**Table of contents**

[Introduction](#_rxn089wjzoxr)

[Part 1: Environment Setup & Software installation](#_yl4r3y8y6zv)

[Part 2: Simple Spark application](#udh8jem1m73p)

[Part 3: PageRank](#k1gmmadbdbg1)

[Datasets tested](#_xp85f1uy8xbf)

[Task 1. Implementation](#_jh9wpr3dezjt)

[Task 2. Adding custom RDD partitioning](#_k6qnsdsncics)

[Task 3. Persist Data](#_nigtbbnt8z72)

[Task 4. Fault Tolerance](#_80wzrwx0sv63)

[Team contributions](#_az3pvke0o0es)

## **Introduction**

For this assignment we first deployed Apache Hadoop (underlying file system) and Apache Spark (execution engine) on a cloudlab experiment. We implemented two applications in Spark, a simple sorting routine for dataframes and the page rank algorithm. We experimented running the page rank method over different workloads and here we report some performance measures and discuss our observations in detail.

## **Part 1: Environment Setup & Software installation**

**A 2-node cluster, each with 30GB RAM and 5 cores was used:**

[**https://www.cloudlab.us/status.php?uuid=9d4be33e-c374-11ee-9f39-e4434b2381fc**](https://www.cloudlab.us/status.php?uuid=9d4be33e-c374-11ee-9f39-e4434b2381fc)

**Node 0 init routine:**

* Java installation:
  + sudo apt update
  + sudo apt -y install openjdk-8-jdk
* Public key setup
  + ssh-keygen -t rsa -f ~/.ssh/id\_rsa -N ""
  + cat ~/.ssh/id\_rsa.pub
  + <copy public key to clipboard>
  + key="<paste public key here>"
  + echo -e $key >> ~/.ssh/authorized\_keys
* Parallel ssh test:
  + hostname0=[ivanjaen@node0.group18-asg1.uwmadison744-s24-pg0.wisc.cloudlab.us](mailto:ivanjaen@node0.group18-asg1.uwmadison744-s24-pg0.wisc.cloudlab.us)
  + hostname1=[ivanjaen@node1.group18-asg1.uwmadison744-s24-pg0.wisc.cloudlab.us](mailto:ivanjaen@node1.group18-asg1.uwmadison744-s24-pg0.wisc.cloudlab.us)
  + hostname2=[ivanjaen@node2.group18-asg1.uwmadison744-s24-pg0.wisc.cloudlab.us](mailto:ivanjaen@node2.group18-asg1.uwmadison744-s24-pg0.wisc.cloudlab.us)
  + echo -e "${hostname0}\n${hostname1}\n${hostname2}" > hostnames\_file.txt
  + parallel-ssh -i -h hostnames\_file.txt -O StrictHostKeyChecking=no pwd

**Nodes 1 and 2 init routine:**

* Java installation:
  + sudo apt update
  + sudo apt -y install openjdk-8-jdk
* Public key setup:
  + key="<paste public key here>"
  + echo -e $key >> ~/.ssh/authorized\_keys

**Part 0: mounting disks**

* Mounting data partition:
  + parallel-ssh -i -h hostnames\_file.txt -O StrictHostKeyChecking=no sudo mkfs.ext4 /dev/xvda4
  + parallel -ssh -i -h hostnames\_file.txt -O StrictHostKeyChecking=no sudo mkdir -p /mnt/data
  + parallel-ssh -i -h hostnames\_file.txt -O StrictHostKeyChecking=no sudo mount /dev/xvda4 /mnt/data
* Verifying is working:
  + parallel-ssh -i -h hostnames\_file.txt -O StrictHostKeyChecking=no df -h | grep "data"
* Change directory ownership:
  + parallel-ssh -i -h hostnames\_file.txt -O StrictHostKeyChecking=no sudo chown -R ivanjaen /mnt/data
* Validate permissions:
  + parallel-ssh -i -h hostnames\_file.txt -O StrictHostKeyChecking=no ls -l /mnt

**Part 1: software deployment**

**Apache hadoop (install)**

* wget https://dlcdn.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
* tar zvxf hadoop-3.3.6.tar.gz

**Apache hadoop (config)**

* hadoop-3.3.6/etc/hadoop/core-site.xml:

<configuration>

<property>

<name>fs.default.name</name>

<value>hdfs://namenode\_IP:9000</value>

</property>

</configuration>

* hadoop-3.3.6/etc/hadoop/hdfs-site.xml

parallel-ssh -i -h hostnames\_file.txt -O StrictHostKeyChecking=no mkdir /mnt/data/namenode/

parallel-ssh -i -h hostnames\_file.txt -O StrictHostKeyChecking=no mkdir /mnt/data/datanode/

<configuration>

<property>

<name>dfs.namenode.name.dir</name>

<value>/mnt/data/namenode/</value>

</property>

<property>

<name>dfs.datanode.data.dir</name>

<value>/mnt/data/datanode/</value>

</property>

</configuration>

* hadoop-3.3.6/etc/hadoop/hadoop-env.sh

export JAVA\_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre

* hadoop-3.3.6/etc/hadoop/workers

10.10.1.1

10.10.1.2

10.10.1.3

* Replicate hadoop configurations to all machines:
  + parallel-scp -h hostnames\_file.txt -O StrictHostKeyChecking=no hadoop-3.3.6/etc/hadoop/core-site.xml ~/hadoop-3.3.6/etc/hadoop/
  + parallel-scp -h hostnames\_file.txt -O StrictHostKeyChecking=no hadoop-3.3.6/etc/hadoop/hdfs-site.xml ~/hadoop-3.3.6/etc/hadoop/
  + parallel-scp -h hostnames\_file.txt -O StrictHostKeyChecking=no hadoop-3.3.6/etc/hadoop/hadoop-env.sh ~/hadoop-3.3.6/etc/hadoop/
  + parallel-scp -h hostnames\_file.txt -O StrictHostKeyChecking=no hadoop-3.3.6/etc/hadoop/workers ~/hadoop-3.3.6/etc/hadoop/
* Format the namenode
  + Add hadoop to the path in node0 only:
    - export PATH=$PATH:hadoop-3.3.6/bin:hadoop-3.3.6/sbin
  + hdfs namenode -format
  + start-dfs.sh
  + stop-dfs.sh
* Check HDFS status (ip should be eth0 inet):
  + <http://128.105.146.106:9870/dfshealth.html>
* Test creating a new directory in HDFS:
  + hdfs dfs -mkdir /test
  + hdfs dfs -ls /

**Apache Spark (install)**

* wget <https://dlcdn.apache.org/spark/spark-3.3.4/spark-3.3.4-bin-hadoop3.tgz>
* tar zvxf spark-3.3.4-bin-hadoop3.tgz
* sudo apt install -y python3.7

**Apache Spark (config)**

* spark-3.3.4-bin-hadoop3/conf/workers

10.10.1.2

10.10.1.3

* spark-3.3.4-bin-hadoop3/conf/spark-env.sh
  + export PYSPARK\_PYTHON=/usr/bin/python3.7
  + export PYSPARK\_DRIVER\_PYTHON=/usr/bin/python3.7
  + export SPARK\_MASTER\_HOST=10.10.1.1

(change for every local worker):

* + export SPARK\_LOCAL\_IP=10.10.1.X
* spark-3.3.4-bin-hadoop3/conf/spark-defaults.conf
  + spark.master spark://10.10.1.1:7077
  + spark.eventLog.enabled true
  + spark.eventLog.dir /mnt/data/spark-event-logs
  + spark.history.fs.logDirectory /mnt/data/spark-event-logs
  + spark.driver.memory 30g
  + spark.executor.memory 30g
  + spark.executor.cores 5
  + spark.task.cpus 1
  + spark.local.dir /mnt/data/
* Replicate spark configurations to all machines:
  + parallel-scp -h hostnames\_file.txt -O StrictHostKeyChecking=no spark-3.3.4-bin-hadoop3/conf/workers ~/spark-3.3.4-bin-hadoop3/conf/
  + parallel-scp -h hostnames\_file.txt -O StrictHostKeyChecking=no spark-3.3.4-bin-hadoop3/conf/spark-env.sh ~/spark-3.3.4-bin-hadoop3/conf/
  + parallel-scp -h hostnames\_file.txt -O StrictHostKeyChecking=no spark-3.3.4-bin-hadoop3/conf/spark-defaults.conf ~/spark-3.3.4-bin-hadoop3/conf/
* Start Spark standalone cluster:
  + spark-3.3.4-bin-hadoop3/sbin/start-all.sh
* Check Spark cluster status (ip should be eth0 inet):
  + <http://128.105.146.106:8080>

## **P****art 2: Simple Spark application**

* Move CSV file to node0
  + scp export.csv [ivanjaen@c220g5-110927vm-1.wisc.cloudlab.us](mailto:ivanjaen@c220g5-110927vm-1.wisc.cloudlab.us):/users/ivanjaen
* Load file into HDFS
  + hdfs dfs -copyFromLocal export.csv /

app.py

from pyspark.sql import SparkSession

import sys

# using command-line arguments

input\_filename\_path = sys.argv[1]

output\_path = sys.argv[2]

# hardcoding variables

#input\_filename\_path = "hdfs://10.10.1.1:9000/export.csv"

#output\_path = "hdfs://10.10.1.1:9000/output"

# The entry point (SparkSession class)

spark = (SparkSession

.builder

.appName("FirstApp")

#.config("some.config.option", "some-value")

.master("spark://c220g5-110927vm-1.wisc.cloudlab.us:7077")

.getOrCreate())

# Read file into DataFrames

df = spark.read.csv(input\_filename\_path,

header='true')

# Sorting (stable sorting) the data firth by country code ("cca2" column) and then by timestamp ("timestamp" column)

df.orderBy(['cca2', 'timestamp'],

ascending=True).show()

#df.show()

df.printSchema()

df.coalesce(1).write.csv(output\_path, header=True, sep=',')

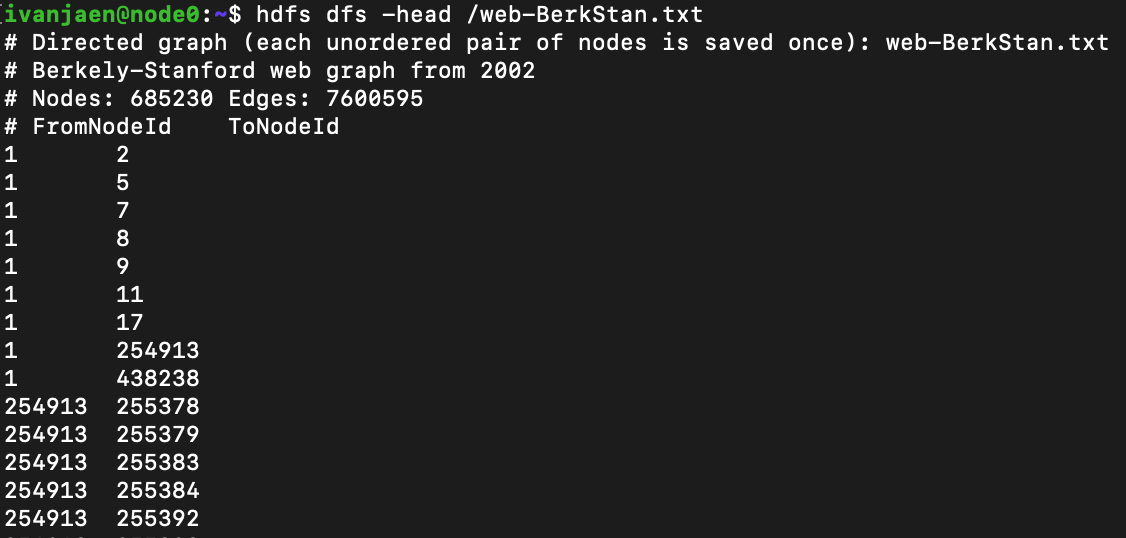
* Invoke script
  + ~/spark-3.3.4-bin-hadoop3/bin/spark-submit --master spark://c220g5-110927vm-1.wisc.cloudlab.us:7077 app.py hdfs://10.10.1.1:9000/export.csv hdfs://10.10.1.1:9000/output

## 

## **Part 3: PageRank**

### Datasets tested

* [Berkeley-Stanford web graph](https://snap.stanford.edu/data/web-BerkStan.html) (105MB): Consisting of 685,230 Nodes and 7,600,595 Edges



* enwiki-20180601-pages-articles (9.9GB): A dump of the Wikipedia link graph, where vertices are entities and edges are the hyperlinks between Wikipedia pages representing these entities.



### Task 1. Implementation

* Download the berley dataset into local and unzip it using
  + gunzip web-BerkStan.txt.gz
* Upload enwiki data set to hdfs
  + hdfs dfs -copyFromLocal /proj/uwmadison744-s24-PG0/data-part3/enwiki-pages-articles /
  + Status can be checked at http://localhost:9870/explorer.html#/
* Upload the dataset file to remote node0
  + scp web-BerkStan.txt ivanjaen@c220g5-110927vm-1.wisc.cloudlab.us:/users/ivanjaen
* Upload the txt file from local to the hdfs cluster
  + hdfs dfs -copyFromLocal web-BerkStan.txt /
* ./spark-3.3.4-bin-hadoop3/bin/pyspark --master spark://c220g5-110927vm-1.wisc.cloudlab.us:7077

Launch interactive python

from pyspark.sql import SparkSession

import sys

import re

if \_\_name\_\_ == "\_\_main\_\_":

# using command-line arguments

input\_path = sys.argv[1]

output\_path = sys.argv[2]

# hardcoding variables

#input\_filename\_path = "hdfs://10.10.1.1:9000/web-BerkStan.txt"

#output\_path = "hdfs://10.10.1.1:9000/output2"

input\_workload\_name = input\_path.split("/")[3]

# The entry point (SparkSession class)

spark = (SparkSession

.builder

.appName("PageRank\_task1\_" + input\_workload\_name)

#.config("some.config.option", "some-value")

#.master("spark://c220g5-110927vm-1.wisc.cloudlab.us:7077")

.getOrCreate())

sc = spark.sparkContext

# Get web graph data into RDD

webNodes = sc.textFile(input\_path)

# filtering out first four lines that contains comments about the dataset

curatedNodes = webNodes.filter(lambda x: not x.startswith("#"))

# Generates an RDD of (source, destinations) pairs

links = curatedNodes.map(lambda row: (row.split("\t")[0], row.split("\t")[1])).groupByKey()

# Generates an RDD of (source, rank) pairs. Initializes each rank to 1.0

ranks = links.map(lambda link: (link[0], 1.0))

# Page Rank algorithm

MAX\_ITER = 5 #number of iterations

for i in range(MAX\_ITER):

# Compute contribution for every destination (page p contribute rank\_p / neighbors\_p ) . It generates a mapping (destination, contribution)

contributions = links.join(ranks).flatMap(

lambda x: [(destination, x[1][1] / len(x[1][0])) for destination in x[1][0]]

)

# Aggregate the contributions and recalculate the rank using given formula

ranks = contributions.reduceByKey(lambda x, y: x + y).mapValues(lambda sum: 0.15 + (0.85 \* sum))

# For debug only: Print all pageranks

#for (link, rank) in ranks.collect():

# print("%s has rank: %s." % (link, rank))

# Save RDD as a text file at user provided output\_path

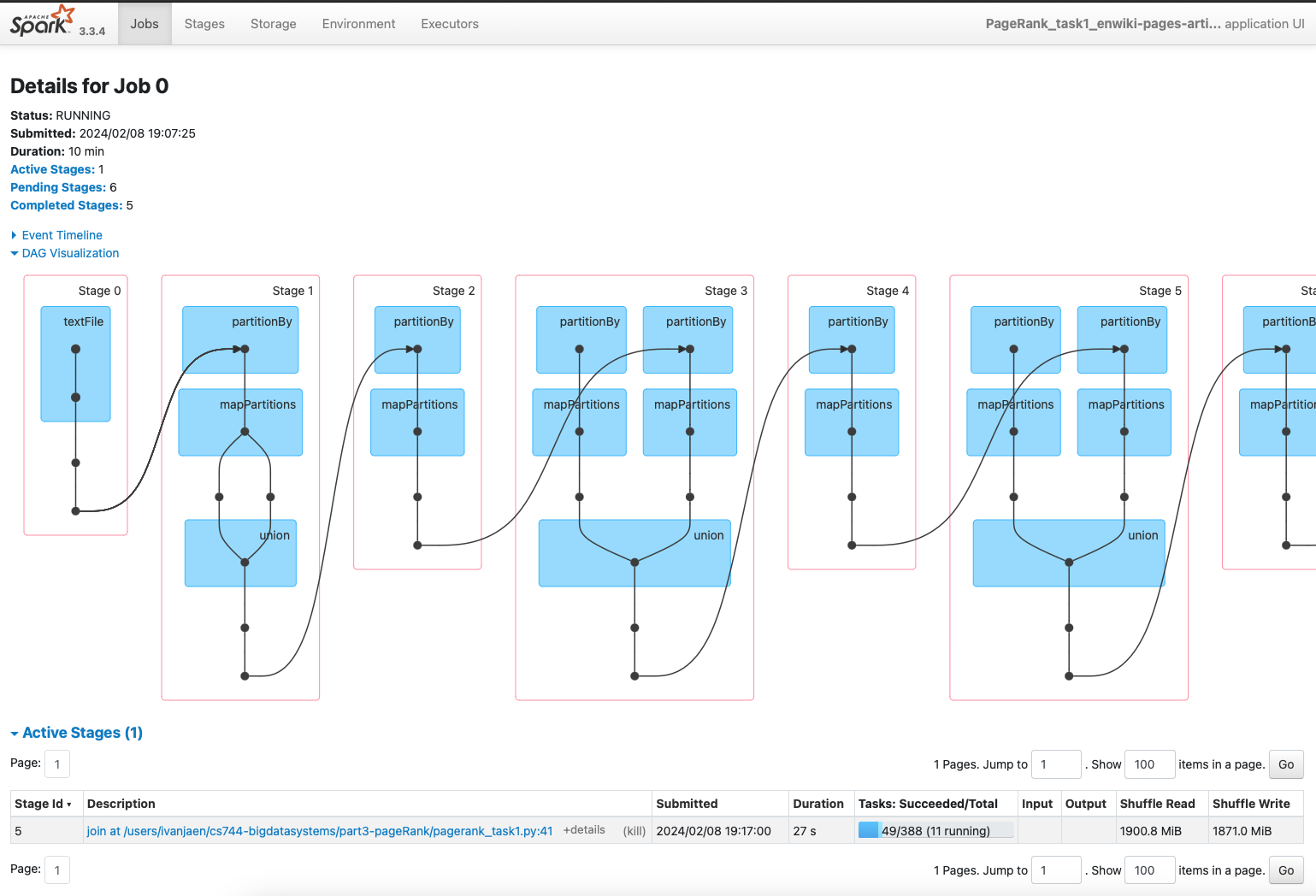
ranks.saveAsTextFile(output\_path)

sc.stop()

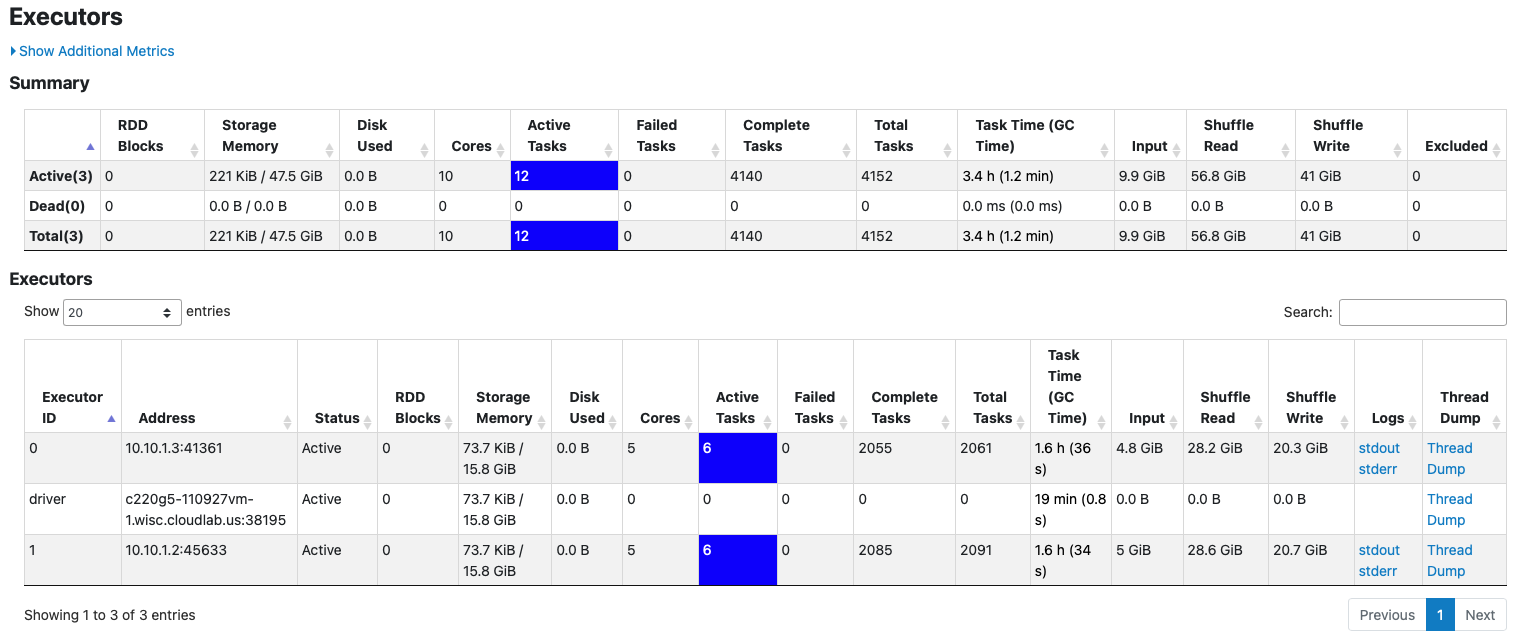
Stats:

We report the time it took for each workload and show the corresponding process graph.

| **Workload** | **Total time** |
| --- | --- |
| * Berkeley-Stanford web graph | 3.4 min |
| * enwiki-20180601-pages-articles | 20 min |

* Berkeley-Stanford web graph

We see the two worker nodes executing the tasks in parallel.



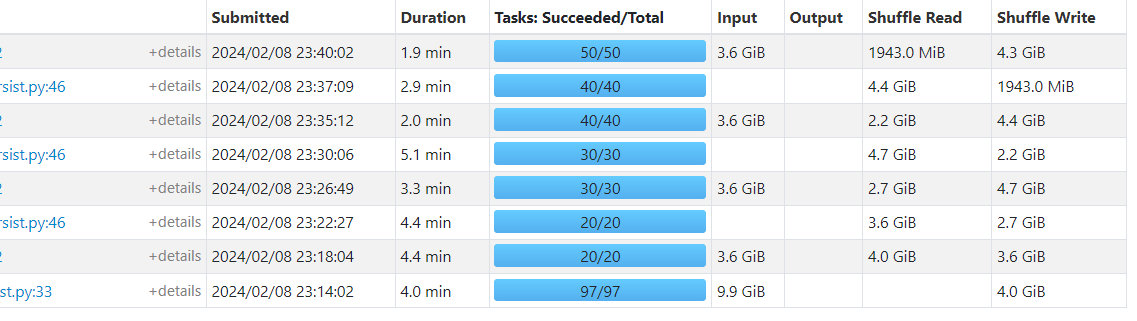
### Task 2. Adding custom RDD partitioning

We try different value for .partitionBy()

For small values, the program will simply fail because there is no space.

**Partitions =** 10

**Time taken =** 35 minutes

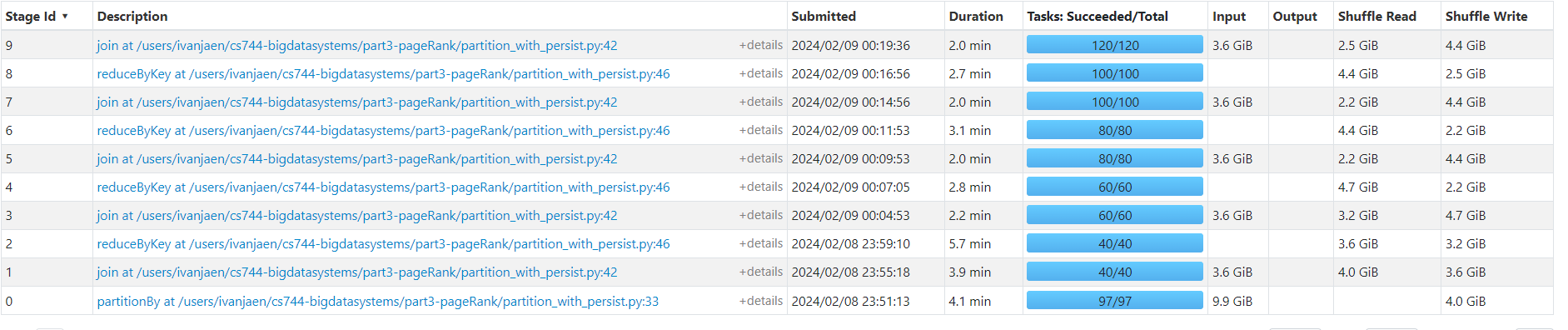
****

**Insights:**

From the data it's evident that time taken is proportional to Shuffle read and write. This can be explained by the fact that shuffle operation is expensive as data has to be transported over the network.

**Partitions**: 20

Time Taken = 34 minutes

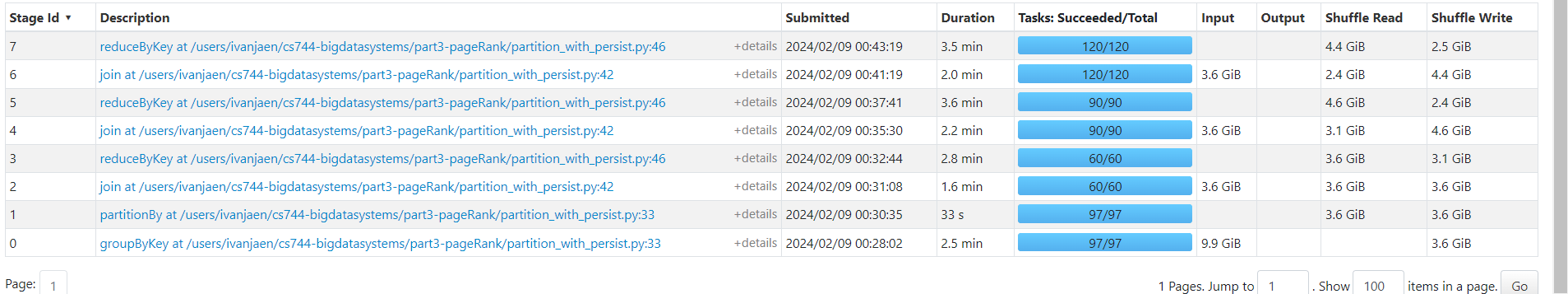


Partition Insights:

The total time taken is more or less the same which might be due to a lot of factors, but on average the shuffle read and shuffle write looks the same. We can notice as the number of partitions increases the number of tasks also increases.

Partitions: 30

Time Taken: 30 Minutes



Partition Insights:

Joins performed better than reduce operations which indicate persisting the data helped joins speed up.

Summary:

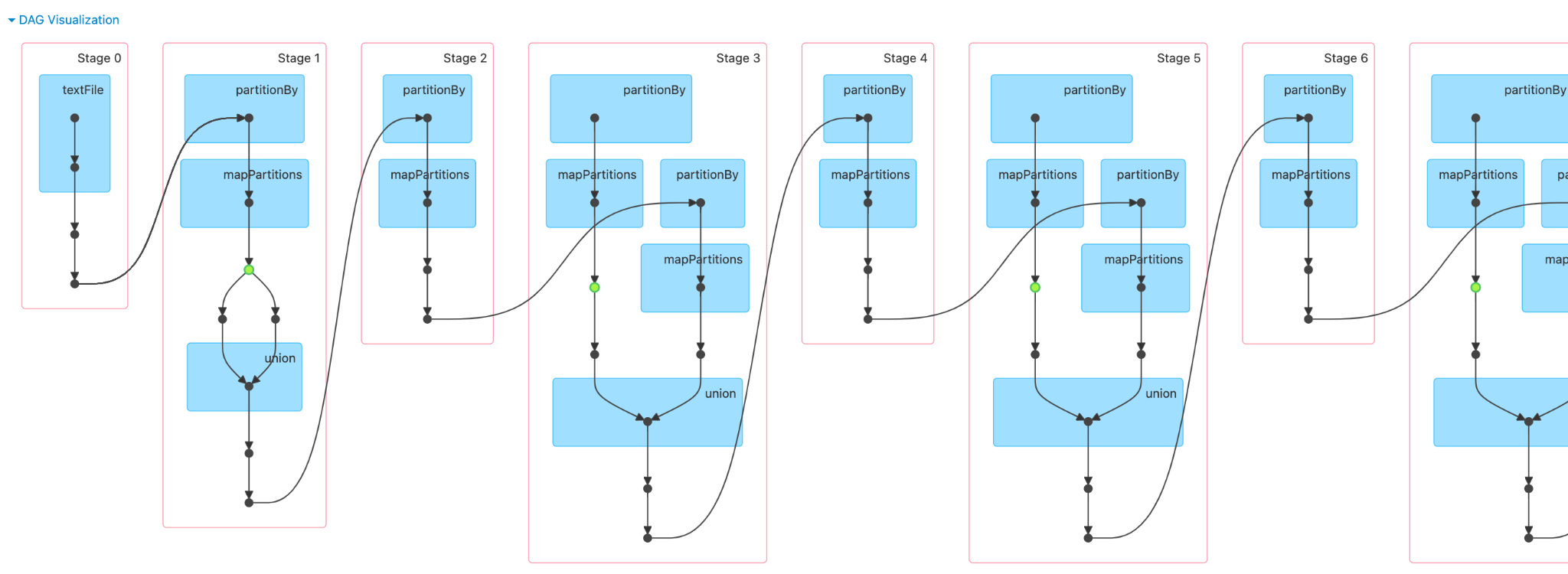
As we can generalize from the observations there is a significant performance gain as we increase the partition size mostly by the fact that data is processed locally before being sent over the network. But, the performance will drop if we choose the partition size to be too large as the records present in each partition would be too less and there might be a significant partition maintenance overhead. So, the optimal partition size actually depends on the specific use case and size of dataset.

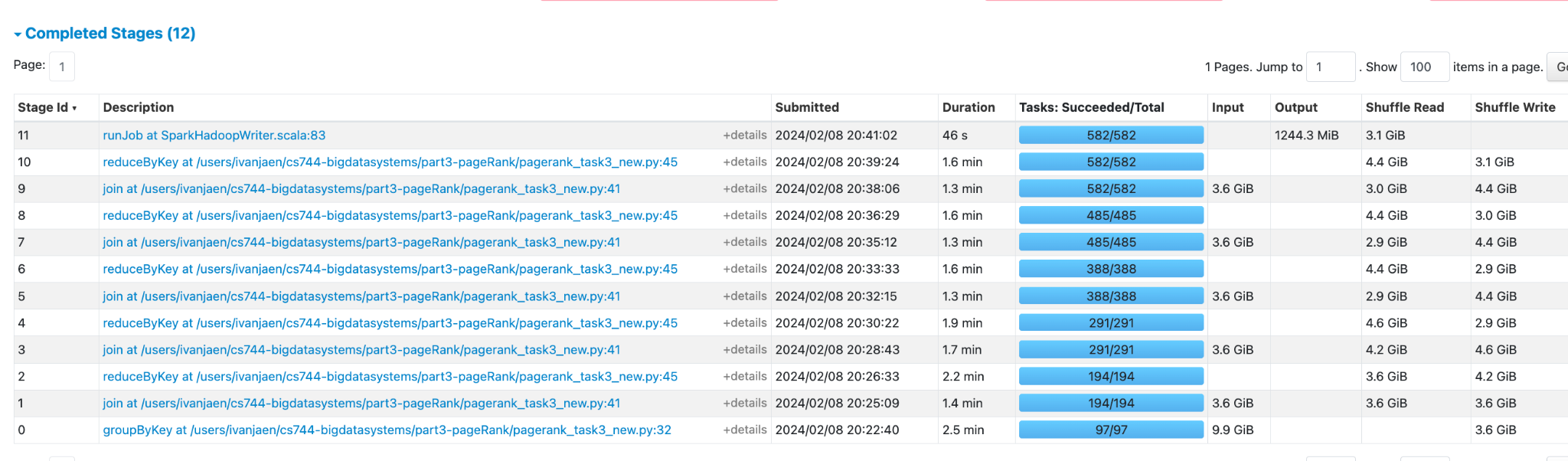
### Task 3. Persist Data

We add .persist() to links to make it an in-memory object. The following result is compared to task1 (same partition as task1):

Joins performed better than reduce operations which indicate persisting the data helped joins speed up.

| **Workload** | **Total time** |
| --- | --- |
| * Berkeley-Stanford web graph | 3 min |
| * enwiki-20180601-pages-articles | 20 min |



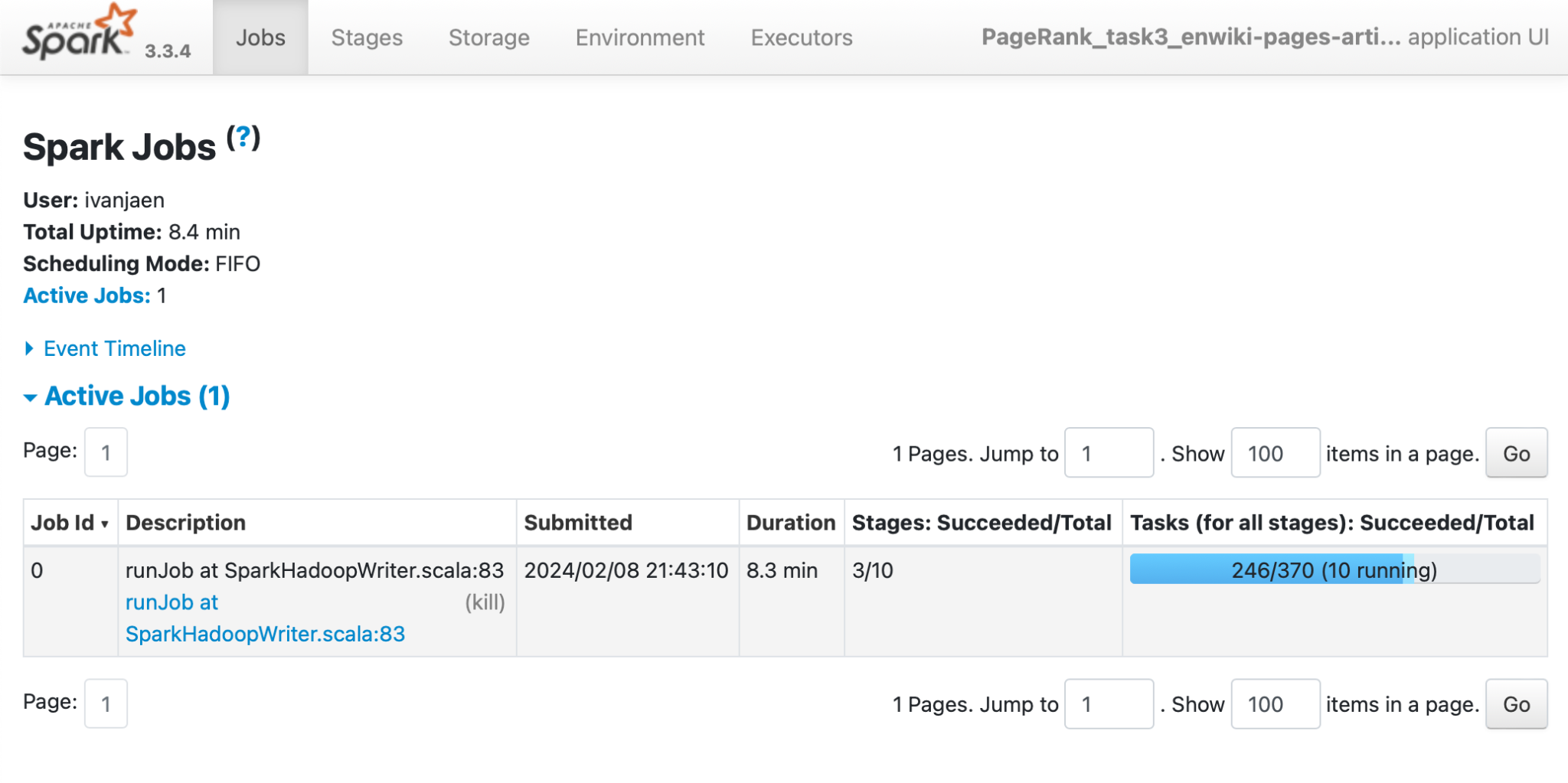


(running on enwiki)

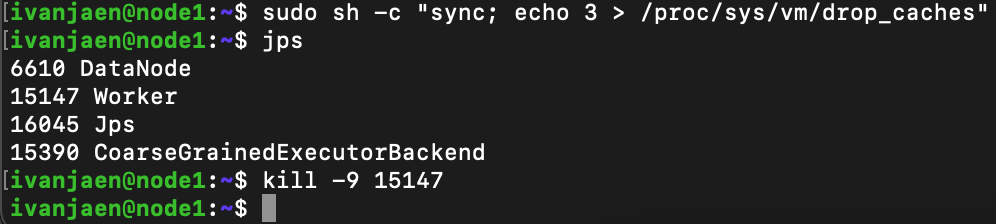
Note that for large data the running time does not change much. For smaller data the running time is improved.

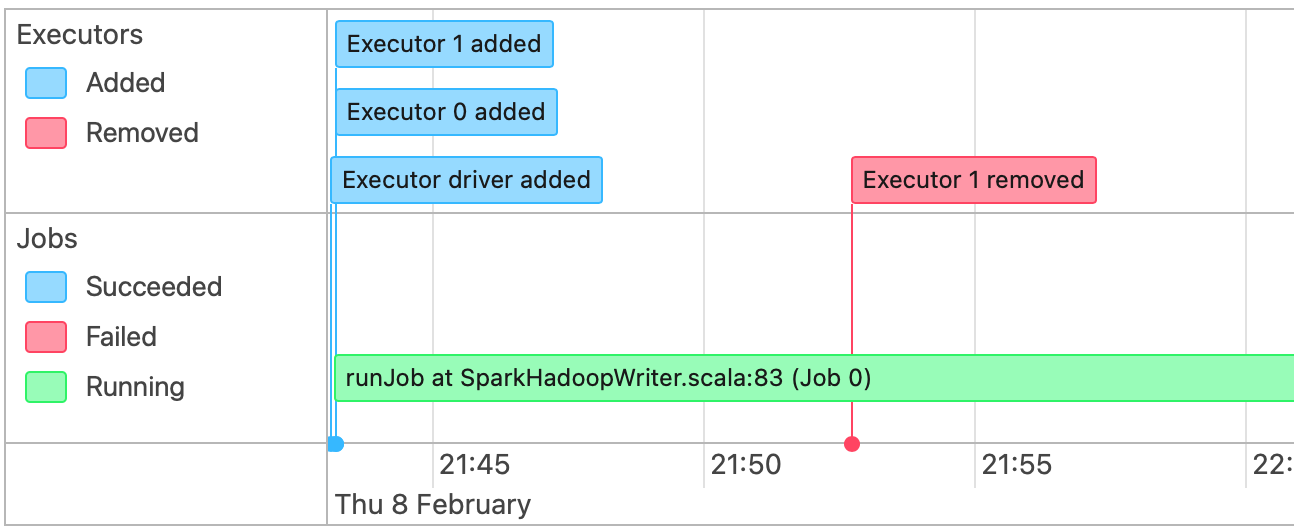
### Task 4. Fault Tolerance

In order to test the fault tolerance of the system we experimented killing one of the 2 worker processes after cleaning the cache. We started a Spark application using the same settings as Task3 (Persistant RDD in cache)

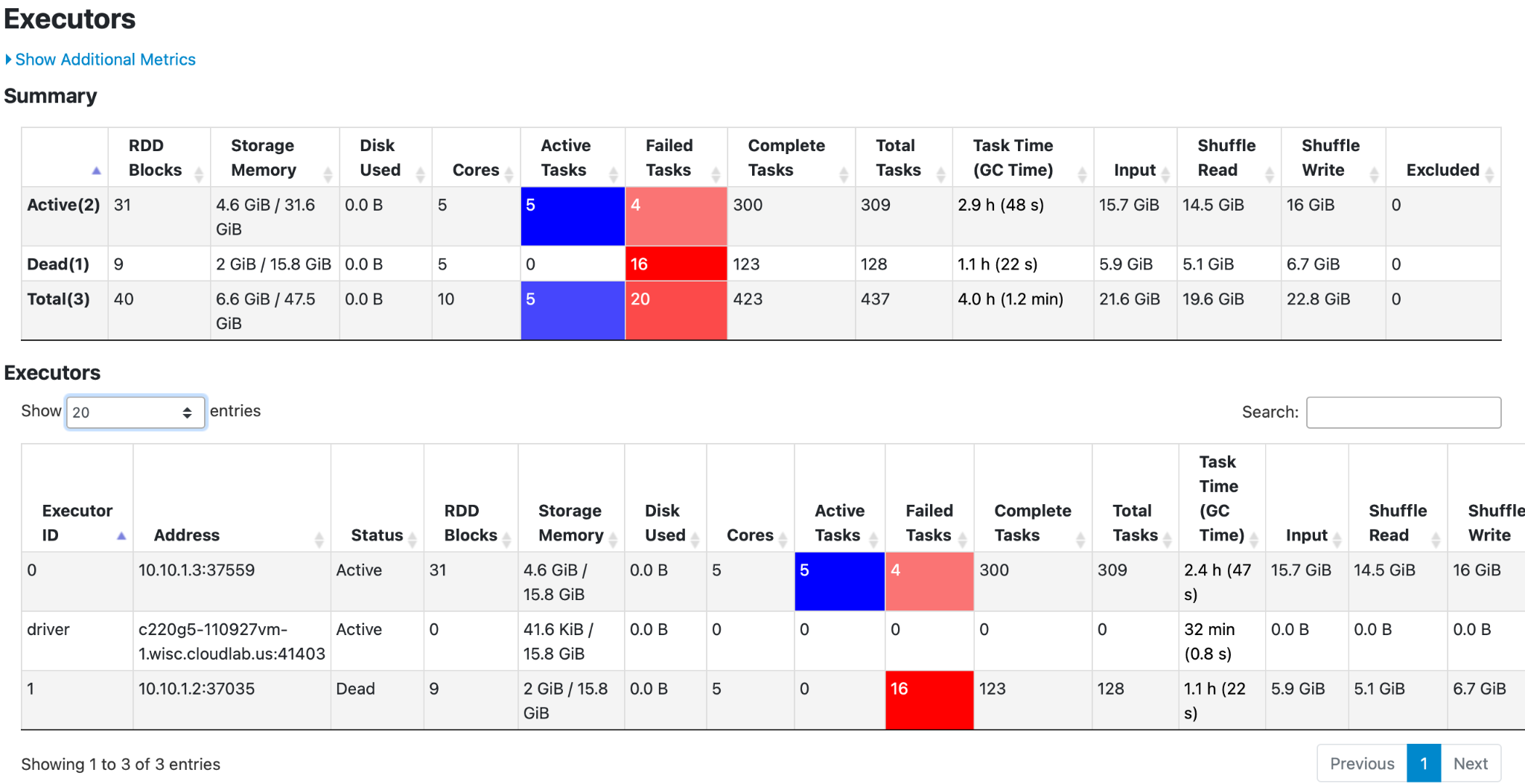


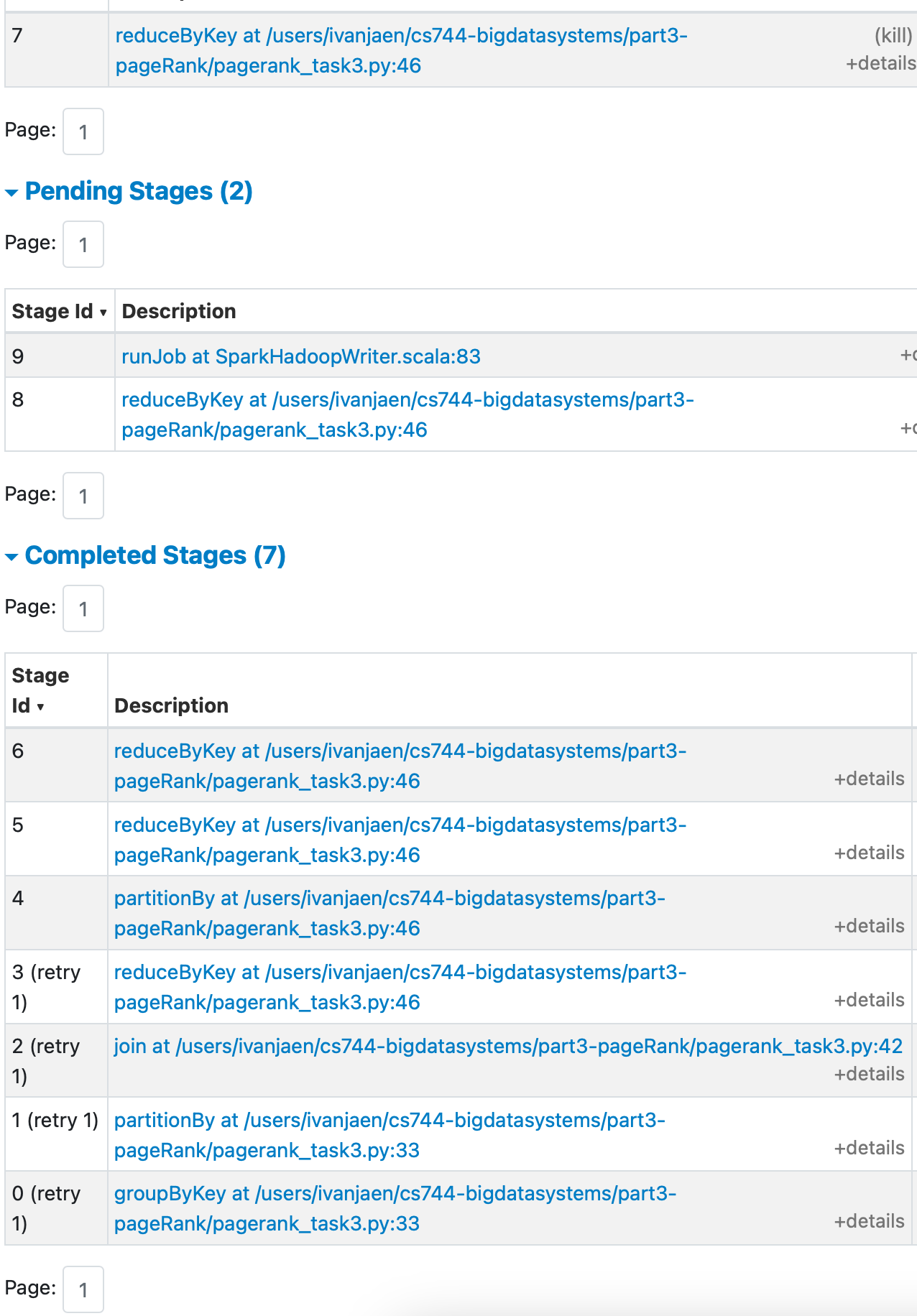
In the middle of the execution we applied the corresponding procedure to inactivate node 1:





Immediately after this, some tasks are marked as failed in the executors view. When this is detected by the Spark Driver, it automatically creates new tasks in order to re-execute previous calculations that were lost by the killed node (marked in red in the web UI).





Reported measurements at different points in time during the task execution:

|  | At 25% of task lifetime | At 75% of task lifetime |
| --- | --- | --- |
| Total time | 42 min | 78 min |

A failure triggered at 75% completion time takes twice the amount of time which can be explained by the fact that it has to replay the entire parent lineage to rebuild the intermediate result for that partition. Given that we have only 2 worker nodes the penalty to calculate the parent RDD from the input file is high.