PART A

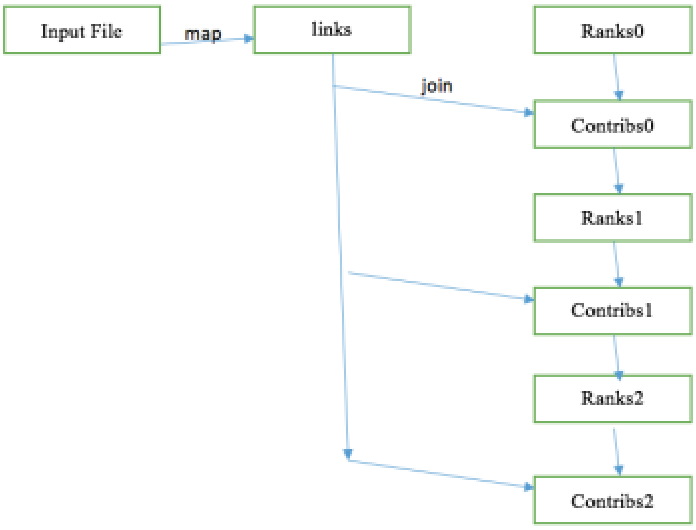
Scenario List

1. No custom partitioning or RDD persistence
2. Custom partitioning
3. Custom partitioning and RDD persistence
4. Increase Custom partitioning RDD partitions from 8 to 100
5. Increase Custom partitioning RDD partitions from 8 to 300
6. Scenario 1 failure – 25%
7. Scenario 1 failure – 75%
8. Scenario 3 failure – 25%
9. Scenario 3 failure – 75%

Various Parameters noted when run for the above scenarios

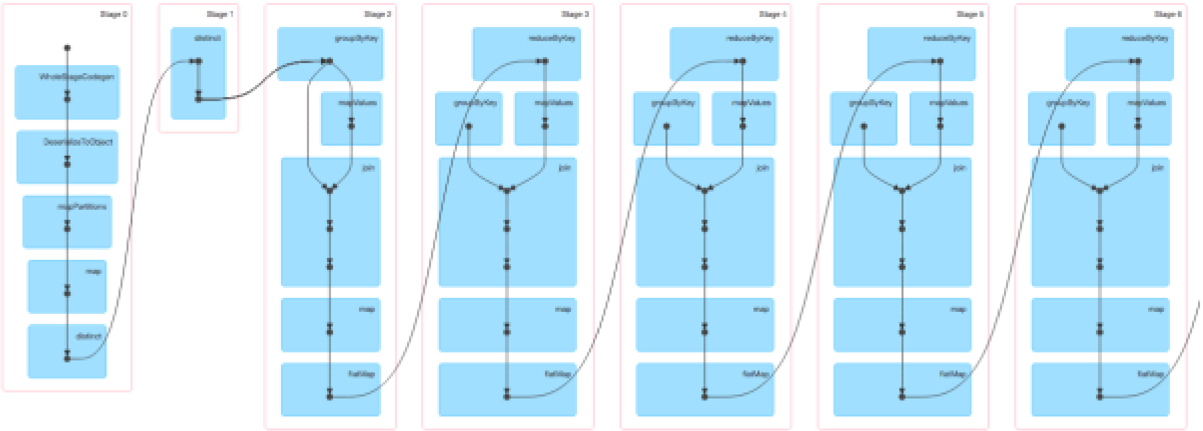
Scenario	Application Completion Time	Network Read-Network Write-Storage Read-Storage Write	Number of tasks
1	46 s	930.6MB – 442.7MB – 160KB – 2.1GB	260
2	59 s	1082.5MB – 858.1MB – 157KB – 2.1GB	220
3	53 s	714.3MB - 858.5MB – 1180.7MB – 10.1GB	220
4	54 s	1372.3MB – 1174.9MB- 172.9KB – 2.1GB	2048
5	1.3 m	1540.6MB - 1385.6MB – 240.5KB – 2.1GB	6060
6	1.2 m	858.5MB- 458.8MB – 177.1KB –2.1GB	264
7	1.2 m	1052.9MB – 523.4MB – 179.6KB– 2.1GB	303
8	1.1 m	664.6MB- 871.4 MB – 720.3MB - 10.1GB	224
9	1.3 m	809.2MB - 1053.9MB – 1187.1MB - 10.1GB	271

Lineage Graph of Scenario 3

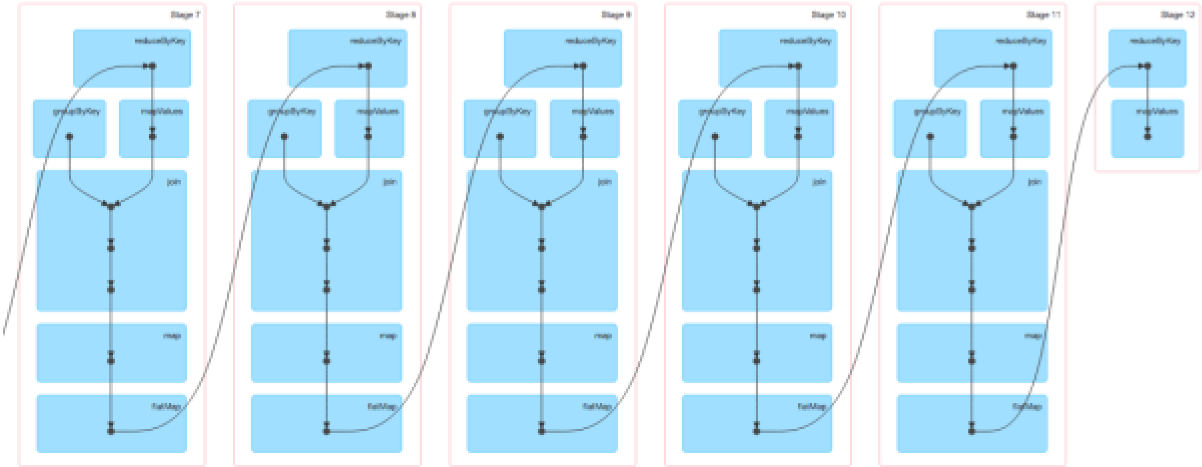


Stage-level Spark Application DAG

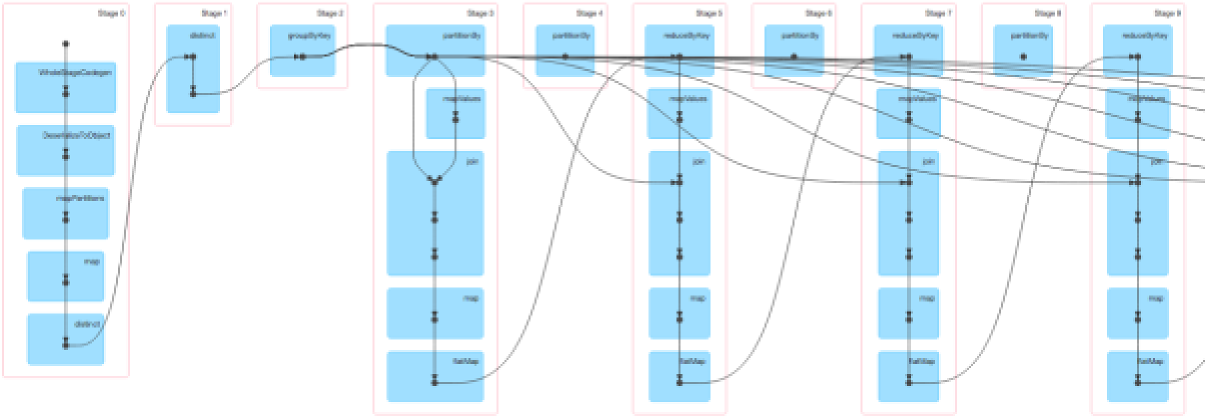
Q1

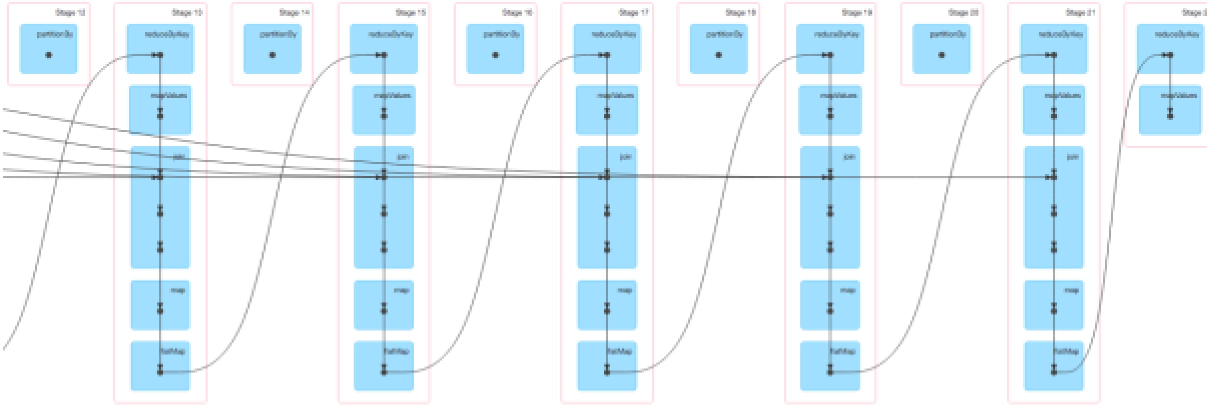


(contd below)

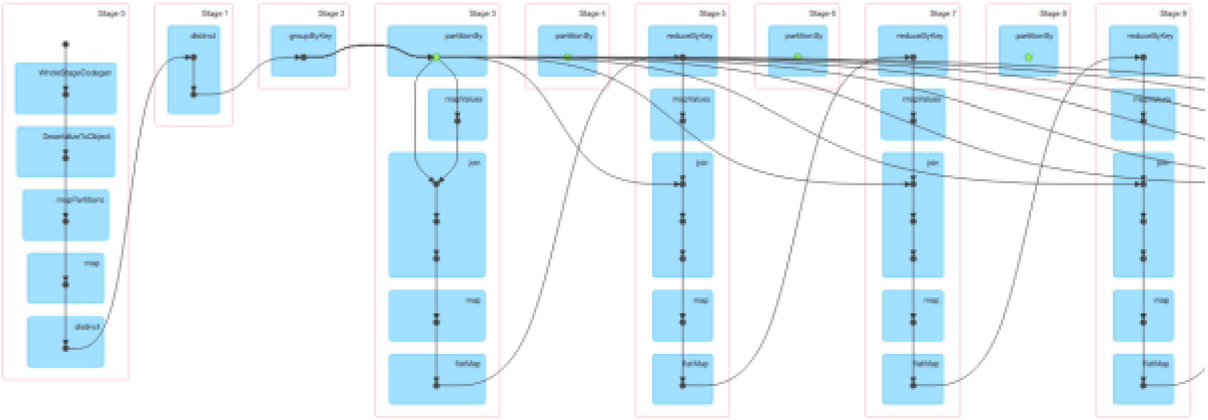


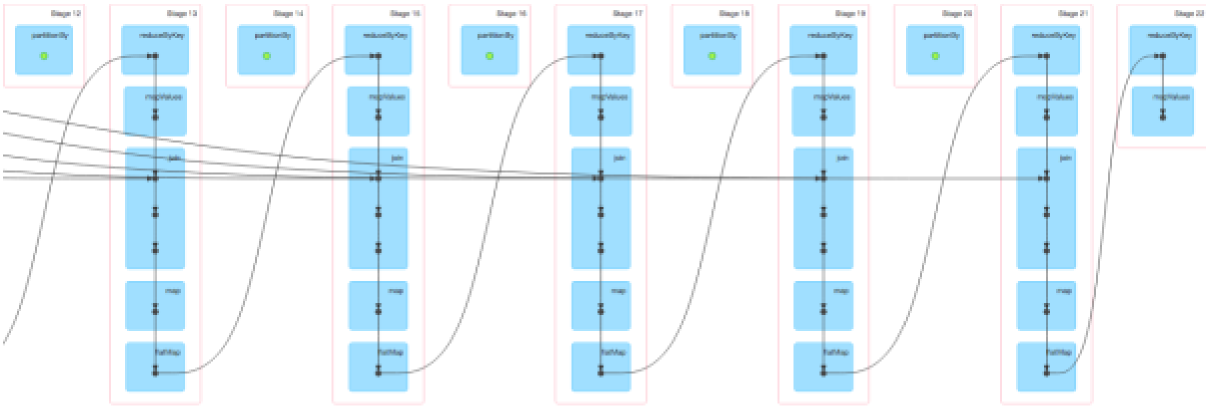
Q2


Stage 0
WholeStageCodegen
DeserializeToObject
mapPartitions
map
distinct
Stage 1
distinct
Stage 2
groupByKey
Stage 3
partitionBy
mapValues
join
map
flatMap
Stage 4
partitionBy
Stage 5
reduceByKey
mapValues
join
map
flatMap
Stage 6
partitionBy
Stage 7
reduceByKey
mapValues
join
map
flatMap
Stage 8
partitionBy
Stage 9
reduceByKey
mapValues
join
map
flatMap


Stage 12
partitionBy
Stage 13
reduceByKey
mapValues
join
map
flatMap
Stage 14
partitionBy
Stage 15
reduceByKey
mapValues
join
map
flatMap
Stage 16
partitionBy
Stage 17
reduceByKey
mapValues
join
map
flatMap
Stage 18
partitionBy
Stage 19
reduceByKey
mapValues
join
map
flatMap
Stage 20
partitionBy
Stage 21
reduceByKey
mapValues
join
map
flatMap
Stage 2
reduceByKey
mapValues

Q3


Stage 0
WholeStageCodegen
DeserializeToObject
mapPartitions
map
distinct
Stage 1
distinct
Stage 2
groupByKey
Stage 3
partitionBy
mapValues
join
map
flatMap
Stage 4
partitionBy
Stage 5
reduceByKey
mapValues
join
map
flatMap
Stage 6
partitionBy
Stage 7
reduceByKey
mapValues
join
map
flatMap
Stage 8
partitionBy
Stage 9
reduceByKey
mapValues
join
map
flatMap


Stage 12
partitionBy
Stage 13
reduceByKey
mapValues
join
map
flatMap
Stage 14
partitionBy
Stage 15
reduceByKey
mapValues
join
map
flatMap
Stage 16
partitionBy
Stage 17
reduceByKey
mapValues
join
map
flatMap
Stage 18
partitionBy
Stage 19
reduceByKey
mapValues
join
map
flatMap
Stage 20
partitionBy
Stage 21
reduceByKey
mapValues
join
map
flatMap
Stage 22
reduceByKey
mapValues

Questions (Observations and Findings)

Question 1

Scala Application in Submission Folder (PartAQuestion1.scala)
The parameters (time taken etc) in Scenario 1 in table above
If too few partitions**,** we will not utilize all of the cores available in the cluster.
If too many partitions, there will be excessive overhead in managing many small tasks.
Generally number of partitions should be set to 3 - 4 times the number of CPUs in our cluster. In this case, we have 20 cores(4*5)
So we tested Q1 with No of partitions = 20 (number of cores) and =80 (4 times number of cores), there was significant improvement of 10 seconds in the latter case (from 50 sec to 40 sec). If we increase beyond 80, performance will keep deteriorating.

Question 2

Scala Application in Submission Folder (PartAQuestion2.scala)
The parameters (time taken etc) in Scenario 2 in table above
Using custom partitioning, results were not as good as with default partitioning
For custom partitioning, we simply returned partition number as modulo by number of partitions. partitionBy() doesn't preserve order for custom partitioning ,which may be the reason for poorer performance

Question 3

Scala Application in Submission Folder (PartAQuestion3.scala)
The parameters (time taken etc) in Scenario 3 in table above
As seen from the table above, applying persist on Q2 code in fact improved the performance. Cache is useful when the lineage of the RDD branches out, which is true in our case so caching helps.

Question 4

The parameters (time taken etc) when increasing number of partitions from 8 to 100 and 300 is in Scenario 4,5 in table above.
As discussed in Q1 above and seen in Scenario results 4,5 , usually number of partitions should be set to 3 - 4 times the number of CPUs in our cluster, otherwise performance deteriorates.

Question 5

Lineage graph for Q3 application shown above.
It will be same for all the applications.

Question 6

Stage-level Spark Application DAG for the 3 applications shown above.
For Q1, DAG graph will be different from Q2 because of change in partitioning function

Question 7

The parameters (time taken etc) in Scenario 6,7,8,9 in table above when failing at different times the Q1 and Q3 code.
Q3 is more resilient when failing early (25%- map phase) but Q1 is more resilient when failing late(75%-somewhere around the phase when map has finished).
It is because Q3 is using caching so worker failing leads to loss in persisted data which must be computed again.