python 3.8

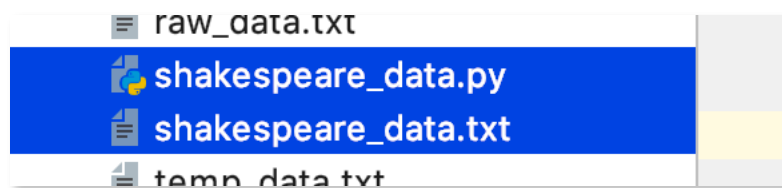
First we need to set up a virtual environment in PyCharm IDE by creating a new python project.



Write a script in Python 'shakespeare_words.py' which generates a list of the words contained in the complete works of Shakespeare and output it to a file. Name this file 'shakespeare_data.txt'

A python file 'shakespeare_words.py' is created in which the unnecessary texts have been removed.

The required texts from the file has been saved by creating a new output file 'shakespeare_data.txt'.



Here we can see the files 'shakespeare_words.py' and 'shakespeare_data.txt'.

Install PySpark using the command line

PySpark had already been installed during the implementation.

Load your 'shakespeare_data.txt' file in the shell and count the number of words in the file.

We will count the number of words in the file by creating a dataframe object and using the count() method.

```
>>> df=spark.read.text('shakespeare_data.txt')
>>> df.count()
922603
>>> █
```

Here we can see that there are 922603 words present in the file 'shakespeare_data.txt'.

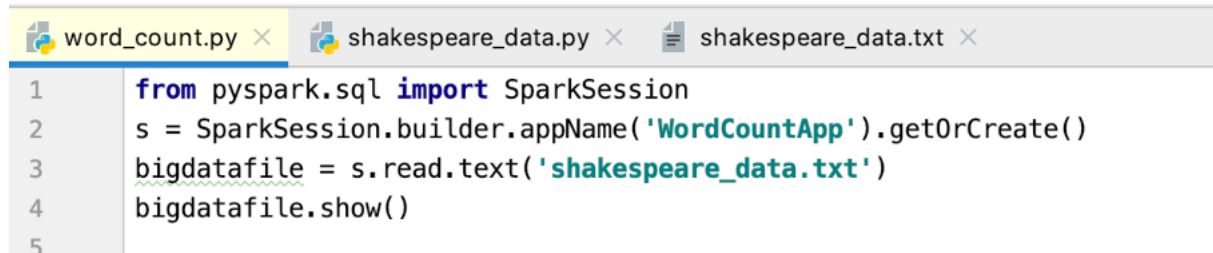
Exit the shell

```
>>> exit()  
(pracs) (base) poorvamorvekar@Poorvas-MBP BigData %
```

We can exit from the Spark shell by just typing in the `exit()` command.

Create a 'word_count.py' file and do the same thing like with the shell (reading the text file and displaying the number of words). For that, you need to create an instance of spark in your script using a SparkSession.

Repeating the same previous step in the python file.

A screenshot of a code editor window with three tabs: 'word_count.py', 'shakespeare_data.py', and 'shakespeare_data.txt'. The 'word_count.py' tab is active and shows the following Python code:

```
1 from pyspark.sql import SparkSession
2 s = SparkSession.builder.appName('WordCountApp').getOrCreate()
3 bigdatafile = s.read.text('shakespeare_data.txt')
4 bigdatafile.show()
5
```

Here we open a SparkSession and read the 'shakespeare_data.txt' file and display the total number of words present in it.

Then add spark code to your script in order to show:

- ❖ The first three values in the text file

```
# First three values in text file
print(bigdatafile.take(3))
```

Output:

```
[Row(value='EWITH'), Row(value='PERMISSION'), Row(value='ELECTRONIC')]
```

- ❖ The 10 longest words, showing their length

```
# 10 longest words and their length
length = s.sql('SELECT word, length(word) AS len FROM data ORDER BY len DESC LIMIT 10')
length.show()
```

Output:

```
+-----+-----+
|          word|len|
+-----+-----+
|honorificabilitud...| 27|
|ShakespeareSHAKES...| 22|
|AttendantsSHAKESP...| 21|
|AttendantsSHAKESP...| 21|
|AttendantsSHAKESP...| 21|
|AttendantsSHAKESP...| 21|
|AttendantsSHAKESP...| 21|
|AttendantsSHAKESP...| 21|
|MessengersSHAKESP...| 21|
|ExeuntKING_HENRY_...| 21|
+-----+-----+
```

-
- ❖ The 10 words having the highest number of occurrences, with their number of occurrences

```
# 10 highest occurred words and count of occurrence
count = s.sql("SELECT word, count(word) AS word_count FROM data GROUP BY word ORDER BY word_count DESC LIMIT 10")
count.show()
```

Output:

word	word_count
the	23241
I	22225
and	18611
to	16337
of	15685
a	12779
you	12160
my	10838
in	10003
d	8954

Run your script locally and observe the results.

The screenshots of the outputs are provided in the previous section.

Create an Amazon EMR cluster (keep default parameters).

We need to create an Amazon EMR cluster by logging into awseducate.com and going into the EMR section.

The screenshot displays the Amazon EMR console for a cluster named 'Poorvafinalcluster'. The cluster is in the 'Waiting' state, indicated by a green status label and the message 'Cluster ready after last step completed.' The console features a navigation bar with tabs for Summary, Application user interfaces, Monitoring, Hardware, Configurations, Events, Steps, and Bootstrap actions. The 'Summary' tab is active, showing the following details:

- ID:** j-1FP0FL1FMX2MS
- Creation date:** 2021-02-11 20:33 (UTC+1)
- Elapsed time:** 5 minutes
- After last step completes:** Cluster waits
- Termination protection:** Off [Change](#)
- Tags:** -- [View All / Edit](#)
- Master public DNS:** ec2-54-80-237-99.compute-1.amazonaws.com [Connect to the Master Node Using SSH](#)

The 'Configuration details' section shows:

- Release label:** emr-6.2.0
- Hadoop distribution:** Amazon
- Applications:** Spark 3.0.1, Zeppelin 0.9.0
- Log URI:** s3://aws-logs-522833485400-us-east-1/elasticmapreduce/ [View](#)
- EMRFS consistent view:** Disabled
- Custom AMI ID:** --

The 'Application user interfaces' section shows:

- Persistent user interfaces:** [Spark history server, YARN timeline server](#)
- On-cluster user interfaces:** Not Enabled [Enable an SSH Connection](#)

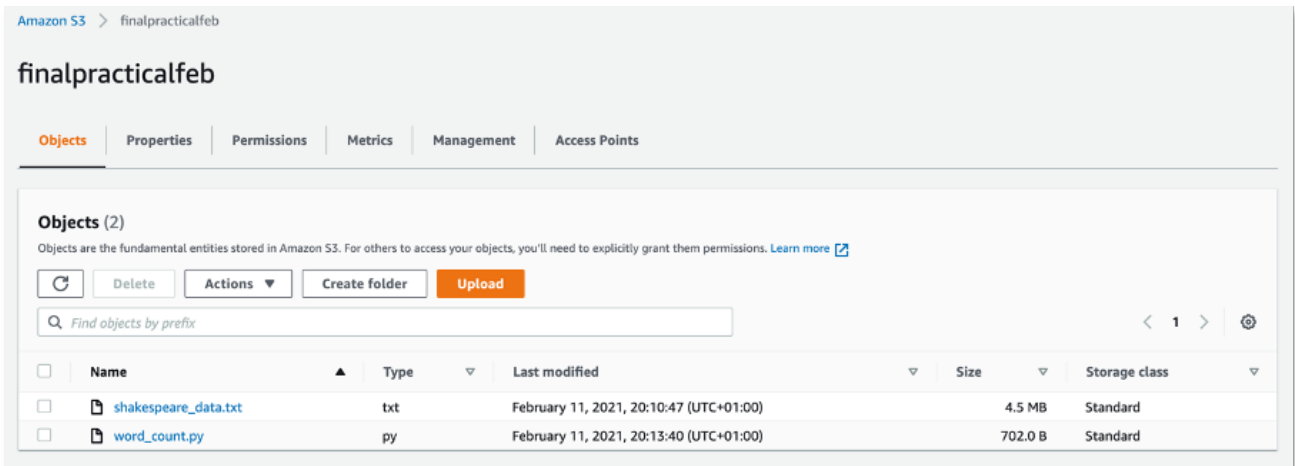
The 'Network and hardware' section shows:

- Availability zone:** us-east-1d
- Subnet ID:** [subnet-71ec5c2e](#)
- Master:** Running 1 m5.xlarge
- Core:** Running 2 m5.xlarge
- Task:** --
- Cluster scaling:** Not enabled

An EMR cluster by the name 'Poorvafinalcluster' has been created by keeping all the parameters as default.

Create a S3 bucket and upload your data to the bucket.

We need to create an S3 bucket by logging into awseducate.com and going into the S3 section.



S3 bucket has been created by the name 'finalpracticalfeb' and the files 'shakespeare_data.txt' and 'word_count.py' has been uploaded into it.

Open the Spark History Server user interface.

Run your script on the cluster using the graphical user interface. Observe the jobs being run in the Spark History Server.

First, we need to go to the steps section of the EMR cluster to add a step.

Add step

Step type: Spark application

Name: WordCountApp

Deploy mode: Cluster

Spark-submit options

Application location*: s3://finalpracticalfeb/word_count.py

Arguments

Action on failure: Continue

Cancel Add

In the 'Step type' we need to select 'Spark application' and select the python file which is located in the S3 bucket.

After adding the step you can see the status of the step change.

ID	Name	Status	Start time (UTC+1)	Elapsed time	Log files
s-3EBK5H91HQM2Q	WordCountApp	Pending		--	View logs
JAR location : command-runner.jar					
Main class : None					
Arguments : spark-submit --deploy-mode cluster s3://finalpracticalfeb/word_count.py					
Action on failure: Continue					

Filter: All steps		Filter steps ...		2 steps (all running)			
	ID	Name	Status	Start time (UTC+1)	Elapsed time	Log files 🔗	
<div><div></div><div></div></div>	s-3EBK5H91HQM2Q	WordCountApp	Completed	2021-02-11 20:48 (UTC+1)	36 seconds	View logs	
<div>JAR location : command-runner.jar</div> <div>Main class : None</div> <div>Arguments : spark-submit --deploy-mode cluster s3://finalpracticalfeb/word_count.py</div> <div>Action on failure: Continue</div>							

The page needs to be refreshed for the status to change from 'pending' to 'completed'.

After the status has been changed to 'completed',

In the summary page of the EMR cluster, we need to click into the Spark History Server

Observe the final state in Spark History Server.

In the Spark History Server,

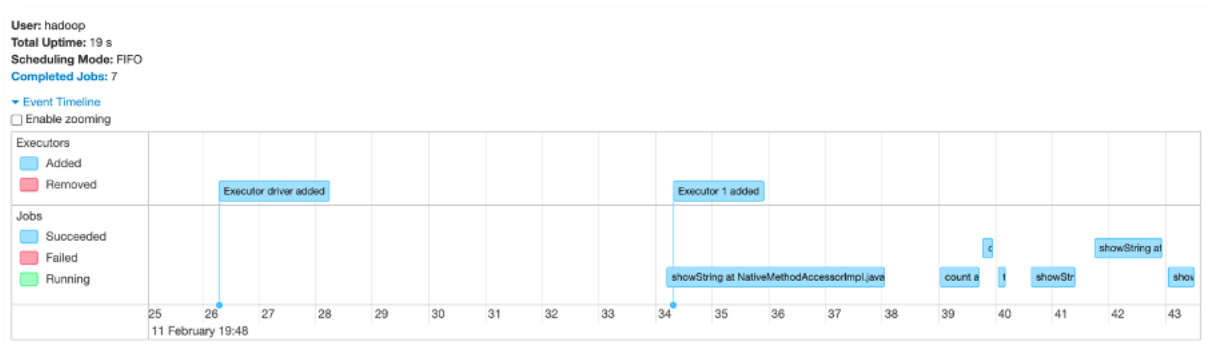
▼ Completed Jobs (7)

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

Job Id ▼	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
6	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2021/02/11 19:48:43	0.4 s	2/2 (1 skipped)	27/27 (2 skipped)
5	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2021/02/11 19:48:41	1 s	1/1	2/2
4	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2021/02/11 19:48:40	0.8 s	2/2	3/3
3	take at word_count.py:13 take at word_count.py:13	2021/02/11 19:48:40	0.1 s	1/1	1/1
2	count at NativeMethodAccessorImpl.java:0 count at NativeMethodAccessorImpl.java:0	2021/02/11 19:48:39	0.2 s	1/1 (1 skipped)	1/1 (2 skipped)
1	count at NativeMethodAccessorImpl.java:0 count at NativeMethodAccessorImpl.java:0	2021/02/11 19:48:39	0.7 s	1/1	2/2
0	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2021/02/11 19:48:34	4 s	1/1	1/1

The above screenshot shows the completed jobs.

Here we can see that it took 27 tasks to complete the job.



The above screenshot shows the event timeline.

Here we can see that the total uptime taken is 19 secs.

Propose and test different optimizations to improve the performances. Show the performance gains attained with each optimization tested.

We need to optimize the performance (decrease the number of tasks from 27 to lesser value).

For that we can make use of repartition() method.

*The **repartition** method can be used to either increase or decrease the number of partitions in a DataFrame. The whole data is taken out from existing partitions and equally distributed into newly formed partitions. Repartition helps to partition the data by balancing the amount of data that has to be mapped across the cluster.*

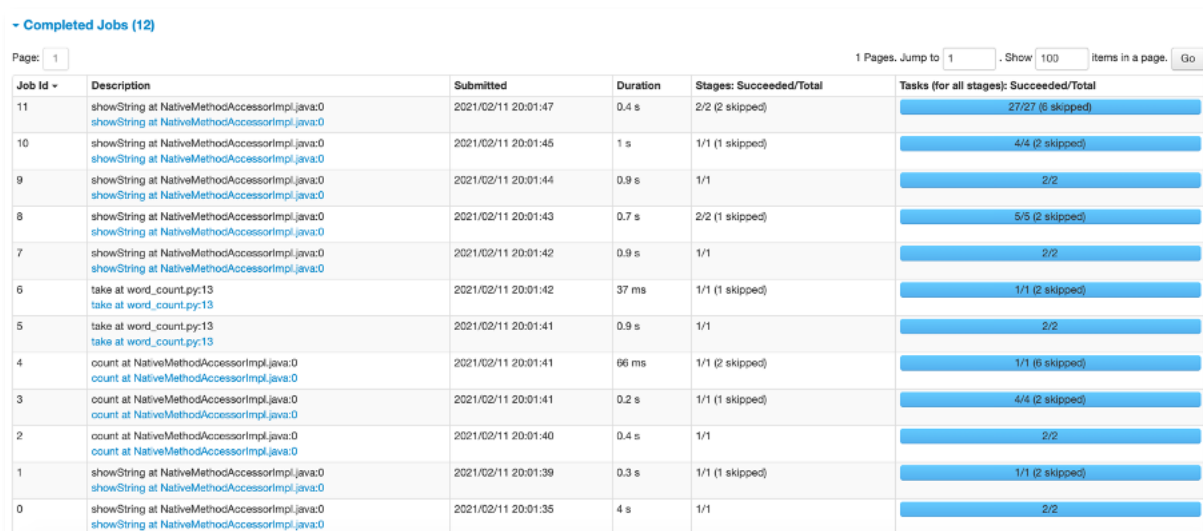
```
from pyspark.sql import SparkSession
s = SparkSession.builder.appName('WordCountApp').getOrCreate()

bigdatafile = s.read.text('s3://finalpracticalfeb/shakespeare_data.txt').repartition(4)

bigdatafile.show()
```

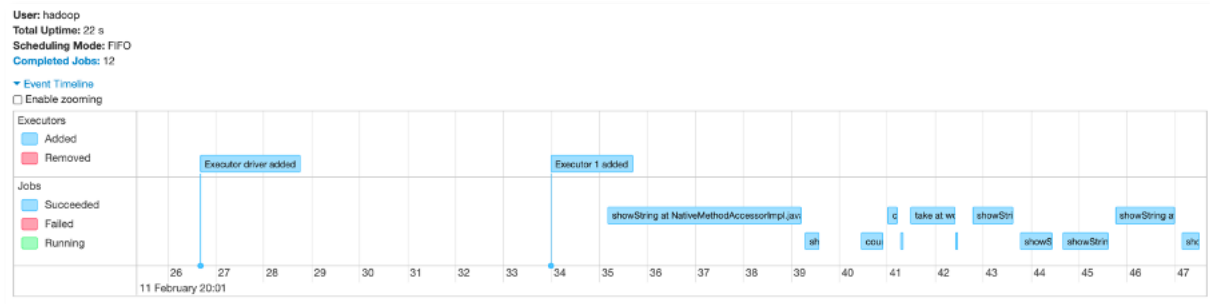
Once the script has been updated with repartition(4), the python file 'word_count.py' in the S3 bucket has to be replaced with this new file containing the repartition() method.

Once that is done, again a new step has to be added and the Spark History Server has to be observed for final state.



Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
11	showString at NativeMethodAccessorImpl.java:0	2021/02/11 20:01:47	0.4 s	2/2 (2 skipped)	27/27 (6 skipped)
10	showString at NativeMethodAccessorImpl.java:0	2021/02/11 20:01:45	1 s	1/1 (1 skipped)	4/4 (2 skipped)
9	showString at NativeMethodAccessorImpl.java:0	2021/02/11 20:01:44	0.9 s	1/1	2/2
8	showString at NativeMethodAccessorImpl.java:0	2021/02/11 20:01:43	0.7 s	2/2 (1 skipped)	5/5 (2 skipped)
7	showString at NativeMethodAccessorImpl.java:0	2021/02/11 20:01:42	0.9 s	1/1	2/2
6	take at word_count.py:13	2021/02/11 20:01:42	37 ms	1/1 (1 skipped)	1/1 (2 skipped)
5	take at word_count.py:13	2021/02/11 20:01:41	0.9 s	1/1	2/2
4	count at NativeMethodAccessorImpl.java:0	2021/02/11 20:01:41	66 ms	1/1 (2 skipped)	1/1 (6 skipped)
3	count at NativeMethodAccessorImpl.java:0	2021/02/11 20:01:41	0.2 s	1/1 (1 skipped)	4/4 (2 skipped)
2	count at NativeMethodAccessorImpl.java:0	2021/02/11 20:01:40	0.4 s	1/1	2/2
1	showString at NativeMethodAccessorImpl.java:0	2021/02/11 20:01:39	0.3 s	1/1 (1 skipped)	1/1 (2 skipped)
0	showString at NativeMethodAccessorImpl.java:0	2021/02/11 20:01:35	4 s	1/1	2/2

Here we can still see that the number of tasks has not been reduced.



The event timeline shows that the total time taken after repartition(4) is 22 secs, which is more.

I tried increasing the repartitioning but still the number of tasks remained same.

Then I tried using another method, coalesce()

*The **coalesce** method reduces the number of partitions in a DataFrame. Coalesce avoids full shuffle, instead of creating new partitions, it shuffles the data using Hash Partitioner (Default), and adjusts into existing partitions, this means it can only decrease the number of partitions.*

```
from pyspark.sql import SparkSession
s = SparkSession.builder.appName('WordCountApp').getOrCreate()

bigdatafile = s.read.text('s3://finalpracticalfeb/shakespeare_data.txt').coalesce(10)

bigdatafile.show()
```

Once the script has been updated with coalesce(10), the python file 'word_count.py' in the S3 bucket has to be replaced with this new file containing the coalesce() method.

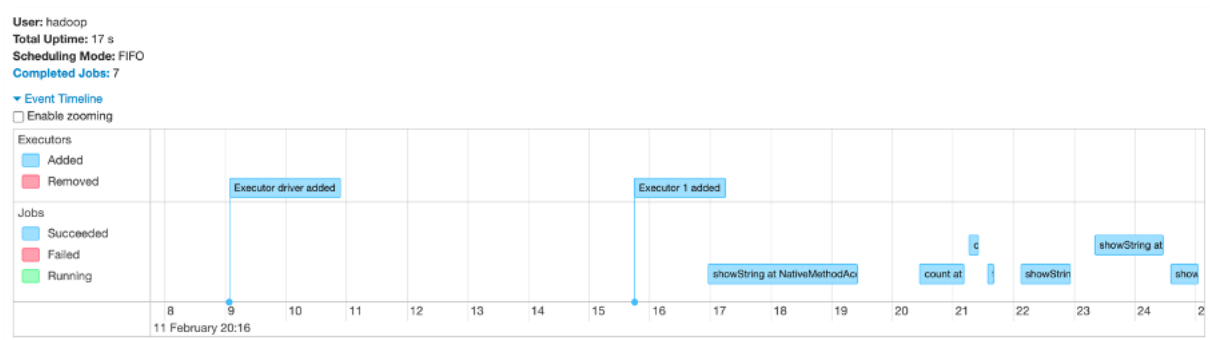
Once that is done, again a new step has to be added and the Spark History Server has to be observed for final state.

▼ Completed Jobs (7)

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

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
6	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2021/02/11 20:16:24	0.5 s	2/2 (1 skipped)	27/27 (2 skipped)
5	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2021/02/11 20:16:23	1 s	1/1	2/2
4	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2021/02/11 20:16:22	0.8 s	2/2	3/3
3	take at word_count.py:15 take at word_count.py:15	2021/02/11 20:16:21	0.1 s	1/1	1/1
2	count at NativeMethodAccessorImpl.java:0 count at NativeMethodAccessorImpl.java:0	2021/02/11 20:16:21	0.2 s	1/1 (1 skipped)	1/1 (2 skipped)
1	count at NativeMethodAccessorImpl.java:0 count at NativeMethodAccessorImpl.java:0	2021/02/11 20:16:20	0.7 s	1/1	2/2
0	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2021/02/11 20:16:16	2 s	1/1	1/1

Even after trying `coalesce(10)`, the number of tasks failed to decrease.



But the event timeline showed total uptime as 17 secs.

Using `coalesce()`, the total uptime has been reduced from 22 secs to 17 secs.

References:

- ❖ <https://www.jetbrains.com/help/pycharm/creating-virtual-environment.html>
- ❖ <https://spark.apache.org/docs/latest/>
- ❖ <https://ocw.mit.edu/ans7870/6/6.006/s08/lecturenotes/files/t8.shakespeare.txt>
- ❖ <https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-gs.html>
- ❖ <https://mageswaran1989.medium.com/spark-optimizations-for-advanced-users-spark-cheat-sheet-d74464618c20>
- ❖ <https://sparkbyexamples.com/spark/spark-performance-tuning/>
- ❖ <https://blog.knoldus.com/apache-spark-repartitioning-v-s-coalesce/>