CS 838 Assignment 2 Report

Group members:
Xiaodong Wang (xwang322@wisc.edu),
Minghao Dai (mdai9@wisc.edu),

Part A: Apache Spark

1. We wrote a Python Script for this application and we record following data.

Time	Tasks #	N/R	N/W	S/R	S/W
6.4 min	572	0.27 GB/min	0.424 GB/min	5.4 MB/min	0.323 GB/min

2. We revised our codes to partition the input file into multiple numbers of partitions, we separate into 2, 10, 50, 100, 300.

Partition #	Time	Tasks #	N/R	N/W	S/R	S/W
2	6.3 min	572	0.28 GB/min	0.42 GB/min	5.1 MB/min	0.3 GB/min
10	5.2 min	1430	0.40 GB/min	0.47 GB/min	6.9 MB/min	0.28 GB/min
50	6.3 min	7150	0.33 GB/min	0.58 GB/min	8.1 MB/min	0.35 GB/min
100	9.2 min	14300	0.36 GB/min	0.61 GB/min	11.3 MB/min	0.41 GB/min
300	22 min	42900	0.52 GB/min	0.77 GB/min	29.4 MB/min	0.77 GB/min

3. As partition 10 gives the best performance time, we decide to proceed with this number with persist RDD setting in the codes.
Partition number: 10

Time	Tasks #	N/R	N/W	S/R	S/W
5 min	1430	0.29 GB/min	0.29 GB/min	127.7 MB/min	0.29 GB/min

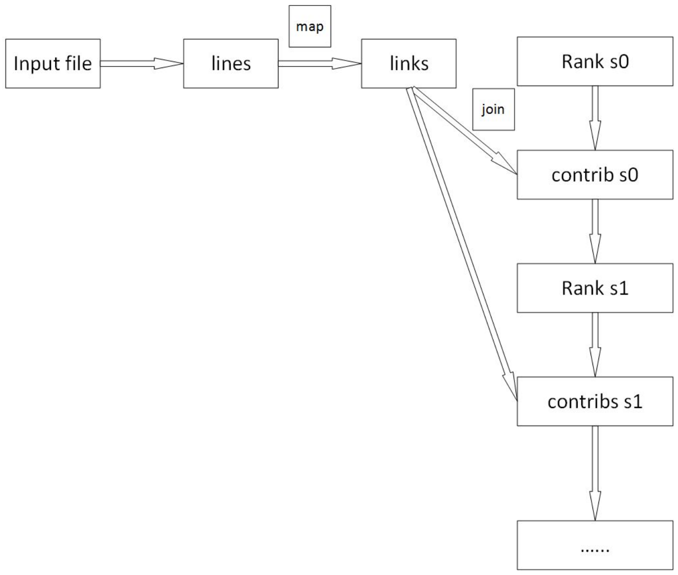
We find some interesting facts as follow.

a. The number of partition is direct ratio with the number of tasks, which is straightforward as more partitioned input files will introduce more map and reduce tasks for the input data. What is more, more partitioned RDDs in earlier iterations will generate more tasks in the following iterations because join functions in Spark API used in our codes will hash partitioned data then shuffle through network. This creates more tasks in future iterations. The sum of all tasks within a job is thus direct ratio with the number of partitions.
b. Having a persistent RDD does not increase the completion time a lot. It will cost some storage read because some RDDs will be persistent. But the network read/write decreases as we gives the possible reason that some copies of RDDs are located on the machine for some next iterations which saves the shuffle read/write through network.
c. We run our codes in Python with Spark API functions and classes, but it is interesting to point out that we also write another Python script with no Spark API functions getting the same results

however, it takes less time to finish. This is fun together with the fact that Scala will spend less time to complete than Python with Spark API. We think achieving the same functions in different languages and how to optimize the script efficiency is worth investigations.

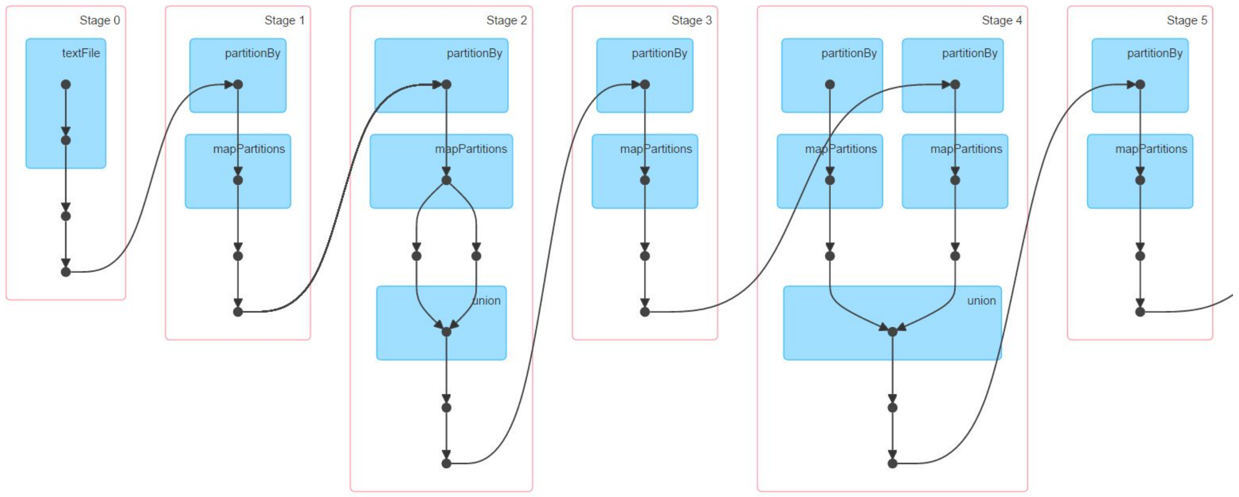
4. As results shown before, it is not always true that more partitions will give better performances, as more partitions will introduce more shuffle read and write through network which causes bandwidth be a bottleneck in completion time. We currently found that partitioned into 10 parts or 50 parts will give the best performances. We have not used customized partition as Spark uses a default hashing partition function which we think it is already good. Above value of 100, completion times increases a lot so we come to this point that it is not suitable to run partitioned RDD at a value higher than 100.

5. We plot the lineage graph for all applications we develop. This is the same for all applications as their differences are located in partitioned textfiles and persist RDD, but they share the same lineage.

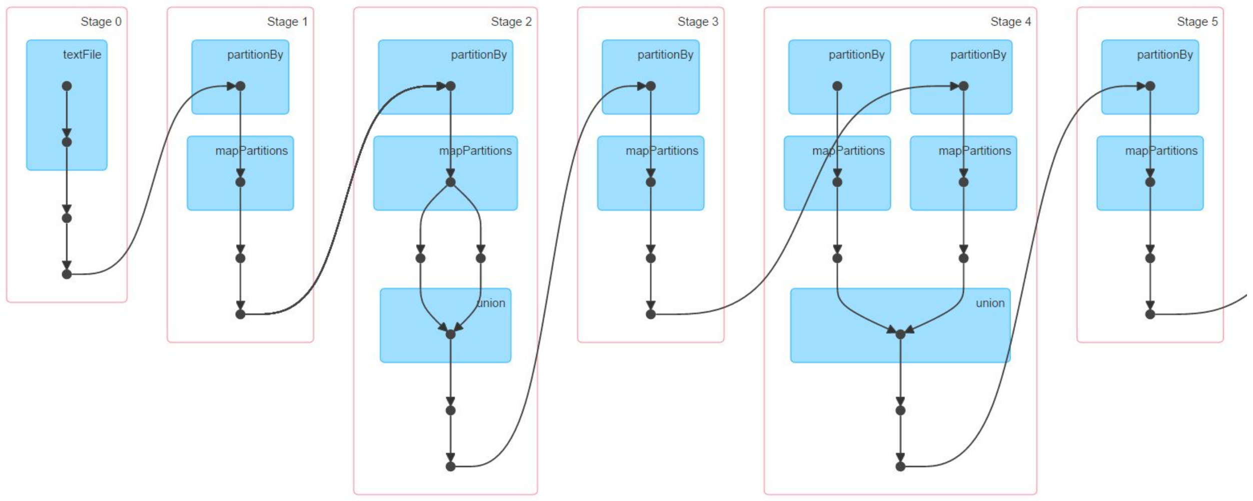


6. We make a screenshot of the stage DAG for all three applications till the second iteration.

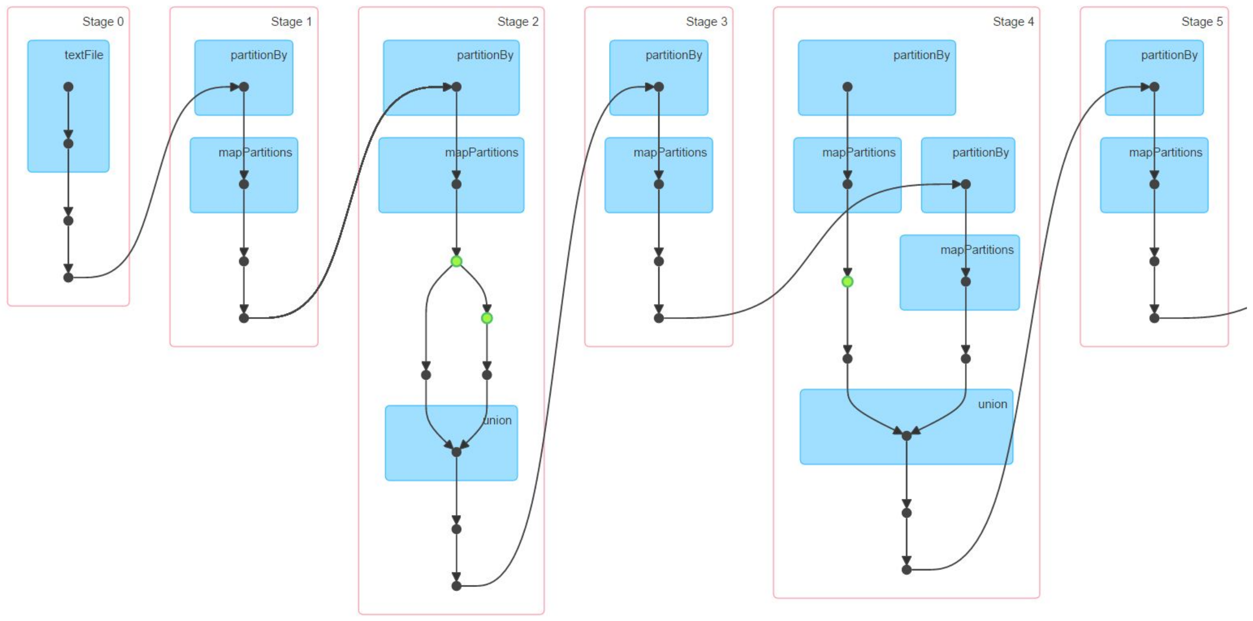
a. For application with no partition and no persist RDD:



b. For application with partition and without persist RDD:



c. For application with partition and with persist RDD:



From what we observe, most of stage DAG is almost same for all applications we have developed. The most differences is there will be green dots on the process flow from one block to another inside one stage. This is due to persist RDD we implement. We think having a different DAG be different for completion time. Because Spark will build DAG based on RDD lineage graph and splits the DAG into stages that contain pipelined transformations with narrow dependencies. So different DAG will have generate different dependencies relationship between RDD which influences completion time.

7. We evaluate the performances of fault tolerance by clear the memory cache and kill the process worker. There are the results we test on application in question 1 with no persistent RDD and application in question 3 with persistent RDD.

a. Clear the memory cache

	25 % lifetime	75% lifetime
Question 1	6.4 min	6.3 min
Question 3	4.7 min	4.6 min

b. Kill the worker process

	25 % lifetime	75% lifetime
Question 1	N/A	N/A
Question 3	5.9 min	8.2 min

What we observe is that clearing the memory cache does not increase the running time much. They are on the same level. There could be possibility that clearing the cache even decreases the completion time. But as what we learnt from assignment 1, running the same codes continuously might decrease the completion time. So we can draw a conclusion that clearing the cache memory is not influencing much on completion time. However, kill the worker gives more effects on completion. Without persistent RDD, application will never get back. With persistent RDD, it takes longer to recover. The later the worker is killed, the more time it takes to recover. Killing at 25% lifetime gives roughly 20% more completion time while killing at 75% will increase the completion time 60% more.