CS744 Report of Assignment 1

Group8: Xin Jin *and* Hongyi Wang *and* Yuzhe Ma

October 2, 2017

1 Part A

The proposed queries {12, 21, 50, 71, 85} have been run for 3 times in this experiment to mitigate biases in each signal experiment trial. The results reported below are averaged based on those 3 experiments.

1.1 Part A-1-a

Plot the query completion time. Your plot should have five queries on the X-axis, and for each query two columns. One describing the job completion time with Hive/MR and one running Hive/Tez. What do you observe? Is Hive/Tez always better than Hive/MR? By how much? Is the difference always constant? If not, why?

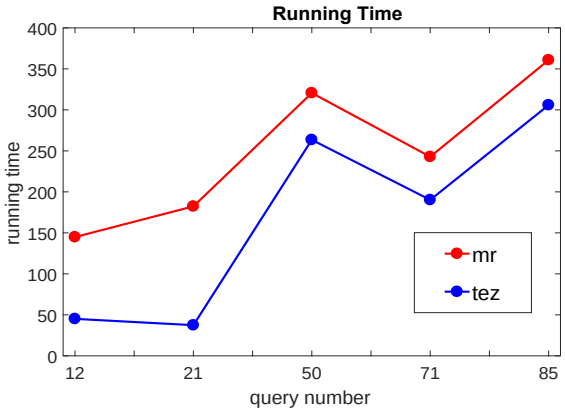


Figure 1: Time costs of Hive/MR vs Hive/Tez

We've observed that for each query Hive/Tez is faster then Hive/MR. but the difference is **not** constant. For query12 and query21 execution, Hive/Tez was much faster than Hive/MR(3.2 and 4.8 times). And the time cost for query12 and query21 are lighter compared to other queries for both Hive/MR and Hive/Tez. With respect to query50, query71, and query85 Hive/Tez is also faster than

Hive/MR, but is only 18%, 21%, 15% faster respectively. Since we knew that, there are many read/write and network communication intensive operations in Hive/MR, so ,there might be a I/O (read and write) and communication optimization strategy in Hive/Tez, which lead to the better performance with respect to time consumption.

1.2 Part A-1-b

Compute the amount of network/storage read/write bandwidth used during the query lifetime. Your plot(s) should clearly describe the amount of data read/written from network/disk for every query and for each framework(Hive/MR and Hive/Tez). What do you observe? Is there a trend? What are the possible explanations for your results/anomalies?

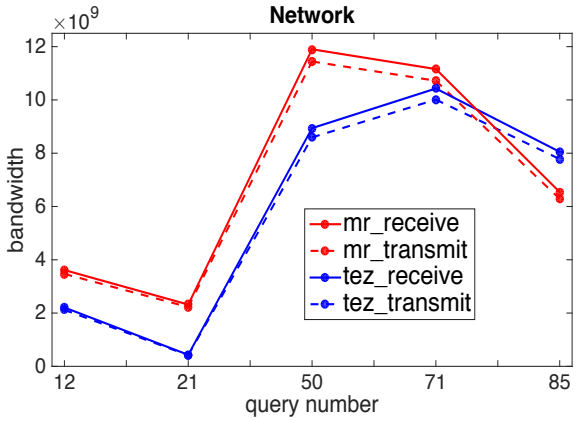


Figure 2: Bandwidth of network for Hive/MR and Hive/Tez.

We've observed that in general the amount of resource (especially for the network bandwidth) required by queries somehow correlates with the query execution time. Roughly, the longer a query execution costs (for both Hive/MR and Hive/Tez), the more resources costs by this query. Since we already observed in *Part A-1-a* the Hive/Tez is usually faster than Hive/MR, we can observe that for query12, query21, query50, the resources cost (both bandwidth and disk usage) of Hive/Tez is lower than Hive/MR. Therefore, there might be an resource assignment optimization in Hive/Tez to make sure low resource usage. For query71 and query85, we can observe that the disk usage of Hive/Tez are higher than Hive/MR, this might cause by that the communication cost for jobs (query71 and query85) is quite high, thus to get global optimum performance Hive/Tez increase the disk resource usage to reduce the communication cost.

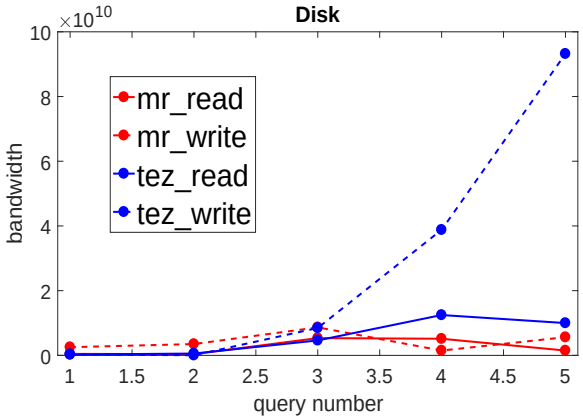


Figure 3: Bandwidth of disk for Hive/MR and Hive/Tez.

1.3 Part A-1-c

Compute various task statistics exploring the log history files stored in HDFS. More precisely, for every query and Hive/MR, Hive/Tez, you should compute the following: total number of tasks; ratio of tasks which aggregates data versus the number of tasks which read data from HDFS; task distribution over query lifetime. Is there any correlation between these metrics and performance? Why/why not?

- Hive/MR

Statistics of Tasks in Hive/MR		
Query index	total number of tasks	ratio(aggregate:read)
12	42	0.076923077
21	22	0.1
50	185	0.907216495
71	174	0.011627907
85	94	0.740740741

- Hive/Tez

Statistics of Tasks in Hive/Tez		
Query index	total number of tasks	ratio(aggregate:read)
12	55	0.058
21	30	0.069
50	110	0.054
71	172	0.116
85	84	0.149

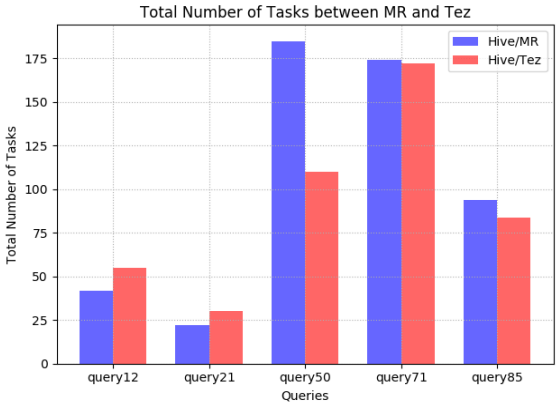


Figure 4: Total Number of Tasks Hive/Mr vs Hive/Tez

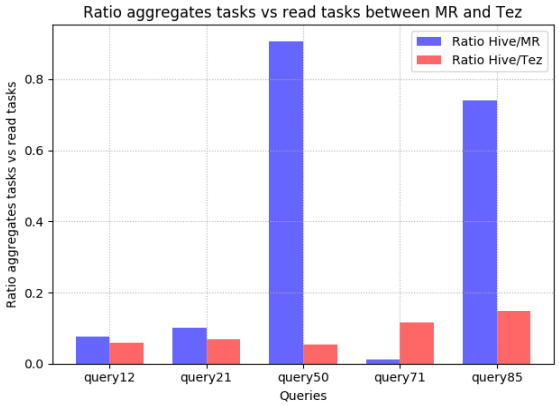
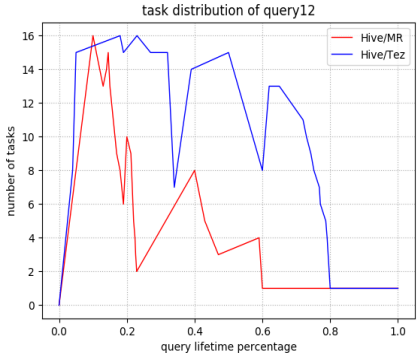
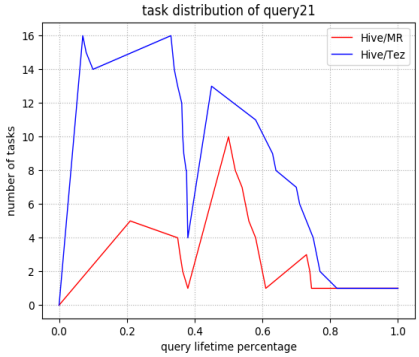


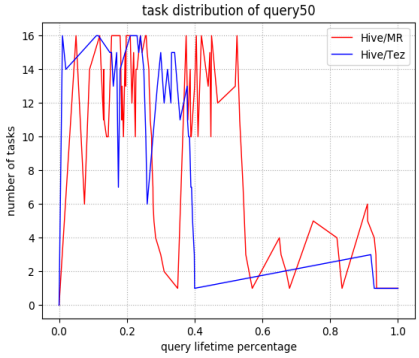
Figure 5: Ratio of Tasks Hive/Mr vs Hive/Tez



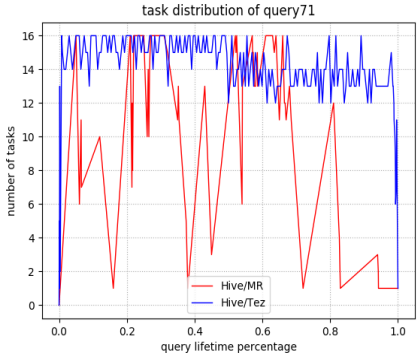
(a) Q12



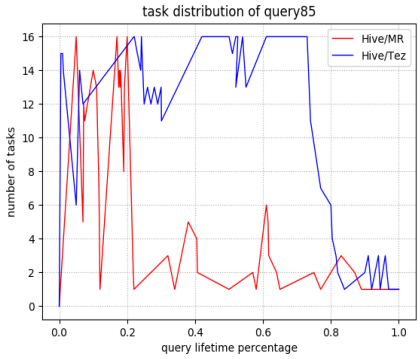
(b) Q21



(c) Q50



(d) Q71



(e) Q85

Figure 6: Task Distribution for Each Query

We firstly analyze Figure 4, which showed the total number of tasks between Hive/MR and Hive/Tez. We found that for query12 and query21, we can't observe an obvious trend (correlation) between number of tasks and performance *wrt* completion time. But for query50, query71, query85, we can tell that, small number of tasks gives faster computation. It might be the case that, for query12 and query21, Hive/Tez parses the whole job to more small local tasks, thus the performance can be slightly better.
Also, we can't observe any obvious trend for ratio of aggregation tasks and read tasks. For query12, query21, query50, and query85, we get higher ratio of Hive/MR than Hive/Tez, but for query71 things is different. It can be the case that higher aggregation/read tasks ratio might lead to worse performance. But that is not always the case. In conclusion, there is no obvious trend according to statistics of tasks we collected currently. We believe that the performance is more relevant to specific job properties and schedule than total number of tasks and aggregation/read task ratio.
For the tasks distributions (Figure 6), we observed that for each time point, number of tasks running for Hive/MR is smaller than number of tasks running for Hive/Tez, which might means that Hive/Tez is more task efficient than Hiva/MR (at a certain time Hive/Tez is always running more tasks), thus in general Hive/Tez performs better than Hive/MR.

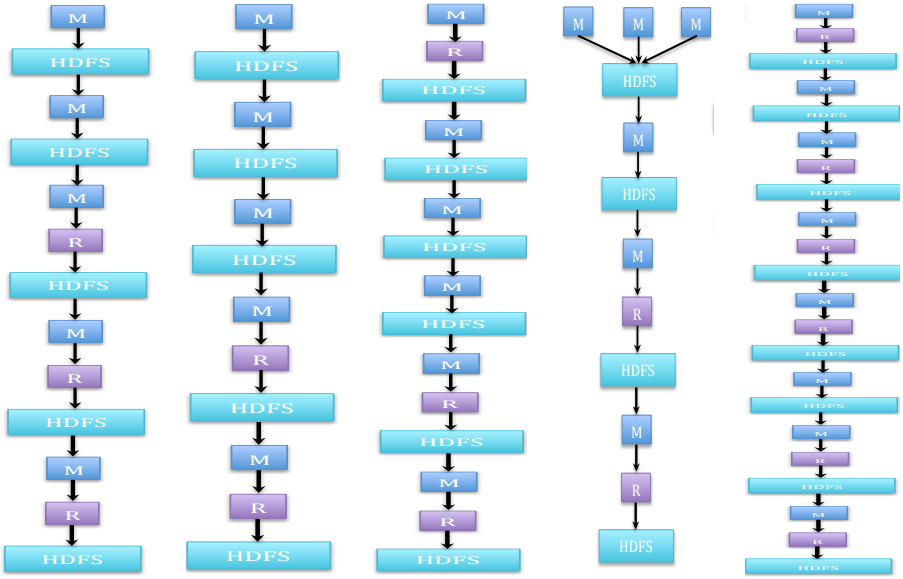
1.4 Part A-1-d

Visually plot the DAGs generated by Hive/MR and Hive/Tez. For Hive/MR you should analyze the stdout output generated by the query and correlate the amount of data read and written from/to HDFS among stages in order to identify their dependencies. What do you observe? How different are the DAGs? Do you think their structure can impact performance? If yes/no, why?
In this part of experiment, we observed that the structure of DAG of a certain job(query) is correlate with the performance of execution on this query. In general, the structure of DAG of certain job correlates with the ability of parallel and comlexity of this query. For instance, we can observe that for query12, query21, query50, the structure is relative simple. And we can also assume that these queries(12, 21, 50) requires less computation and with simple computation complexity. Moreover, we could see that for Hive/Tez, there are several nodes communication with the same node i.e. parallelization, which indicates why Hive/Tez is usually better than Hive/MR. Structure of DAG of query71 and 85 are more complex and thus in Hive/Tez, there are more parallel operations.

1.5 Part A-2

In this experiment, we can observe that killing the data node will always lead more running time. And the earlier we kill, the more extra time will cost.

Program	raw time	25% kill	75% kill
Hive/MR	239.203	251.187	241.208
Hive/Tez	207/317	234.405	224.248



(a) Q12 (b) Q21 (c) Q50 (d) Q71 (e) Q85

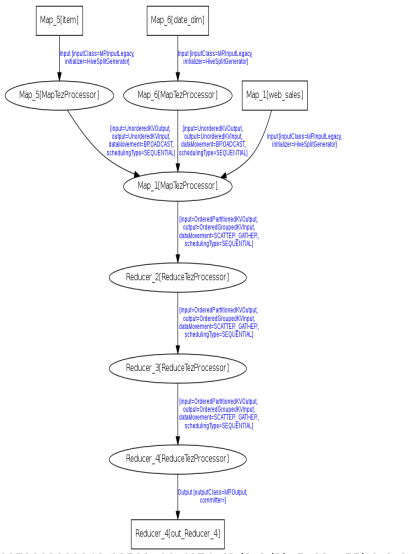
Figure 7: DAG of MR.

2 Part B

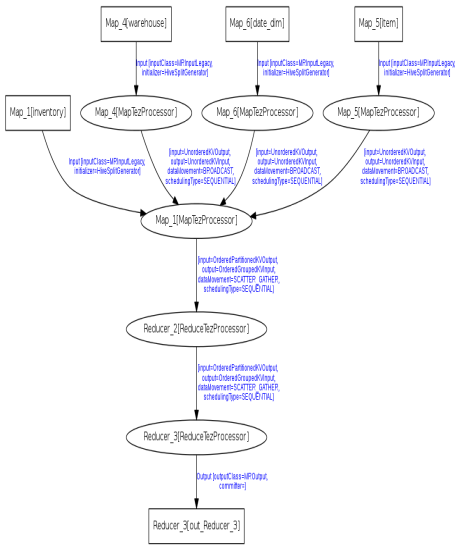
2.1 Problem Description

Here, you will write a simple MapReduce application that, given an input containing a single word per line, will group words that are anagrams of each other and sort the groups based on their size in descending order. Shown below is a sample of how the input will and output should look like.

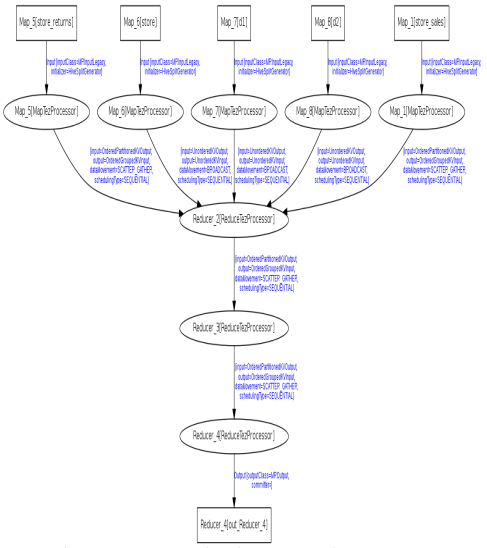
Input	**Intermediate Output**	**Final Output**
abroad	abroad aboard	early leary layer relay
early	early leary layer relay	natured unrated untread
natured	natured unrated untread	abroad aboard
unrated		
layer		
aboard		
untread		
leary		
relay		



(a) Query12 Tez DAG

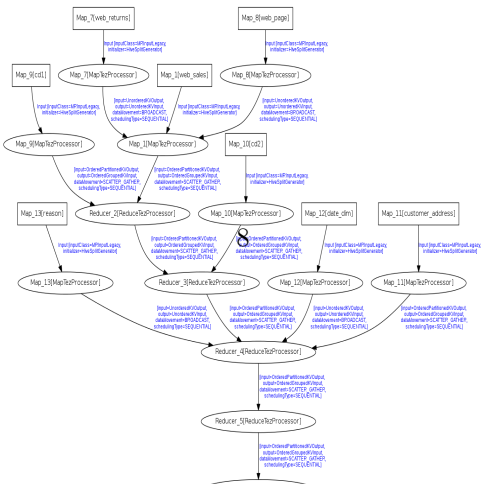


(b) Query21 Tez DAG



(c) Query50 Tez DAG

(d) Query71 Tez DAG



Your MR application (name it as AnagramSorter) should run on a multi-node cluster. First, you will load the input file (input.txt) into HDFS and run your application as:

hadoop jar ac.jar AnagramSorter /input /output

where input and output are the input and ouput directories in HDFS. You will submit your entire source code (all java classes you created) and a brief documentation (README.txt) for the classes you implement, please reference to "Deliverables" section for detail. Please use the exact names as specified; we will automate tests to verify correctness.

2.2 Solution

We solve the problem in two steps. The first step is to group words that have the same alphabets. This can be done by a Map/Reduce program as follows:

1. Map: Produce (key, value) pairs from the input text file, where the key is generated by reordering word alphabetically and value is simply the word itself.

2. Reduce: Group (key, value) pairs such that all the words with the same key are in the same list. The output (key, value) pair is then (size, list), where list is the grouped words and size is the number of words in list.

Note that the words that are anagrams of each other have the same key, thus they will be grouped together. The output of reduce function contains size information of the grouped anagrams, and will be used in the next round of Map/Reduce procedure. The output will be written into an intermediate HDFS file /tempoutput.

The second step is to sort the group based on the group size. We use another Map/Reduce procedure as follows:

1. Map2: Take /tempoutput as the input file, and produce (key, value) pairs from the input text file, where key is the size of each group and value is the group itself.

2. Reduce2: Pass the output (key, value) pair of the Map2 function to the HDFS output file, but with additional sort function that can sort pairs by key value.

Note that in Map2, the input (key, value) pair is not (size, group), but something different. The input key will be the offset of each line in the input file, while the input value is a string containing both the size and the group of words. Thus we need to extract the output key and output value from the input value. Then we pass the output to the Reduce2. In Reduce2, we use comparator that sort keys in descending order.

2.3 Implementation

First run the following setup commands:

1. Go to the user directory /home/ubuntu.
2. Run source run.sh.
3. Export hadoop classpath as:
   export HADOOP_CLASSPATH=$HADOOP_CLASSPATH$JAVA_HOME/lib/tools.jar

To compile the AnagramSorter.java, run the following two commands:

1. hadoop com.sun.tools.javac.Main AnagramSorter.java
2. jar cf ac.jar AnagramSorter*.class

Then upload the input file from local disk to HDFS file system:
hadoop -put ~/input.txt /input

Finally, run the .jar file:
hadoop jar ac.jar AnagramSorter /input /output

The source file is AnagramSorter.java and is placed in home/ubuntu/Part-B/.
To compile the source file, run the following command:
hadoop jar ac.jar AnagramSorter /input /output

3 Part C

3.1 Question1

Write a Scala/Python/Java Spark application that implements the PageRank algorithm without any custom partitioning (RDDs are not partitioned the same way) or RDD persistence. Your application should utilize the cluster resources to it's full capacity. Explain how did you ensure that the cluster resources are used efficiently. (Hint: Try looking at how the number of partitions of a RDD play a role in the application performance)

To utilize the cluster resources to it's full capacity, there are three factors should be taken into account. The first one is CPU utilization, we make full use of the 4 cores in each worker. The second one is memory, we make use of 15GB out of 18GB in each worker. Last but not least, we choose the most reasonable RDD partition numbers 20 by observing different running time under different RDD partition number in the Spark UI. It seems that by choosing RDD partition number 20 could highly avoid of unevenly RDD allocation in different nodes. When the RDD partition number is too small, some executor may idle, when the partition number is too large, there may unevenly allocate partitions to different workers.

3.2 Question2

Modify the Spark application developed in Question 1 to implement the PageRank algorithm with appropriate custom partitioning. Is there any benefit of custom partitioning? Explain. (Hint: Do not assume that all operations on a RDD preserve the partitioning)

Obviously, custom partitioning is beneficial as it could reduce the time cost in shuffling. Originally, links and ranks are repeatedly joined and each join requires a full shuffle over the network. In this case, we could pre-partition the links RDD so that links for URLS with the same hash code are on the same node. In other word, each RDD has an optional partitioner object, and any shuffle operation on a RDD with a partitioner will respect that partitioner, and any shuffle operation on two RDDS will take on the partitioner of one of them. To conclude, good custom partitioning could avoid unnecessary all-to-all communication, saving the cost in communication.

3.3 Question3

Extend the Spark application developed in Question 2 to leverage the flexibility to persist the appropriate RDD as in-memory objects. Is there any benefit of persisting RDDs as in-memory objects in the context of your application? Explain.

By caching data in the memory, it is definitely beneficial to the overall performance as the iterative algorithm will reuse the same data for computation in each iteration directly through memory instead of creating I/O to disk. More officially, caching or persistence are the optimization techniques for interactive and iterative Spark computations. It helps save intermediate results so we can reuse them in the following stages. Generally, these intermediate results as RDDs are kept in memory and could provide time efficiency.

3.4 Analyze of 1-3

With respect to Question 1-3, for your report you should:

- **Report the application completion time under the three different scenarios.**
- **Compute the amount of network/storage read/write bandwidth used during the application lifetime under the four different scenarios.**
- **Compute the number of tasks for every execution.**
- **Present / reason about your findings and answer the above questions. Apart from that you should compare the applications in terms of the above metrics and reason out the difference in performance, if any.**

Refer Figure 9.a to see the completion time under the three different scenarios. It is obvious that Time(P3) < Time(P2) < Time(P1) as in program 2 we add custom partitioning and program 3 adds cache based on program 2 which could accelerate the execution time.

Refer Figure 9.b and 9.c to see the amount of network/storage read/write bandwidth during the application lifetime. The disk read/write and network receive and transmission of Program 2 and Program 3 are similar and are much smaller than Program 1 that is because the all to all communications are reduced by adding custom RDD partitioning. In addition, Program 3 reads the least data from disk because it uses cache to store the intermediate data in the memory.

Refer Figure 9.d to see the number of tasks for every execution of Program 1, Program 2 and Program 3. Program 2 and Program 3 have only 15 stages and 432 tasks with custom partitioning base on 32, while Program 1 has 23 stages and 2288 tasks without custom partitioning.

3.5 Question 4

Analyze the performance of CS-744-Assignment1-PartC-Question1 by varying the number of RDD partitions from 2 to 100 to 300. Does increasing the number of RDD partitions always help? If no, could you find a value where it has a negative impact on performance and reason about the same.

Increasing the number of RDD partitions does not always help. Look into figure 10, we could find that the execution time decreases dramatically with the increment of RDD partitions at the beginning, then starts to increase after the number of RDD partitions become greater than 32. It is reasonable as we only have 16 cores of CPUS. If the number of RDD partitions is too small, some cores may idle. However, if the number of RDD partitions is too large, then the number of tasks will also be too large and increase the overhead of managing these tasks and thus increases the execution time.

3.6 Question 5

Visually plot the lineage graph of the CS-744-Assignment1-PartC-Question3 application. Is the same lineage graph observed for all the applications developed by you? If yes/no, why? The Spark UI does provide some useful visualizations.

Refer Figure 11, the lineage graphs for the three applications are similar, especially for Program 2 and Program 3. The difference between lineage graph of Program 1 and the lineage graphs of Program 2 and Program 3 are easy to reason: the number of stages of Program 1 is 23 while the number of stages of program 2 and 3 is only 15 causes the lineage graph of Program 1 is much longer. This is because Program 2 and Program 3 use custom partitioning to get rid of shuffling at each iteration. The lineage graph of Program 2 and Program 3 are very similar, but they also have little difference because Program 3 use cache which has less transformation.

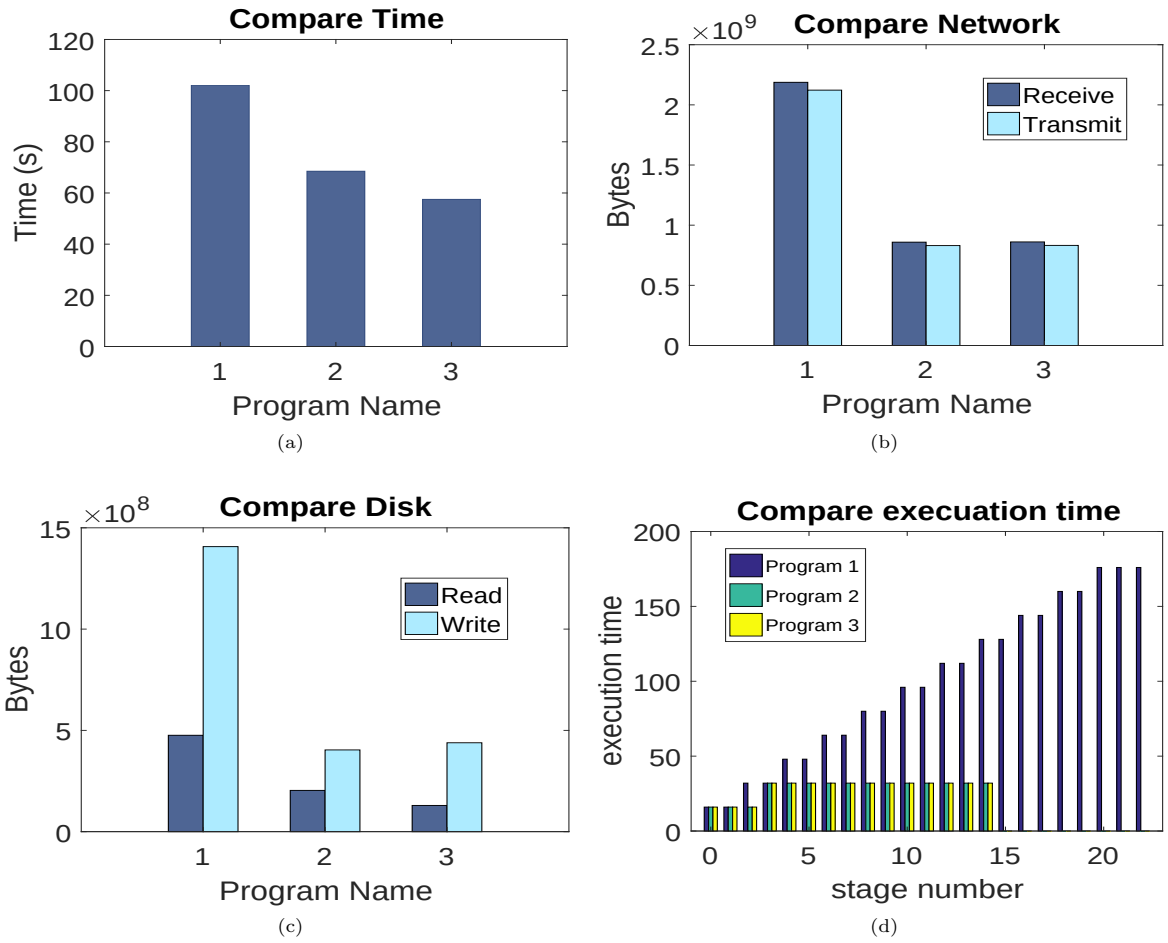


Figure 9: Comparison between performance of three programs in Part C.

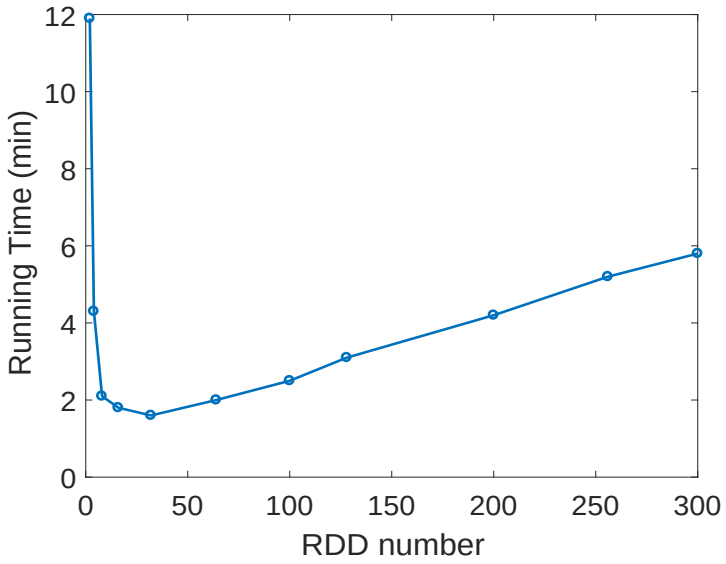


Figure 10: Running time comparison under different RDD partitions number.

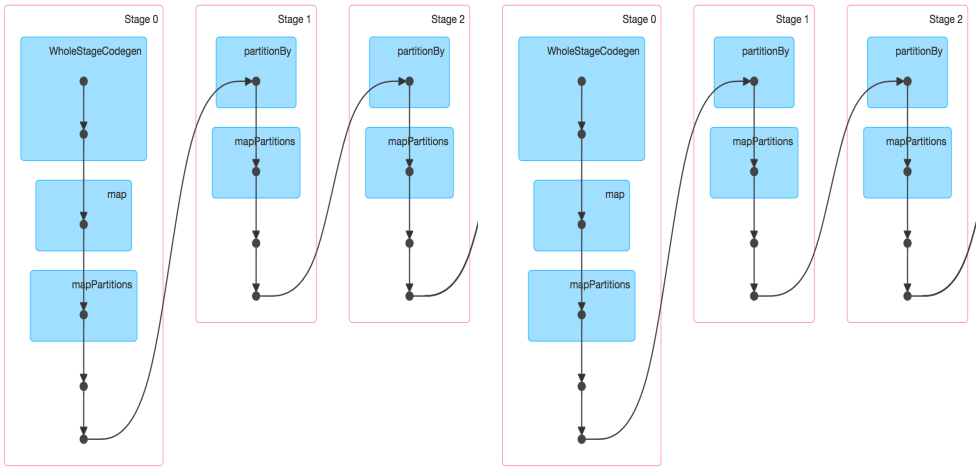
3.7 Question 6

Visually plot the Stage-level Spark Application DAG (with the appropriate dependencies) for all the applications developed by you till the second iteration of PageRank. The Spark UI does provide some useful visualizations. Is it the same for all the applications? Id yes/no, why? What impact does the DAG have on the performance of the application?

The DAG of second iteration is **not** the same for all the applications. The DAG of Program 2 and Program 3 are similar as they both use custom partitioning to get rid of shuffling for each join. Program 1 has one more stage for second iteration as it need to do the join of rank and links at the beginning of every iteration. Refer figure 12 to see the details. By carefully analyzing the DAGs, we could tell whether we have cached the right RDDs and whether we have used the right custom partitioning so that we could adjust the implementation of our Spark program correspondingly to improve the performance.

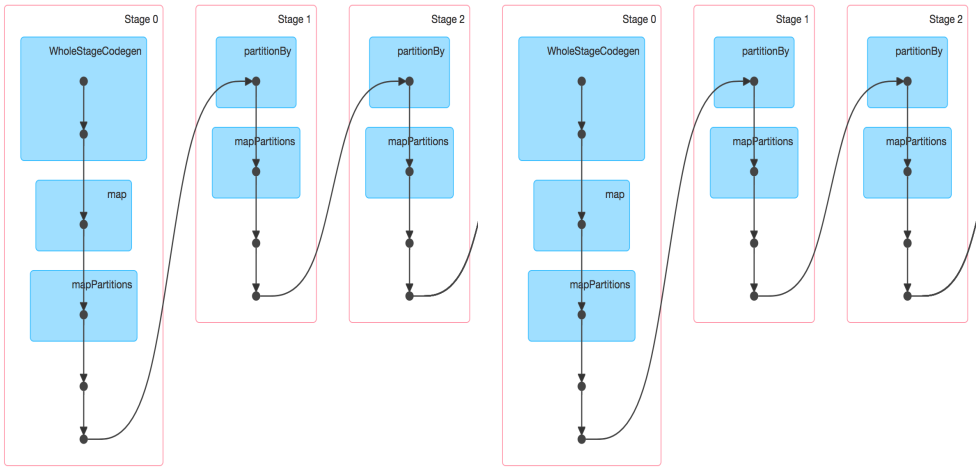
3.8 Question 7

Analyze the performance of CS-744-Assignment1-PartC-Question3 and CS-744-Assignment1-PartC-Question1 in the presence of failures. For each application, you should trigger two types of failures on a desired Worker VM when the application reaches 25% and 75% of its lifetime. The two failure scenarios you should evaluate are the following:



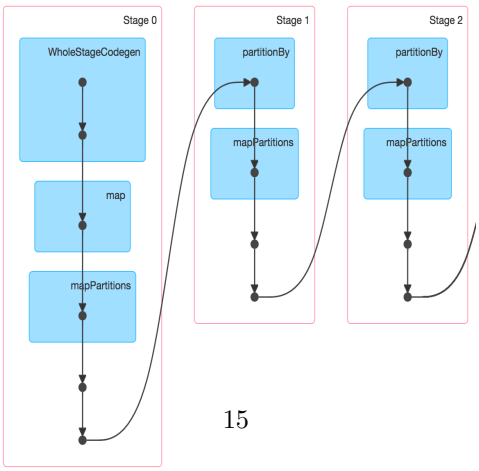
(a)

(b)



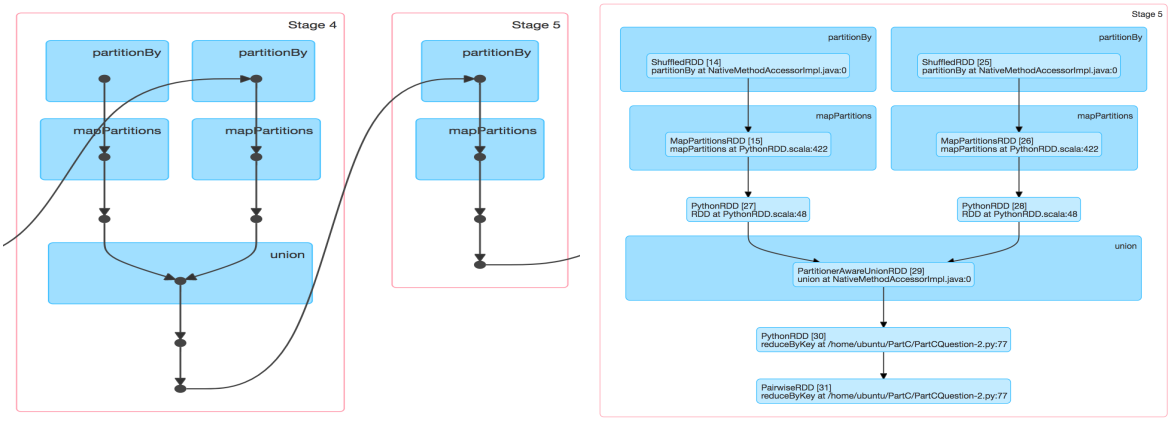
(c)

(d)

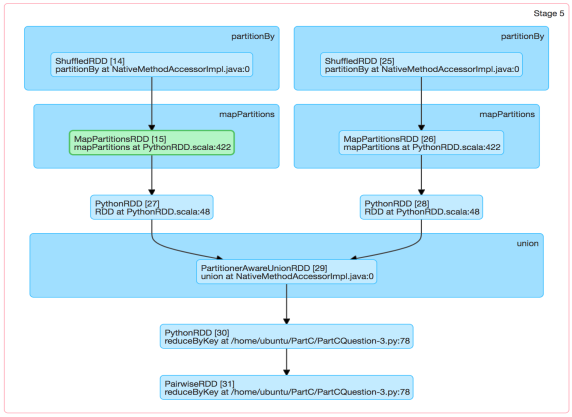


(e)

Figure 11: Linear graph of program 3.



(a) (b)



(c)

Figure 12: DAG of three programs in iteration 2.

1. **Clear the memory cache using:**
   sudo sh -c "sync; echo 3 ¿ /proc/sys/vm/drop_caches".

2. **Kill the Worker process.**

Based on Table below, the failure scenery of clear the memory cache has no import on the execution time of Program 1 and Program 2 while it could slightly increase the execution time of Program 3 which use cache. The failure of killing worker increase the execution time of all the three programs by around 50 to 60 percent.

Program	**original execution** time(s)	**clean cache (s) 15%**	**clean cache (s) 7**5%	**kill worker(s) 15**%	**kill** worker(s) **75%**
P1	102	105	104	164	163
P2	66.5	66	67	85	104
P3	57.5	64	68	99	88