CS 744 Homework1

Rui Huang, Zhenyu Zou, Zheng Liu

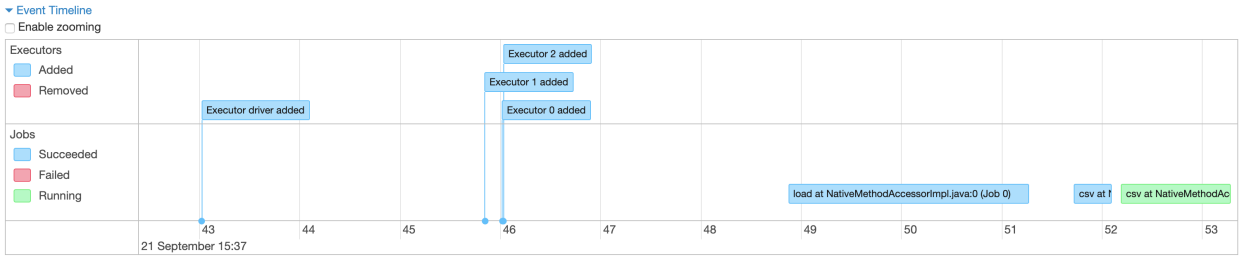
September 21, 2019

1 Part 2

We read in the data as Spark DataFrame. And we mainly rely on orderBy() method of Spark DataFrame to do the sorting. Here's the result of sorting on export.csv dataset. We show the screenshot from Spark WebUI.

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20190921153423-0006	ExportSort	15	24.0 GB	2019/09/21 15:34:23	huangrui	FINISHED	16 s

And here's the screenshot of event timeline.



Active Jobs (1)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
2	csv at NativeMethodAccessorImpl.java:0 csv at NativeMethodAccessorImpl.java:0 (kill)	2019/09/21 15:37:52	1 s	1/2	1/201 (6 running)

Completed Jobs (2)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
1	csv at NativeMethodAccessorImpl.java:0 csv at NativeMethodAccessorImpl.java:0	2019/09/21 15:37:51	0.4 s	1/1	1/1
0	load at NativeMethodAccessorImpl.java:0 load at NativeMethodAccessorImpl.java:0	2019/09/21 15:37:48	2 s	1/1	1/1

2 Part 3

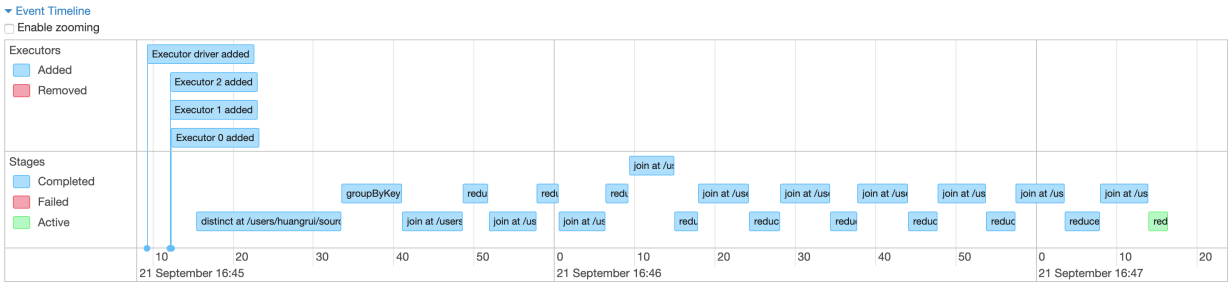
2.1 Task 1

After some time of thinking, we come to the data structure that using two RDDs storing links and pagerank separately. During each iteration, we join the RDDs by key and calculate modifications to rank and build new rank.

The completion time for BerkStan dataset is shown below.

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20190921164509-0027	PageRank	15	24.0 GB	2019/09/21 16:45:09	huangrui	FINISHED	2.2 min

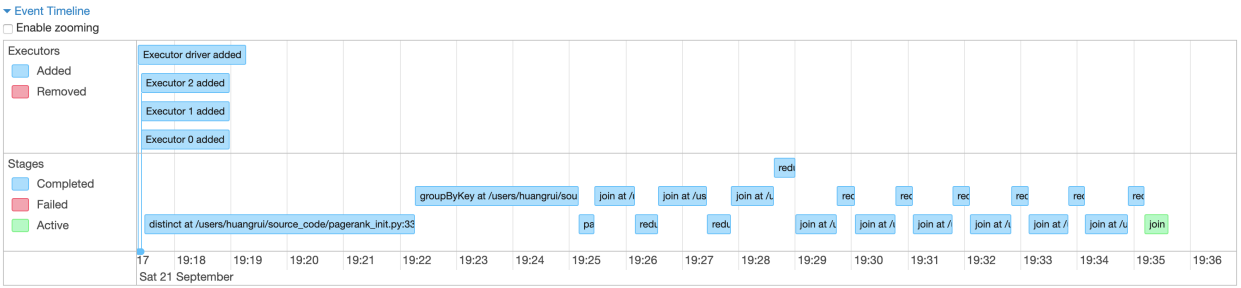
The timeline is shown below.



The completion time for enwiki dataset is shown below. (Note due to time limit of cloudlab, we used 1/3 of all the data in enwiki. Since running over all dataset will take around 1 hour.)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20190921191721-0007	PageRank_enwiki	15	26.0 GB	2019/09/21 19:17:21	huangrui	FINISHED	19 min

The corresponding timeline:



2.2 Task 2

We compared different partitions: 1, 5, 15, 30, 50.

Partition	join time(s)	reduce time(s)
1	210	258
5	56	96
15	43	25
30	22	27
50	20	27

We can see that 15 partitions have lowest reduce time. This is reasonable, since there are in total 15 cores. In terms of join time, we can see 30 or 50 partitions are better. But anyway, 1 partition is far worse than others.

2.3 Task 3

Without persisting RDD in memory, the completion time and detailed time by stage are shown as follows.

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20190921194044-0008	PageRank_enwiki	15	26.0 GB	2019/09/21 19:40:44	huangrui	FINISHED	20 min

▾ Completed Stages (21)

Stage Id ▾	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
20	reduceByKey at /users/huangrui/source_code/pagerank_init.py:40	+details	2019/09/21 19:58:57	17 s	150/150			1467.4 MB	701.8 MB
19	join at /users/huangrui/source_code/pagerank_init.py:39	+details	2019/09/21 19:58:13	44 s	150/150			1889.1 MB	1467.4 MB
18	reduceByKey at /users/huangrui/source_code/pagerank_init.py:40	+details	2019/09/21 19:57:53	20 s	135/135			1468.5 MB	708.2 MB
17	join at /users/huangrui/source_code/pagerank_init.py:39	+details	2019/09/21 19:57:02	51 s	135/135			1887.4 MB	1468.5 MB
16	reduceByKey at /users/huangrui/source_code/pagerank_init.py:40	+details	2019/09/21 19:56:42	20 s	120/120			1468.1 MB	706.5 MB
15	join at /users/huangrui/source_code/pagerank_init.py:39	+details	2019/09/21 19:55:54	48 s	120/120			1882.0 MB	1468.1 MB
14	reduceByKey at /users/huangrui/source_code/pagerank_init.py:40	+details	2019/09/21 19:55:32	21 s	105/105			1467.3 MB	701.1 MB
13	join at /users/huangrui/source_code/pagerank_init.py:39	+details	2019/09/21 19:54:43	50 s	105/105			1892.4 MB	1467.3 MB
12	reduceByKey at /users/huangrui/source_code/pagerank_init.py:40	+details	2019/09/21 19:54:23	19 s	90/90			1471.6 MB	711.5 MB
11	join at /users/huangrui/source_code/pagerank_init.py:39	+details	2019/09/21 19:53:40	43 s	90/90			1892.2 MB	1471.6 MB
10	reduceByKey at /users/huangrui/source_code/pagerank_init.py:40	+details	2019/09/21 19:53:21	19 s	75/75			1478.7 MB	711.2 MB
9	join at /users/huangrui/source_code/pagerank_init.py:39	+details	2019/09/21 19:52:32	48 s	75/75			1911.4 MB	1478.7 MB
8	reduceByKey at /users/huangrui/source_code/pagerank_init.py:40	+details	2019/09/21 19:52:11	21 s	60/60			1507.9 MB	730.5 MB
7	join at /users/huangrui/source_code/pagerank_init.py:39	+details	2019/09/21 19:51:20	51 s	60/60			1886.1 MB	1507.9 MB
6	reduceByKey at /users/huangrui/source_code/pagerank_init.py:40	+details	2019/09/21 19:50:56	24 s	45/45			1615.7 MB	705.2 MB
5	join at /users/huangrui/source_code/pagerank_init.py:39	+details	2019/09/21 19:50:06	50 s	45/45			2.1 GB	1615.7 MB
4	reduceByKey at /users/huangrui/source_code/pagerank_init.py:40	+details	2019/09/21 19:49:45	21 s	30/30			1194.2 MB	969.3 MB
3	join at /users/huangrui/source_code/pagerank_init.py:39	+details	2019/09/21 19:49:04	41 s	30/30			2.3 GB	1194.2 MB
2	partitionBy at /users/huangrui/source_code/pagerank_init.py:34	+details	2019/09/21 19:48:49	15 s	24/24			1352.7 MB	1180.9 MB
1	groupByKey at /users/huangrui/source_code/pagerank_init.py:33	+details	2019/09/21 19:45:56	2.9 min	24/24			1961.1 MB	1352.7 MB
0	distinct at /users/huangrui/source_code/pagerank_init.py:33	+details	2019/09/21 19:40:50	5.1 min	24/24	2.7 GB			1961.1 MB

Persisting RDD in memory, the completion time and detailed time by stage are shown as follows.

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20190921191721-0007	PageRank_enwiki	15	26.0 GB	2019/09/21 19:17:21	huangrui	FINISHED	19 min

▾ Completed Stages (24)

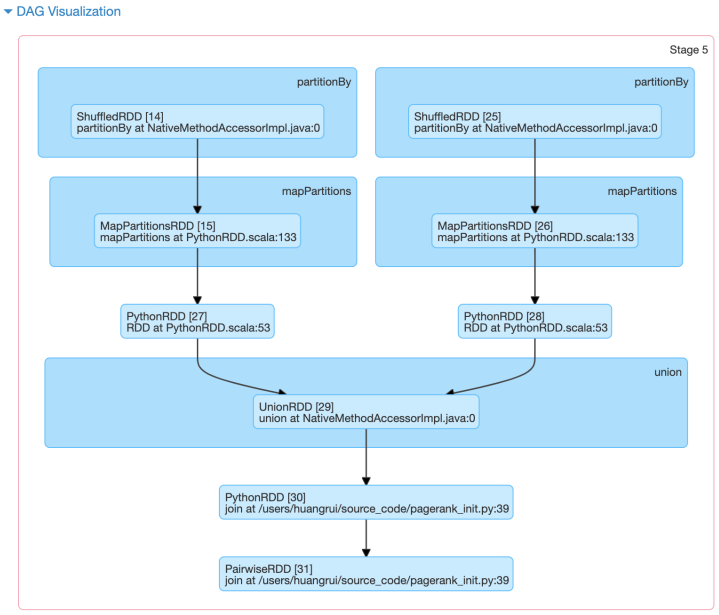
Stage Id ▾	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
23	runJob at PythonRDD.scala:153	+details	2019/09/21 19:36:13	0.2 s	1/1			4.1 MB	
22	reduceByKey at /users/huangrui/source_code/pagerank_init.py:40	+details	2019/09/21 19:35:55	18 s	165/165			1460.8 MB	696.0 MB
21	join at /users/huangrui/source_code/pagerank_init.py:39	+details	2019/09/21 19:35:11	44 s	165/165	1180.9 MB		701.8 MB	1460.8 MB
20	reduceByKey at /users/huangrui/source_code/pagerank_init.py:40	+details	2019/09/21 19:34:53	18 s	150/150			1467.5 MB	701.8 MB
19	join at /users/huangrui/source_code/pagerank_init.py:39	+details	2019/09/21 19:34:07	45 s	150/150	1180.9 MB		708.2 MB	1467.5 MB
18	reduceByKey at /users/huangrui/source_code/pagerank_init.py:40	+details	2019/09/21 19:33:50	18 s	135/135			1468.5 MB	708.2 MB
17	join at /users/huangrui/source_code/pagerank_init.py:39	+details	2019/09/21 19:33:08	42 s	135/135	1180.9 MB		706.5 MB	1468.5 MB
16	reduceByKey at /users/huangrui/source_code/pagerank_init.py:40	+details	2019/09/21 19:32:49	19 s	120/120			1468.0 MB	706.5 MB
15	join at /users/huangrui/source_code/pagerank_init.py:39	+details	2019/09/21 19:32:06	43 s	120/120	1180.9 MB		701.1 MB	1468.0 MB
14	reduceByKey at /users/huangrui/source_code/pagerank_init.py:40	+details	2019/09/21 19:31:47	19 s	105/105			1467.3 MB	701.1 MB
13	join at /users/huangrui/source_code/pagerank_init.py:39	+details	2019/09/21 19:31:05	42 s	105/105	1180.9 MB		711.5 MB	1467.3 MB
12	reduceByKey at /users/huangrui/source_code/pagerank_init.py:40	+details	2019/09/21 19:30:46	19 s	90/90			1471.5 MB	711.5 MB
11	join at /users/huangrui/source_code/pagerank_init.py:39	+details	2019/09/21 19:30:03	43 s	90/90	1180.9 MB		711.2 MB	1471.5 MB
10	reduceByKey at /users/huangrui/source_code/pagerank_init.py:40	+details	2019/09/21 19:29:44	19 s	75/75			1478.7 MB	711.2 MB
9	join at /users/huangrui/source_code/pagerank_init.py:39	+details	2019/09/21 19:28:59	45 s	75/75	1180.9 MB		727.6 MB	1478.7 MB
8	reduceByKey at /users/huangrui/source_code/pagerank_init.py:40	+details	2019/09/21 19:28:37	23 s	60/60			1507.9 MB	727.6 MB
7	join at /users/huangrui/source_code/pagerank_init.py:39	+details	2019/09/21 19:27:52	45 s	60/60	1180.9 MB		705.2 MB	1507.9 MB
6	reduceByKey at /users/huangrui/source_code/pagerank_init.py:40	+details	2019/09/21 19:27:26	26 s	45/45			1615.7 MB	705.2 MB
5	join at /users/huangrui/source_code/pagerank_init.py:39	+details	2019/09/21 19:26:34	52 s	45/45	1180.9 MB		969.3 MB	1615.7 MB
4	reduceByKey at /users/huangrui/source_code/pagerank_init.py:40	+details	2019/09/21 19:26:09	25 s	30/30			1194.1 MB	969.3 MB
3	join at /users/huangrui/source_code/pagerank_init.py:39	+details	2019/09/21 19:25:26	43 s	30/30	1180.9 MB		1180.9 MB	1194.1 MB
2	partitionBy at /users/huangrui/source_code/pagerank_init.py:34	+details	2019/09/21 19:25:09	17 s	24/24			1352.7 MB	1180.9 MB
1	groupByKey at /users/huangrui/source_code/pagerank_init.py:33	+details	2019/09/21 19:22:15	2.9 min	24/24			1960.9 MB	1352.7 MB
0	distinct at /users/huangrui/source_code/pagerank_init.py:33	+details	2019/09/21 19:17:28	4.8 min	24/24	2.7 GB			1960.9 MB

We can see that from our experiment, in terms of completion time, there seems not to be significant difference. Maybe this is because reading from disk is not much slower than memory for dataset of this size.

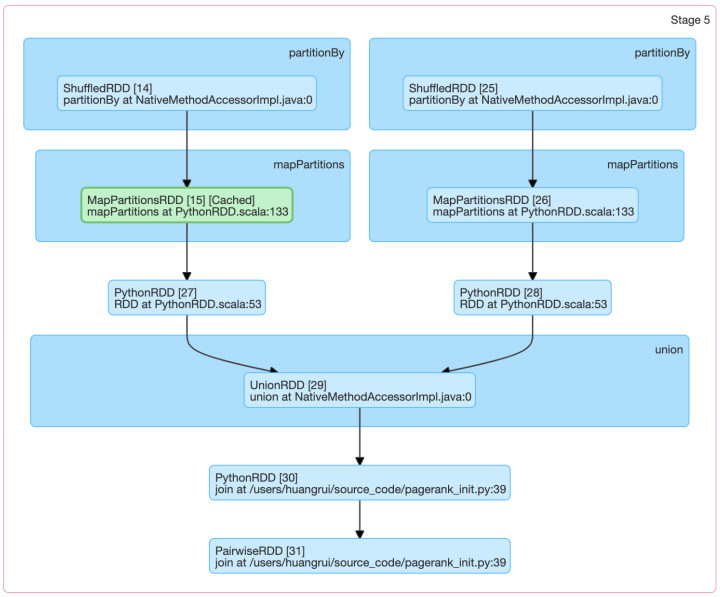
But there's obvious difference in shuffle read. Since persisting RDD in memory can significantly reduce disk read. In our case, it reduces about half disk read(from 1400MB down to 700MB).

Also there's some difference in DAG. We show the figures here.

Without caching:



With caching:



2.4 Task 4

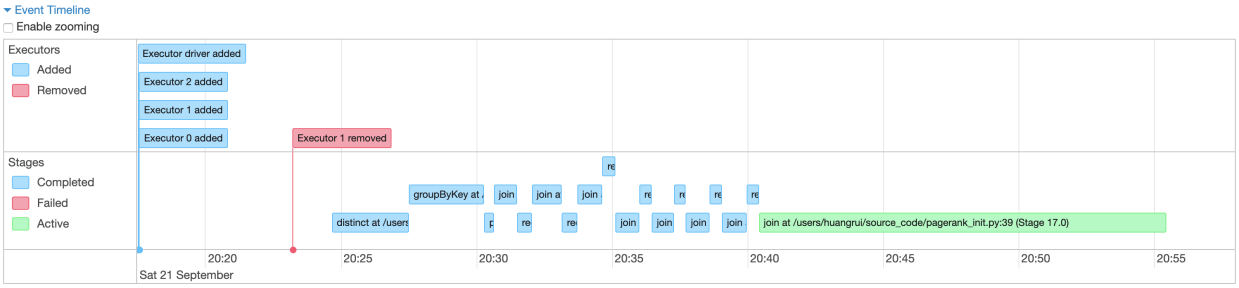
We killed the worker process at 25% of the application lifetime. When killing it, the error log is shown as follows:

```
19/09/21 20:23:48 INFO TransportClientFactory: Found inactive connection to /
128.104.223.104:40997, creating a new one.
19/09/21 20:23:48 ERROR RetryingBlockFetcher: Exception while beginning fetch of 3
outstanding blocks
java.io.IOException: Failed to connect to /128.104.223.104:40997
        at
org.apache.spark.network.client.TransportClientFactory.createClient(TransportClientFactory
.java:245)
        at
org.apache.spark.network.client.TransportClientFactory.createClient(TransportClientFactory
.java:187)
        at org.apache.spark.network.netty.NettyBlockTransferService$
$anon$2.createAndStart(NettyBlockTransferService.scala:114)
        at
org.apache.spark.network.shuffle.RetryingBlockFetcher.fetchAllOutstanding(RetryingBlockFet
cher.java:141)
        at
org.apache.spark.network.shuffle.RetryingBlockFetcher.start(RetryingBlockFetcher.java:121)
        at
org.apache.spark.network.netty.NettyBlockTransferService.fetchBlocks(NettyBlockTransferSer
vice.scala:124)
        at
org.apache.spark.storage.ShuffleBlockFetcherIterator.sendRequest(ShuffleBlockFetcherIterat
or.scala:260)
        at
org.apache.spark.storage.ShuffleBlockFetcherIterator.org$apache$spark$storage$ShuffleBlock
FetcherIterator$$send$1(ShuffleBlockFetcherIterator.scala:531)
        at
org.apache.spark.storage.ShuffleBlockFetcherIterator.fetchUpToMaxBytes(ShuffleBlockFetcher
Iterator.scala:526)
```

On the worker side, the error log is:

```
19/09/21 20:23:48 INFO TransportClientFactory: Found inactive connection to /
128.104.223.104:40997, creating a new one.
19/09/21 20:23:48 ERROR RetryingBlockFetcher: Exception while beginning fetch of 3
outstanding blocks
java.io.IOException: Failed to connect to /128.104.223.104:40997
        at
org.apache.spark.network.client.TransportClientFactory.createClient(TransportClientFactory
.java:245)
        at
org.apache.spark.network.client.TransportClientFactory.createClient(TransportClientFactory
.java:187)
        at org.apache.spark.network.netty.NettyBlockTransferService$
$anon$2.createAndStart(NettyBlockTransferService.scala:114)
        at
org.apache.spark.network.shuffle.RetryingBlockFetcher.fetchAllOutstanding(RetryingBlockFet
cher.java:141)
        at
org.apache.spark.network.shuffle.RetryingBlockFetcher.start(RetryingBlockFetcher.java:121)
        at
org.apache.spark.network.netty.NettyBlockTransferService.fetchBlocks(NettyBlockTransferSer
vice.scala:124)
        at
org.apache.spark.storage.ShuffleBlockFetcherIterator.sendRequest(ShuffleBlockFetcherIterat
or.scala:260)
        at
org.apache.spark.storage.ShuffleBlockFetcherIterator.org$apache$spark$storage$ShuffleBlock
FetcherIterator$$send$1(ShuffleBlockFetcherIterator.scala:531)
        at
org.apache.spark.storage.ShuffleBlockFetcherIterator.fetchUpToMaxBytes(ShuffleBlockFetcher
Iterator.scala:526)
```

We successfully killed it but the application kept running. We can see Spark is resistent to worker node failure. The timeline is shown as bellow.



But actually, later on, at 7th iteration, the program stopped running.

Active Stages (1)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
17	join at /users/huangrui/source_code/pagerank_init.py:39 +details (kill)	2019/09/21 20:40:24	11 min	130/135 (26 failed)	1498.3 MB		836.5 MB	1302.5 MB

Pending Stages (6)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
23	runJob at PythonRDD.scala:153 +details	Unknown	Unknown	0/1				
22	reduceByKey at /users/huangrui/source_code/pagerank_init.py:40 +details	Unknown	Unknown	0/165				
21	join at /users/huangrui/source_code/pagerank_init.py:39 +details	Unknown	Unknown	0/165				
20	reduceByKey at /users/huangrui/source_code/pagerank_init.py:40 +details	Unknown	Unknown	0/150				
19	join at /users/huangrui/source_code/pagerank_init.py:39 +details	Unknown	Unknown	0/150				
18	reduceByKey at /users/huangrui/source_code/pagerank_init.py:40 +details	Unknown	Unknown	0/135				

We figured out the bug: because of running out of space on disk. We think it is because stopping on worker will lead to more intermediate data to be put on remaining two workers. Due to our limited disk space, the application could not proceed.