1. **Pseudo-code for PageRank Program in Spark Scala**

* The following is the Pseudo-code which heavily borrowed Scala syntax.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

1. **Source code for PageRank Program in Spark Scala**

* <https://github.com/CS6240/hw-4-spark-jill666666/blob/master/src/main/scala/pr/PageRank.scala>

1. **PageRank Program Output for k=100 and iterations=10**

텍스트이(가) 표시된 사진

자동 생성된 설명

* For each tuple (p1, p2), p1 is a page, and p2 is its corresponding PageRank.

1. **PageRank RDD Lineage Report**

* Iteration 1

텍스트이(가) 표시된 사진

자동 생성된 설명

* Iteration 2

텍스트이(가) 표시된 사진

자동 생성된 설명

* Iteration 3

텍스트이(가) 표시된 사진

자동 생성된 설명

테이블이(가) 표시된 사진

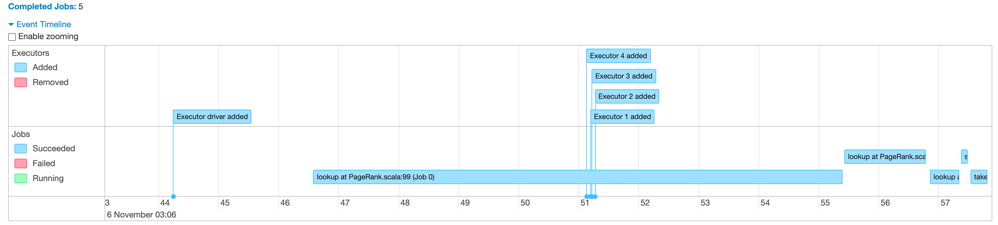
자동 생성된 설명

* If we look at the details for Job 0, it parallelizes the Ranks RDD which is then followed by map and lookup operations.

테이블이(가) 표시된 사진

자동 생성된 설명

* Job 1 and Job 2 is almost identical in terms of the operations. We can also check skipped stages since they have already been computed in the previous round.



* Event timeline shows how each job is being triggered. The jobs include joining the Ranks RDD with the graph RDD, mapping to get the PageRank mass and calculate total dangling pages PageRank values.

테이블이(가) 표시된 사진

자동 생성된 설명

* And we have Job 3 which sums up the PageRank values of all pages. This is for the debugging purpose to check whether the result sums up to 1. Finally, we have Job 4 that ‘take’ first 20 records (page 0 ~ 19) which is our final output.

1. **Spark Cache and Reusability**
2. Below is the lineage of Ranks RDD after 3 iterations.

텍스트이(가) 표시된 사진

자동 생성된 설명

Adding the print statement didn’t change the lineage, and each of the iteration print statement has been printed only once (e.g., 1, 2, 3, …). We can also find out that Spark is smart enough to re-use the pre-computed results.

1. We can certainly see from the log how Spark manages the jobs efficiently by performing the lookup actions. Below is part of lookup logs copied from the log file.

* 2021-11-05 21:55:16,253 INFO spark.SparkContext: Starting job: lookup at PageRank.scala:99
* 2021-11-05 21:55:16,583 INFO scheduler.DAGScheduler: Got job 0 (lookup at PageRank.scala:99) with 1 output partitions
* 2021-11-05 21:55:16,583 INFO scheduler.DAGScheduler: Final stage: ResultStage 5 (lookup at PageRank.scala:99)
* 2021-11-05 21:55:18,001 INFO scheduler.DAGScheduler: ResultStage 5 (lookup at PageRank.scala:99) finished in 0.075 s
* 2021-11-05 21:55:18,006 INFO scheduler.DAGScheduler: Job 0 finished: lookup at PageRank.scala:99, took 1.752810 s
* 2021-11-05 21:55:18,024 INFO spark.SparkContext: Starting job: lookup at PageRank.scala:99
* 2021-11-05 21:55:18,029 INFO scheduler.DAGScheduler: Got job 1 (lookup at PageRank.scala:99) with 1 output partitions
* 2021-11-05 21:55:18,029 INFO scheduler.DAGScheduler: Final stage: ResultStage 14 (lookup at PageRank.scala:99)

…

* 2021-11-05 21:55:19,402 INFO scheduler.DAGScheduler: Job 8 finished: lookup at PageRank.scala:99, took 0.118823 s
* 2021-11-05 21:55:19,418 INFO scheduler.DAGScheduler: Got job 9 (lookup at PageRank.scala:99) with 1 output partitions
* 2021-11-05 21:55:19,418 INFO scheduler.DAGScheduler: Final stage: ResultStage 194 (lookup at PageRank.scala:99)
* 2021-11-05 21:55:19,525 INFO scheduler.DAGScheduler: ResultStage 194 (lookup at PageRank.scala:99) finished in 0.012 s
* 2021-11-05 21:55:19,525 INFO scheduler.DAGScheduler: Job 9 finished: lookup at PageRank.scala:99, took 0.110396 s

As we can see from the logs, even without Cache() or Persist() operations, Spark re-uses Ranks RDD result that has been pre-computed in the previous iteration.

1. Refer to the link to the log files created after running two programs, one without Cache(), and the other one using Cache().

* Link: <https://github.com/CS6240/hw-4-spark-jill666666/tree/master/logs>

One most prominent difference between two cases is that the caching makes Spark program finds the blocks and store the blocks as values in memory to re-use them.

Below is some of the logs that only appears when cached.

* 2021-11-05 22:22:58,946 INFO storage.BlockManager: Found block rdd\_66\_2 locally
* 2021-11-05 22:22:58,946 INFO storage.BlockManager: Found block rdd\_66\_0 locally
* 2021-11-05 22:22:58,946 INFO storage.BlockManager: Found block rdd\_66\_3 locally
* 2021-11-05 22:22:58,946 INFO storage.BlockManager: Found block rdd\_66\_1 locall

Even though program without Cache() stores blocks such as broadcast\_8 as values in memory, following are the logs that are only distinct to the program with Cache().

* 2021-11-05 22:22:57,028 INFO memory.MemoryStore: Block rdd\_12\_0 stored as values in memory (estimated size 87.9 KiB, free 366.2 MiB)
* 2021-11-05 22:22:57,029 INFO memory.MemoryStore: Block rdd\_12\_1 stored as values in memory (estimated size 87.9 KiB, free 366.1 MiB)

One another interesting thing is that we can see the CachedPartitions in the lineage of the Ranks RDD.

텍스트, 신문, 스크린샷이(가) 표시된 사진

자동 생성된 설명

1. **Spark Scala PageRank Program on EMR (1 Master & 5 Workers)**

* k = 10,000 caused java heap space ‘OutOfMemory’ error. Setting k = 1,000 instead with 10 iterations resulted in the following output.
* Output & Log: <https://github.com/CS6240/hw-4-spark-jill666666/tree/master/aws-output>
* Running Time: Approximately 1 minute

텍스트이(가) 표시된 사진

자동 생성된 설명

* Thoughts: There are some approaches I would like to take, if more time has been allowed, to address the memory problem. One thing can be generating the input in parallel and create input RDD that can save a way much more memory. Another solution can be storing the intermediate output every iteration in the file system, like what we did in the MapReduce PageRank program.

1. **Pseudo-code for PageRank Program in MapReduce**

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

1. **Source code for PageRank Program in MapReduce**

* <https://github.com/CS6240/hw-4-mapreduce-jill666666/blob/master/src/main/java/pr/PageRank.java>

1. **MapReduce PageRank Program on EMR (1 Master & 5 Workers)**

* Logs: <https://github.com/CS6240/hw-4-mapreduce-jill666666/tree/master/aws-log>
* Output: <https://github.com/CS6240/hw-4-mapreduce-jill666666/tree/master/aws-output>
* Running Time: Approximately 50 minutes

