CS744 : Big Data Systems
Spring 2024

Assignment 1

Lakshika Rathi, Aboli Pai, Tanisha Hegde, Aanandita Dhawan

February 9, 2024

1 Introduction

In this assignment, we build a 3-node cluster. We then deploy Apache Spark and Apache Hadoop on the cluster. This was followed by implementing two applications in Spark - a simple one to sort the data given some sample data collected by IoT devices, and the other to implement PageRank algorithm and run it on two datasets. We then did a couple of experiments on top of it and recorded the observations and inferences.

2 Environment Setup

As given in the assignment description, we set up a simple 3-node cluster with Ubuntu on each machine.

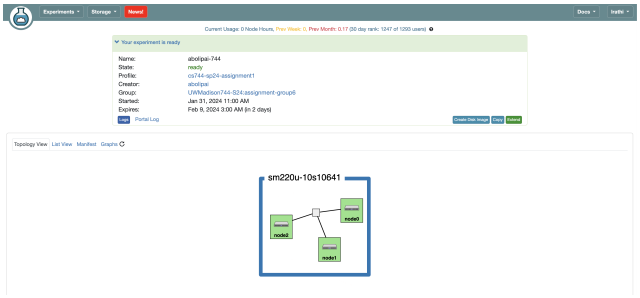


Figure 1: Setup of the simple 3-node cluster

Firstly, we enable SSH service among all the nodes in the server. Then, we copy files from node0 to the other nodes using parallel ssh. As a first step, we enable an additional mount point to have extra storage space on each node.

3 Part 1: Software Deployment

This part involved the installation of Apache Hadoop and Apache Spark. Apache Hadoop is a collection of open-source software utilities that provide distributed programming models to process large data sets. For this assignment, we specifically use the Hadoop Distributed File System (HDFS). Apache Spark is an open-source analytics engine for big data processing.

We were able to install both successfully using the instructions given in the assignment description.

4 Part 2: A simple Spark application

This part involved the creation of a simple spark application. We first load the file data into HDFS. Data frames are then created from these files, on which operations can be applied. We use the PySpark API to implement the sorting operation. To run the application, we need to submit it using spark-submit script. Finally, the output files are in the csv format in HDFS.

5 Part 3: PageRank

In this part, we write the PageRank algorithm used by search engines to evaluate the quality of links to a webpage. We run the algorithm on two data-sets. The Berkeley-Stanford web graph is a smaller dataset to test the algorithm implementation, while the enwiki-20180601-pages-articles is the larger dataset.

5.1 Task 1

This task involved the implementation of the PageRank algorithm. We followed the algorithm proposed in the spark paper and ran it for 10 iterations.

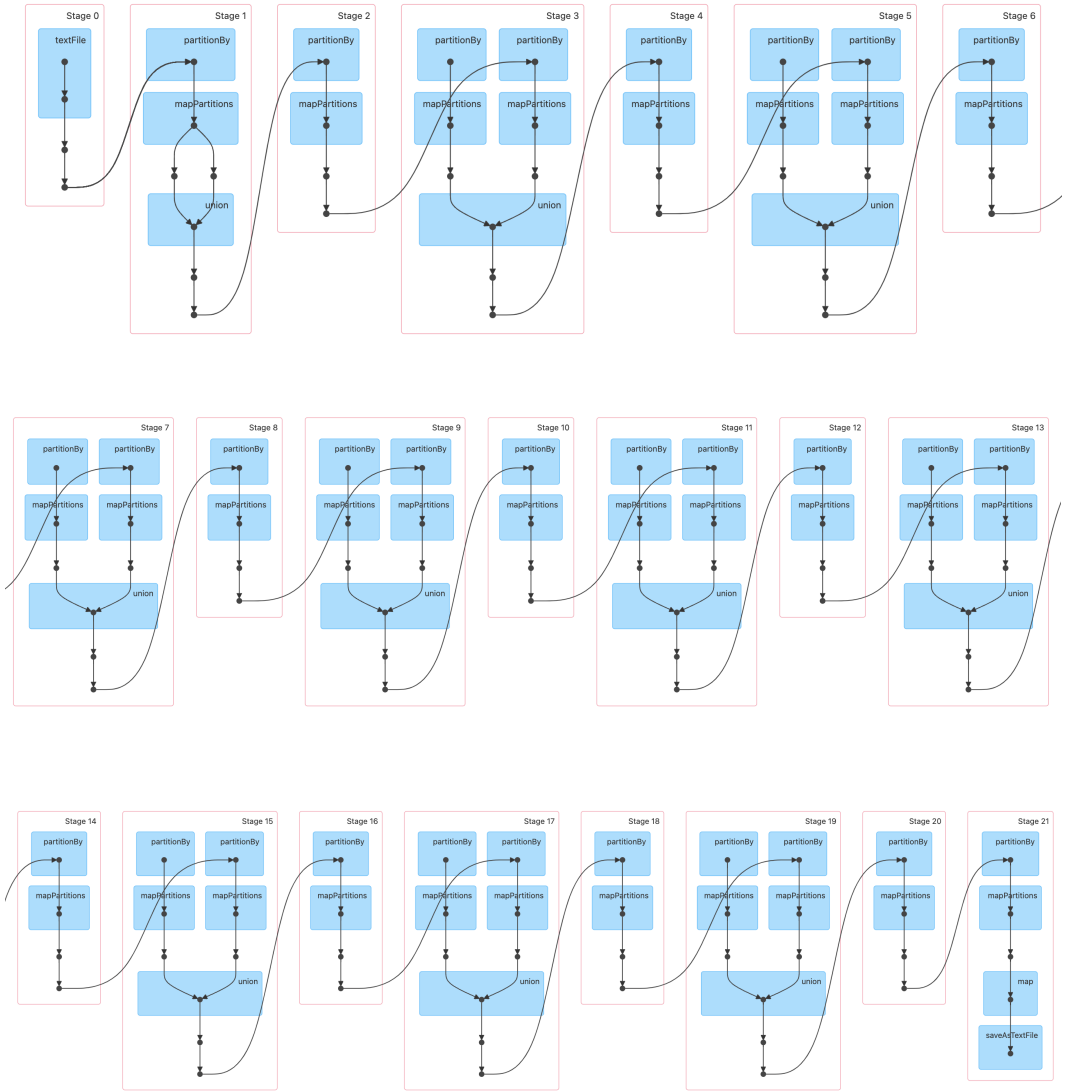


Figure 2: DAG visualization of Lineage Graph for PageRank algorithm (10 iterations)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
21	runJob at SparkHadoopWriter.scala:83 +details	2024/02/06 16:51:35	2 s	22/22		17.5 MiB	32.3 MiB	
20	reduceByKey at /users/abolipai/part3-1.py:34 +details	2024/02/06 16:51:30	5 s	22/22			46.3 MiB	32.3 MiB
19	join at /users/abolipai/part3-1.py:31 +details	2024/02/06 16:51:06	24 s	22/22			45.4 MiB	46.3 MiB
18	reduceByKey at /users/abolipai/part3-1.py:34 +details	2024/02/06 16:51:01	5 s	20/20			45.9 MiB	31.1 MiB
17	join at /users/abolipai/part3-1.py:31 +details	2024/02/06 16:50:36	25 s	20/20			44.4 MiB	45.9 MiB
16	reduceByKey at /users/abolipai/part3-1.py:34 +details	2024/02/06 16:50:31	6 s	18/18			45.4 MiB	30.1 MiB
15	join at /users/abolipai/part3-1.py:31 +details	2024/02/06 16:50:07	24 s	18/18			43.0 MiB	45.4 MiB
14	reduceByKey at /users/abolipai/part3-1.py:34 +details	2024/02/06 16:50:01	6 s	16/16			44.9 MiB	28.7 MiB
13	join at /users/abolipai/part3-1.py:31 +details	2024/02/06 16:49:37	24 s	16/16			41.5 MiB	44.9 MiB
12	reduceByKey at /users/abolipai/part3-1.py:34 +details	2024/02/06 16:49:32	5 s	14/14			44.3 MiB	27.2 MiB
11	join at /users/abolipai/part3-1.py:31 +details	2024/02/06 16:49:07	24 s	14/14			39.9 MiB	44.3 MiB
10	reduceByKey at /users/abolipai/part3-1.py:34 +details	2024/02/06 16:49:01	6 s	12/12			43.6 MiB	25.6 MiB
9	join at /users/abolipai/part3-1.py:31 +details	2024/02/06 16:48:37	24 s	12/12			38.1 MiB	43.6 MiB
8	reduceByKey at /users/abolipai/part3-1.py:34 +details	2024/02/06 16:48:34	4 s	10/10			42.7 MiB	23.8 MiB
7	join at /users/abolipai/part3-1.py:31 +details	2024/02/06 16:48:10	24 s	10/10			35.5 MiB	42.7 MiB
6	reduceByKey at /users/abolipai/part3-1.py:34 +details	2024/02/06 16:48:03	7 s	8/8			41.6 MiB	21.2 MiB
5	join at /users/abolipai/part3-1.py:31 +details	2024/02/06 16:47:39	24 s	8/8			32.1 MiB	41.6 MiB
4	reduceByKey at /users/abolipai/part3-1.py:34 +details	2024/02/06 16:47:27	11 s	6/6			39.0 MiB	17.9 MiB
3	join at /users/abolipai/part3-1.py:31 +details	2024/02/06 16:47:03	24 s	6/6			26.5 MiB	39.0 MiB
2	reduceByKey at /users/abolipai/part3-1.py:34 +details	2024/02/06 16:46:55	8 s	4/4			33.9 MiB	12.2 MiB
1	join at /users/abolipai/part3-1.py:31 +details	2024/02/06 16:46:30	25 s	4/4			28.6 MiB	33.9 MiB
0	groupByKey at /users/abolipai/part3-1.py:23 +details	2024/02/06 16:46:13	16 s	2/2	105.1 MiB			14.3 MiB

Figure 3: Stage details for PageRank algorithm

The spark execution gave 22 stages. Our initial implementation added another step to calculate the unique node IDs using the 'distinct' function. This added 2 more stages to the spark execution. To optimize this, we removed the 'distinct' function and used the 'links' rdd to map the nodes to the rank. Stage 0 uses the groupByKey function to create the 'links' RDD. This requires it to perform a shuffle write as the data is grouped by key and partitions accordingly. Stages 1 - 20 denote the 10 iterations of the PageRank algorithm that perform *join* operation on the 'ranks' and 'links' RDDs and then perform a *reduceByKey* operation on the computation. Both of these require a shuffle read and write across partitions. Stage 21 is the last step that saves data in a text file and produces the final output.

Our implementation gave an execution time of **5.5 minutes** for the smaller Berkeley-Stanford, while an execution time of **37 minutes** for the larger Wiki pages dataset. The total number of tasks for Berkeley-Stanford were **284** and for the Wiki dataset were **13871**.

While executing the Berkeley-Stanford dataset, we observed that there were at most 2 tasks running for every stage. The 2 tasks would often run on a single executor at a time. This could be due to locality of data. As this was a smaller dataset, all keys could be collected on a single node when performing operations. For the Wiki dataset, there were 12 tasks running for every stage, and 6 tasks would run on each executor. As the data is large, spark likely tries to allocate equal workload to every executor.

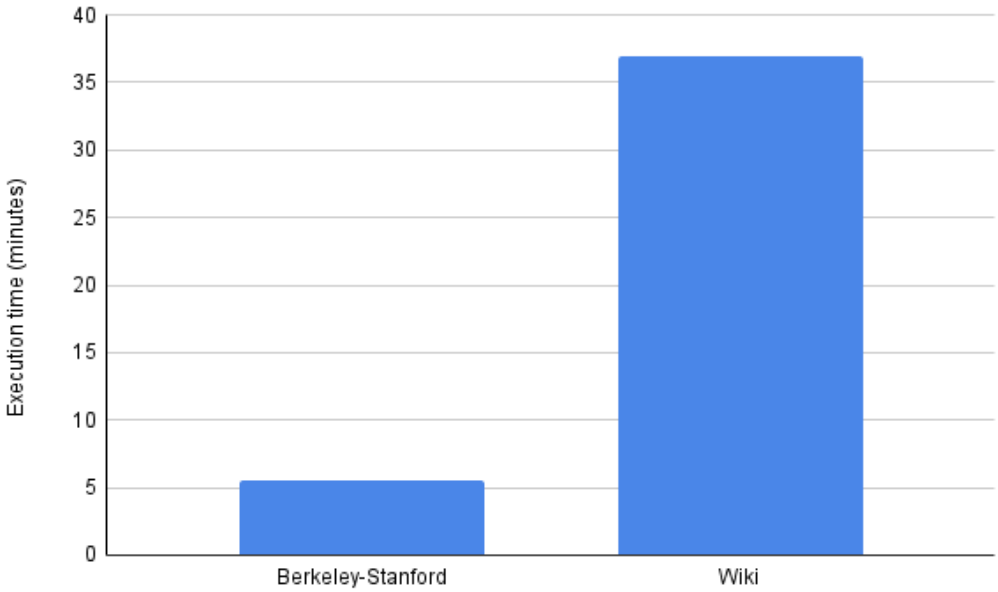


Figure 4: Execution times for both datasets

5.2 Task 2

In this task, we vary the number of partitions in spark for the larger dataset. We record the total execution time, the time taken for "join" operation, the time taken for "partitionBy" stage which is done after the join operation in each iteration, the total stages, shuffle read and shuffle write. As it was taking a long time to run the PageRank algorithm for 10 iterations, we ran it uptil 5 iterations itself and recorded the results for 2, 10, 100, 512 and 1024 partitions.

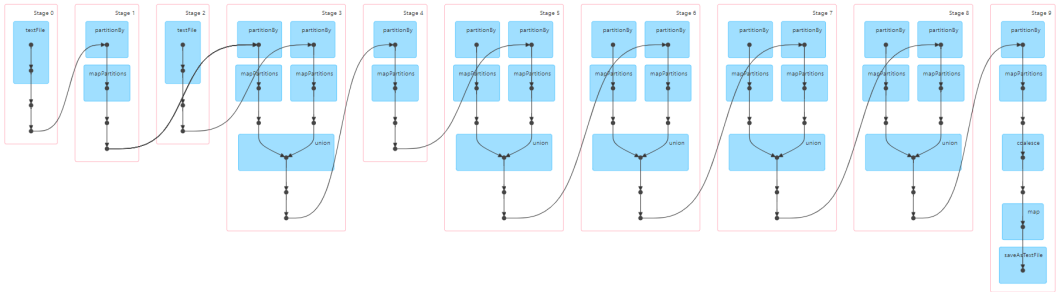
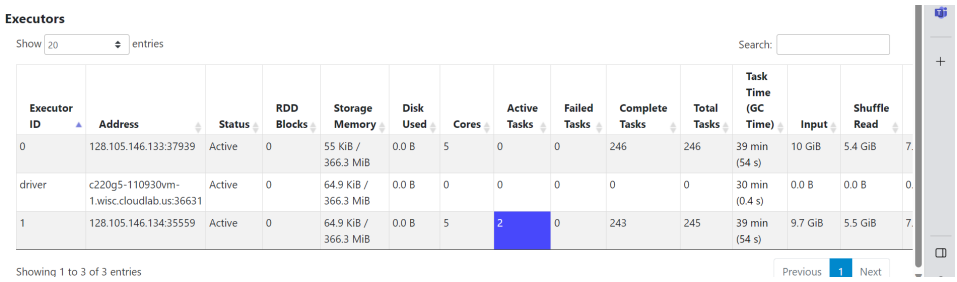


Figure 5: DAG Visualization for partitioning

5.2.1 Number of partitions = 2

The total execution time was 2.4 hours. A single executor was used at a time.



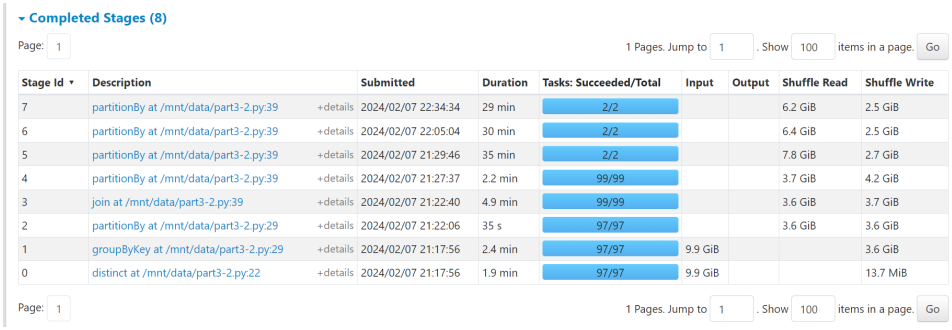
Executors

Show 20 entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	
0	128.105.146.133:37939	Active	0	55 KiB / 366.3 MiB	0.0 B	5	0	0	246	246	39 min (54 s)	10 GiB	5.4 GiB	7.
driver	c220g5-110930vm-1.wisc.cloudlab.us:36631	Active	0	64.9 KiB / 366.3 MiB	0.0 B	0	0	0	0	0	30 min (0.4 s)	0.0 B	0.0 B	0.
1	128.105.146.134:35559	Active	0	64.9 KiB / 366.3 MiB	0.0 B	5	2	0	243	245	39 min (54 s)	9.7 GiB	5.5 GiB	7.

Showing 1 to 3 of 3 entries Previous 1 Next

Figure 6: Figure depicting the shuffle statistics when number of partitions = 2



Completed Stages (8)

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

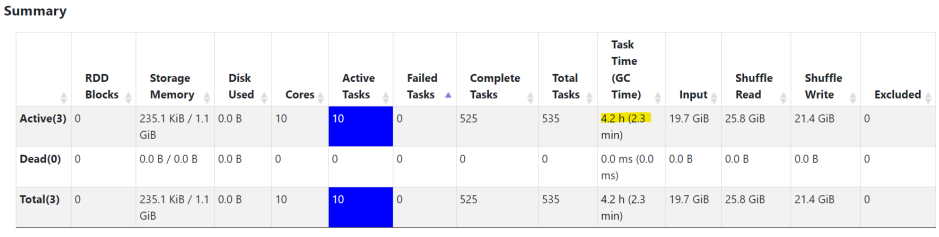
Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
7	partitionBy at /mnt/data/part3-2.py:39 +details	2024/02/07 22:34:34	29 min	2/2			6.2 GiB	2.5 GiB
6	partitionBy at /mnt/data/part3-2.py:39 +details	2024/02/07 22:05:04	30 min	2/2			6.4 GiB	2.5 GiB
5	partitionBy at /mnt/data/part3-2.py:39 +details	2024/02/07 21:29:46	35 min	2/2			7.8 GiB	2.7 GiB
4	partitionBy at /mnt/data/part3-2.py:39 +details	2024/02/07 21:27:37	2.2 min	99/99			3.7 GiB	4.2 GiB
3	join at /mnt/data/part3-2.py:39 +details	2024/02/07 21:22:40	4.9 min	99/99			3.6 GiB	3.7 GiB
2	partitionBy at /mnt/data/part3-2.py:29 +details	2024/02/07 21:22:06	35 s	97/97			3.6 GiB	3.6 GiB
1	groupByKey at /mnt/data/part3-2.py:29 +details	2024/02/07 21:17:56	2.4 min	97/97	9.9 GiB			3.6 GiB
0	distinct at /mnt/data/part3-2.py:22 +details	2024/02/07 21:17:56	1.9 min	97/97	9.9 GiB			13.7 MiB

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

Figure 7: Figure depicting the completed stages of PageRank algorithm when number of partitions = 2

5.2.2 Number of partitions = 10

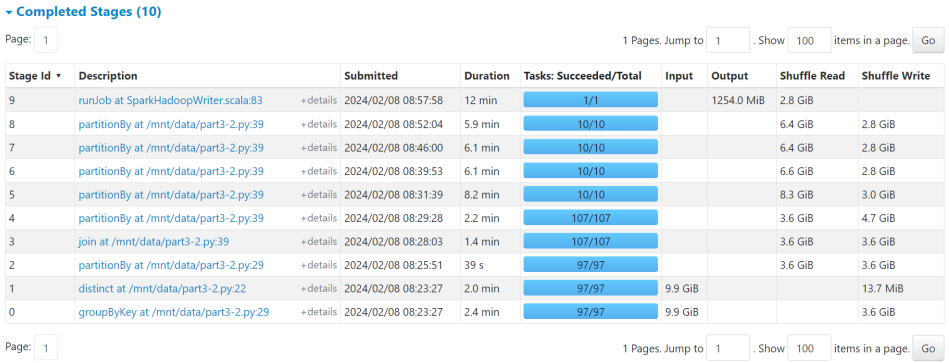
The total execution time was 47 mins. There was equal load on both the executors. The writing took 12 mins, making the overall job slow.



Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Excluded
Active(3)	0	235.1 KiB / 1.1 GiB	0.0 B	10	10	0	525	535	4.2 h (2.3 min)	19.7 GiB	25.8 GiB	21.4 GiB	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	0
Total(3)	0	235.1 KiB / 1.1 GiB	0.0 B	10	10	0	525	535	4.2 h (2.3 min)	19.7 GiB	25.8 GiB	21.4 GiB	0

Figure 8: Figure depicting the shuffle statistics when number of partitions = 10



▾ Completed Stages (10)

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

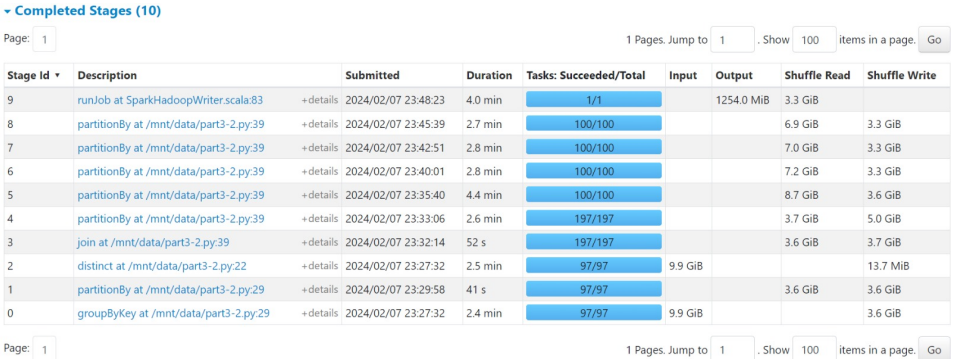
Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
9	runJob at SparkHadoopWriter.scala:83 +details	2024/02/08 08:57:58	12 min	1/1		1254.0 MiB	2.8 GiB	
8	partitionBy at /mnt/data/part3-2.py:39 +details	2024/02/08 08:52:04	5.9 min	10/10			6.4 GiB	2.8 GiB
7	partitionBy at /mnt/data/part3-2.py:39 +details	2024/02/08 08:46:00	6.1 min	10/10			6.4 GiB	2.8 GiB
6	partitionBy at /mnt/data/part3-2.py:39 +details	2024/02/08 08:39:53	6.1 min	10/10			6.6 GiB	2.8 GiB
5	partitionBy at /mnt/data/part3-2.py:39 +details	2024/02/08 08:31:39	8.2 min	10/10			8.3 GiB	3.0 GiB
4	partitionBy at /mnt/data/part3-2.py:39 +details	2024/02/08 08:29:28	2.2 min	107/107			3.6 GiB	4.7 GiB
3	join at /mnt/data/part3-2.py:39 +details	2024/02/08 08:28:03	1.4 min	107/107			3.6 GiB	3.6 GiB
2	partitionBy at /mnt/data/part3-2.py:29 +details	2024/02/08 08:25:51	39 s	97/97			3.6 GiB	3.6 GiB
1	distinct at /mnt/data/part3-2.py:22 +details	2024/02/08 08:23:27	2.0 min	97/97	9.9 GiB			13.7 MiB
0	groupByKey at /mnt/data/part3-2.py:29 +details	2024/02/08 08:23:27	2.4 min	97/97	9.9 GiB			3.6 GiB

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

Figure 9: Figure depicting the completed stages of PageRank algorithm when number of partitions = 10

5.2.3 Number of partitions = 100

The total execution time was 25 mins. There was equal load on both the executors.



▾ Completed Stages (10)

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

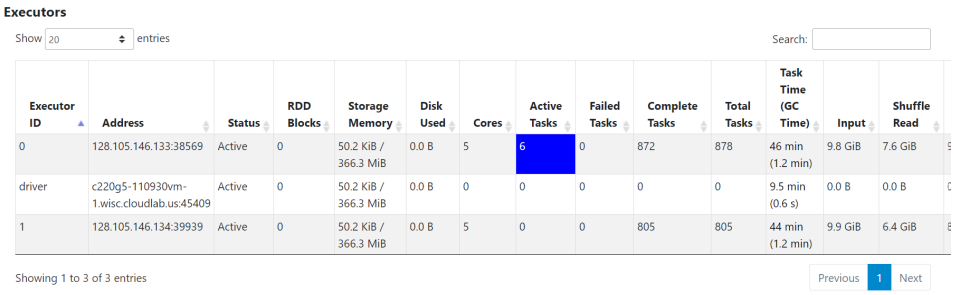
Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
9	runJob at SparkHadoopWriter.scala:83 +details	2024/02/07 23:48:23	4.0 min	1/1		1254.0 MiB	3.3 GiB	
8	partitionBy at /mnt/data/part3-2.py:39 +details	2024/02/07 23:45:39	2.7 min	100/100			6.9 GiB	3.3 GiB
7	partitionBy at /mnt/data/part3-2.py:39 +details	2024/02/07 23:42:51	2.8 min	100/100			7.0 GiB	3.3 GiB
6	partitionBy at /mnt/data/part3-2.py:39 +details	2024/02/07 23:40:01	2.8 min	100/100			7.2 GiB	3.3 GiB
5	partitionBy at /mnt/data/part3-2.py:39 +details	2024/02/07 23:35:40	4.4 min	100/100			8.7 GiB	3.6 GiB
4	partitionBy at /mnt/data/part3-2.py:39 +details	2024/02/07 23:33:06	2.6 min	197/197			3.7 GiB	5.0 GiB
3	join at /mnt/data/part3-2.py:39 +details	2024/02/07 23:32:14	52 s	197/197			3.6 GiB	3.7 GiB
2	distinct at /mnt/data/part3-2.py:22 +details	2024/02/07 23:27:32	2.5 min	97/97	9.9 GiB			13.7 MiB
1	partitionBy at /mnt/data/part3-2.py:29 +details	2024/02/07 23:29:58	41 s	97/97			3.6 GiB	3.6 GiB
0	groupByKey at /mnt/data/part3-2.py:29 +details	2024/02/07 23:27:32	2.4 min	97/97	9.9 GiB			3.6 GiB

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

Figure 10: Figure depicting the completed stages of PageRank algorithm when number of partitions = 100

5.2.4 Number of partitions = 512

The total execution time was 26 mins. Sometimes only one executor was being used. There was an uneven distribution of tasks between the two executors.



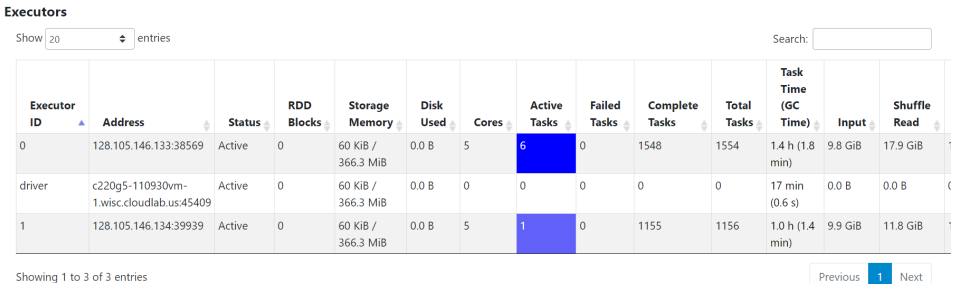
Executors

Show 20 entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read
0	128.105.146.133:38569	Active	0	50.2 KiB / 366.3 MiB	0.0 B	5	6	0	872	878	46 min (1.2 min)	9.8 GiB	7.6 GiB
driver	c220g5-110930vm-1.wisc.cloudlab.us:45409	Active	0	50.2 KiB / 366.3 MiB	0.0 B	0	0	0	0	0	9.5 min (0.6 s)	0.0 B	0.0 B
1	128.105.146.134:39939	Active	0	50.2 KiB / 366.3 MiB	0.0 B	5	0	0	805	805	44 min (1.2 min)	9.9 GiB	6.4 GiB

Showing 1 to 3 of 3 entries Previous 1 Next

Figure 11: Figure depicting that sometimes only one executor was used when number of partitions = 512



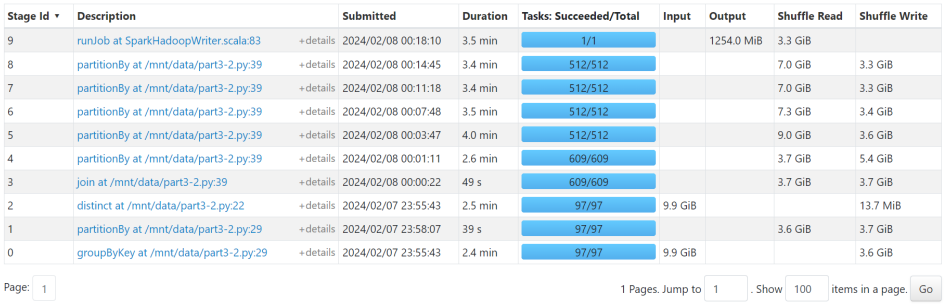
Executors

Show 20 entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read
0	128.105.146.133:38569	Active	0	60 KiB / 366.3 MiB	0.0 B	5	6	0	1548	1554	1.4 h (1.8 min)	9.8 GiB	17.9 GiB
driver	c220g5-110930vm-1.wisc.cloudlab.us:45409	Active	0	60 KiB / 366.3 MiB	0.0 B	0	0	0	0	0	17 min (0.6 s)	0.0 B	0.0 B
1	128.105.146.134:39939	Active	0	60 KiB / 366.3 MiB	0.0 B	5	1	0	1155	1156	1.0 h (1.4 min)	9.9 GiB	11.8 GiB

Showing 1 to 3 of 3 entries Previous 1 Next

Figure 12: Figure depicting uneven task distribution on both executors when number of partitions = 512



Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
9	runJob at SparkHadoopWriter.scala:83 +details	2024/02/08 00:18:10	3.5 min	1/1		1254.0 MiB	3.3 GiB	
8	partitionBy at /mnt/data/part3-2.py:39 +details	2024/02/08 00:14:45	3.4 min	512/512			7.0 GiB	3.3 GiB
7	partitionBy at /mnt/data/part3-2.py:39 +details	2024/02/08 00:11:18	3.4 min	512/512			7.0 GiB	3.3 GiB
6	partitionBy at /mnt/data/part3-2.py:39 +details	2024/02/08 00:07:48	3.5 min	512/512			7.3 GiB	3.4 GiB
5	partitionBy at /mnt/data/part3-2.py:39 +details	2024/02/08 00:03:47	4.0 min	512/512			9.0 GiB	3.6 GiB
4	partitionBy at /mnt/data/part3-2.py:39 +details	2024/02/08 00:01:11	2.6 min	609/609			3.7 GiB	5.4 GiB
3	join at /mnt/data/part3-2.py:39 +details	2024/02/08 00:00:22	49 s	609/609			3.7 GiB	3.7 GiB
2	distinct at /mnt/data/part3-2.py:22 +details	2024/02/07 23:55:43	2.5 min	97/97	9.9 GiB			13.7 MiB
1	partitionBy at /mnt/data/part3-2.py:29 +details	2024/02/07 23:58:07	39 s	97/97			3.6 GiB	3.7 GiB
0	groupByKey at /mnt/data/part3-2.py:29 +details	2024/02/07 23:55:43	2.4 min	97/97	9.9 GiB			3.6 GiB

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

Figure 13: Figure depicting the completed stages of PageRank algorithm when number of partitions = 512

5.2.5 Number of partitions = 1024

The total execution time was 26 mins.

▾ Completed Stages (10)

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
9	runJob at SparkHadoopWriter.scala:83 +details	2024/02/08 00:45:43	4.5 min	1/1		1254.0 MiB	3.9 GiB	
8	partitionBy at /mnt/data/part3-2.py:39 +details	2024/02/08 00:42:34	3.1 min	1024/1024			7.6 GiB	3.9 GiB
7	partitionBy at /mnt/data/part3-2.py:39 +details	2024/02/08 00:39:24	3.2 min	1024/1024			7.6 GiB	3.9 GiB
6	partitionBy at /mnt/data/part3-2.py:39 +details	2024/02/08 00:36:11	3.2 min	1024/1024			7.9 GiB	3.9 GiB
5	partitionBy at /mnt/data/part3-2.py:39 +details	2024/02/08 00:32:30	3.7 min	1024/1024			9.8 GiB	4.2 GiB
4	partitionBy at /mnt/data/part3-2.py:39 +details	2024/02/08 00:29:31	3.0 min	1121/1121			3.9 GiB	6.1 GiB
3	join at /mnt/data/part3-2.py:39 +details	2024/02/08 00:28:35	56 s	1121/1121			3.7 GiB	3.9 GiB
2	distinct at /mnt/data/part3-2.py:22 +details	2024/02/08 00:23:54	2.5 min	97/97	9.9 GiB			13.7 MiB
1	partitionBy at /mnt/data/part3-2.py:29 +details	2024/02/08 00:26:19	41 s	97/97			3.6 GiB	3.7 GiB
0	groupByKey at /mnt/data/part3-2.py:29 +details	2024/02/08 00:23:54	2.4 min	97/97	9.9 GiB			3.6 GiB

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

Figure 14: Figure depicting the completed stages of PageRank algorithm when number of partitions = 1024

5.2.6 Observations and Inference

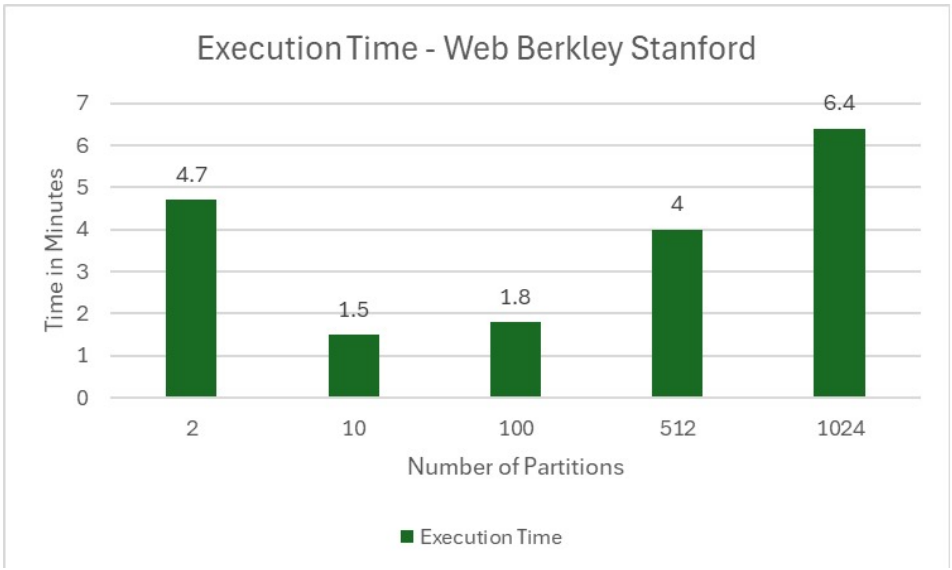


Figure 15: Execution Times for Berkeley-Stanford Dataset across different number of partitions

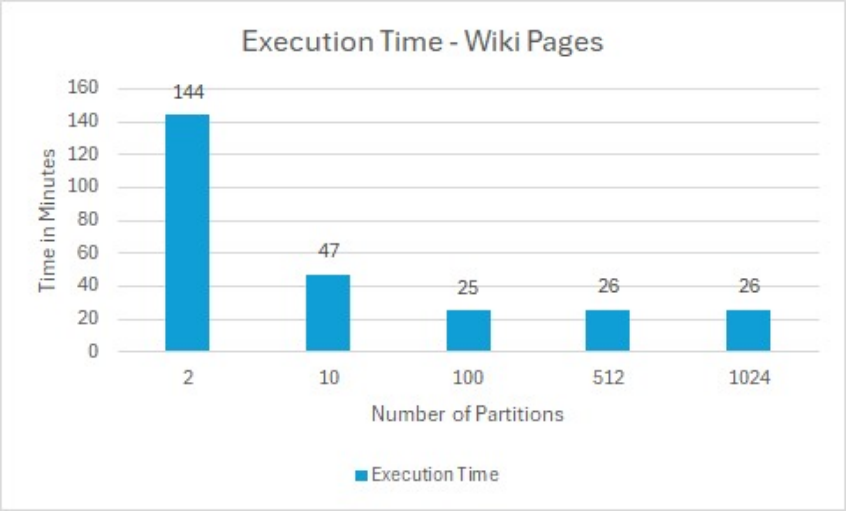


Figure 16: Execution Times for Wiki-Pages Dataset across different number of partitions

Number of partitions	"partitionBy" time	"Join" time
2	30 mins	5 mins
10	6 mins	1.4 mins
100	3 mins	52 secs
512	3.7 mins	47 secs
1024	3.2 mins	56 secs

Table 1: Table showing the "partitionBy" time and "Join time" across different partitions for 5 iterations

Thus, we observe that in the wiki-pages dataset, when the partitions are less (say 2 or 10), the time taken for each "partitionBy" stage is higher. It takes 35 mins for each "partitionBy" stage with 2 partitions, by using one executor, which causes the overall job to take more than 2 hours. When we use 100 partitions, we see a good speed up in the job, as it takes only 25 mins to complete. The "partitionBy" stage takes 3 mins, which is quite decent. Further increasing the number of partitions to 512 and 1024, doesn't change the execution time significantly. In the case of Barkley Stanford dataset, we observe that with a partition of 10 the execution time has reduced by 0.5X compared to execution without custom partitioning. As the number of partitions increase to 100, 512, 1024 the execution time increases. This is due to the increased time in writing the output, which increased to 2 mins for 1024 partitions.

5.3 Task 3

In this task, we need to persist RDDs and evaluate the performance. In our code, we repeatedly use the 'links' RDD and perform join on it. Hence we decided to persist this RDD.

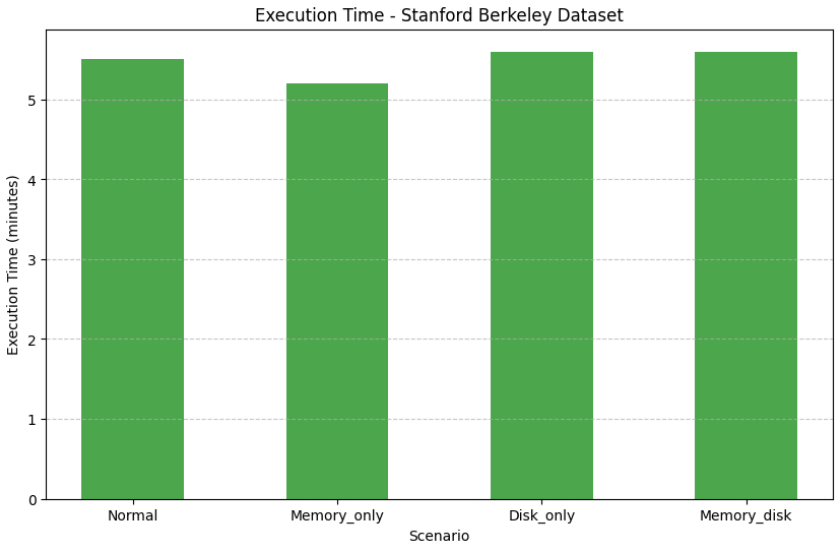


Figure 17: Execution times for the Stanford-Berkeley Dataset

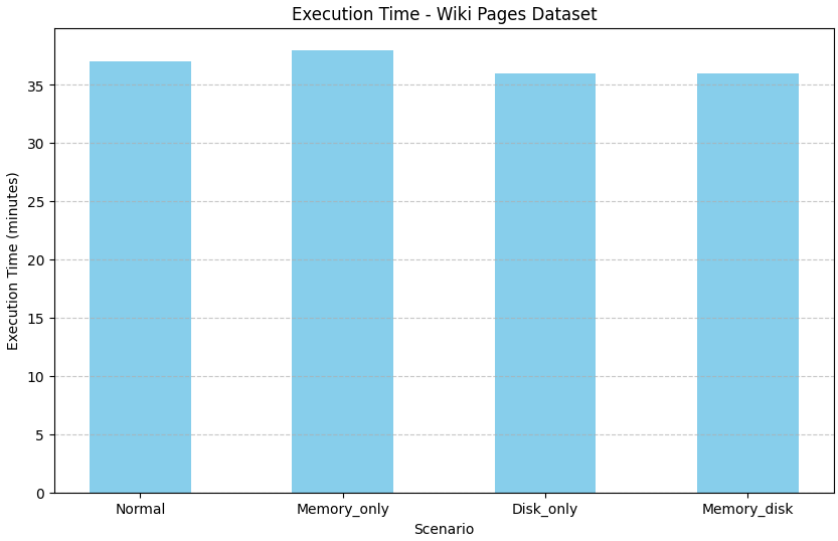
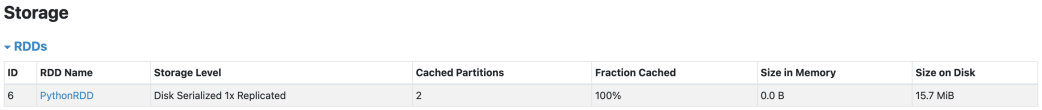


Figure 18: Execution times for the Wiki Pages Dataset

Spark offers various options for the persist function. We decided to compare the performance for 3 options - MEMORY_ONLY, DISK_ONLY and MEMORY_AND_DISK.

MEMORY_ONLY persists the RDD to memory in the JVM heap. DISK_ONLY persists the RDD to the disk. MEMORY_AND_DISK first stores partitions to memory and spills to disk as required. Since the data is always serialized on the Python side, these options use the serialized formats by default.



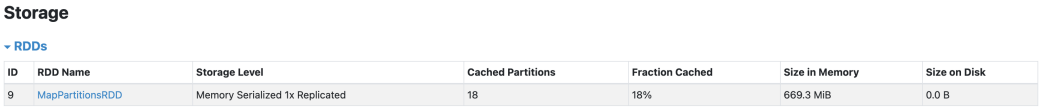
Storage

▾ RDDs

ID	RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size on Disk
6	PythonRDD	Disk Serialized 1x Replicated	2	100%	0.0 B	15.7 MiB

Figure 19: Storage showing RDD persisted in disk for Berkeley-Stanford dataset

For the Berkeley-Stanford dataset, we do not see much improvement when the 'links' RDD is persisted in the memory or the disk. This may be because the size of data is small and the time it takes to compute the RDD is not much. Hence caching it does not offer much benefit.

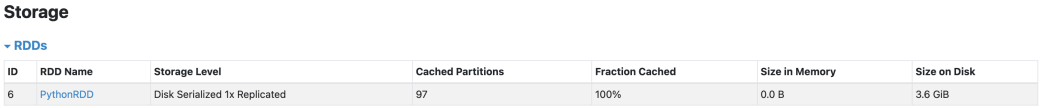


Storage

▾ RDDs

ID	RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size on Disk
9	MapPartitionsRDD	Memory Serialized 1x Replicated	18	18%	669.3 MiB	0.0 B

Figure 20: Storage showing RDD persisted in memory for Wiki dataset

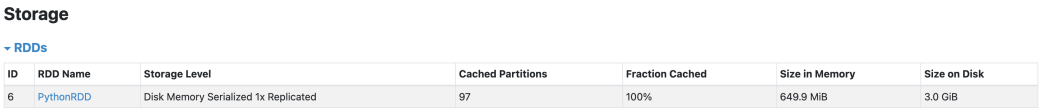


Storage

▾ RDDs

ID	RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size on Disk
6	PythonRDD	Disk Serialized 1x Replicated	97	100%	0.0 B	3.6 GiB

Figure 21: Storage showing RDD persisted in disk for Wiki dataset



Storage

▾ RDDs

ID	RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size on Disk
6	PythonRDD	Disk Memory Serialized 1x Replicated	97	100%	649.9 MiB	3.0 GiB

Figure 22: Storage showing RDD persisted in memory and disk for Wiki dataset

For the Wiki dataset Figure 18 and Figure 19, we see that the number of partitions cached as well as the memory cached is much higher when we persist in disk compared to when we persist in memory. . It is about 700 MB in memory while around 4 GB in disk. This parity is because a lot more data can be spilled on the disk and saved while the memory is limited.

Similar to the Berkeley-Stanford dataset, persisting RDDs did not offer much performance benefit for the Wiki dataset. When we persist in memory, the space is less and hence Spark may not be able to cache all partitions. This might be the reason we do not see any improvement. We also noted that for both the datasets, a few number of partitions for the 'links' RDD are cached in memory. This number keeps changing throughout the execution. This is due to the fact that Spark dynamically changes the partitions that are cached based on the execution. If some partitions are not required, Spark will remove them from cache. It also depends on the memory requirements. If might remove some partitions from cache if it needs to reclaim memory.

When we persist in disk, although it is able to cache all partitions, the overhead of reading data from the disk may be the reason why it does not offer much benefit. Saving to the disk still performs little better than saving to memory.

Figure 20 shows that the MEM_AND_DISK option performs similarly to the DISK_ONLY option as Spark persists most of the data on disk and less in memory.

Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
6	reduceByKey at /mnt/data/part3-3.py:33 +details	2024/02/08 19:00:40	1.6 min	388/388			4.4 GiB	2.9 GiB
5	join at /mnt/data/part3-3.py:30 +details	2024/02/08 18:59:20	1.3 min	388/388	3.6 GiB		2.9 GiB	4.4 GiB
4	reduceByKey at /mnt/data/part3-3.py:33 +details	2024/02/08 18:57:27	1.9 min	291/291			4.6 GiB	2.9 GiB
3	join at /mnt/data/part3-3.py:30 +details	2024/02/08 18:55:46	1.7 min	291/291	3.6 GiB		4.2 GiB	4.6 GiB
2	reduceByKey at /mnt/data/part3-3.py:33 +details	2024/02/08 18:53:37	2.2 min	194/194			3.6 GiB	4.2 GiB
1	join at /mnt/data/part3-3.py:30 +details	2024/02/08 18:52:13	1.4 min	194/194	3.6 GiB		3.6 GiB	3.6 GiB
0	groupByKey at /mnt/data/part3-3.py:20 +details	2024/02/08 18:49:47	2.4 min	97/97	9.9 GiB			3.6 GiB

Figure 23: Stages taking input from cached RDD for disk persistence

Figure 21 shows that the input for different stages is being taken from the cached RDD. We observe that the data shuffled for each stage reduces considerably after caching. After persisting the RDD on disk, the data shuffled reduced to around 4GB from 7GB.

5.4 Task 4

In this task, we trigger faults to observe how Spark ensures fault tolerance using RDDs. In our setup, we have 2 worker nodes and 1 master node. We killed one of the worker processes at different times. Thus, our job ran with only one executor for the remaining time.We performed this experiment on the Berkeley-Stanford and the Wiki dataset. For both the datasets, we see similar results.

When the worker node is killed at 25% application completion time (around stage 5), the spark job recomputes all the lost tasks and performs a retry on all stages. When the worker is killed at 75% application completion time (around stage 16), the spark job has to recompute more lost stages. This re-computation takes place on the remaining worker node. As the job has to recompute more number of stages when the worker is killed at 75% completion, the job takes more time to complete.

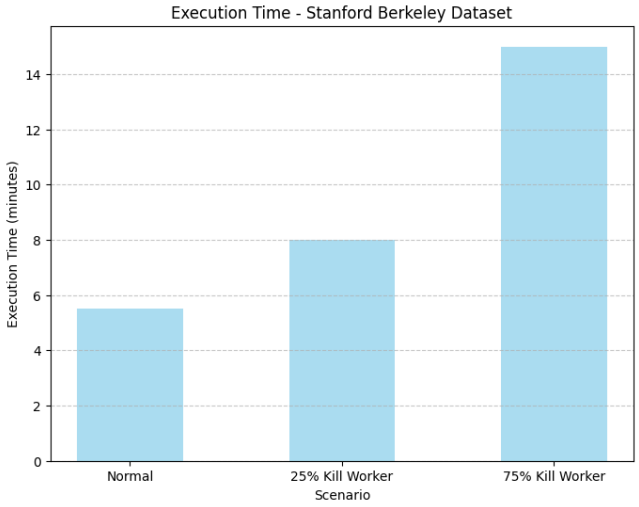


Figure 24: Execution times for the Stanford-Berkeley Dataset after killing a worker process

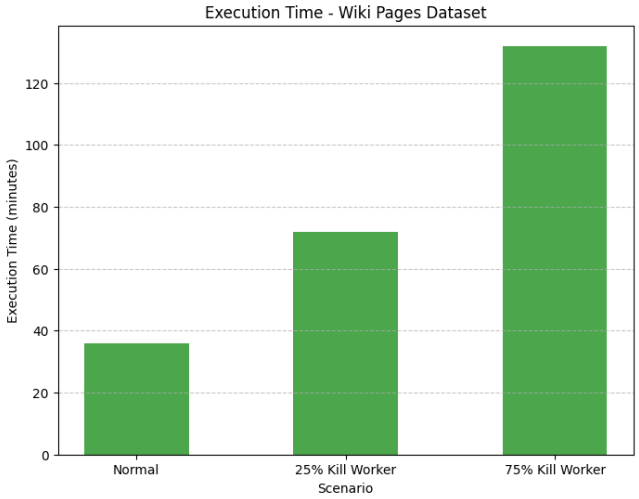


Figure 25: Execution times for the Wiki-Pages Dataset after killing a worker process

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs
0	128.105.146.153:38103	Active	0	0.0 B / 366.3 MiB	0.0 B	5	0	25	11833	11863	3.6 h (4.2 min)	9.9 GiB	109 GiB	77.2 GiB	stdout stderr
driver	sm220u-10s10641vm-1.wisc.cloudlab.us:33057	Active	0	0.0 B / 366.3 MiB	0.0 B	0	0	0	0	0	44 min (0.0 ms)	0.0 B	0.0 B	0.0 B	
1	128.105.146.154:45725	Dead	0	0.0 B / 366.3 MiB	0.0 B	5	5	266	5067	5072	1.7 h (2.0 min)	5 GiB	49.1 GiB	34.9 GiB	stdout stderr

Showing 1 to 3 of 3 entries

Previous 1 Next

Figure 26: Executor statistics when one worker is killed at 75% application time

With our setup, the job was not able to complete when one worker was killed during 75% application completion time, even after 2.5 hours for the Wiki dataset. This is because recomputing 16 stages on the single worker increased the memory load. Around 100 GB of data was shuffled across (read and write) for this executor. This means that many intermediate files were written for the shuffled data on the existing worker node, hence, the memory load on this node increased drastically. Thus, this node ran out of space (even after using the extra mounted directory for temporary files)

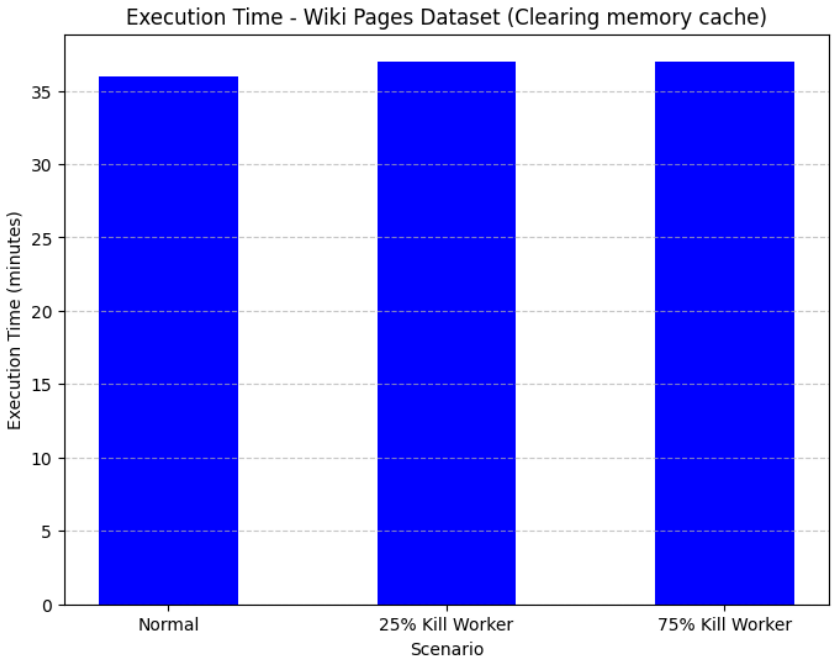


Figure 27: Execution times for the Wiki-Pages Dataset after clearing the memory cache

We also ran the given command to clear the memory cache in one of the workers. We did not kill any worker process this time. Only clearing the memory cache on one of the workers did not affect the performance. Spark stores some RDD data

in memory, hence performance could be affected by clearing the memory cache. However, as we perform many shuffle operations, Spark writes intermediate data in files to the disk. Hence, clearing the memory cache did not have an impact on the execution time as data from the intermediate files is read and no re-computation is needed.

6 Contributions

- **Environment setup** : Aanandita Dhawan, Aboli Pai, Tanisha Hegde and Lakshika Rathi
- **Sorting Application** : Aanandita Dhawan, Aboli Pai and Tanisha Hegde
- **Task-1 code and execution** : Aboli Pai and Lakshika Rathi
- **Task-2 code and execution** : Tanisha Hegde
- **Task-3 code** : Aanandita Dhawan
- **Task-3 and Task-4 execution** : Aboli Pai
- **Report writing, Result collection and Graphs** : Lakshika Rathi and Aboli Pai