

Spark Part C

Question 1. 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 its 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)

Solution 1: We had 5 workers in the cluster. To utilize the cluster to its full capacity, RDD should have at least 5 partitions. Every partition is executed by a task. We got least running time with number of Partitions = 50.

Number of tasks created = 751

Following are the IO and Network stats for 4 runs of Spark App(Spark driver and 5 workers):

Time	IO_Read (in KB)	IO_Write (in KB)	Net_Send (in B)	Net_Recv (in KB)
51	384296	89216	89216	184116168
66	371128	57036	57264	211206387
66	371860	158356	158760	209757881
55	374136	152496	152496	210491896

Average:

Completion Time=59.5s

IO_Read = 362.92 MB

IO_Write = 133.89 MB

Net_send = 134.68 KB

Net_recv = 603.78 MB

The above stats are for entire cluster (1 Driver + 5 Workers).

Question 2. 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)

Solution2:

We have used RangePartitioning to replace Spark's Default partitioning scheme. We also considered HashPartitioning but found that Range Partitioning offered better performance.

Number of partitions = 50

Number of tasks created = 751

Time	IO_Read	IO_Write	Net_Send	Net_Recv
59	373264	206080	206080	861086615
59	371412	99320	181536	889569678

56	372580	160452	251864	874254597
52	373212	70656	70656	882879520

Average:

Completion Time=56.5s

IO_Read = 363.88 MB

IO_Write = 130.98 MB

Net_send = 173.37 KB

Net_rcv = 836.32 MB

Why partitioning increased the Job turn-around time?

The Spark App with default partitioning took 59.5 sec (of total 52 sec from Run 4) to calculate all the Stages.

The Spark App with Range partitioning had 1 extra job of creating Partitioner (Stage 0) which took 8 sec. But, the core computation of Page Rank stages was completed in 39sec!

The RangePartitioning improved the completion time by 3sec.

Question 3. 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.

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.

Number of partitions = 50

Number of tasks created = 751

Time	IO_Read	IO_Write	Net_Send	Net_Recv
43	372908	14528	15136	489547063
39	369840	14144	14604	534893531
39	371912	120292	120292	528281961
40	371008	29844	30124	554625993

Average:

Completion Time=40.25s

IO_Read = 362.71 MB

IO_Write = 43.65 MB

Net_send = 43.98 KB

Net_rcv = 502.43 MB

Few important RDD's which were being re-used are now persisted in Spark with persistence level of MEMORY_ONLY via api 'cache()'.
The persisted RDDs are being re-used and hence, we got speed up in the job execution.

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.

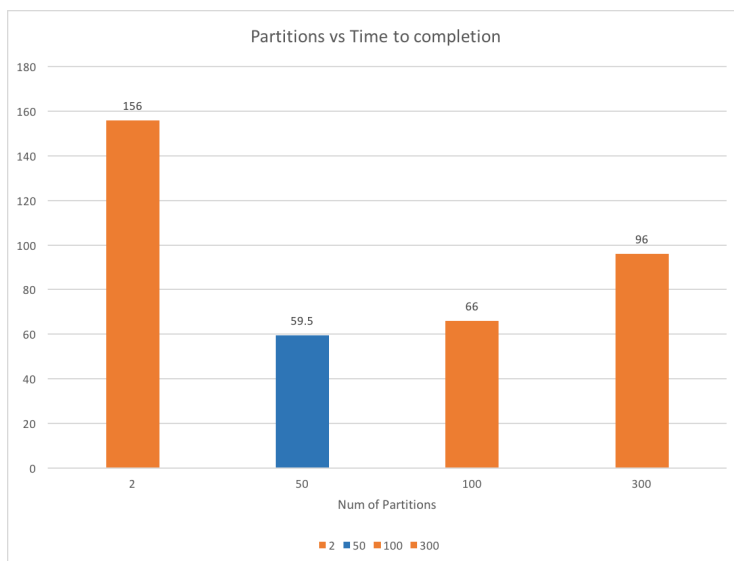
Solution4:

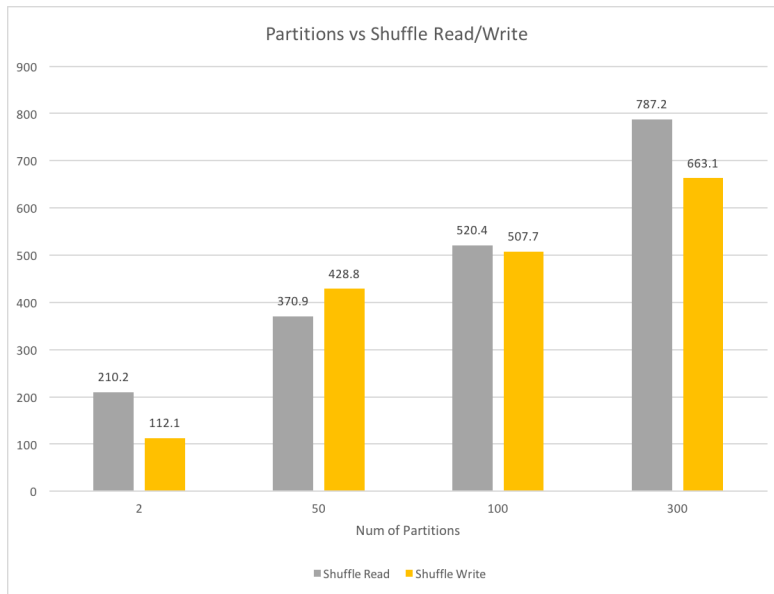
We ran PartC-1 Spark App (with default partitioning) for 4 times with 2,50,100 and 300 partitions.

Partitions	Time (in sec)	Shuffle Read (in MB)	Shuffle Write (in MB)
2	156	210.2	112.1
50	59.5	370.9	428.8
100	66	520.4	507.7
300	96	787.2	663.1

For partitions=2, the cluster was being under-utilized as only 2 tasks were being spawned. Also, since only 2 tasks shared the load, the task size was high. As a result, the Spark Worker Garbage collection time was high.

For partitions=100 and 300, the overhead of Shuffle read and write beats the benefits of data parallelism. We got the best performance for 50 partitions.

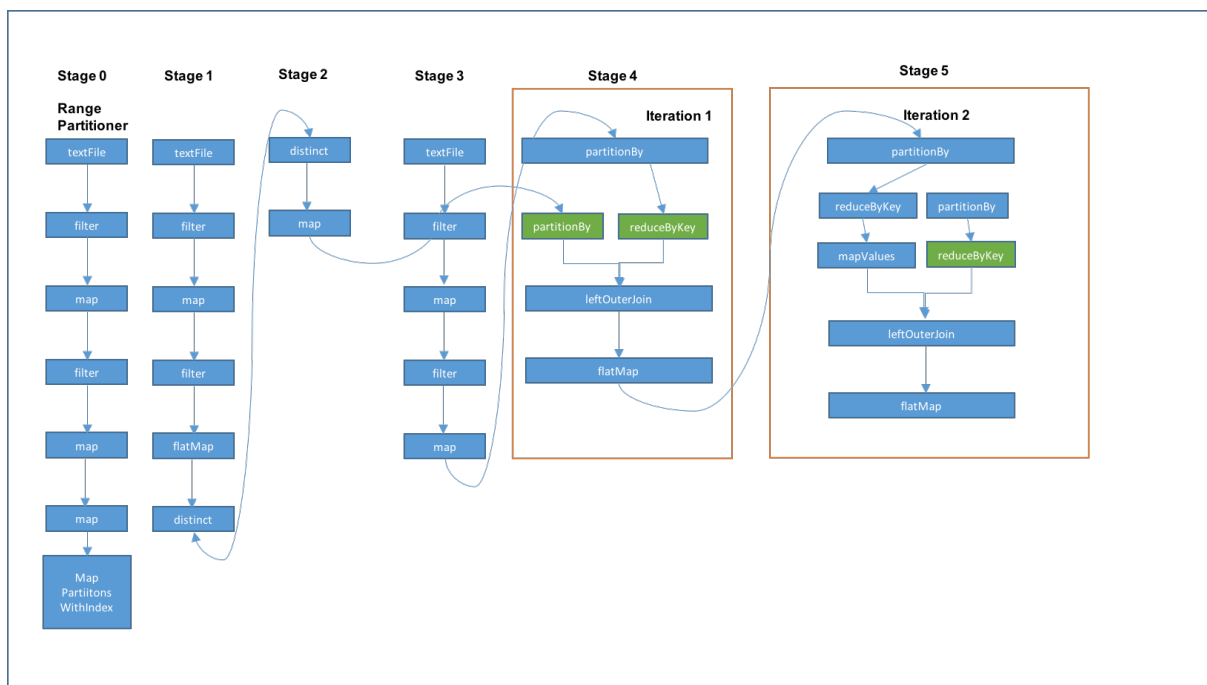




Thus, increasing number of partitions to a large number doesn't benefit the turn-around time for the job execution.

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.

Solution 5:



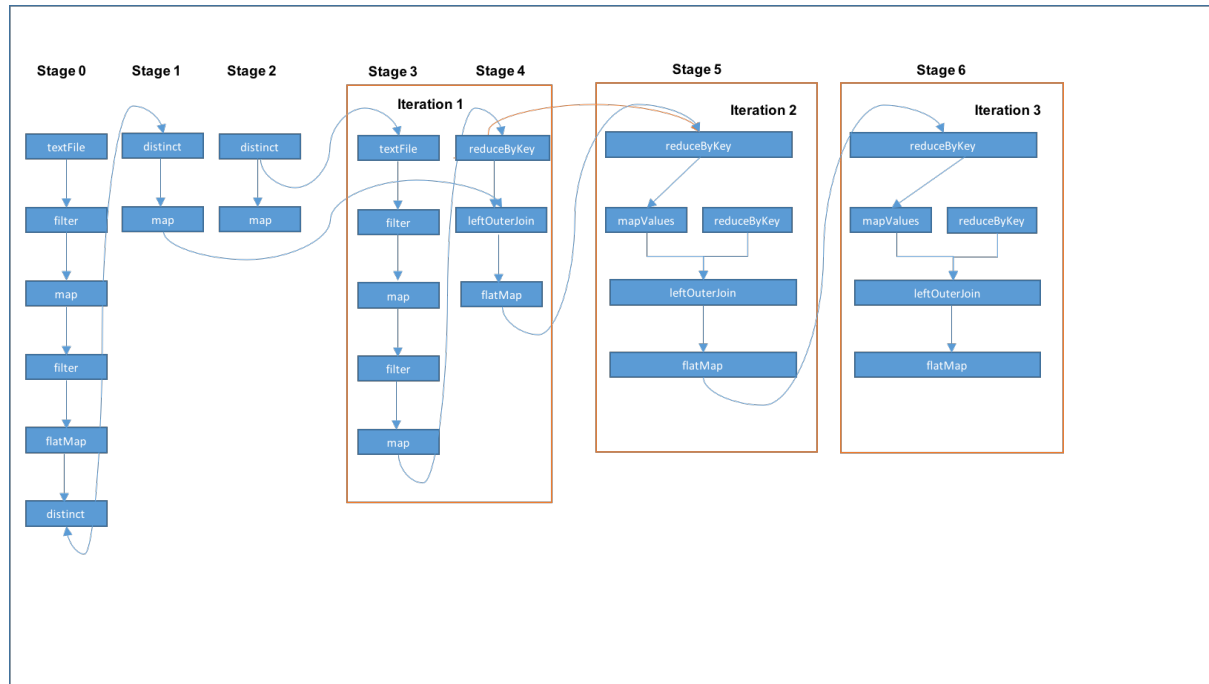
Remaining iterations are not shown in the graph. The Stage 0 represents Range Partitioner computation. Stages 1-3 represent data cleaning (Eliminate irrelevant lines), rank initialization (= 1), etc. Stages 4 and 5 represent iterations of Page Rank.

The lineage graphs for PartC2 and PartC3 are almost same as both employ same partitioning technique. PartC3 persists some of the crucial RDDs and hence, the green highlights represent cached stages.

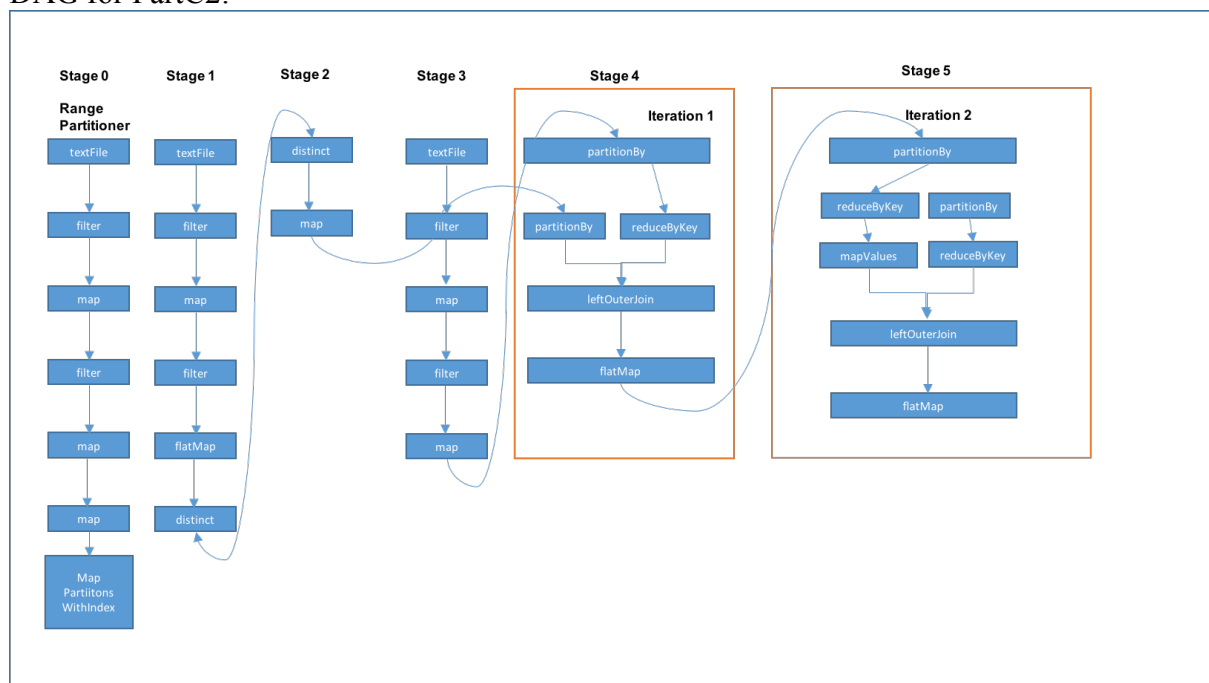
PartC1 uses default partitioning and hence, its Lineage graph is slightly different.

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?

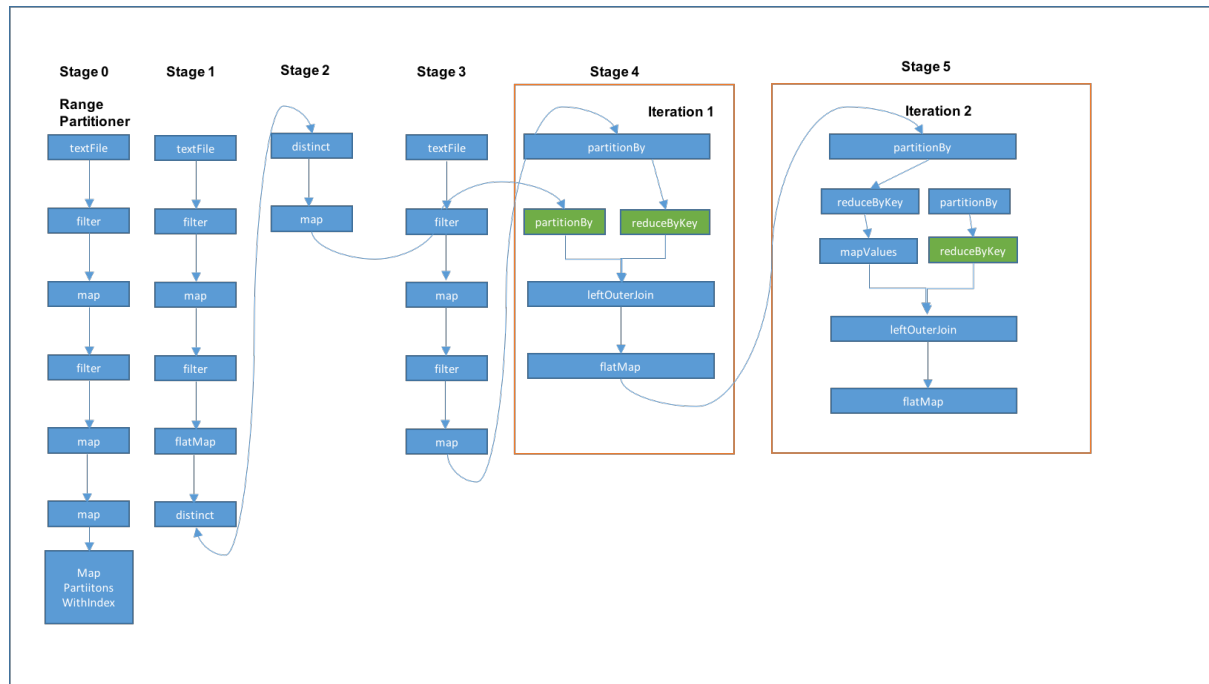
DAG for PartC1:



DAG for PartC2:



DAG for PartC3:



DAGs for Parts C2 and C3 are almost same. PartC3 uses RDD persistence (in Memory) and hence the green boxes. PartC1 uses different partitioning technique. Hence, Part C1 doesn't have Partitioner stage and PartitionBy stages in each iteration.

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.

Solution 7:

All the metrics are in seconds.

	Part C1	Recovery Time	Part C3	Recovery Time
Without Failure	59.5		40.25	
25%	78	18.5	54	13.75
75%	90	30.5	55	14.75

The recovery time for PartC3 is smaller than that of PartC1.

Also, recovery time for failure at 75% is much larger than failure at 25% for PartC1.

But, for PartC3, the recovery time is almost same for 25% and 75%. This is probably due to persisted RDDs cutting down recovery time for PartC3.