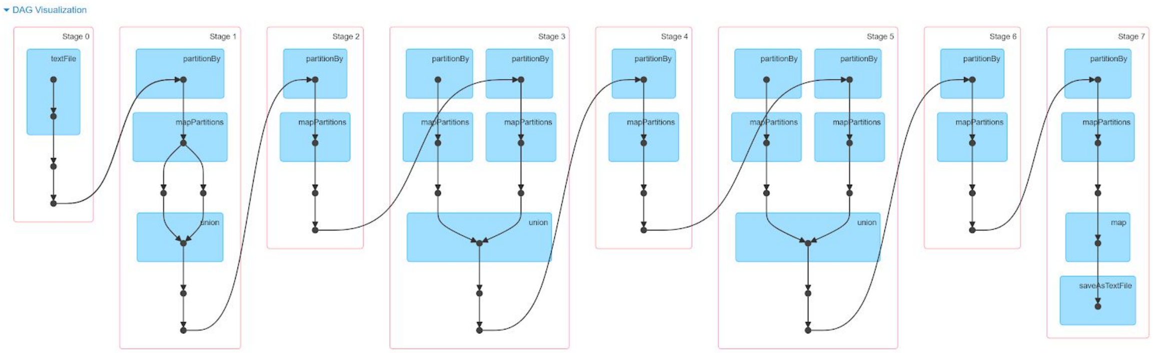
Running time and set up summary table

	Completion Time	Setup	Task Count
task1	15min	No cache default partition(~390)	● Task count : 2254
task2	55min	No cache, partitionBy(5)	● Task count : 1148
task2	20min	No cache, partitionBy(100)	● Task count : 1616
task2	9min	No cache, partitionBy(900)	● Task count : 6518
task3	15min	Cache one rdd	● Task count : 2254
task3	31min	Cache two rdd	● Task count : 2254
task4	30min	No cache, default partitions, kill workers on 25%, 75%	● Task count : 2254

- *All experiments above are based on finishing 3 iterations for test time convenience. We observed that it takes 9 min to finish the first iteration and 6 min for remaining 2 iterations, so if we run in 10 iterations, the estimate time should be 9 + 3 * 9 = 40 (min).*



DAG diagram for task1

Analysis

Task2

Observation:

We can see from the data that as the partition number grows the task count as well as the running efficiency increases. So it means the partition number for default is much smaller than optimal one. When the input size is small, the partitions are also small. We guess the partition number may depend on the size of files, so in the later process (map, reduce, join), the default setting is to preserve original partitions if set.

We note that each time the data is processed in a 128MB block.It is optimal if the data is located on the same machine but not guaranteed. Thus, the time for each iteration fluctuates.

Task3

We explicitly store the rdd for <source: targeList>, but the running time does not change(compared to task1). We did some research finding that Spark automatically cache the rdd when encountering reduceByKey() action. So it could explain that during the first iteration our reduceByKey cache the similar RDD as we do explicitly caching.

When we cache the result <link, rank>pairs together with <source: target> pairs for each iteration, the running time is slower, and writing increases significantly, so there might be eviction if we keep too many things in memory. However, persist() in disk takes even more time so perhaps we should not use this unless machine failure is common.

Task4 and Fault-tolerance analysis:

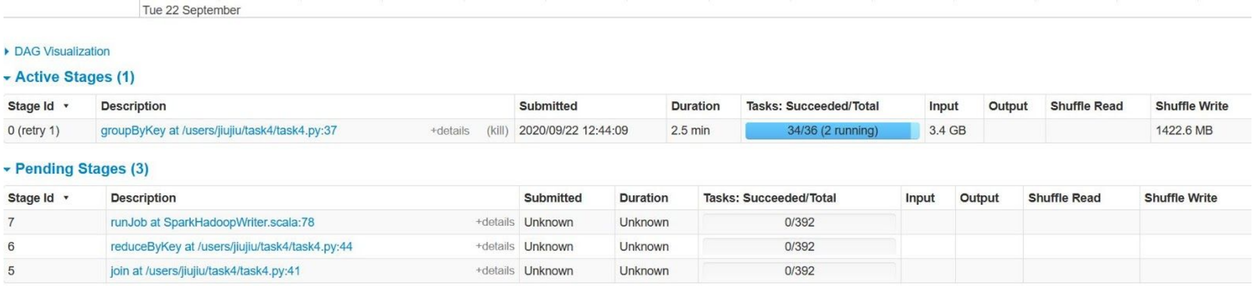
Summary

After killing the process at 25%, the task is reverted, and redo afterwards. In this case, only the failed part will be reloaded so recovery only takes about a third of original time to redo the previous tasks. In this case, the cache in disk might be useful.

Joining after failure takes more time than the usual one, interesting

Possible reason: # of worker is smaller but remote writing might be smaller

Worker killed in different phases and retrying...



Tue 22 September

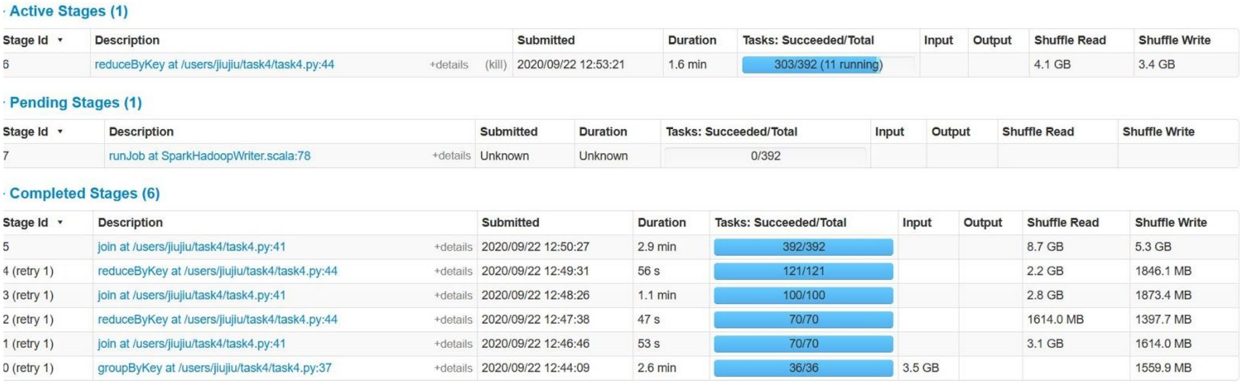
▸ DAG Visualization

▾ Active Stages (1)

Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
0 (retry 1)	groupByKey at /users/jiujiu/task4/task4.py:37 +details (kill)	2020/09/22 12:44:09	2.5 min	34/36 (2 running)	3.4 GB			1422.6 MB

▾ Pending Stages (3)

Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
7	runJob at SparkHadoopWriter.scala:78 +details	Unknown	Unknown	0/392				
6	reduceByKey at /users/jiujiu/task4/task4.py:44 +details	Unknown	Unknown	0/392				
5	join at /users/jiujiu/task4/task4.py:41 +details	Unknown	Unknown	0/392				



Active Stages (1)

Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
6	reduceByKey at /users/jiujiu/task4/task4.py:44 +details (kill)	2020/09/22 12:53:21	1.6 min	303/392 (11 running)			4.1 GB	3.4 GB

Pending Stages (1)

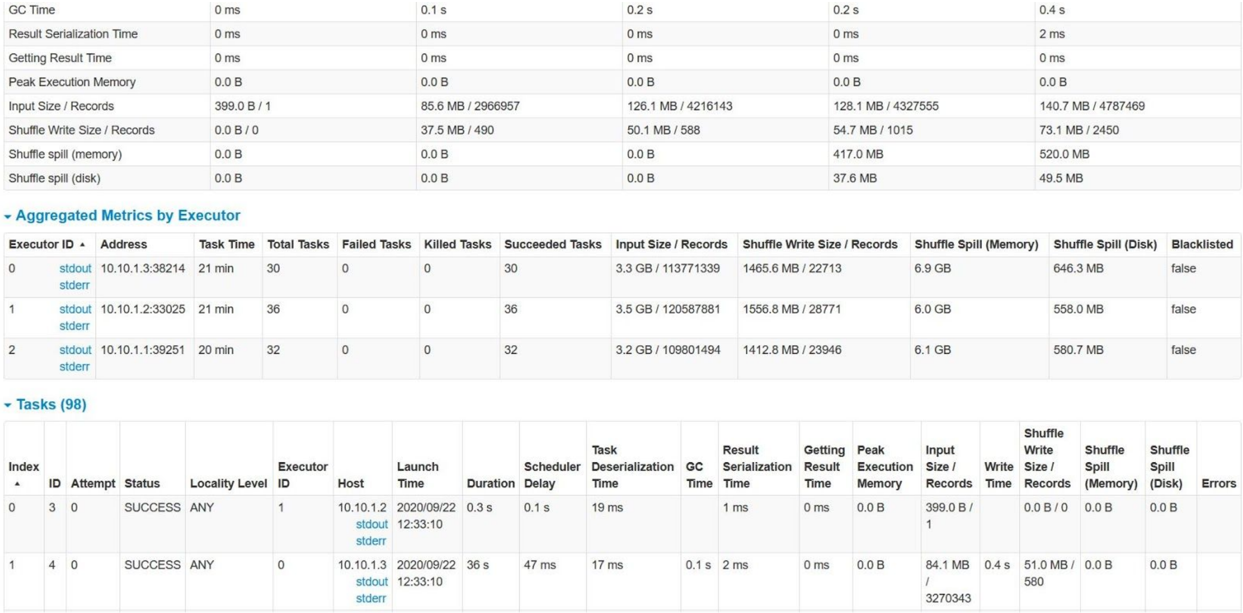
Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
7	runJob at SparkHadoopWriter.scala:78 +details	Unknown	Unknown	0/392				

Completed Stages (6)

Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
5	join at /users/jiujiu/task4/task4.py:41 +details	2020/09/22 12:50:27	2.9 min	392/392			8.7 GB	5.3 GB
4 (retry 1)	reduceByKey at /users/jiujiu/task4/task4.py:44 +details	2020/09/22 12:49:31	56 s	121/121			2.2 GB	1846.1 MB
3 (retry 1)	join at /users/jiujiu/task4/task4.py:41 +details	2020/09/22 12:48:26	1.1 min	100/100			2.8 GB	1873.4 MB
2 (retry 1)	reduceByKey at /users/jiujiu/task4/task4.py:44 +details	2020/09/22 12:47:38	47 s	70/70			1614.0 MB	1397.7 MB
1 (retry 1)	join at /users/jiujiu/task4/task4.py:41 +details	2020/09/22 12:46:46	53 s	70/70			3.1 GB	1614.0 MB
0 (retry 1)	groupByKey at /users/jiujiu/task4/task4.py:37 +details	2020/09/22 12:44:09	2.6 min	36/36	3.5 GB			1559.9 MB

When restarting, only the lost parts are recovered, so there is no need to roll back all process and process time is reduced.

Metrics



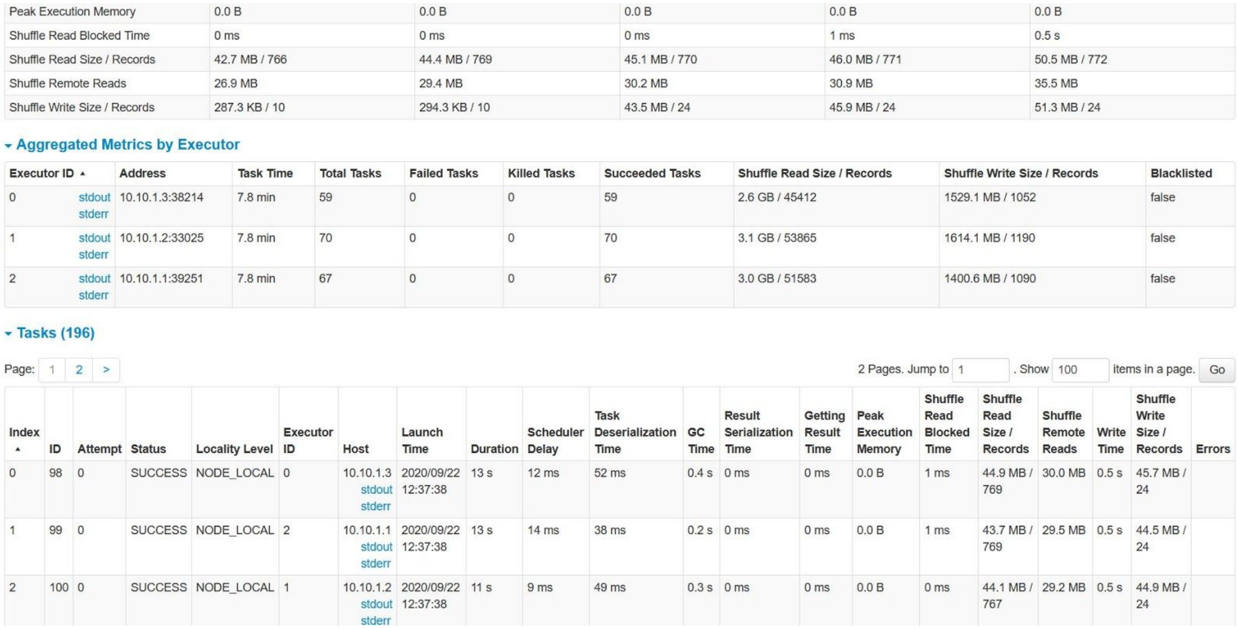
GC Time	0 ms	0.1 s	0.2 s	0.2 s	0.4 s
Result Serialization Time	0 ms	0 ms	0 ms	0 ms	2 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Input Size / Records	399.0 B / 1	85.6 MB / 2966957	126.1 MB / 4216143	128.1 MB / 4327555	140.7 MB / 4787469
Shuffle Write Size / Records	0.0 B / 0	37.5 MB / 490	50.1 MB / 588	54.7 MB / 1015	73.1 MB / 2450
Shuffle spill (memory)	0.0 B	0.0 B	0.0 B	417.0 MB	520.0 MB
Shuffle spill (disk)	0.0 B	0.0 B	0.0 B	37.6 MB	49.5 MB

▾ Aggregated Metrics by Executor

Executor ID ▴	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Input Size / Records	Shuffle Write Size / Records	Shuffle Spill (Memory)	Shuffle Spill (Disk)	Blacklisted
0 stdout stderr	10.10.1.3:38214	21 min	30	0	0	30	3.3 GB / 113771339	1465.6 MB / 22713	6.9 GB	646.3 MB	false
1 stdout stderr	10.10.1.2:33025	21 min	36	0	0	36	3.5 GB / 120587881	1556.8 MB / 28771	6.0 GB	558.0 MB	false
2 stdout stderr	10.10.1.1:39251	20 min	32	0	0	32	3.2 GB / 109801494	1412.8 MB / 23946	6.1 GB	580.7 MB	false

▾ Tasks (98)

Index ▴	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	Scheduler Delay	Task Deserialization Time	GC Time	Result Serialization Time	Getting Result Time	Peak Execution Memory	Input Size / Records	Write Time	Shuffle Write Size / Records	Shuffle Spill (Memory)	Shuffle Spill (Disk)	Errors
0	3	0	SUCCESS	ANY	1	10.10.1.2 stdout stderr	2020/09/22 12:33:10	0.3 s	0.1 s	19 ms		1 ms	0 ms	0.0 B	399.0 B / 1		0.0 B / 0	0.0 B	0.0 B	
1	4	0	SUCCESS	ANY	0	10.10.1.3 stdout stderr	2020/09/22 12:33:10	36 s	47 ms	17 ms	0.1 s	2 ms	0 ms	0.0 B	84.1 MB / 3270343	0.4 s	51.0 MB / 580	0.0 B	0.0 B	



Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Shuffle Read Blocked Time	0 ms	0 ms	0 ms	1 ms	0.5 s
Shuffle Read Size / Records	42.7 MB / 766	44.4 MB / 769	45.1 MB / 770	46.0 MB / 771	50.5 MB / 772
Shuffle Remote Reads	26.9 MB	29.4 MB	30.2 MB	30.9 MB	35.5 MB
Shuffle Write Size / Records	287.3 KB / 10	294.3 KB / 10	43.5 MB / 24	45.9 MB / 24	51.3 MB / 24

▾ Aggregated Metrics by Executor

Executor ID ▴	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Shuffle Read Size / Records	Shuffle Write Size / Records	Blacklisted
0 stdout stderr	10.10.1.3:38214	7.8 min	59	0	0	59	2.6 GB / 45412	1529.1 MB / 1052	false
1 stdout stderr	10.10.1.2:33025	7.8 min	70	0	0	70	3.1 GB / 53865	1614.1 MB / 1190	false
2 stdout stderr	10.10.1.1:39251	7.8 min	67	0	0	67	3.0 GB / 51583	1400.6 MB / 1090	false

▾ Tasks (196)

Index ▴	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	Scheduler Delay	Task Deserialization Time	GC Time	Result Serialization Time	Getting Result Time	Peak Execution Memory	Shuffle Read Blocked Time	Shuffle Read Size / Records	Shuffle Remote Reads	Write Time	Shuffle Write Size / Records	Errors
0	98	0	SUCCESS	NODE_LOCAL	0	10.10.1.3 stdout stderr	2020/09/22 12:37:38	13 s	12 ms	52 ms	0.4 s	0 ms	0 ms	0.0 B	1 ms	44.9 MB / 769	30.0 MB	0.5 s	45.7 MB / 24	
1	99	0	SUCCESS	NODE_LOCAL	2	10.10.1.1 stdout stderr	2020/09/22 12:37:38	13 s	14 ms	38 ms	0.2 s	0 ms	0 ms	0.0 B	1 ms	43.7 MB / 769	29.5 MB	0.5 s	44.5 MB / 24	
2	100	0	SUCCESS	NODE_LOCAL	1	10.10.1.2 stdout stderr	2020/09/22 12:37:38	11 s	9 ms	49 ms	0.3 s	0 ms	0 ms	0.0 B	0 ms	44.1 MB / 767	29.2 MB	0.5 s	44.9 MB / 24	

Contributions:

Jiujiu Pan: Setup the HDFS and Spark and part 2, implements the initial pagerank algorithm with Zhikang, research the effects with groupbykey

Ruoyu He: Debug and finish task 3 and 4 of part 3, analyze the lineage graph. Research the effects of different partition functions.

Zhikang Hao: Sorting algorithm for part2. Completed Task 3, 4 of part 3, implements the initial pagerank algorithm, researching the effects of partitioning and different cache policies.