Assignment I - Page Rank Algorithm

Group V
Badri Narayanan Murali Krishnan
Shreyas Subramanian
Rohan Purandhar
FNU Nevin John Selby
bmuralikrish@wisc.edu, ss7@wisc.edu
purandhar@wisc.edu, nselby@wisc.edu

Github Link: https://github.com/MBadriNarayanan/CS744

February 9, 2024

1 Tasks

1.1 Task One

Write a Scala/Python/Java Spark application that implements the PageRank algorithm.

1.1.1 Observation

- The initial PageRank implementation can be checked for accuracy using the smaller Berkeley graph.
- The wider Wikipedia graph evaluates the performance and scalability of utilizing Spark to execute distributed PageRank on a cluster. On both datasets, PageRank scores converge during ten iterations.
- Performance is enhanced by caching important RDDs to prevent recomputation.
- On real-world graphs, custom partitioning helps prevent skew.

1.1.2 Inference

- Spark provides a scalable way to run PageRank on large graphs by parallelizing computation.
- PageRank and other iterative graph algorithms can be expressed well with RDD API.

- Spark execution configuration, segmentation, and caching can all be used to maximize performance.

1.2 Task Two

In order to achieve high parallelism, Spark will split the data into smaller chunks called partitions which are distributed across different nodes in the cluster. Partitions can be changed in several ways. For example, any shuffle operation on a DataFrame (e.g., join()) will result in a change in partitions (customizable via user's configuration). In addition, one can also decide how to partition data when writing DataFrames back to disk. For this task, add appropriate custom DataFrame/RDD partitioning and see what changes.

1.2.1 Observation

- Compared to default, partitioning after joins reduces skew and boosts performance.
- More parallelism is possible with more partitions, but overhead increases as a result.
- Data is distributed equally when hash/range partitioning is applied to the appropriate columns.
- Shuffle IO and runtime can decrease with ideal partitioning, but they can also increase if improperly executed.

1.2.2 Inference

- For some workloads, such as PageRank, partitioning can have a major effect on how well a job performs.
- It is necessary to assess the trade-offs between overhead, shuffle costs, and partitioning granularity.
- Performance profiling along with iteration are necessary to determine the optimal partitioning technique.
- Tailoring partitions to individual needs offers adjustment options to maximize Spark workload efficiency.

1.3 Task Three

: Persist the appropriate DataFrame/RDD(s) as in-memory objects and see what changes.

1.3.1 Observation

- Caching the edges and linkages Every iteration, DataFrame prevents rereading from disk.
- Reaching the top Every iteration, DataFrame/RDD prevents complete recomputation.
- Caching is intended to enhance runtimes, but memory use will go up.
- Benefits will become more apparent with larger datasets because disk input operations cost more.
- Large data sizes results in out-of-memory issues due to excessive caching.
- Taking cache invalidation into account when data modifications may be necessary.

1.3.2 Inference

- Because persisting RDDs minimize disk IO, they improve optimization for iterative algorithms.
- Each use case must assess the trade-off between memory and compute.
- Performance and cost can be optimized by adjusting storage levels and persistence mechanisms.
- Viability is provided via caching when datasets grow and disk input becomes a bottleneck.

1.4 Task Four

: Kill a Worker process and see the changes. You should trigger the failure to a desired worker VM when the application reaches 25% and 75% of its lifetime.

1.4.1 Part 1 - Triggering failure at 25% progress

- On other nodes, the job will pause and resume failing tasks.
- When caching is enabled, a slowdown occurs since lost data would need to be recalculated.
- The work will take longer overall and computation continues and finishes.
- Spark re-executes failed tasks, making it resilient to worker failures.
- Caching lessens the effect by preventing whole recomputation.
- Failures do not guarantee the completion of the job.

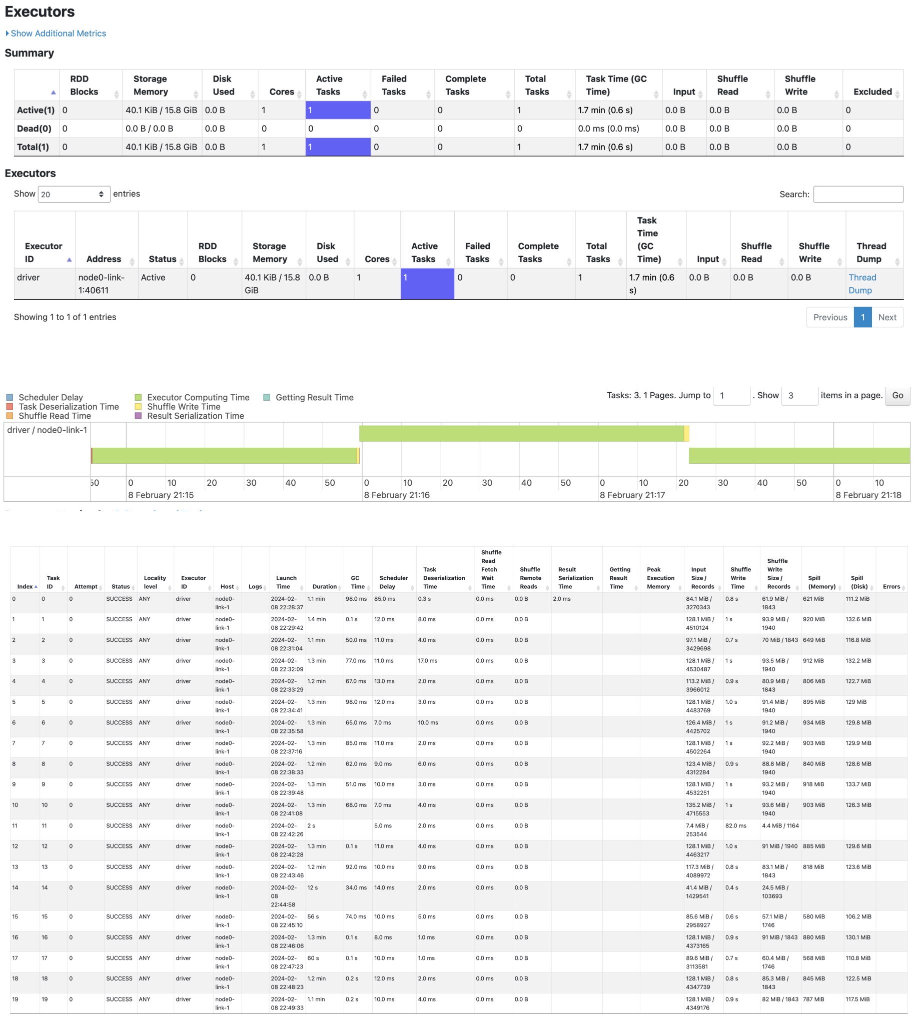
1.4.2 Part 2 - Triggering failure at 75% progress

- Impact on work time is less because there are fewer iterations remaining.
- Few calculations are wasted due to caching not being used because most data was ephemeral.
- More data loss that requires recalculating if caching was enabled.
- Failures by later workers have less of an effect because most of the work is done.
- The effects of job failures later on can be amplified by caching larger datasets.

2 Contribution

- Badri
  1. Acted as the team lead and formulated the majority of the code.
  2. Maintained version control with Git.
  3. Generated the report using LaTeX.
  4. Gave the idea to set up a config bash file with the input arguments already present to replicate results.
- Shreyas
  1. Formulated the code for Simple Page Rank.
  2. Implemented the config bash script to replicate results.
  3. Wrote the content for the report.
- Rohan
  1. Monitored the Spark UI and gave insights.
  2. Worked on code documentation.
  3. Formulated the code for custom partitioning.
- Nevin
  1. Formulated the code for custom partitioning.
  2. Worked on the content for the report.
  3. Documented the code

  The entire team worked on setting up the environment for Spark and Hadoop and making sure the simple program ran seamlessly. We also had an in-depth discussion about the results of our run.



Executors

▸ Show Additional Metrics

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(1)	0	40.1 KiB / 15.8 GiB	0.0 B	1	1	0	0	1	1.7 min (0.6 s)	0.0 B	0.0 B	0.0 B	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(1)	0	40.1 KiB / 15.8 GiB	0.0 B	1	1	0	0	1	1.7 min (0.6 s)	0.0 B	0.0 B	0.0 B	0

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	Shuffle Write	Thread Dump
driver	node0-link-1:40611	Active	0	40.1 KiB / 15.8 GiB	0.0 B	1	1	0	0	1	1.7 min (0.6 s)	0.0 B	0.0 B	0.0 B	Thread Dump

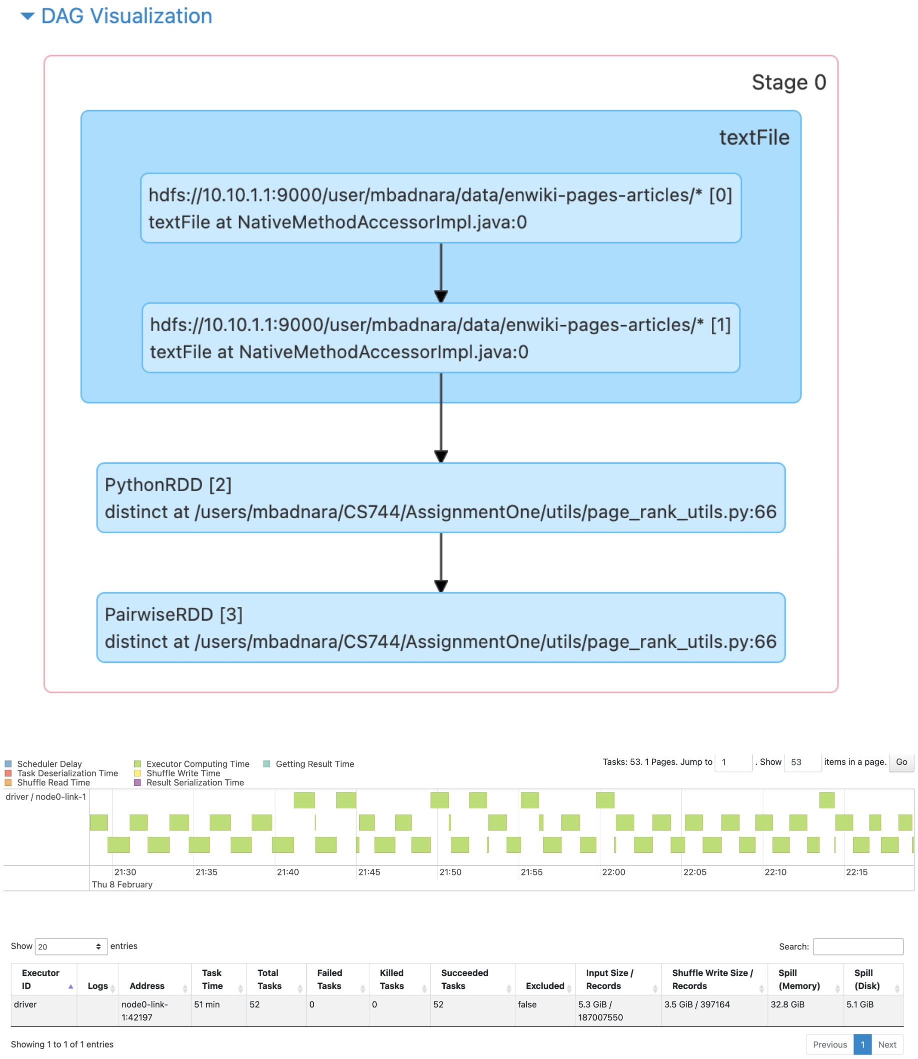
Showing 1 to 1 of 1 entries

Previous 1 Next

Tasks: 3. 1 Pages. Jump to 1 . Show 3 items in a page. Go

Index	Task ID	Attempt	Status	Locality level	Executor ID	Host	Logs	Launch Time	Duration	GC Time	Scheduler Delay	Task Deserialization Time	Shuffle Read Fetch Wait Time	Shuffle Remote Reads	Result Serialization Time	Getting Result Time	Peak Execution Memory	Input Size / Records	Shuffle Write Time	Shuffle Write Size / Records	Spill (Memory)	Spill (Disk)	Errors
0	0	0	SUCCESS	ANY	driver	node0-link-1		2024-02-08 22:28:37	1.1 min	98.0 ms	85.0 ms	0.3 s	0.0 ms	0.0 B	2.0 ms			84.1 MiB / 3270343	0.8 s	61.9 MiB / 1843	621 MiB	111.2 MiB	
1	1	0	SUCCESS	ANY	driver	node0-link-1		2024-02-08 22:29:42	1.4 min	0.1 s	12.0 ms	8.0 ms	0.0 ms	0.0 B				128.1 MiB / 4510124	1 s	93.9 MiB / 1940	920 MiB	132.6 MiB	
2	2	0	SUCCESS	ANY	driver	node0-link-1		2024-02-08 22:31:04	1.1 min	50.0 ms	11.0 ms	4.0 ms	0.0 ms	0.0 B				97.1 MiB / 3429698	0.7 s	70 MiB / 1843	649 MiB	116.8 MiB	
3	3	0	SUCCESS	ANY	driver	node0-link-1		2024-02-08 22:32:09	1.3 min	77.0 ms	11.0 ms	17.0 ms	0.0 ms	0.0 B				128.1 MiB / 4530487	1 s	93.5 MiB / 1940	912 MiB	132.2 MiB	
4	4	0	SUCCESS	ANY	driver	node0-link-1		2024-02-08 22:33:29	1.2 min	67.0 ms	13.0 ms	2.0 ms	0.0 ms	0.0 B				113.2 MiB / 3966012	0.9 s	80.9 MiB / 1843	806 MiB	122.7 MiB	
5	5	0	SUCCESS	ANY	driver	node0-link-1		2024-02-08 22:34:41	1.3 min	98.0 ms	12.0 ms	3.0 ms	0.0 ms	0.0 B				128.1 MiB / 4483769	1.0 s	91.4 MiB / 1940	895 MiB	129 MiB	
6	6	0	SUCCESS	ANY	driver	node0-link-1		2024-02-08 22:35:58	1.3 min	65.0 ms	7.0 ms	10.0 ms	0.0 ms	0.0 B				126.4 MiB / 4425702	1 s	91.2 MiB / 1940	934 MiB	129.8 MiB	
7	7	0	SUCCESS	ANY	driver	node0-link-1		2024-02-08 22:37:16	1.3 min	85.0 ms	11.0 ms	2.0 ms	0.0 ms	0.0 B				128.1 MiB / 4502264	1 s	92.2 MiB / 1940	903 MiB	129.9 MiB	
8	8	0	SUCCESS	ANY	driver	node0-link-1		2024-02-08 22:38:33	1.2 min	62.0 ms	9.0 ms	6.0 ms	0.0 ms	0.0 B				123.4 MiB / 4312284	0.9 s	88.8 MiB / 1940	840 MiB	128.6 MiB	
9	9	0	SUCCESS	ANY	driver	node0-link-1		2024-02-08 22:39:48	1.3 min	51.0 ms	10.0 ms	3.0 ms	0.0 ms	0.0 B				128.1 MiB / 4532251	1 s	93.2 MiB / 1940	918 MiB	133.7 MiB	
10	10	0	SUCCESS	ANY	driver	node0-link-1		2024-02-08 22:41:08	1.3 min	68.0 ms	7.0 ms	4.0 ms	0.0 ms	0.0 B				135.2 MiB / 4715553	1 s	93.6 MiB / 1940	903 MiB	126.3 MiB	
11	11	0	SUCCESS	ANY	driver	node0-link-1		2024-02-08 22:42:26	2 s		5.0 ms	2.0 ms	0.0 ms	0.0 B				7.4 MiB / 253544	82.0 ms	4.4 MiB / 1164			
12	12	0	SUCCESS	ANY	driver	node0-link-1		2024-02-08 22:42:28	1.3 min	0.1 s	11.0 ms	4.0 ms	0.0 ms	0.0 B				128.1 MiB / 4463217	1.0 s	91 MiB / 1940	885 MiB	129.6 MiB	
13	13	0	SUCCESS	ANY	driver	node0-link-1		2024-02-08 22:43:46	1.2 min	92.0 ms	10.0 ms	9.0 ms	0.0 ms	0.0 B				117.3 MiB / 4089972	0.8 s	83.1 MiB / 1843	818 MiB	123.6 MiB	
14	14	0	SUCCESS	ANY	driver	node0-link-1		2024-02-08 22:44:58	12 s	34.0 ms	14.0 ms	2.0 ms	0.0 ms	0.0 B				41.4 MiB / 1429541	0.4 s	24.5 MiB / 103693			
15	15	0	SUCCESS	ANY	driver	node0-link-1		2024-02-08 22:45:10	56 s	74.0 ms	10.0 ms	5.0 ms	0.0 ms	0.0 B				85.6 MiB / 2958927	0.6 s	57.1 MiB / 1746	580 MiB	106.2 MiB	
16	16	0	SUCCESS	ANY	driver	node0-link-1		2024-02-08 22:46:06	1.3 min	0.1 s	8.0 ms	1.0 ms	0.0 ms	0.0 B				128.1 MiB / 4373165	0.9 s	91 MiB / 1843	880 MiB	130.1 MiB	
17	17	0	SUCCESS	ANY	driver	node0-link-1		2024-02-08 22:47:23	60 s	0.1 s	10.0 ms	1.0 ms	0.0 ms	0.0 B				89.6 MiB / 3113581	0.7 s	60.4 MiB / 1746	568 MiB	110.8 MiB	
18	18	0	SUCCESS	ANY	driver	node0-link-1		2024-02-08 22:48:23	1.2 min	0.2 s	12.0 ms	2.0 ms	0.0 ms	0.0 B				128.1 MiB / 4347739	0.8 s	85.3 MiB / 1843	845 MiB	122.5 MiB	
19	19	0	SUCCESS	ANY	driver	node0-link-1		2024-02-08 22:49:33	1.1 min	0.2 s	10.0 ms	4.0 ms	0.0 ms	0.0 B				128.1 MiB / 4349176	0.9 s	82 MiB / 1843	787 MiB	117.5 MiB	

Figure 1: Output snapshots - Part One



DAG Visualization

Stage 0

textFile

hdfs://10.10.1.1:9000/user/mbadnara/data/enwiki-pages-articles/* [0]
textFile at NativeMethodAccessorImpl.java:0

hdfs://10.10.1.1:9000/user/mbadnara/data/enwiki-pages-articles/* [1]
textFile at NativeMethodAccessorImpl.java:0

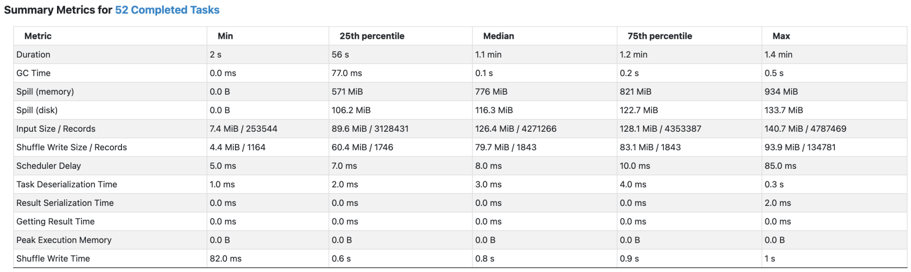
PythonRDD [2]
distinct at /users/mbadnara/CS744/AssignmentOne/utils/page_rank_utils.py:66

PairwiseRDD [3]
distinct at /users/mbadnara/CS744/AssignmentOne/utils/page_rank_utils.py:66

Executor ID	Logs	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Excluded	Input Size / Records	Shuffle Write Size / Records	Spill (Memory)	Spill (Disk)
driver		node0-link-1:42197	51 min	52	0	0	52	false	5.3 GiB / 187007550	3.5 GiB / 397164	32.8 GiB	5.1 GiB

Showing 1 to 1 of 1 entries

Figure 2: Output snapshots - Part Two



Summary Metrics for 52 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	2 s	56 s	1.1 min	1.2 min	1.4 min
GC Time	0.0 ms	77.0 ms	0.1 s	0.2 s	0.5 s
Spill (memory)	0.0 B	571 MiB	776 MiB	821 MiB	934 MiB
Spill (disk)	0.0 B	106.2 MiB	116.3 MiB	122.7 MiB	133.7 MiB
Input Size / Records	7.4 MiB / 253544	89.6 MiB / 3128431	126.4 MiB / 4271266	128.1 MiB / 4353387	140.7 MiB / 4787469
Shuffle Write Size / Records	4.4 MiB / 1164	60.4 MiB / 1746	79.7 MiB / 1843	83.1 MiB / 1843	93.9 MiB / 134781
Scheduler Delay	5.0 ms	7.0 ms	8.0 ms	10.0 ms	85.0 ms
Task Deserialization Time	1.0 ms	2.0 ms	3.0 ms	4.0 ms	0.3 s
Result Serialization Time	0.0 ms	0.0 ms	0.0 ms	0.0 ms	2.0 ms
Getting Result Time	0.0 ms	0.0 ms	0.0 ms	0.0 ms	0.0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Shuffle Write Time	82.0 ms	0.6 s	0.8 s	0.9 s	1 s

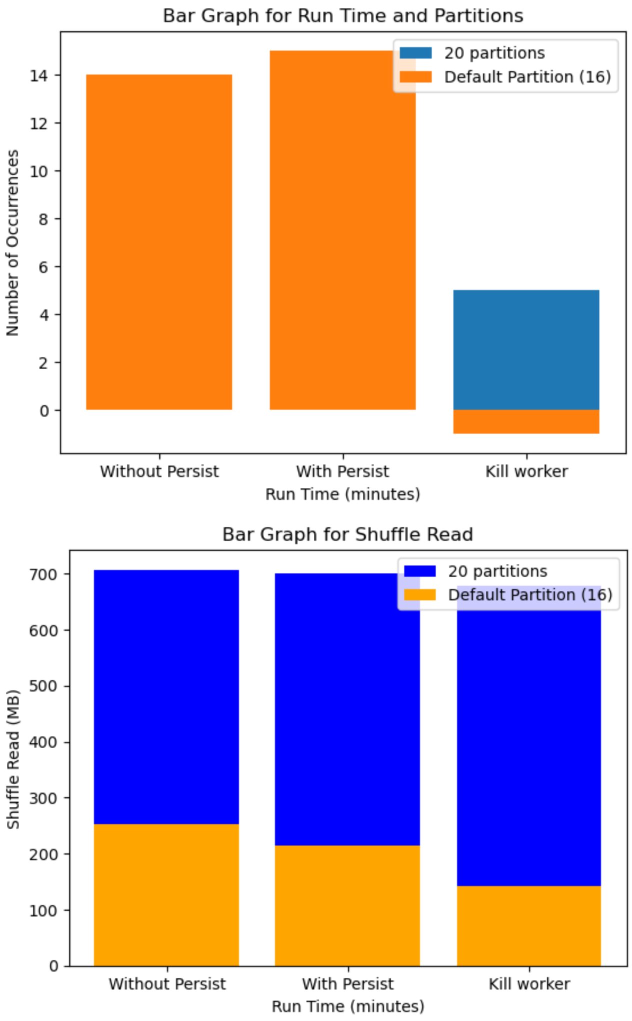


Figure 3: Output snapshots - Part Three

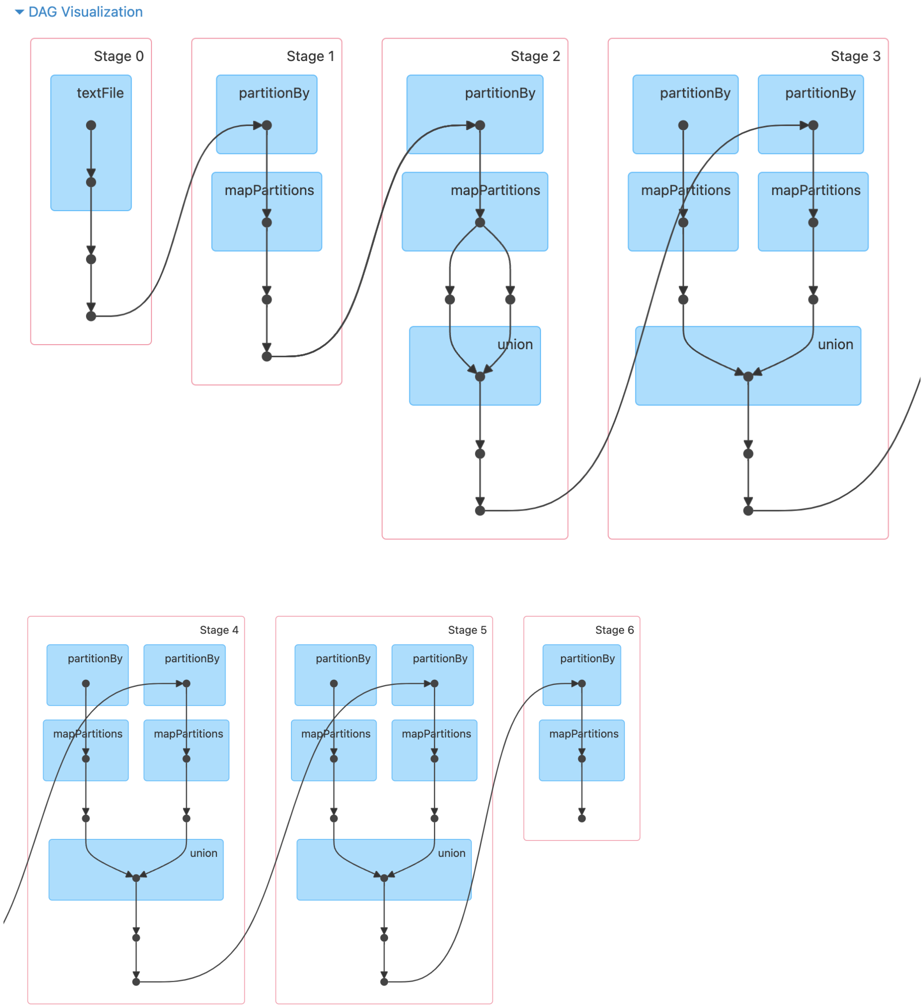


Figure 4: Output snapshots - Part Four