Assignment 1 Report

Yufei Wang, Zifan Liu, Jiefeng Chen

Application Completion Time

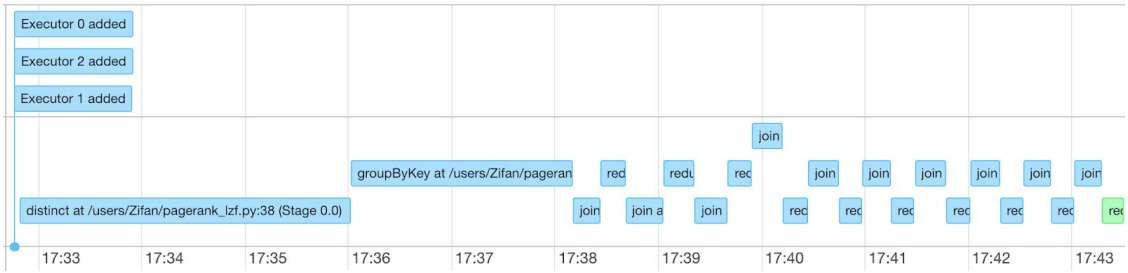
1. Berkeley-Stanford Web Graph Dataset (without persisting)

Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
PageRank Berkeley-Stanford	15	8.0 GB	2019/02/14 17:30:17	Zifan	FINISHED	1.8 min

2. Enwiki-pages-articles Dataset (without persisting)

Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
PageRank Wiki non-persist	15	8.0 GB	2019/02/14 17:32:43	Zifan	FINISHED	11 min

The event timeline:

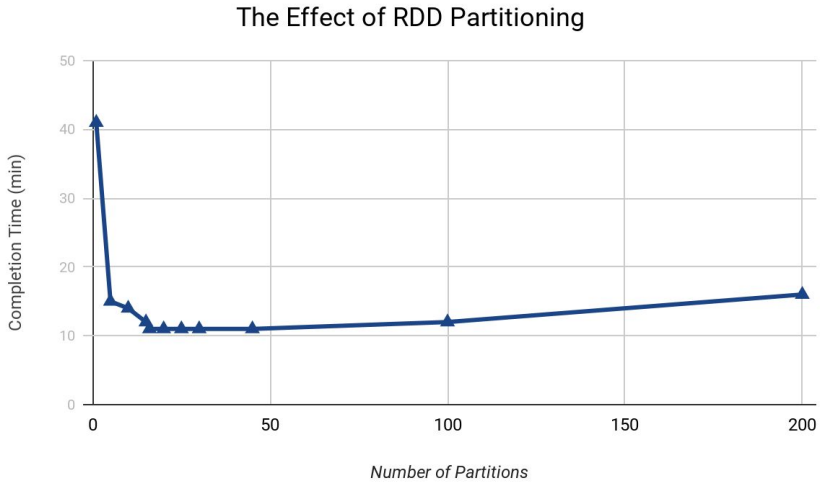


From the timeline, we could see that the distinct operation in the preprocessing stage (which is used for removing potential duplicate records) takes a long time (3.2 min). Then the groupByKey operation takes 2.2 min. After that, map and reduce operation run alternatively, each taking roughly 5 seconds.

Performance Changes

1. adding appropriate custom RDD partitioning

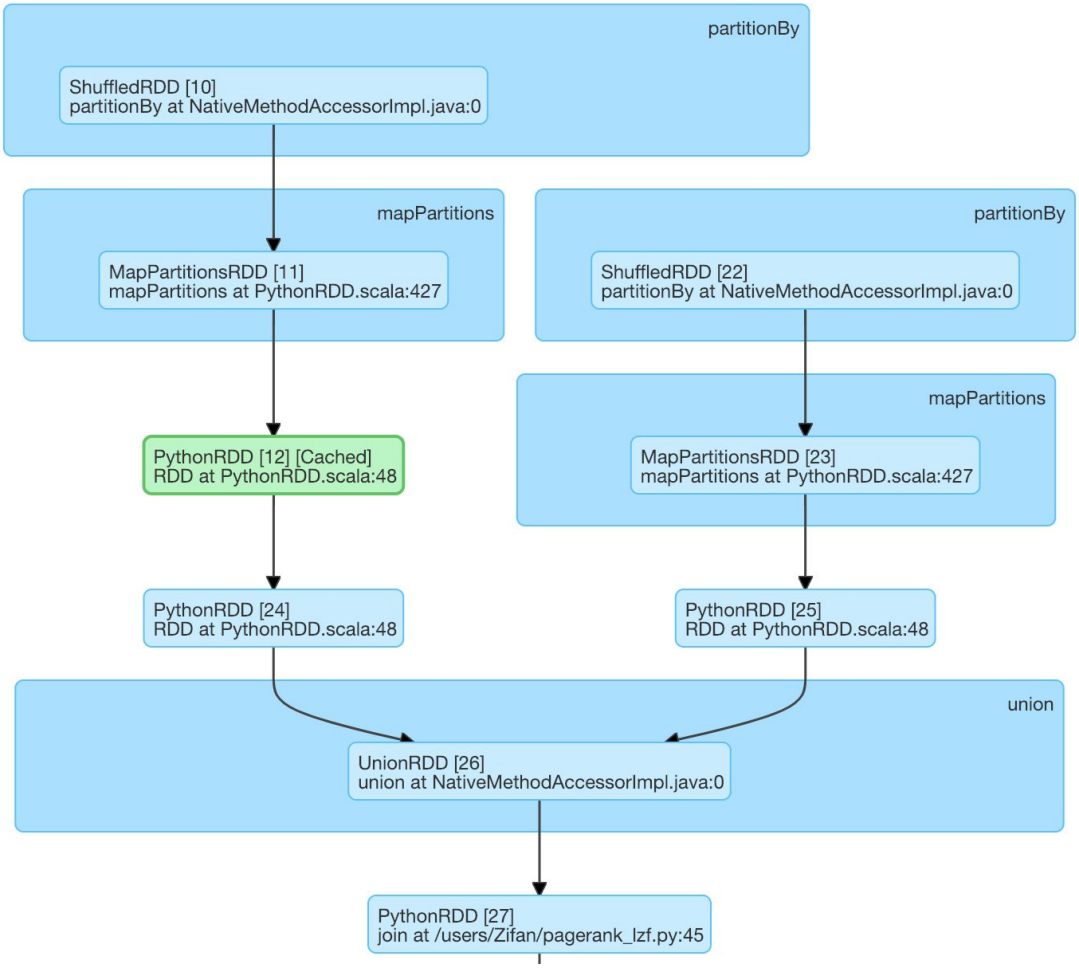
We tune the number of partitions and see how it affects the completion time. See the following figure.



From the figure, we can see when the number of partitions is small (less than 10), the processing is very slow. This is because we only use a small number of cores and the tasks are not fully parallelized. When the number of partitions becomes large (larger than 10 and less than 16), the processing becomes substantially fast. This is because the data are now laid out, the network traffic is minimized. Aslo, now we fully utilize the power of cores. When the number of partitions is larger than 15 but less than 45, the performance gain is very small. This is because Spark can only run 1 concurrent task for every partition of the RDD, up to the maximum number of cores in the cluster, and the total number of cores in our cluster is 15. When the number of partitions becomes very large (larger than 100), the processing becomes slow again. This is because now the time to create RDD partitions cannot be neglected and the partitions are not well-distributed, the parallelization efficiency goes down.

2. persisting the appropriate RDD as in-memory objects

After we persist the graph in the memory, we could see that in the DAG, one of the input for join is kept in memory.



Below are some statistics for each stage.

Without persisting:

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
20	join at /users/Zifan/pagerank_lzf.py:45 +details	2019/02/14 17:43:01	16 s	176/176			1458.4 MB	979.8 MB
19	reduceByKey at /users/Zifan/pagerank_lzf.py:46 +details	2019/02/14 17:42:47	14 s	160/160			980.0 MB	644.2 MB
18	join at /users/Zifan/pagerank_lzf.py:45 +details	2019/02/14 17:42:31	16 s	160/160			1464.4 MB	980.0 MB
17	reduceByKey at /users/Zifan/pagerank_lzf.py:46 +details	2019/02/14 17:42:18	13 s	144/144			980.8 MB	650.2 MB
16	join at /users/Zifan/pagerank_lzf.py:45 +details	2019/02/14 17:42:00	17 s	144/144			1467.0 MB	980.8 MB
15	reduceByKey at /users/Zifan/pagerank_lzf.py:46 +details	2019/02/14 17:41:47	14 s	128/128			987.1 MB	652.8 MB
14	join at /users/Zifan/pagerank_lzf.py:45 +details	2019/02/14 17:41:28	18 s	128/128			1468.1 MB	987.1 MB
13	reduceByKey at /users/Zifan/pagerank_lzf.py:46 +details	2019/02/14 17:41:14	14 s	112/112			985.7 MB	653.9 MB
12	join at /users/Zifan/pagerank_lzf.py:45 +details	2019/02/14 17:40:58	17 s	112/112			1456.5 MB	985.7 MB
11	reduceByKey at /users/Zifan/pagerank_lzf.py:46 +details	2019/02/14 17:40:44	13 s	96/96			983.6 MB	642.3 MB
10	join at /users/Zifan/pagerank_lzf.py:45 +details	2019/02/14 17:40:26	18 s	96/96			1458.2 MB	983.6 MB
9	reduceByKey at /users/Zifan/pagerank_lzf.py:46 +details	2019/02/14 17:40:12	15 s	80/80			985.3 MB	644.0 MB
8	join at /users/Zifan/pagerank_lzf.py:45 +details	2019/02/14 17:39:54	18 s	80/80			1453.8 MB	985.3 MB
7	reduceByKey at /users/Zifan/pagerank_lzf.py:46 +details	2019/02/14 17:39:40	14 s	64/64			995.7 MB	639.6 MB
6	join at /users/Zifan/pagerank_lzf.py:45 +details	2019/02/14 17:39:21	19 s	64/64			1456.0 MB	995.7 MB
5	reduceByKey at /users/Zifan/pagerank_lzf.py:46 +details	2019/02/14 17:39:02	18 s	48/48			1034.9 MB	641.8 MB
4	join at /users/Zifan/pagerank_lzf.py:45 +details	2019/02/14 17:38:40	22 s	48/48			1465.9 MB	1034.9 MB
3	reduceByKey at /users/Zifan/pagerank_lzf.py:46 +details	2019/02/14 17:38:26	15 s	32/32			750.0 MB	651.7 MB
2	join at /users/Zifan/pagerank_lzf.py:45 +details	2019/02/14 17:38:10	16 s	32/32			1628.4 MB	750.0 MB
1	groupByKey at /users/Zifan/pagerank_lzf.py:38 +details	2019/02/14 17:36:01	2.2 min	16/16			1219.8 MB	814.2 MB
0	distinct at /users/Zifan/pagerank_lzf.py:38 +details	2019/02/14 17:32:49	3.2 min	16/16	1531.3 MB			1219.8 MB

With persisting:

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
20	join at /users/Zifan/pagerank_lzf.py:45 +details	2019/02/14 19:14:13	14 s	176/176	742.5 MB		644.2 MB	979.8 MB
19	reduceByKey at /users/Zifan/pagerank_lzf.py:46 +details	2019/02/14 19:13:59	14 s	160/160			980.1 MB	644.2 MB
18	join at /users/Zifan/pagerank_lzf.py:45 +details	2019/02/14 19:13:44	16 s	160/160	742.5 MB		650.2 MB	980.1 MB
17	reduceByKey at /users/Zifan/pagerank_lzf.py:46 +details	2019/02/14 19:13:30	14 s	144/144			980.7 MB	650.2 MB
16	join at /users/Zifan/pagerank_lzf.py:45 +details	2019/02/14 19:13:15	15 s	144/144	742.5 MB		652.7 MB	980.7 MB
15	reduceByKey at /users/Zifan/pagerank_lzf.py:46 +details	2019/02/14 19:13:02	13 s	128/128			987.1 MB	652.7 MB
14	join at /users/Zifan/pagerank_lzf.py:45 +details	2019/02/14 19:12:47	15 s	128/128	742.5 MB		653.9 MB	987.1 MB
13	reduceByKey at /users/Zifan/pagerank_lzf.py:46 +details	2019/02/14 19:12:32	14 s	112/112			985.7 MB	653.9 MB
12	join at /users/Zifan/pagerank_lzf.py:45 +details	2019/02/14 19:12:16	16 s	112/112	742.5 MB		642.3 MB	985.7 MB
11	reduceByKey at /users/Zifan/pagerank_lzf.py:46 +details	2019/02/14 19:12:01	15 s	96/96			983.5 MB	642.3 MB
10	join at /users/Zifan/pagerank_lzf.py:45 +details	2019/02/14 19:11:44	17 s	96/96	742.5 MB		644.0 MB	983.5 MB
9	reduceByKey at /users/Zifan/pagerank_lzf.py:46 +details	2019/02/14 19:11:29	15 s	80/80			985.3 MB	644.0 MB
8	join at /users/Zifan/pagerank_lzf.py:45 +details	2019/02/14 19:11:13	16 s	80/80	742.5 MB		639.6 MB	985.3 MB
7	reduceByKey at /users/Zifan/pagerank_lzf.py:46 +details	2019/02/14 19:10:59	14 s	64/64			995.7 MB	639.6 MB
6	join at /users/Zifan/pagerank_lzf.py:45 +details	2019/02/14 19:10:41	18 s	64/64	742.5 MB		641.7 MB	995.7 MB
5	reduceByKey at /users/Zifan/pagerank_lzf.py:46 +details	2019/02/14 19:10:23	19 s	48/48			1034.9 MB	641.7 MB
4	join at /users/Zifan/pagerank_lzf.py:45 +details	2019/02/14 19:10:03	20 s	48/48	742.5 MB		654.8 MB	1034.9 MB
3	reduceByKey at /users/Zifan/pagerank_lzf.py:46 +details	2019/02/14 19:09:48	14 s	32/32			749.9 MB	654.8 MB
2	join at /users/Zifan/pagerank_lzf.py:45 +details	2019/02/14 19:09:29	19 s	32/32	697.3 MB		863.8 MB	749.9 MB
1	groupByKey at /users/Zifan/pagerank_lzf.py:38 +details	2019/02/14 19:07:19	2.2 min	16/16			1221.0 MB	814.2 MB
0	distinct at /users/Zifan/pagerank_lzf.py:38 +details	2019/02/14 19:04:00	3.3 min	16/16	1531.3 MB			1221.0 MB

From the table above we could see that with persisting, the size of shuffle read in each map operation reduces. Obviously that is due to the persisted graph in the memory. However, there is no remarkable difference in running time, probably because the reading overhead is not a bottleneck for this task in this system.

3. killing a worker process

If one worker is shut down, firstly other workers who are fetching some blocks from it (if any) will report some errors as those blocks are missing:

```
19/02/14 16:41:44 WARN TaskSetManager: Lost task 16.0 in stage 5.0 (TID 124, 128.104.223.197, executor 1): FetchFailed(BlockManagerId(2, 128.104.223.196, 33044, None), shuffleId=21, mapId=25, reduceId=1, message='some java error message'
```

Then the master node will detect that this worker is down and report something in the log:

```
19/02/13 21:40:35 ERROR TaskSchedulerImpl: Lost executor 0 on 128.104.223.196: Remote RPC client disassociated. Likely due to containers exceeding thresholds, or network issues. Check driver logs for WARN messages.

19/02/13 21:40:35 WARN TaskSetManager: Lost task 14.0 in stage 7.0 (TID 211, 128.104.223.196, executor 0): ExecutorLostFailure (executor 0 exited caused by one of the running tasks) Reason: Remote RPC client disassociated. Likely due to containers exceeding thresholds, or network issues. Check driver logs for WARN messages.

19/02/13 21:40:35 INFO StandaloneAppClient$ClientEndpoint: Executor updated: app-20190213213318-0075/0 is now FAILED (Worker shutting down)

19/02/13 21:40:35 INFO StandaloneSchedulerBackend: Executor app-20190213213318-0075/0 removed: Worker shutting down
```

Then the master node will remove that worker from the valid worker list. As the master already know that worker is gone, it will rerun all computing processes which are related to that worker, something like:

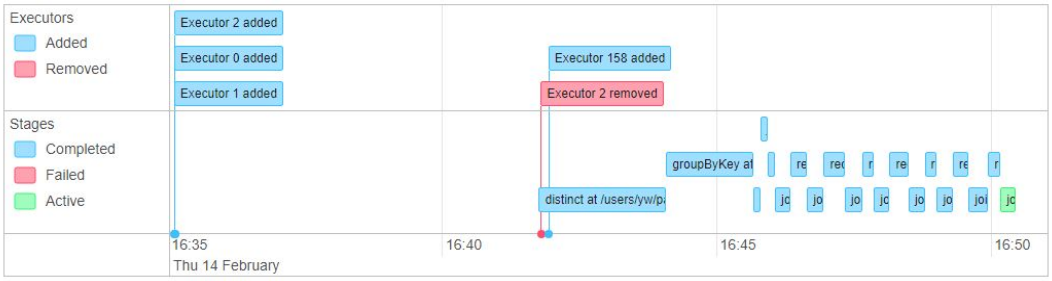
```
19/02/13 21:40:40 WARN TaskSetManager: Lost task 17.0 in stage 7.0 (TID 215, 128.104.223.102, executor 1): FetchFailed(null, shuffleId=17, mapId=-1, reduceId=2, message='some java error message'

19/02/13 21:40:40 INFO TaskSetManager: Task 17.0 in stage 7.0 (TID 215) failed, but the task will not be re-executed (either because the task failed with a shuffle data fetch failure, so the previous stage needs to be re-run, or because a different copy of the task has already succeeded).

19/02/13 21:40:40 INFO DAGScheduler: Resubmitting failed stages

19/02/13 21:40:40 INFO DAGScheduler: Submitting ShuffleMapStage 0 (PairwiseRDD[5] at distinct at /users/yw/pagerank.py:38), which has no missing parents
```

Tasks will be submitted according to the DAG graph until all lost blocks are recovered. In the MapReduce task, unfortunately, the whole task should re-execute as we do not save any intermediate results. The whole process should look like this:



That executor 158 looks a little weird. It seems that the master node was trying to build a new executor on the machine where the old executor was shut down and keeps trying for lots of times until it succeeds. The whole task takes about 18 minutes, which is reasonable as that worker is killed when the application reaches 50% of its lifetime - in normal cases, our application takes about 12 minutes to finish.