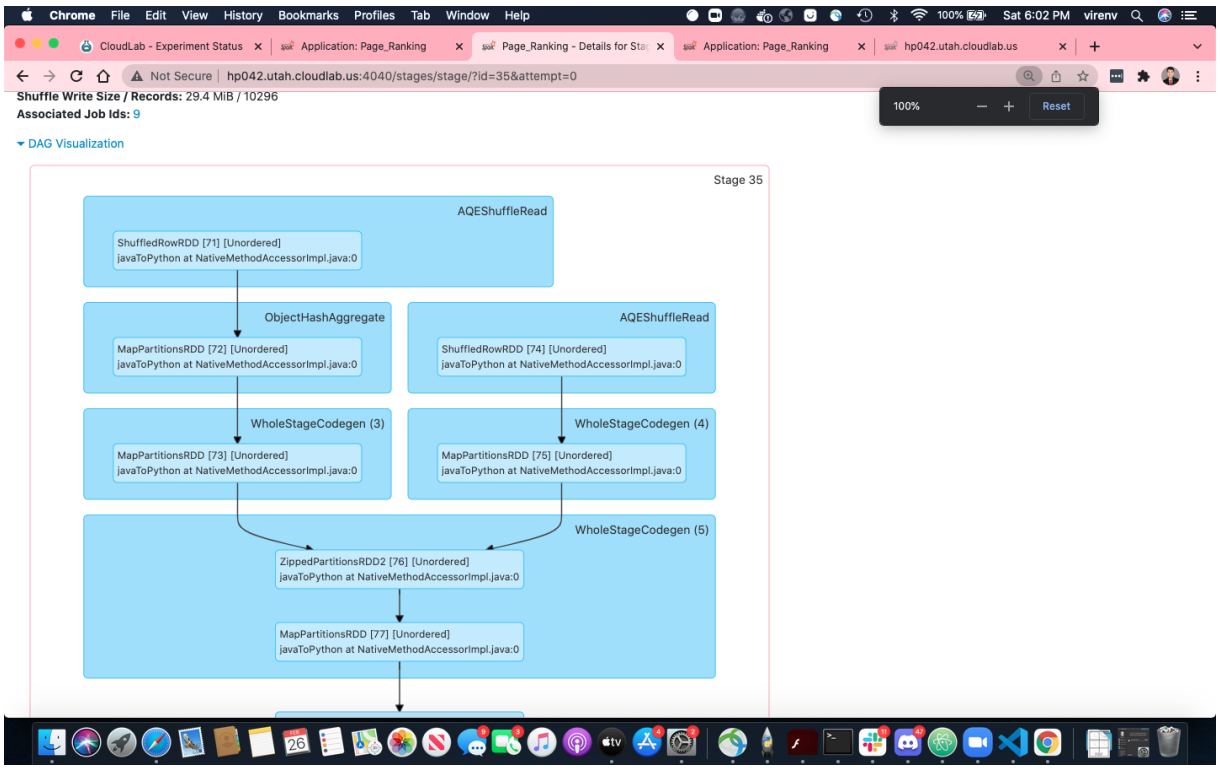
Pranav Akinepalli
Sameer Haniyur
Viren Velacheri
February 27th,2022

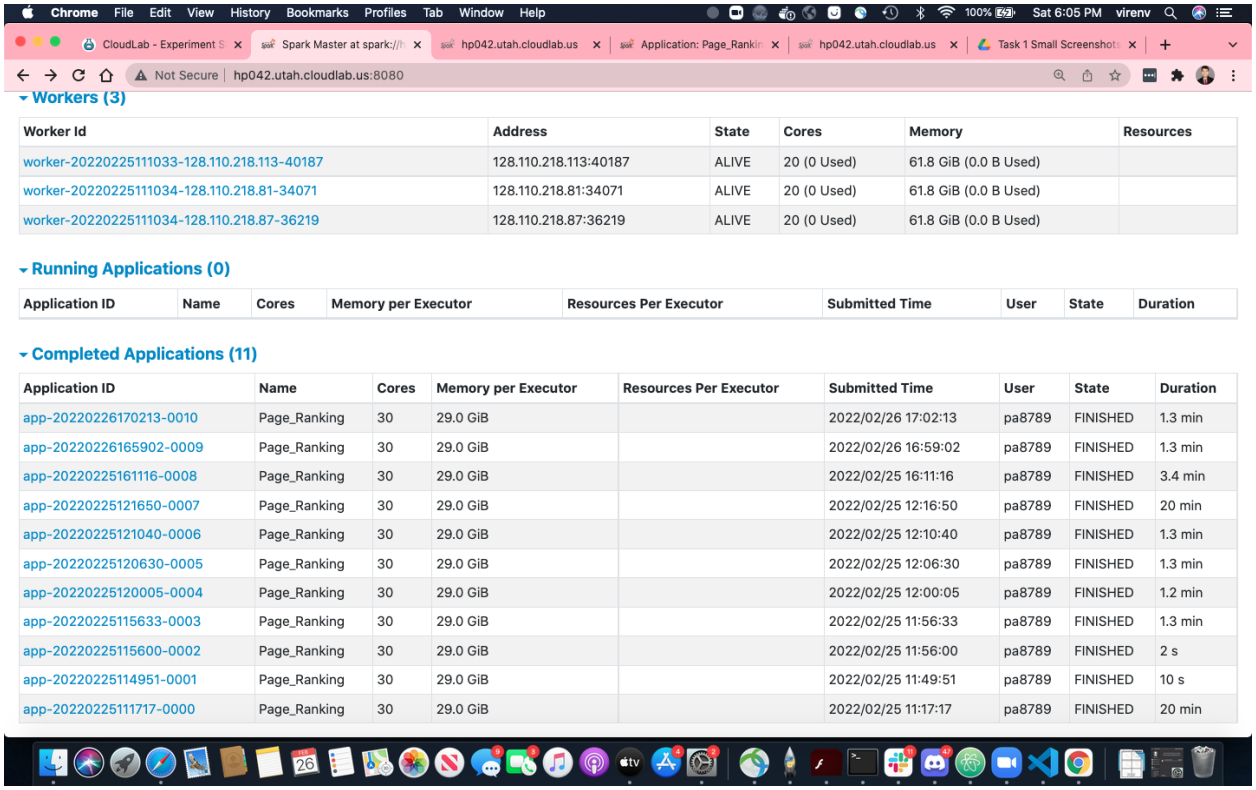
Assignment 2 Report

Task 1 Analysis

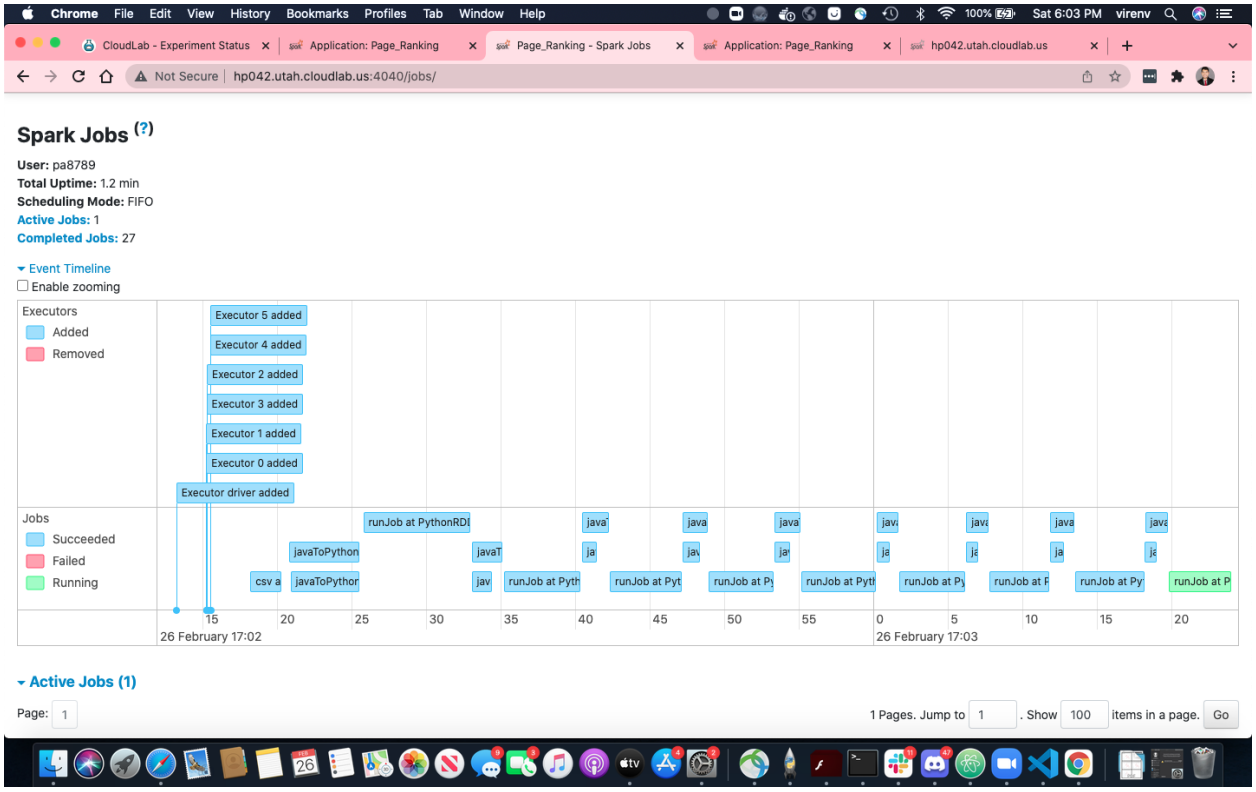
Small Dataset Analysis



(JavaToPython Job Stage Details)



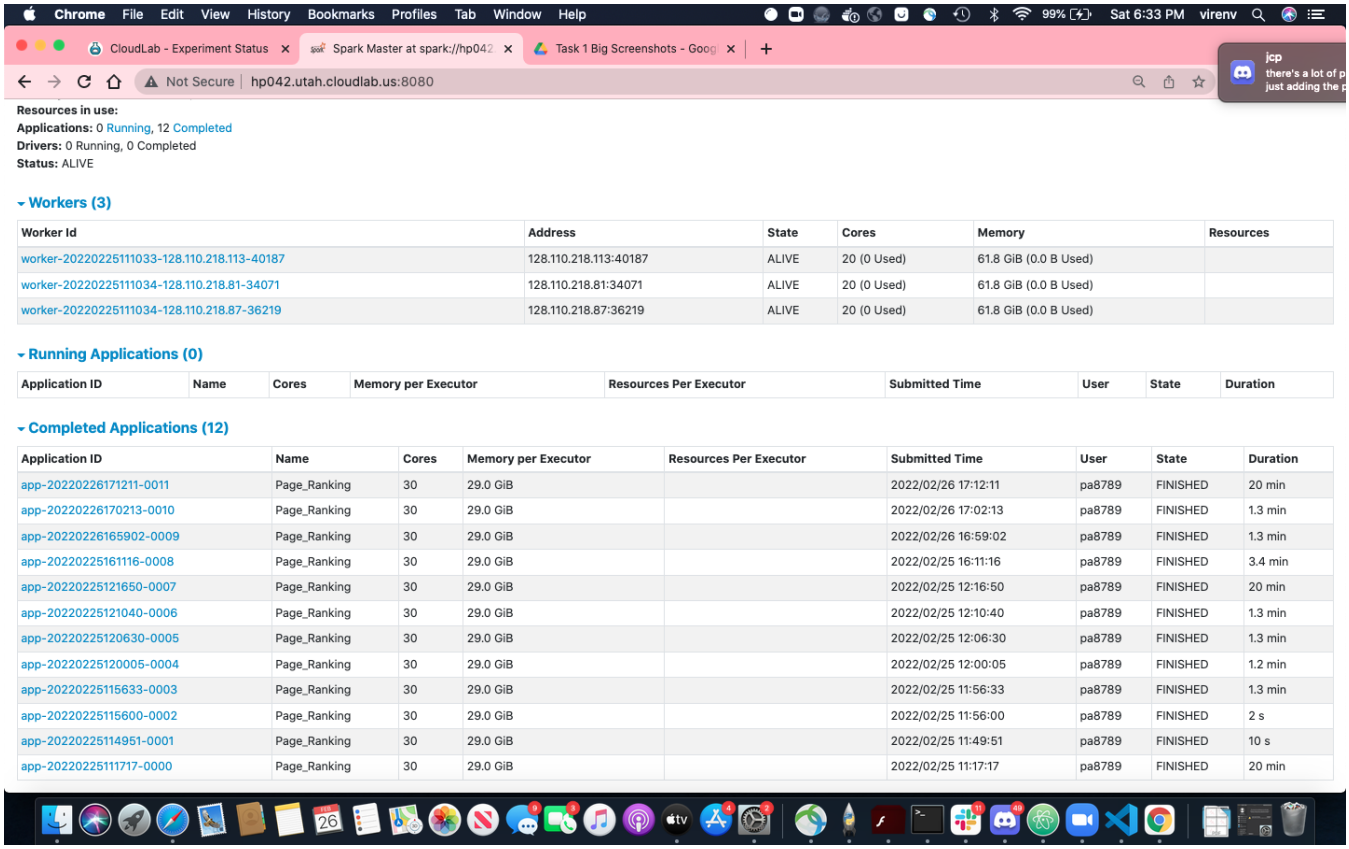
(1.3 minutes to completion)



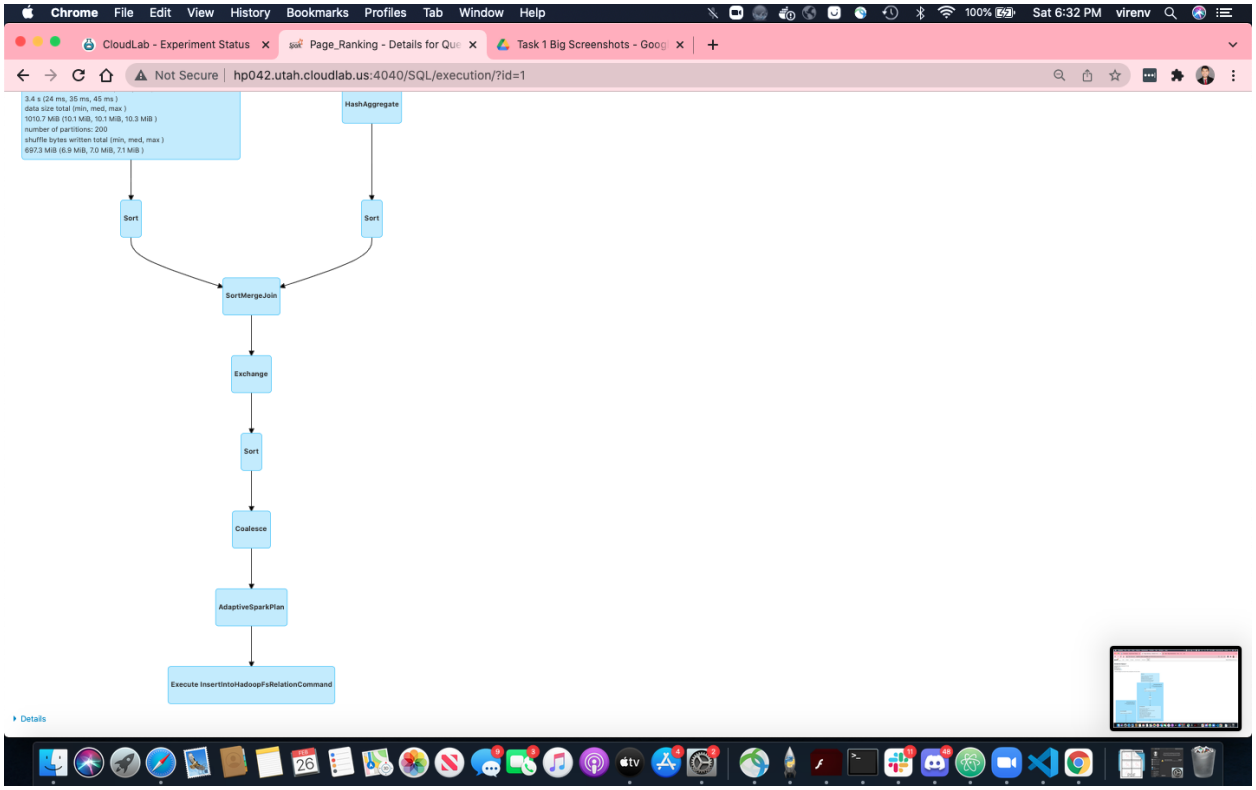
(Application Timeline)

Takeaway: The algorithm runs pretty fast. With that said, as shown by the Spark Jobs timeline, many jobs are taking place. In this case, about 30 jobs were completed by the end of the application's lifetime. Further, it was interesting seeing how the jobs were broken up further into stages. A good example is the DAG visualization for one of the common javaToPython jobs where we can see that it is being broken down into different operations like shuffle read and so forth. The timeline above also shows that there is a pattern of repetition in the jobs being run which is probably representative of the 10 iterations that the algorithm runs for. Overall, the page ranking algorithm for the small dataset took around 1.3 minutes (78 seconds) to complete.

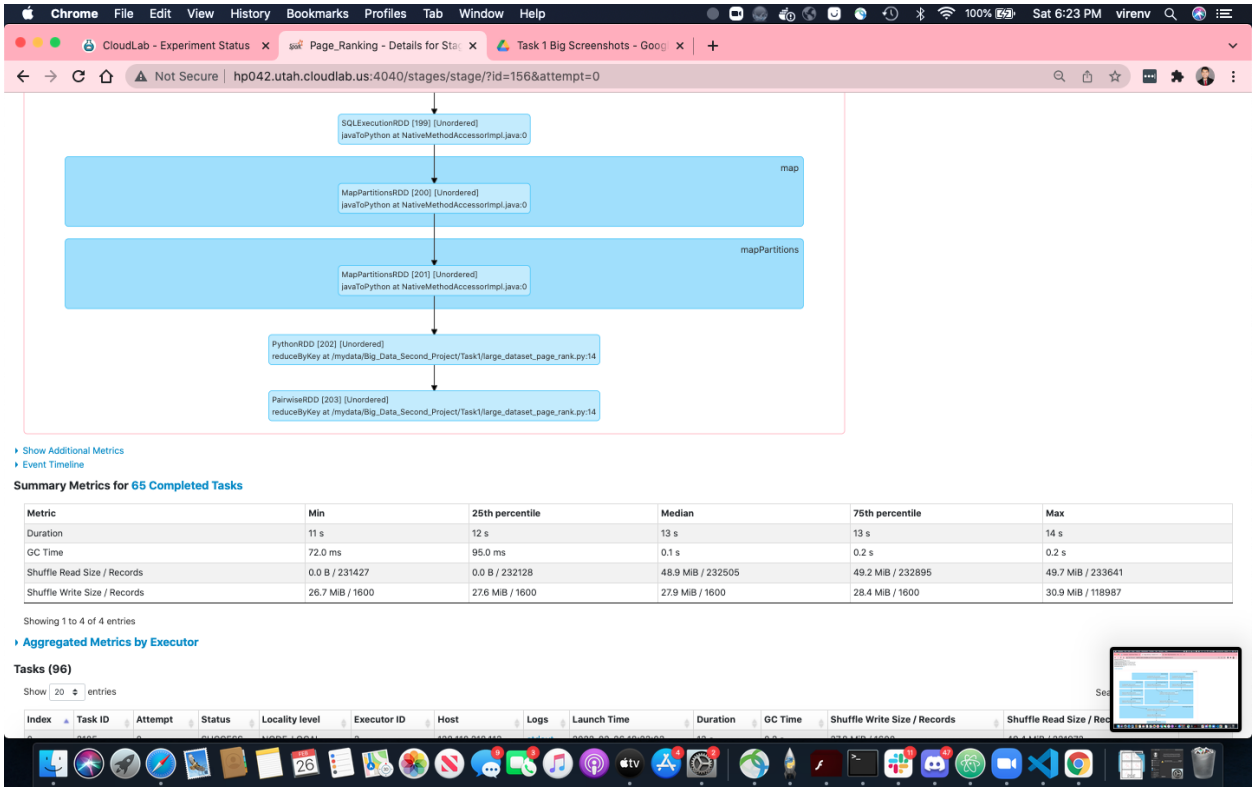
Large Dataset Analysis



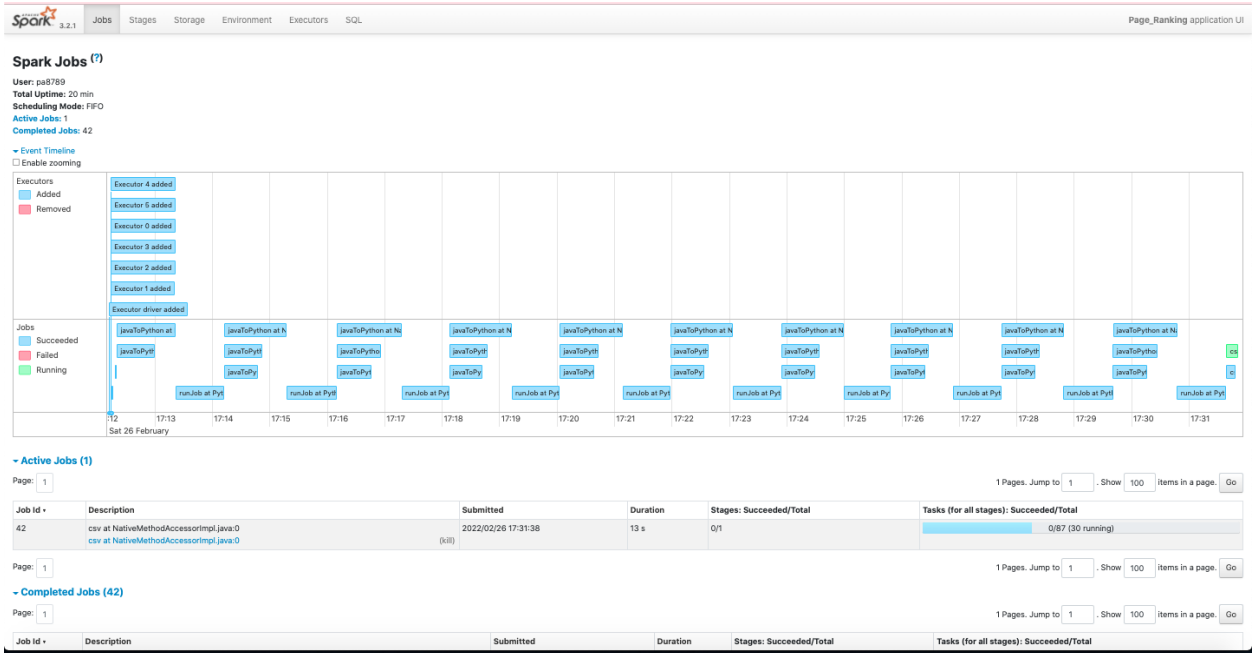
(20 min completion time)



(SQL Execution plan)



(Summary Statistics, ReduceByKey in lineage graph)

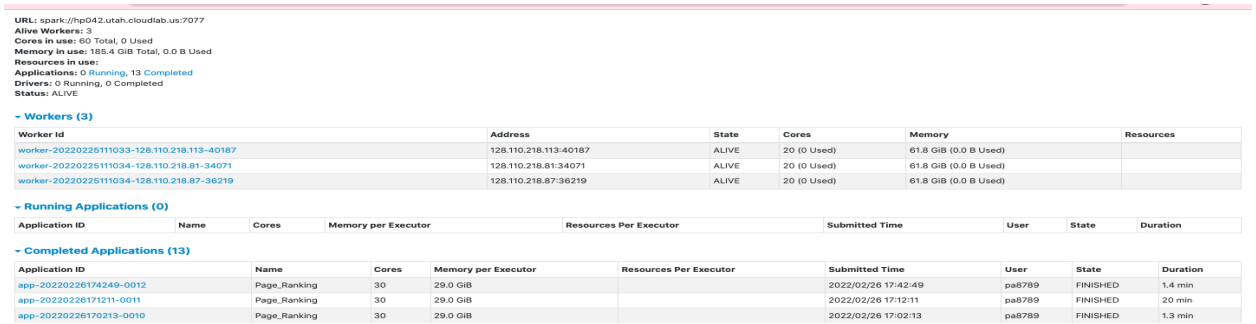


(Timeline)

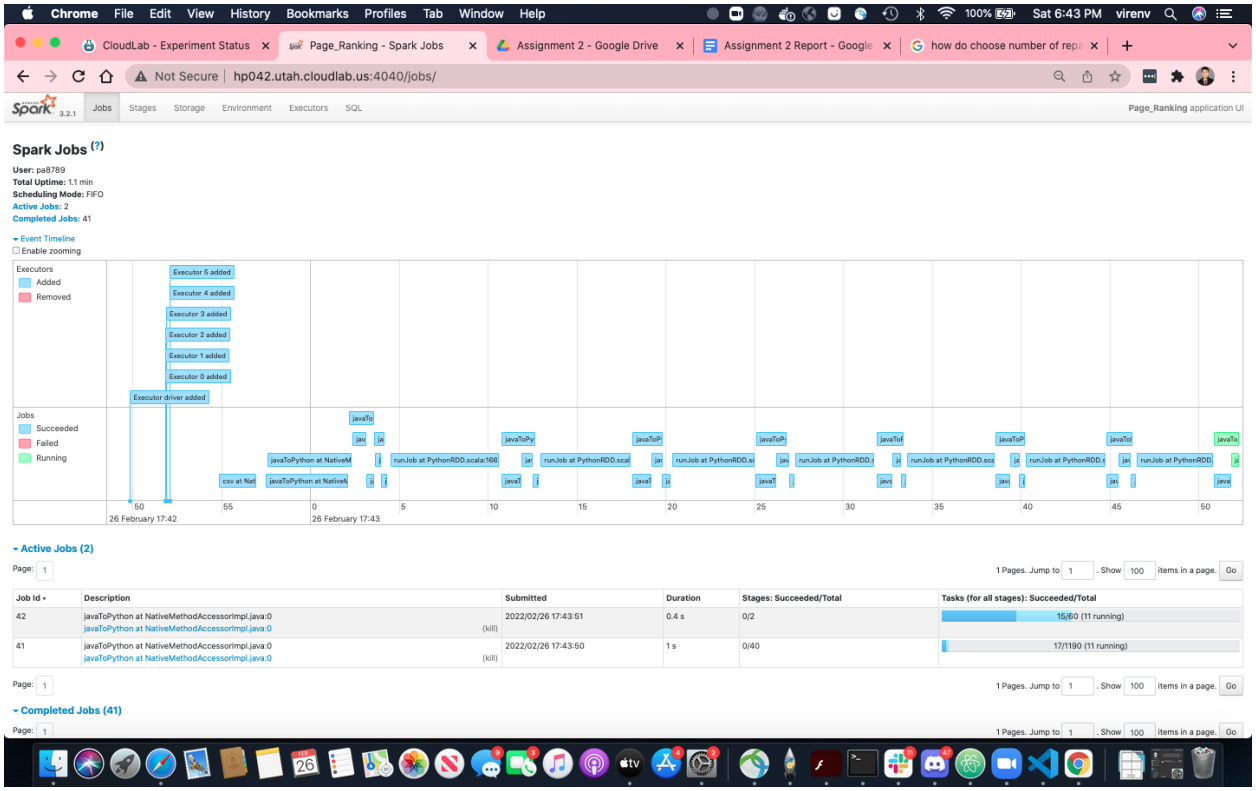
Takeaway: As expected given the large dataset size, the algorithm completed in a much longer period of time, approximately 20 minutes total. We can see that running through this dataset resulted in over 65 completed tasks at one instance in a given job. It is also interesting to visualize the execution plan for what appears to be a single job in the application. We can clearly see various stages in the physical execution plan generated by the query planner, such as the SortMergeJoin algorithm for joining, likely a result of our join query with the links and ranks datasets.

Task 2 Analysis

Small Dataset - 30 Repartitions

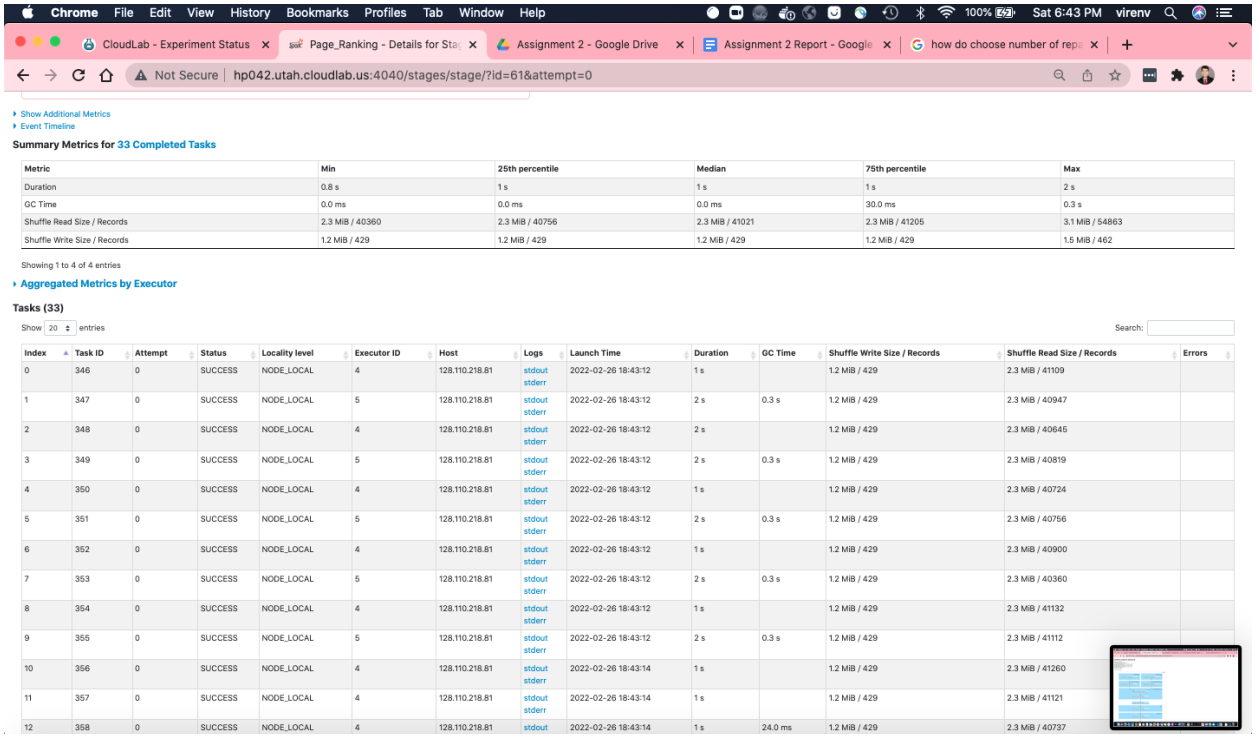


(Completion time 1.4 min, a bit slower than Task 1 small)



(Timeline)

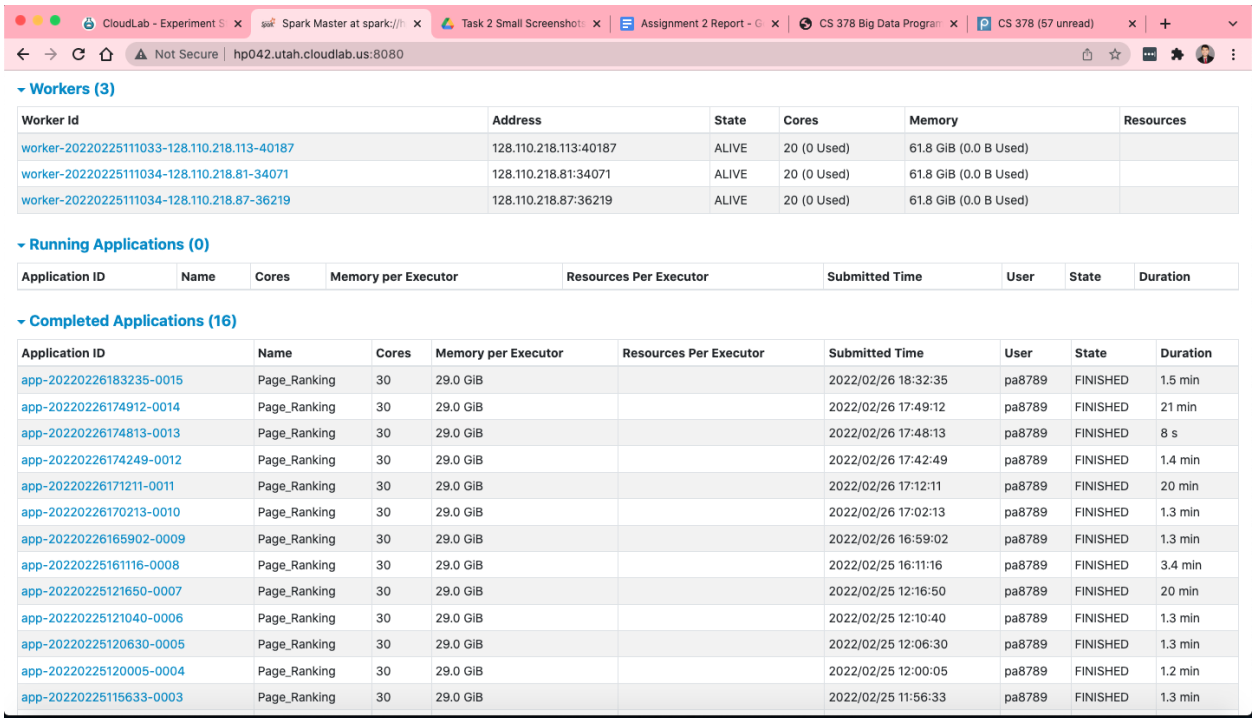
The above timeline shows the numbers of jobs run through at 41 respectively. This is quite a bit more than the previous run of the program with no repartitions.



(Summary statistics for 30 partition small dataset)

Takeaway: Compared to Task 1 where there was no repartitioning, the page ranking algorithm on the small dataset with 30 partitions took longer (1.4 minutes instead of 1.3 or around 6 seconds longer). Looking at the timeline, the overall pattern is still the same as with no repartitioning in terms of having a repeated structure of jobs probably representing the 10 iterations being run. However, the main difference between this run and having no repartitioning is the number of jobs that had to be run. Around the same time, this run had 41 jobs completed whereas the run with no repartitioning had 27 jobs completed. Thus, there were significantly more jobs that ran which we suspect is due to repartitioning and having more partitions meant more jobs and tasks that had to be delegated to handle the increased number of partitions. We also think that the algorithm took a bit longer to run due to the overhead of repartitioning and all the shuffling that it is required to perform.

Small Dataset - 60 Repartitions:



Not Secure | hp042.utah.cloudlab.us:8080

Workers (3)

Worker Id	Address	State	Cores	Memory	Resources
worker-20220225111033-128.110.218.113-40187	128.110.218.113:40187	ALIVE	20 (0 Used)	61.8 GiB (0.0 B Used)	
worker-20220225111034-128.110.218.81-34071	128.110.218.81:34071	ALIVE	20 (0 Used)	61.8 GiB (0.0 B Used)	
worker-20220225111034-128.110.218.87-36219	128.110.218.87:36219	ALIVE	20 (0 Used)	61.8 GiB (0.0 B Used)	

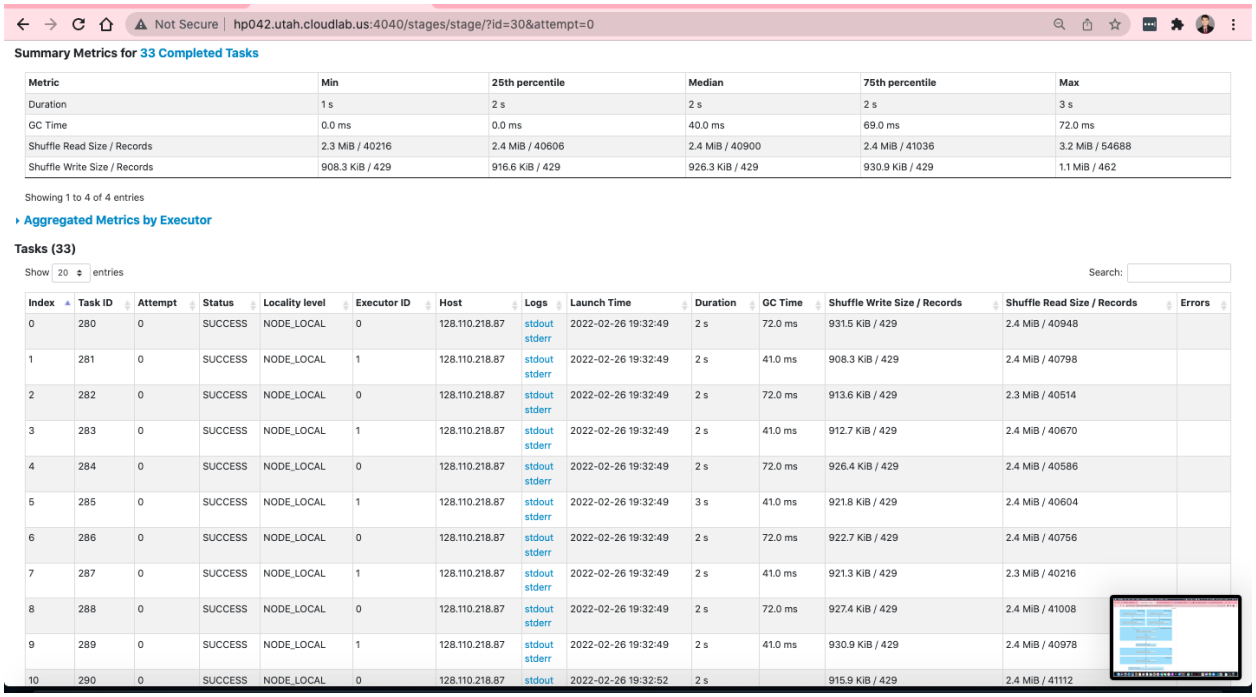
Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration

Completed Applications (16)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20220226183235-0015	Page_Ranking	30	29.0 GiB		2022/02/26 18:32:35	pa8789	FINISHED	1.5 min
app-20220226174912-0014	Page_Ranking	30	29.0 GiB		2022/02/26 17:49:12	pa8789	FINISHED	21 min
app-20220226174813-0013	Page_Ranking	30	29.0 GiB		2022/02/26 17:48:13	pa8789	FINISHED	8 s
app-20220226174249-0012	Page_Ranking	30	29.0 GiB		2022/02/26 17:42:49	pa8789	FINISHED	1.4 min
app-20220226171211-0011	Page_Ranking	30	29.0 GiB		2022/02/26 17:12:11	pa8789	FINISHED	20 min
app-20220226170213-0010	Page_Ranking	30	29.0 GiB		2022/02/26 17:02:13	pa8789	FINISHED	1.3 min
app-20220226165902-0009	Page_Ranking	30	29.0 GiB		2022/02/26 16:59:02	pa8789	FINISHED	1.3 min
app-20220225161116-0008	Page_Ranking	30	29.0 GiB		2022/02/25 16:11:16	pa8789	FINISHED	3.4 min
app-20220225121650-0007	Page_Ranking	30	29.0 GiB		2022/02/25 12:16:50	pa8789	FINISHED	20 min
app-20220225121040-0006	Page_Ranking	30	29.0 GiB		2022/02/25 12:10:40	pa8789	FINISHED	1.3 min
app-20220225120630-0005	Page_Ranking	30	29.0 GiB		2022/02/25 12:06:30	pa8789	FINISHED	1.3 min
app-20220225120005-0004	Page_Ranking	30	29.0 GiB		2022/02/25 12:00:05	pa8789	FINISHED	1.2 min
app-20220225115633-0003	Page_Ranking	30	29.0 GiB		2022/02/25 11:56:33	pa8789	FINISHED	1.3 min

(Completion time for small dataset at 60 repartition, slightly slower still than 30 repartition)



Not Secure | hp042.utah.cloudlab.us:4040/stages/stage/?id=30&attempt=0

Summary Metrics for 33 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	1 s	2 s	2 s	2 s	3 s
GC Time	0.0 ms	0.0 ms	40.0 ms	69.0 ms	72.0 ms
Shuffle Read Size / Records	2.3 MiB / 40216	2.4 MiB / 40606	2.4 MiB / 40900	2.4 MiB / 41036	3.2 MiB / 54688
Shuffle Write Size / Records	908.3 KiB / 429	916.6 KiB / 429	926.3 KiB / 429	930.9 KiB / 429	1.1 MiB / 462

Showing 1 to 4 of 4 entries

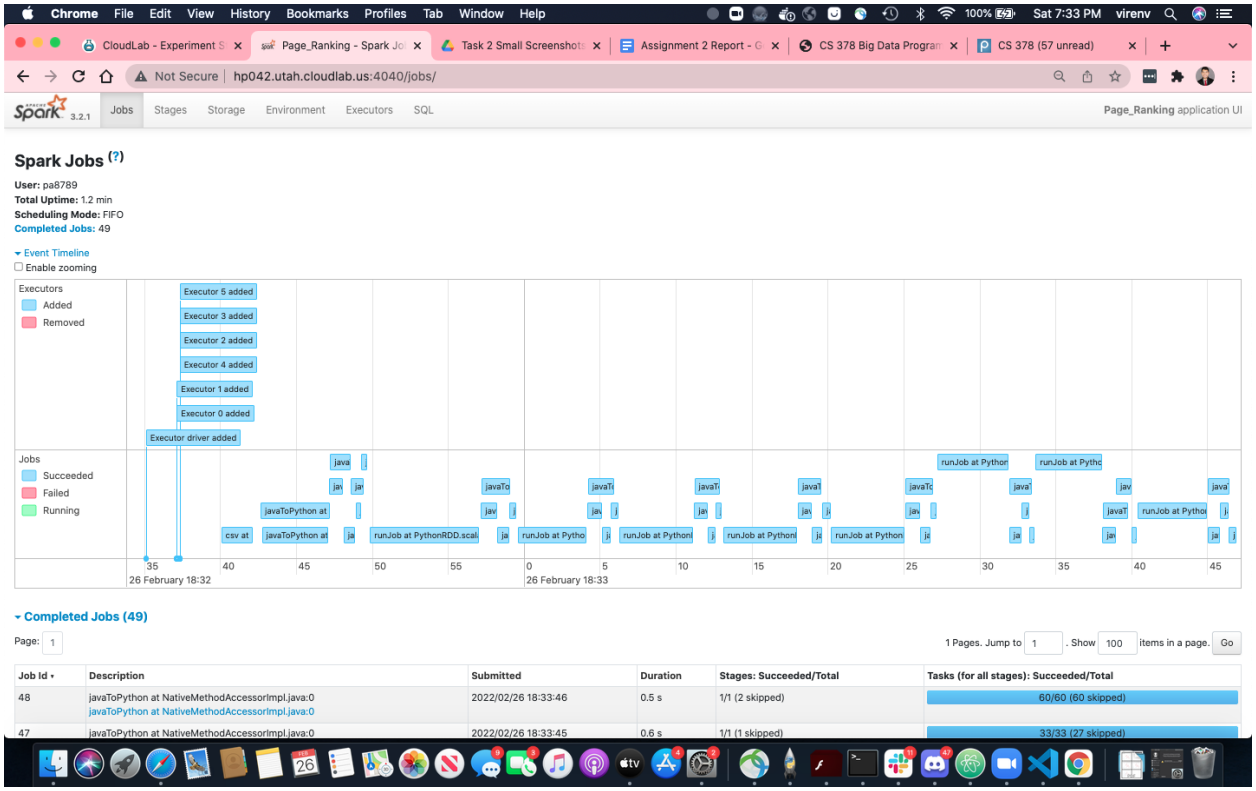
Aggregated Metrics by Executor

Tasks (33)

Show 20 entries Search:

Index	Task ID	Attempt	Status	Locality level	Executor ID	Host	Logs	Launch Time	Duration	GC Time	Shuffle Write Size / Records	Shuffle Read Size / Records	Errors
0	280	0	SUCCESS	NODE_LOCAL	0	128.110.218.87	stdout stderr	2022-02-26 19:32:49	2 s	72.0 ms	931.5 KiB / 429	2.4 MiB / 40948	
1	281	0	SUCCESS	NODE_LOCAL	1	128.110.218.87	stdout stderr	2022-02-26 19:32:49	2 s	41.0 ms	908.3 KiB / 429	2.4 MiB / 40798	
2	282	0	SUCCESS	NODE_LOCAL	0	128.110.218.87	stdout stderr	2022-02-26 19:32:49	2 s	72.0 ms	913.6 KiB / 429	2.3 MiB / 40514	
3	283	0	SUCCESS	NODE_LOCAL	1	128.110.218.87	stdout stderr	2022-02-26 19:32:49	2 s	41.0 ms	912.7 KiB / 429	2.4 MiB / 40670	
4	284	0	SUCCESS	NODE_LOCAL	0	128.110.218.87	stdout stderr	2022-02-26 19:32:49	2 s	72.0 ms	926.4 KiB / 429	2.4 MiB / 40586	
5	285	0	SUCCESS	NODE_LOCAL	1	128.110.218.87	stdout stderr	2022-02-26 19:32:49	3 s	41.0 ms	921.8 KiB / 429	2.4 MiB / 40604	
6	286	0	SUCCESS	NODE_LOCAL	0	128.110.218.87	stdout stderr	2022-02-26 19:32:49	2 s	72.0 ms	922.7 KiB / 429	2.4 MiB / 40756	
7	287	0	SUCCESS	NODE_LOCAL	1	128.110.218.87	stdout stderr	2022-02-26 19:32:49	2 s	41.0 ms	921.3 KiB / 429	2.3 MiB / 40216	
8	288	0	SUCCESS	NODE_LOCAL	0	128.110.218.87	stdout stderr	2022-02-26 19:32:49	2 s	72.0 ms	927.4 KiB / 429	2.4 MiB / 41008	
9	289	0	SUCCESS	NODE_LOCAL	1	128.110.218.87	stdout stderr	2022-02-26 19:32:49	2 s	41.0 ms	930.9 KiB / 429	2.4 MiB / 40978	
10	290	0	SUCCESS	NODE_LOCAL	0	128.110.218.87	stdout	2022-02-26 19:32:52	2 s		915.9 KiB / 429	2.4 MiB / 41112	

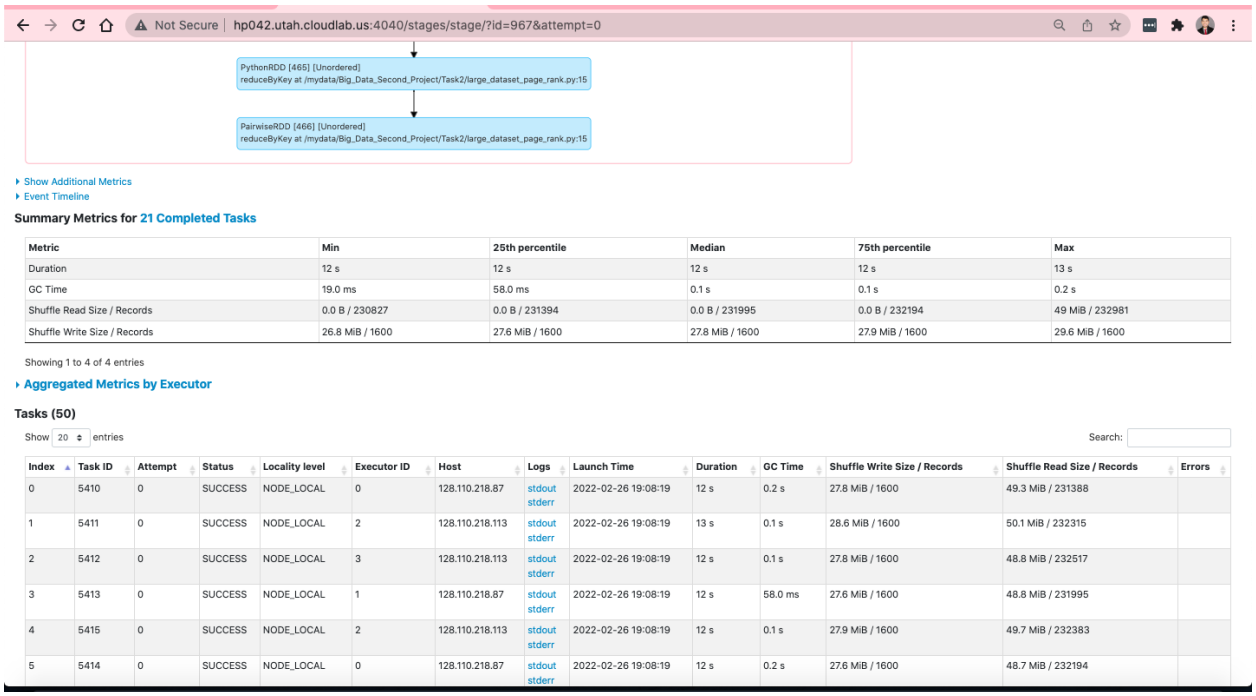
(Summary stats for 60 repartition small dataset)



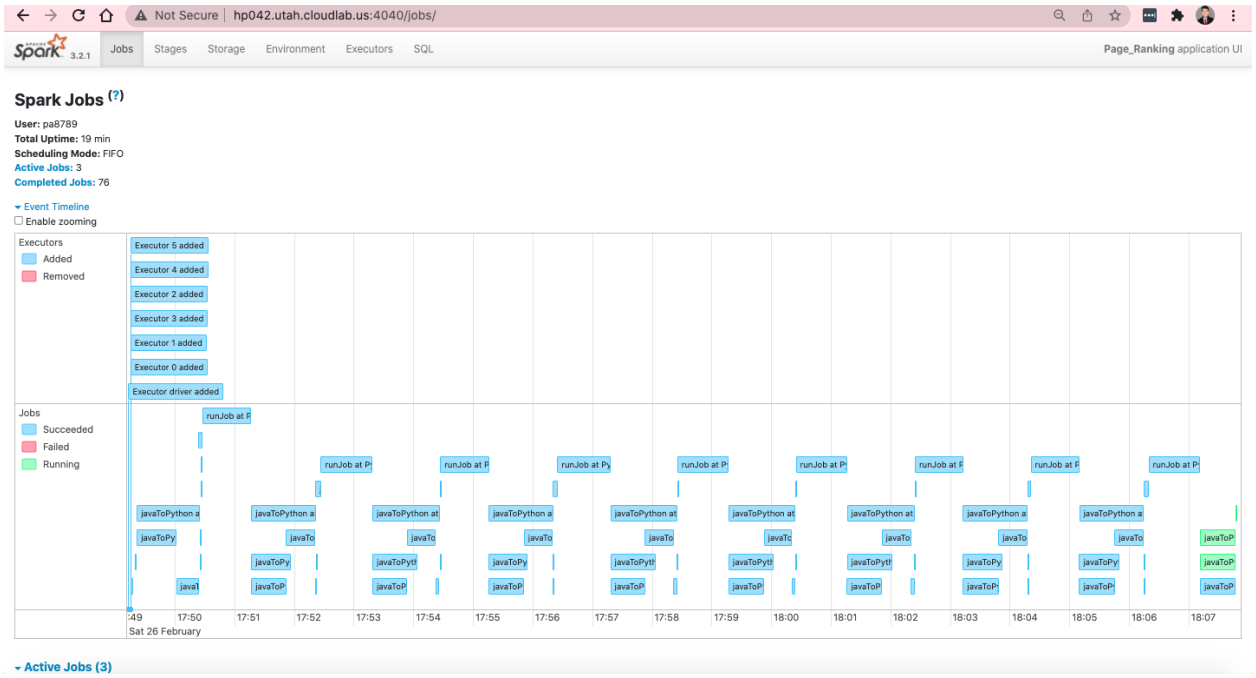
(Timeline)

Takeaway: Doubling the number of repartitions made the application even slower as it finished in 1.5 minutes instead of 1.4 minutes. It appears that adding more and more repartitions, at least in the context of this Page Ranking application, adds overhead that leads to the application running slower than before. Based on the summary metrics, it is apparent that things like garbage collection took more time. Adding more repartitions led to more jobs having to be done as about 50 or so jobs were probably completed by the end of application based on the above timeline.

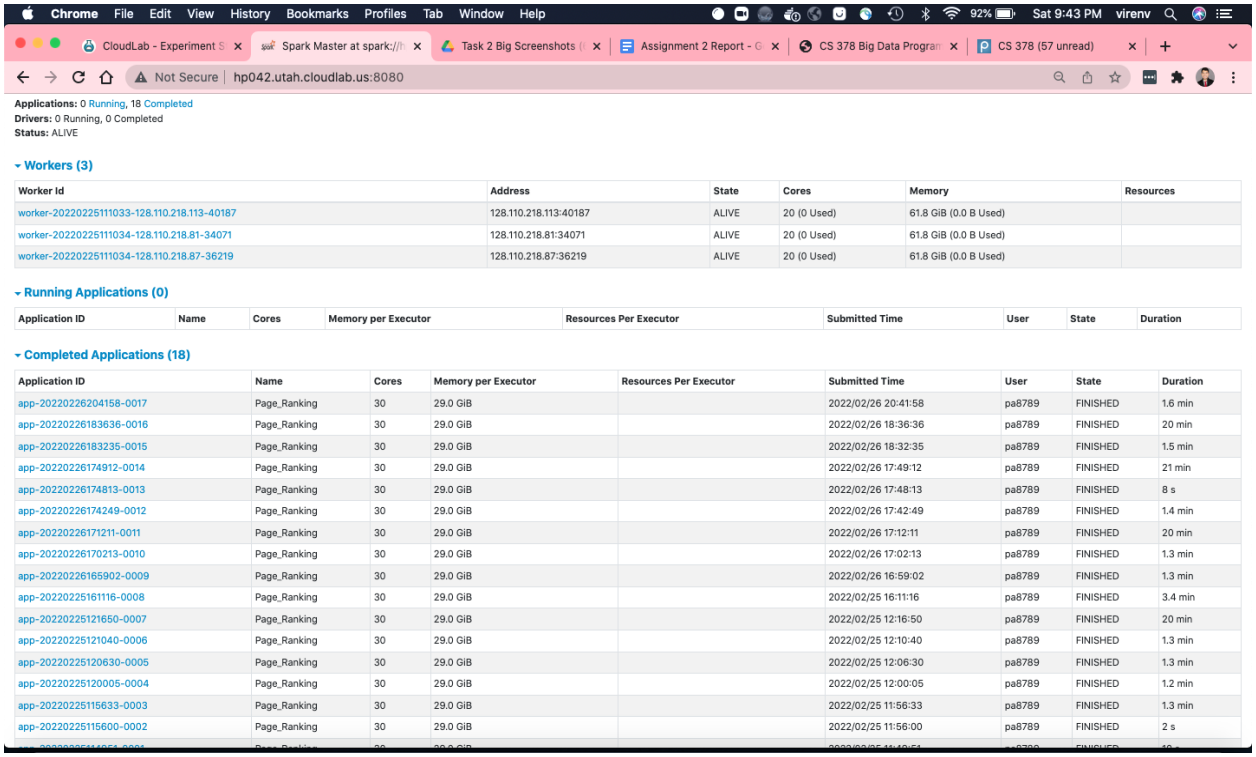
Large Dataset 30 Repartitions



(Summary metrics for large dataset, 30 repartitions: low shuffle read size but large shuffle write size compared to small dataset at 30 partitions. Garbage collection time > small dataset GC time)



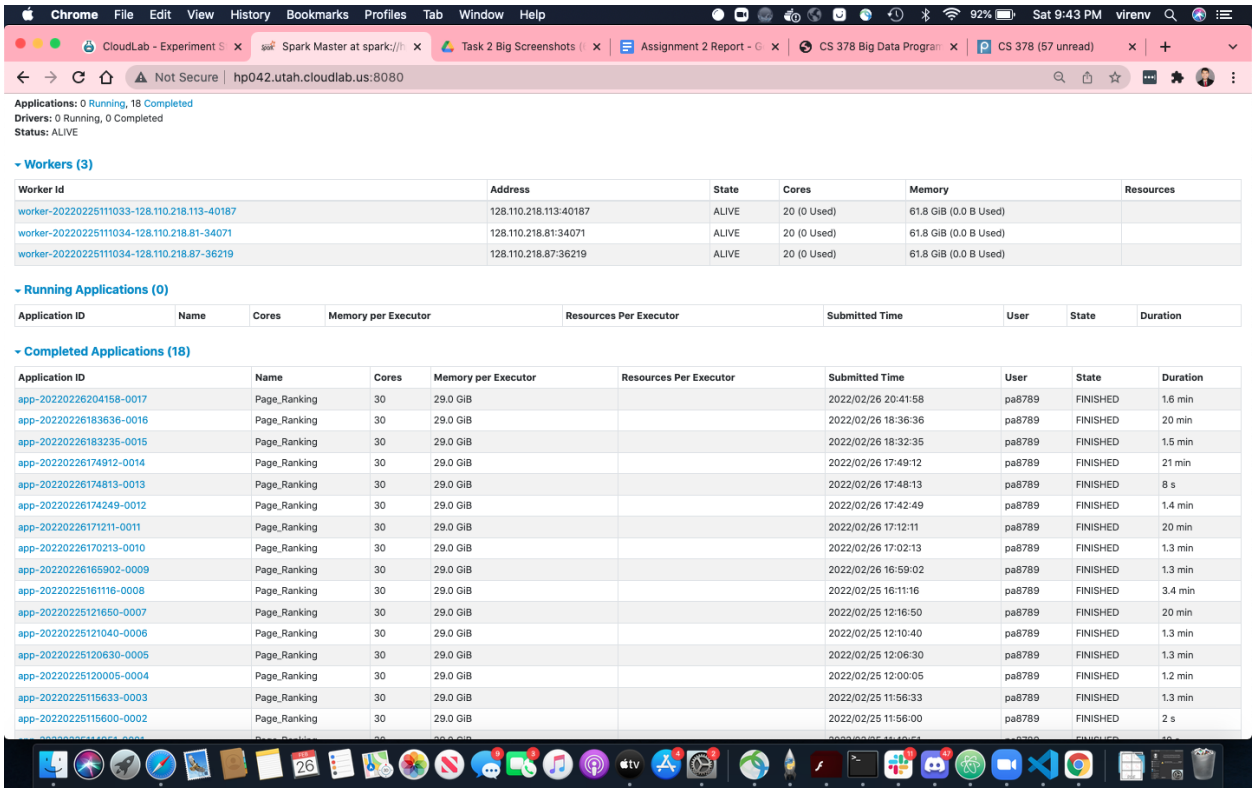
(Job timeline/quantity for Big dataset 30 partition)



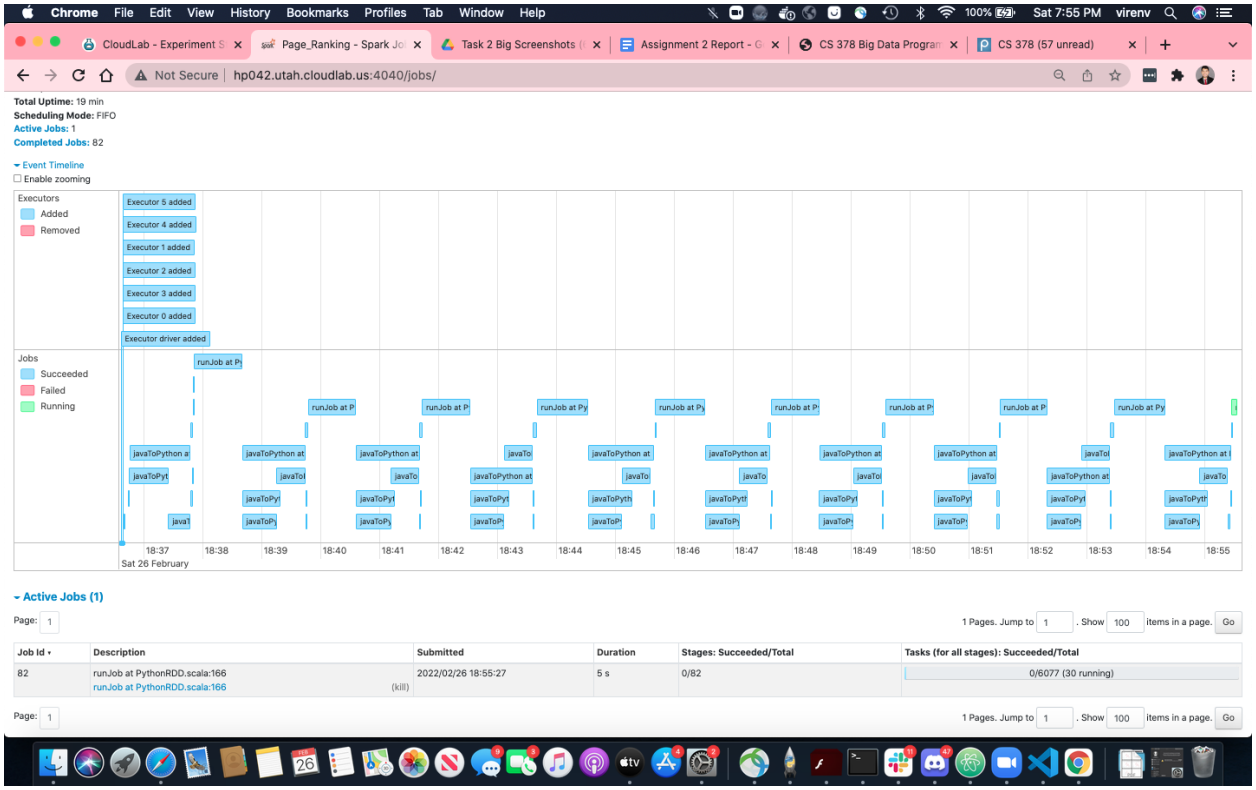
(Total time for application was 21 minutes, so just a minute slower than without any repartitions, but still around the same time)

Takeaway: It seems like the page ranking algorithm repartitioned for 30 partitions run on the large dataset mostly shares the same qualities of the one run on the small dataset with amplified effects. For example, it still shares the quality of taking slightly longer than the original run on the dataset (21 minutes vs 20 minutes so a minute longer). This again is most likely due to the overhead of repartitioning with the shuffling involved. It also has the same pattern in the timeline representing the 10 iterations being run. We also see that it completes significantly more jobs than the original run on the large dataset (around 80 jobs vs 40 jobs so double the amount). Compared to the 30 repartitioning run on the small dataset, we can see that the tasks on average take much longer to run (around 12 seconds each versus 1 second each) and the GC time is also much longer on average (100s of ms versus 10s of ms). However, the shuffle read size / records is much lower (close to 0 B vs a consistent 2.3 MiB) while the shuffle write size / records is much higher (28 MiB versus 1.2 MiB).

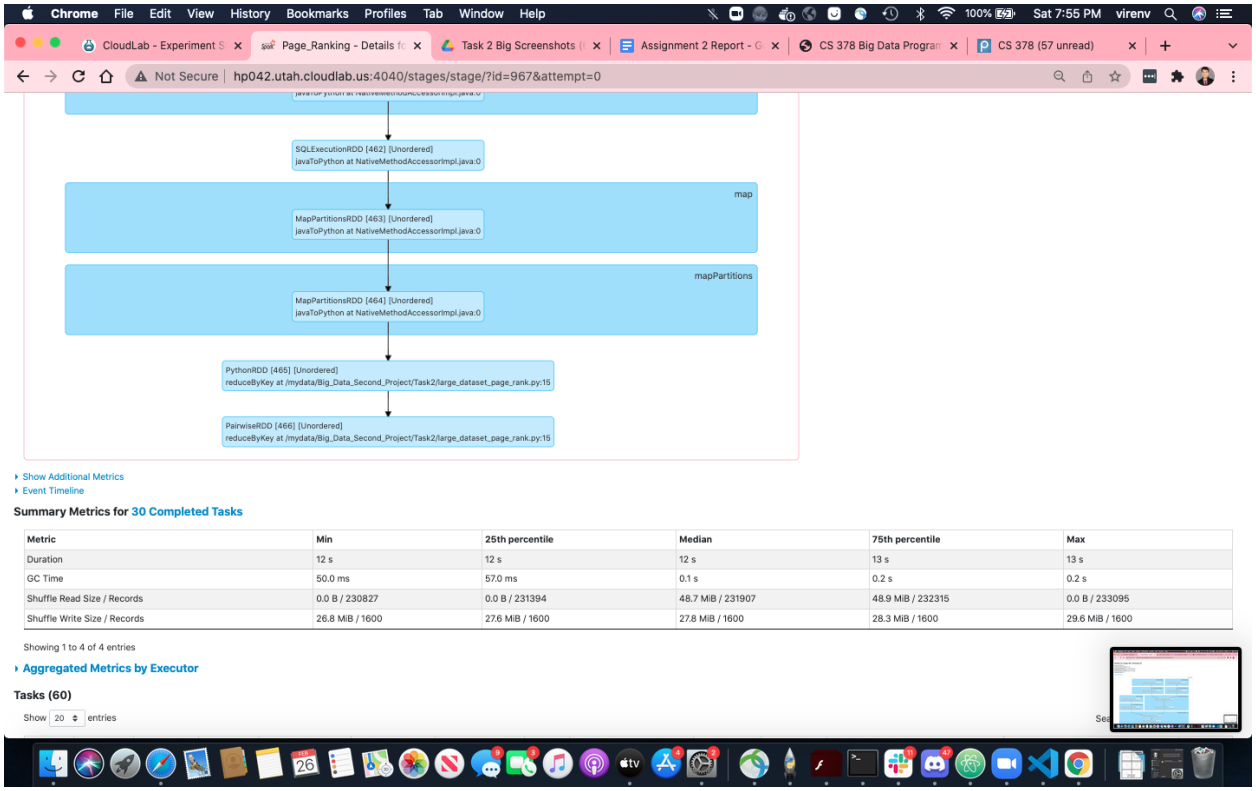
Large Dataset 60 Repartitions



(Total time for application was 20 minutes, so a little better than one with 30 repartitions, but essentially the same as application without any repartitions. Goes to show that repartitions for this kind of application give back pretty much the same or slightly worse results)



(Total number of jobs for Application)

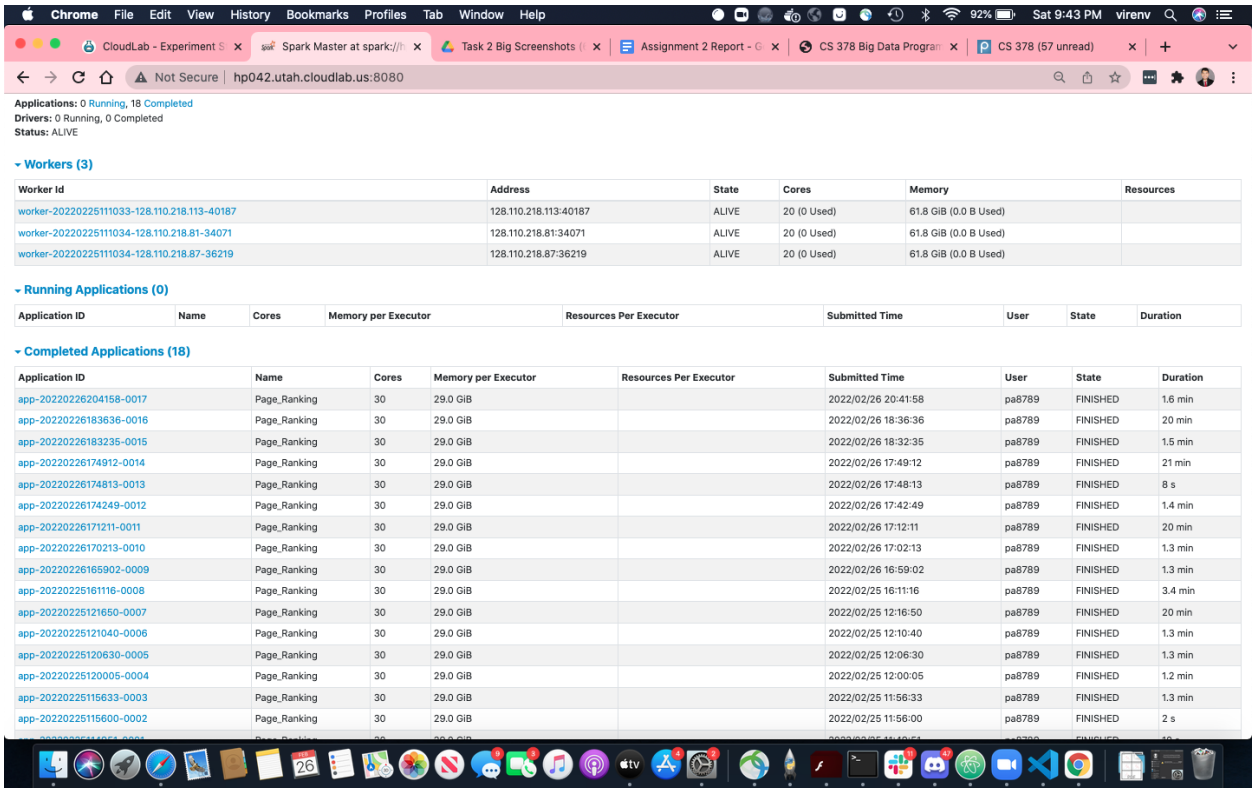


(Lineage graph and summary stats)

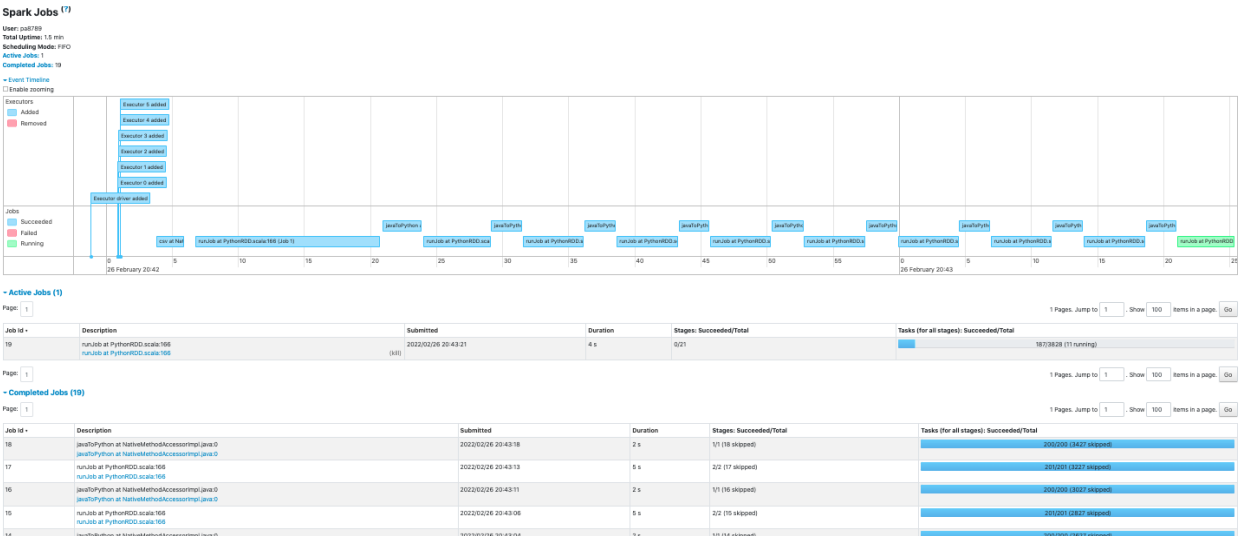
Takeaway: We observed that despite doubling the number of partitions, the completion time for PageRank with the large dataset was not significantly affected. As a matter of fact, it was rather similar to the time from Task 1 at around 20 minutes, about 5% faster than the completion time with 30 partitions. The network shuffling overhead that results from partitioning could have been slightly negated by simply having more partitions across more nodes at work, which could subsequently lead to a greater leveraging of parallelism to compensate for network latency. We see that 82 jobs were completed with this particular run, slightly more than the 76 jobs with 30 partitions and almost double the job count for the large dataset in Task 1. The average completion time for each job is relatively similar to running the large dataset with 30 partitions at 12 seconds on average.

Task 3:

Small Dataset Analysis



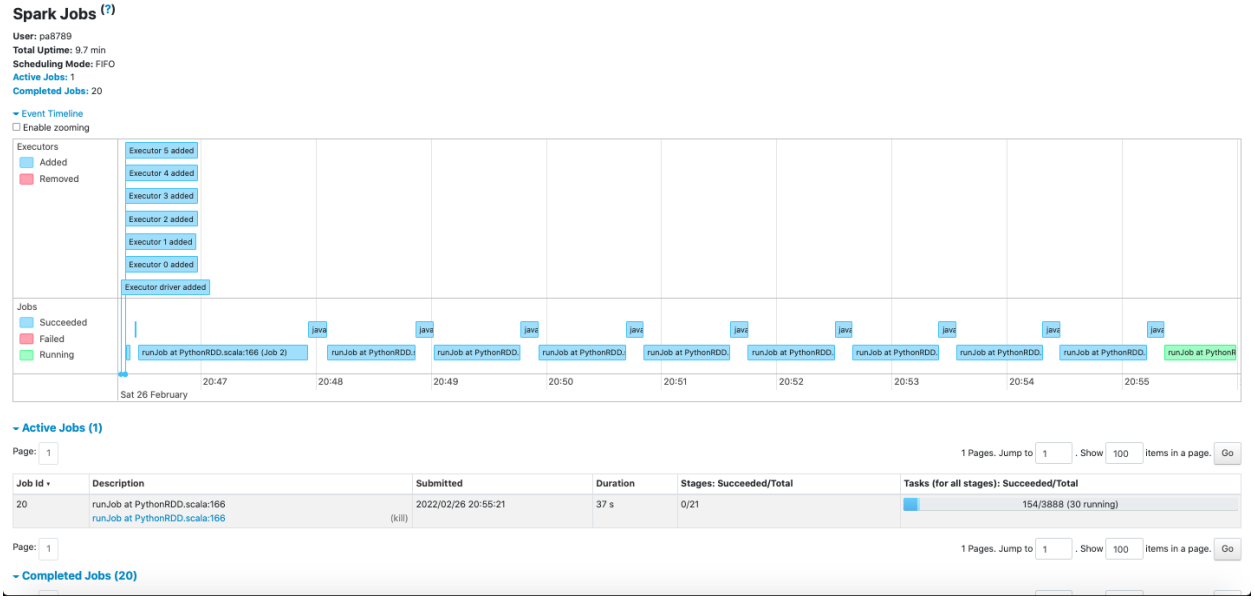
(Completion Time)



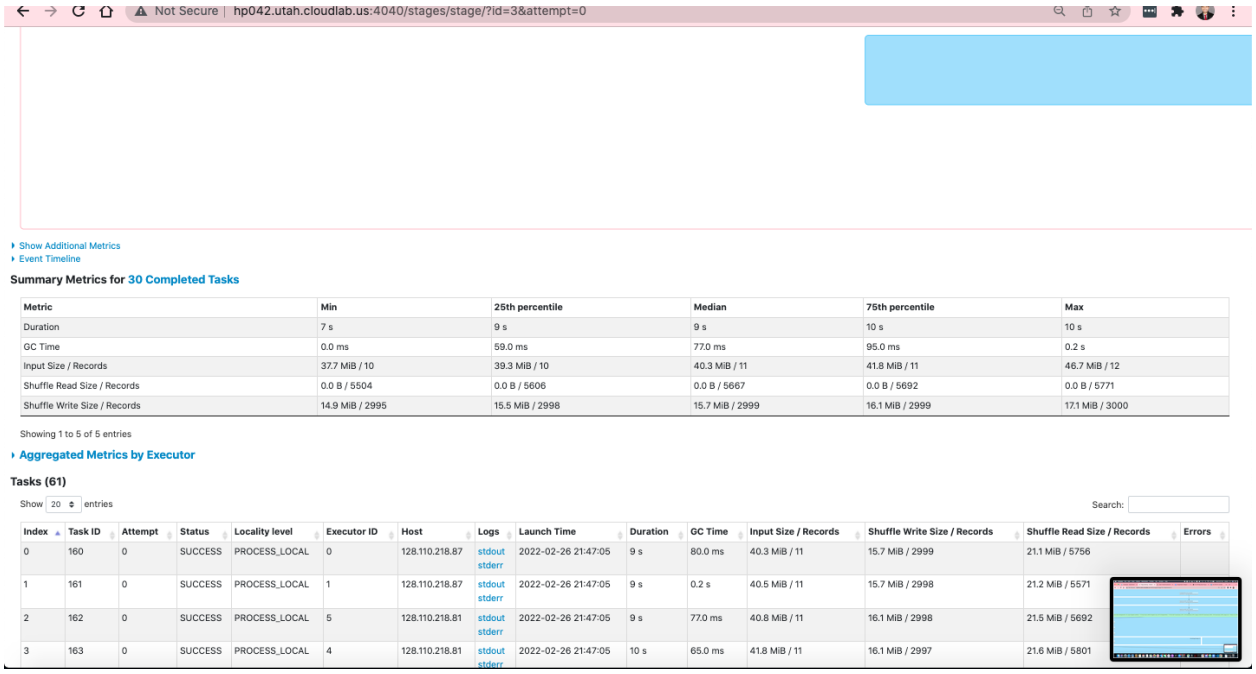
(Job/Application Timeline)

Takeaway: It appears that even though we were caching/persisting, the dataset is so small that it isn't worth doing so as it makes the application run slower at 1.6 minutes as opposed to 1.3. The number of jobs run though was half of before as only 15 jobs were needed to be completed as opposed to around 30 with no caching/persisting. The degradation in completion time could be due to some additional overhead caused by creating and maintaining the links dataset in cached copies.

Big Dataset Analysis



(Half the jobs as without persisting. Forty originally, now only 20 completed jobs, job completion time was recorded as 10 minutes, half of the time as without the cache)

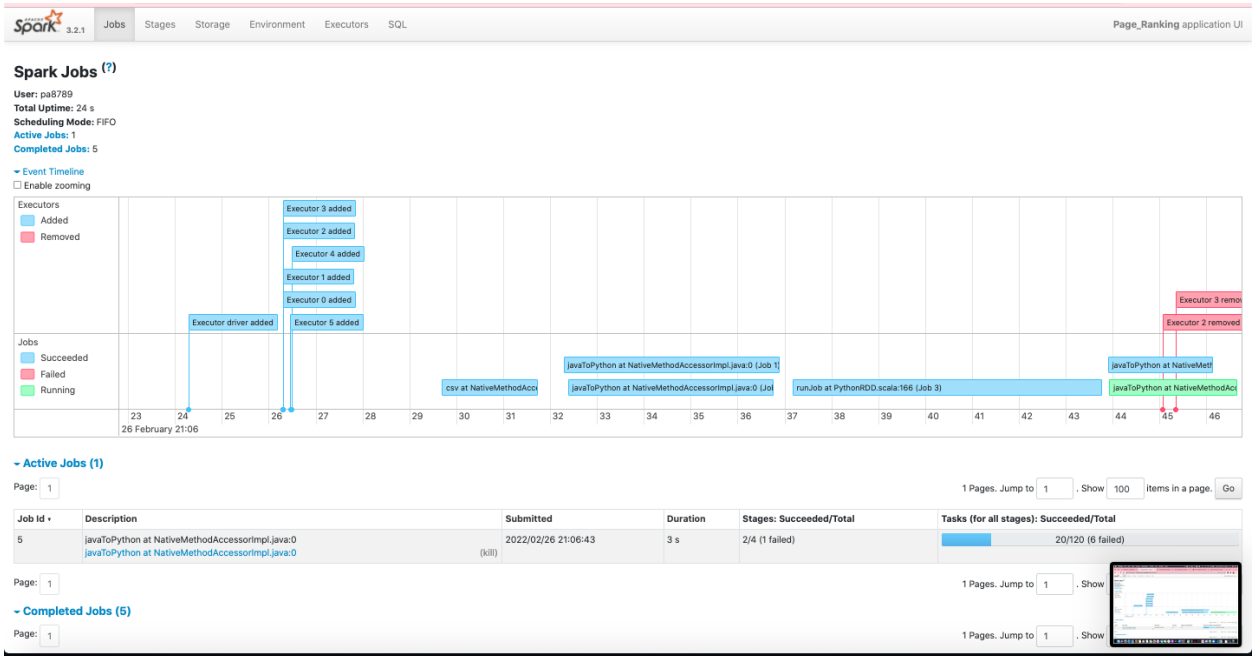


(summary metrics for big dataset after persisting links)

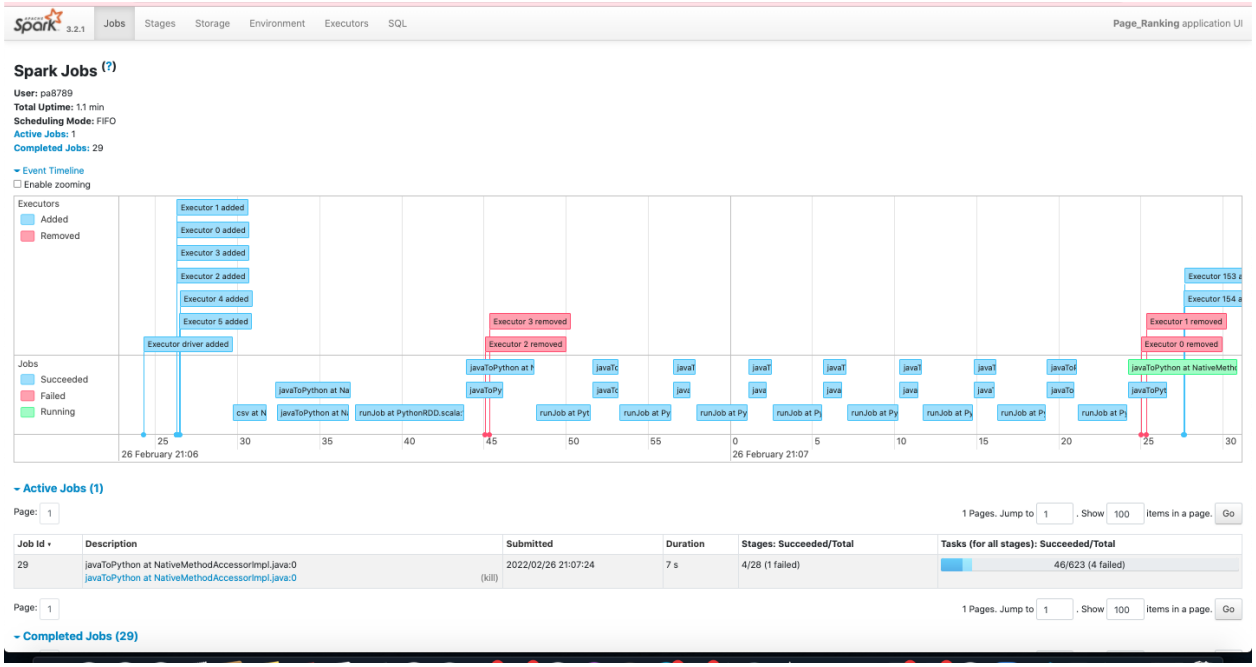
Takeaway: We see the major difference caching/persisting has on a large dataset such as this one. It only took 10 minutes for the application to run to completion as opposed to 20 minutes, a 50% reduction. The number of jobs also run was about half as before as only 20 jobs were needed to be completed. This could have been the result of some computations or HDFS reads that were eliminated along the way as a result of caching, so any part of the lineage that was originally cached could have been quickly accessed if any recomputation was needed.

Task 4

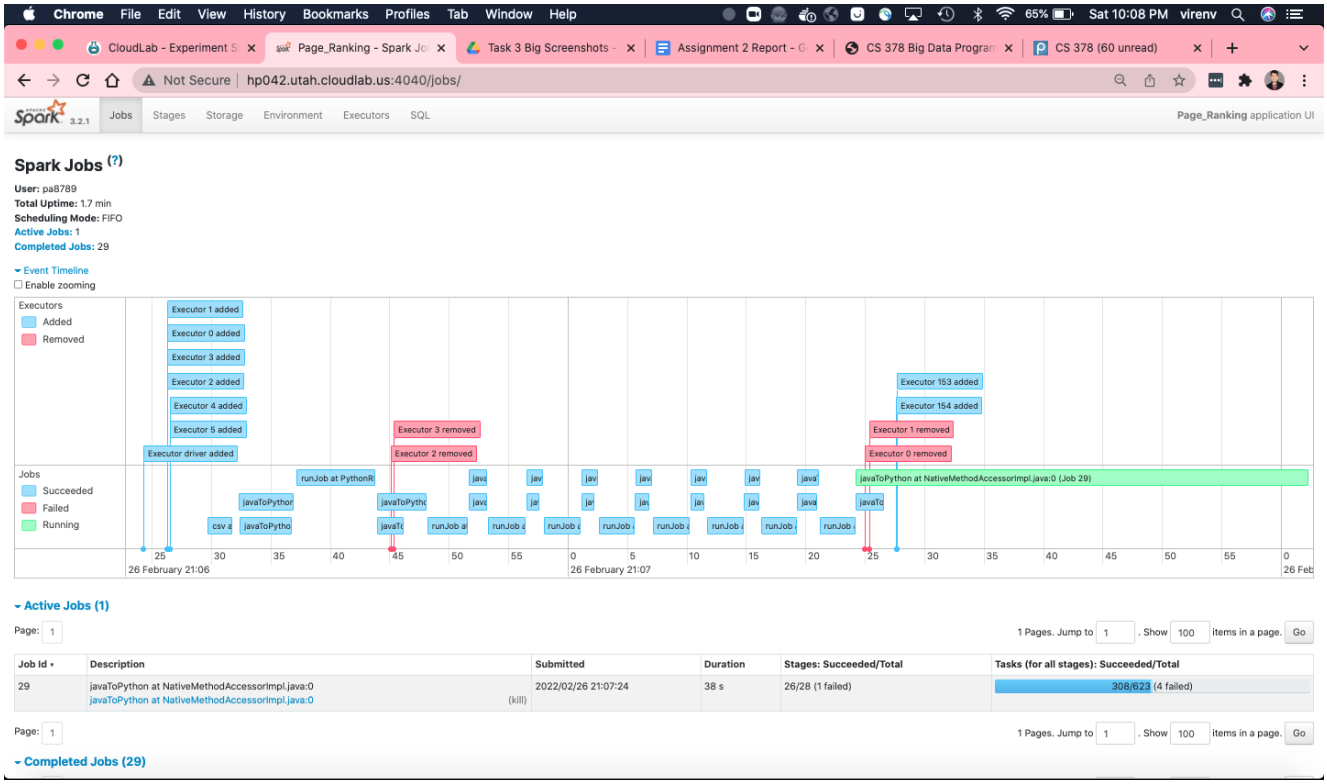
Small Dataset Analysis



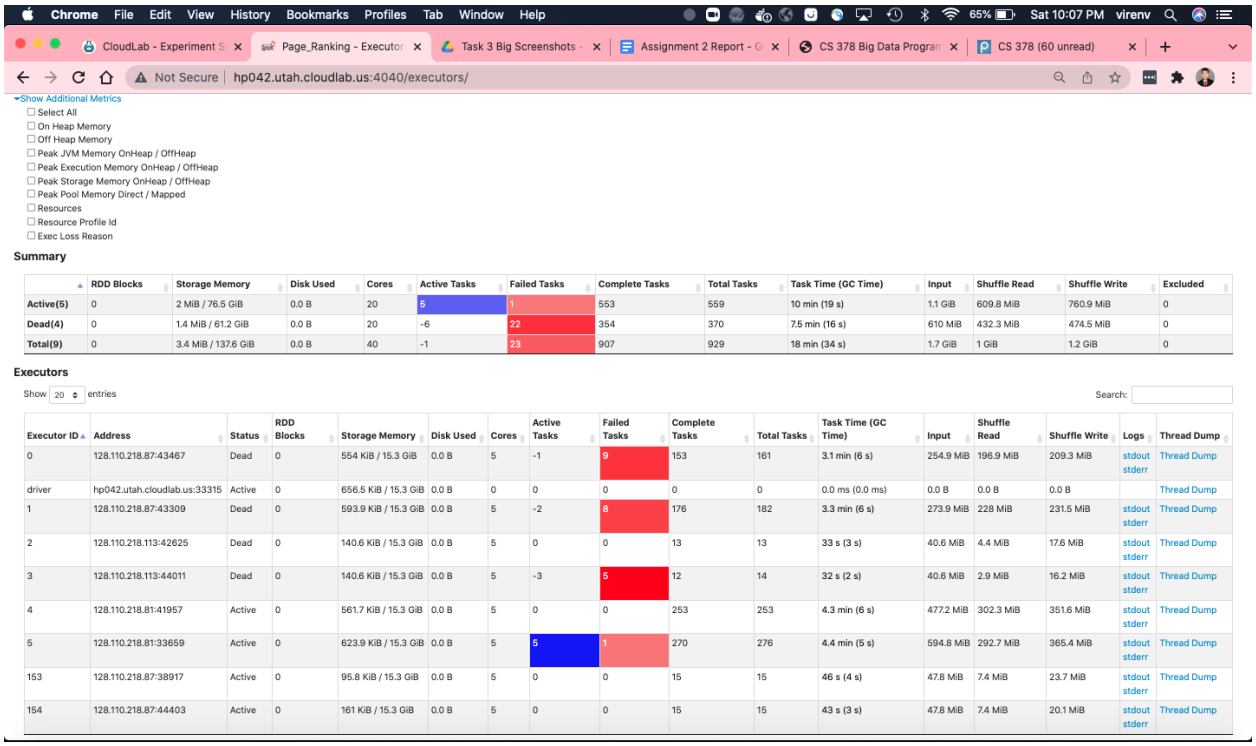
(Timeline showing worker killed at 25% mark of application (20 seconds))



(Timeline showing worker killed at 75% mark of application (1 minute))



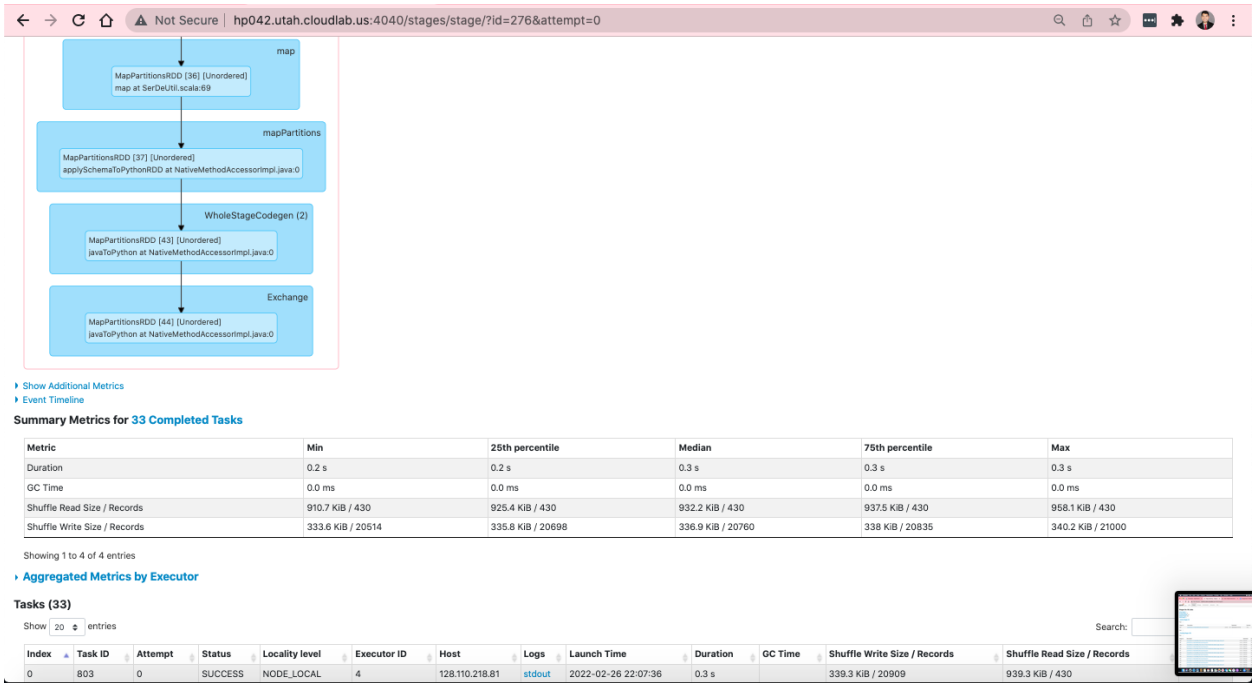
(Can clearly see the killed workers highlighted in red, note the increased total time. Overall completion time ended up being 1.8 minutes)



	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Excluded
Active(5)	0	2 MiB / 76.5 GiB	0.0 B	20	5	1	553	559	10 min (19 s)	1.1 GiB	609.8 MiB	760.9 MiB	0
Dead(4)	0	1.4 MiB / 61.2 GiB	0.0 B	20	-6	22	354	370	7.5 min (16 s)	610 MiB	432.3 MiB	474.5 MiB	0
Total(9)	0	3.4 MiB / 137.6 GiB	0.0 B	40	-1	23	907	929	18 min (34 s)	1.7 GiB	1 GiB	1.2 GiB	0

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump
0	128.110.218.87:43467	Dead	0	554 KiB / 15.3 GiB	0.0 B	5	-1	9	153	161	3.1 min (6 s)	254.9 MiB	196.9 MiB	209.3 MiB	stdout stderr	Thread Dump
driver	hp042.utah.cloudlab.us:33315	Active	0	656.5 KiB / 15.3 GiB	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B		Thread Dump
1	128.110.218.87:43309	Dead	0	593.9 KiB / 15.3 GiB	0.0 B	5	-2	8	176	182	3.3 min (6 s)	273.9 MiB	228 MiB	231.5 MiB	stdout stderr	Thread Dump
2	128.110.218.113:42625	Dead	0	140.6 KiB / 15.3 GiB	0.0 B	5	0	0	13	13	33 s (3 s)	40.6 MiB	4.4 MiB	17.6 MiB	stdout stderr	Thread Dump
3	128.110.218.113:44011	Dead	0	140.6 KiB / 15.3 GiB	0.0 B	5	-3	5	12	14	32 s (2 s)	40.6 MiB	2.9 MiB	16.2 MiB	stdout stderr	Thread Dump
4	128.110.218.81:41957	Active	0	561.7 KiB / 15.3 GiB	0.0 B	5	0	0	253	253	4.3 min (6 s)	477.2 MiB	302.3 MiB	351.6 MiB	stdout stderr	Thread Dump
5	128.110.218.81:33659	Active	0	623.9 KiB / 15.3 GiB	0.0 B	5	5	1	270	276	4.4 min (5 s)	594.8 MiB	292.7 MiB	365.4 MiB	stdout stderr	Thread Dump
153	128.110.218.87:38917	Active	0	95.8 KiB / 15.3 GiB	0.0 B	5	0	0	15	15	46 s (4 s)	47.8 MiB	7.4 MiB	23.7 MiB	stdout stderr	Thread Dump
154	128.110.218.87:44403	Active	0	161 KiB / 15.3 GiB	0.0 B	5	0	0	15	15	43 s (3 s)	47.8 MiB	7.4 MiB	20.1 MiB	stdout stderr	Thread Dump

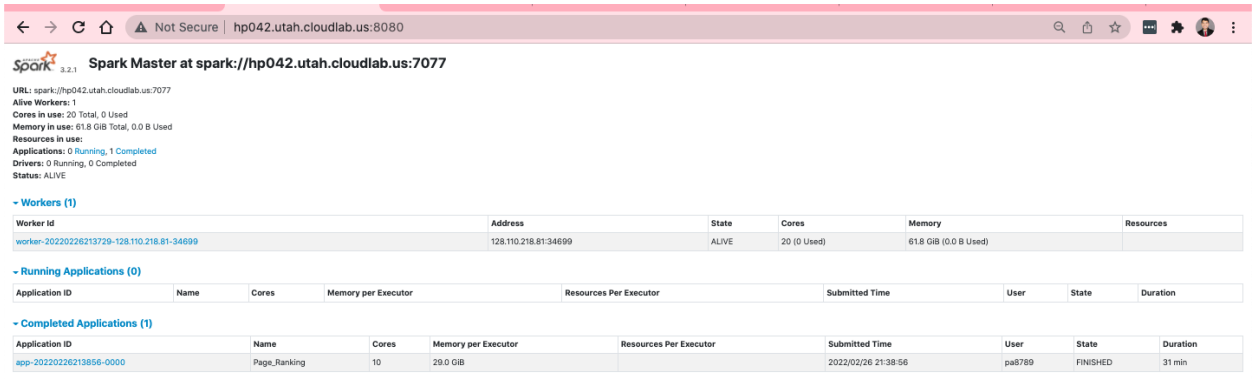
(Executor Statistics)



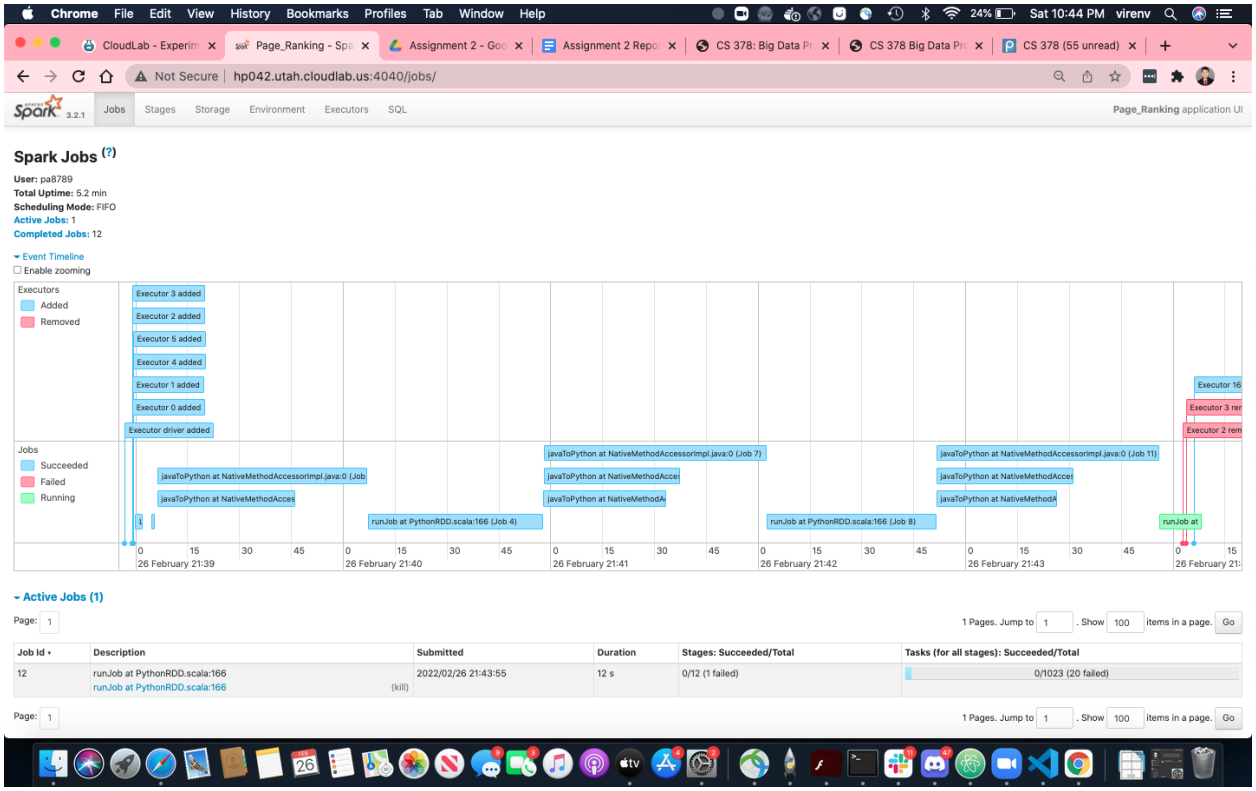
(Summary statistics for small dataset)

Takeaway: It is not a surprise at all that killing a worker at both 25% and 75% lifetime mark of application would lead to a much slower runtime. We can see the effect in the two timelines when a worker was killed as shown by the removal of a couple of the executors in red. This led to some failed tasks which can be seen in the Executor statistics screenshot. It took the application 1.8 minutes to reach completion which is about a 38% increase as the baseline application from Task 1 only took 1.3 minutes. The number of jobs was the same as the baseline application, with just a longer completion time as expected.

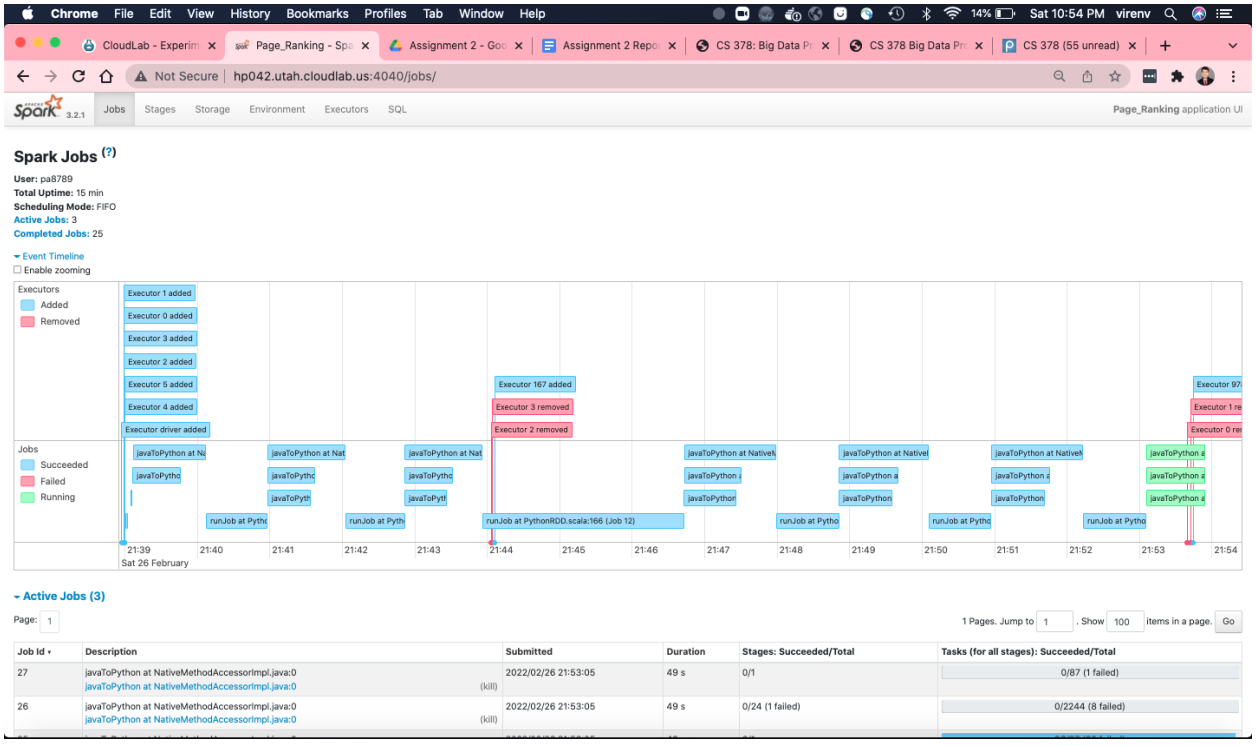
Big Dataset Analysis



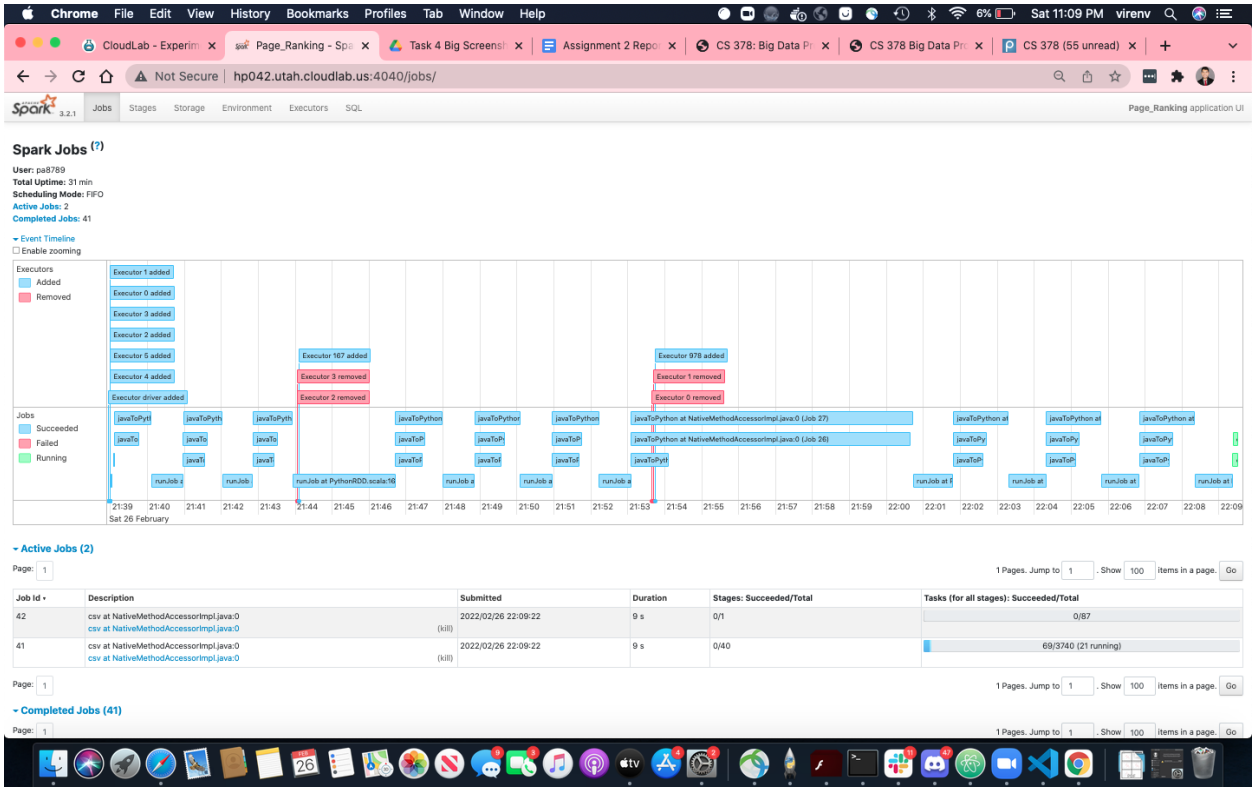
(Completion time of 31 minutes)



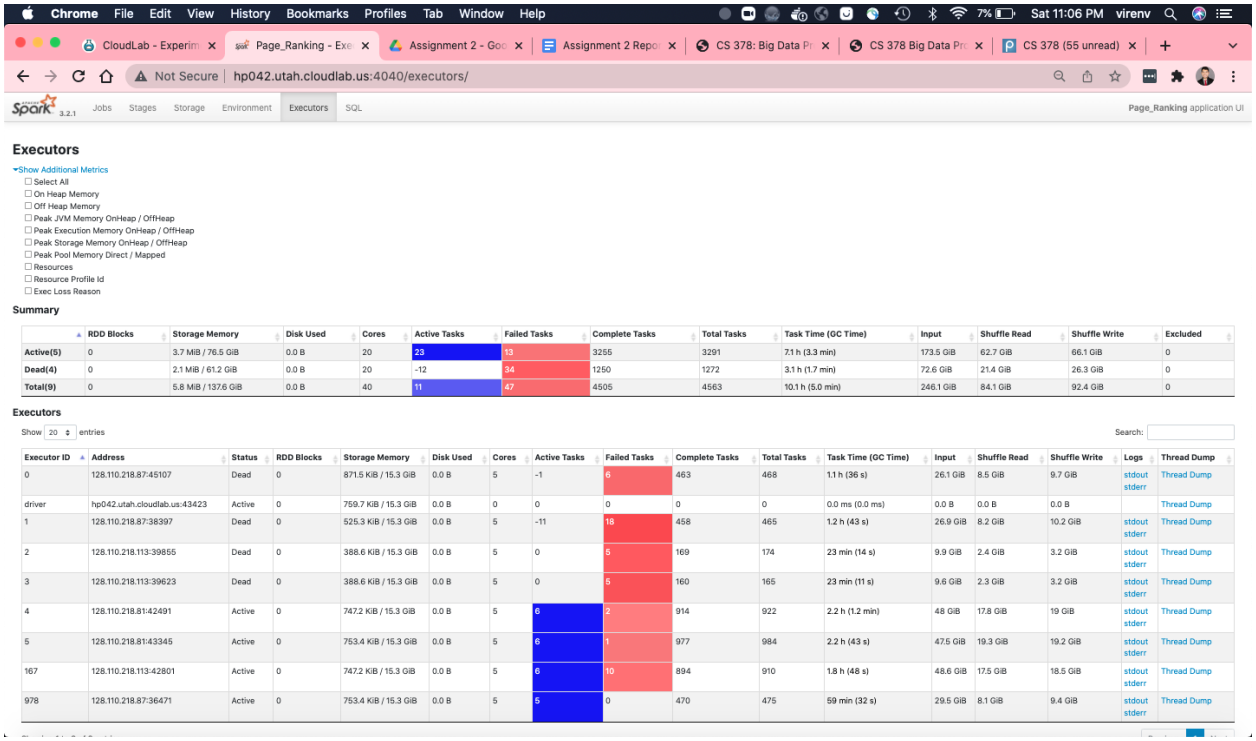
(Timeline at 5 min, killed worker shown)



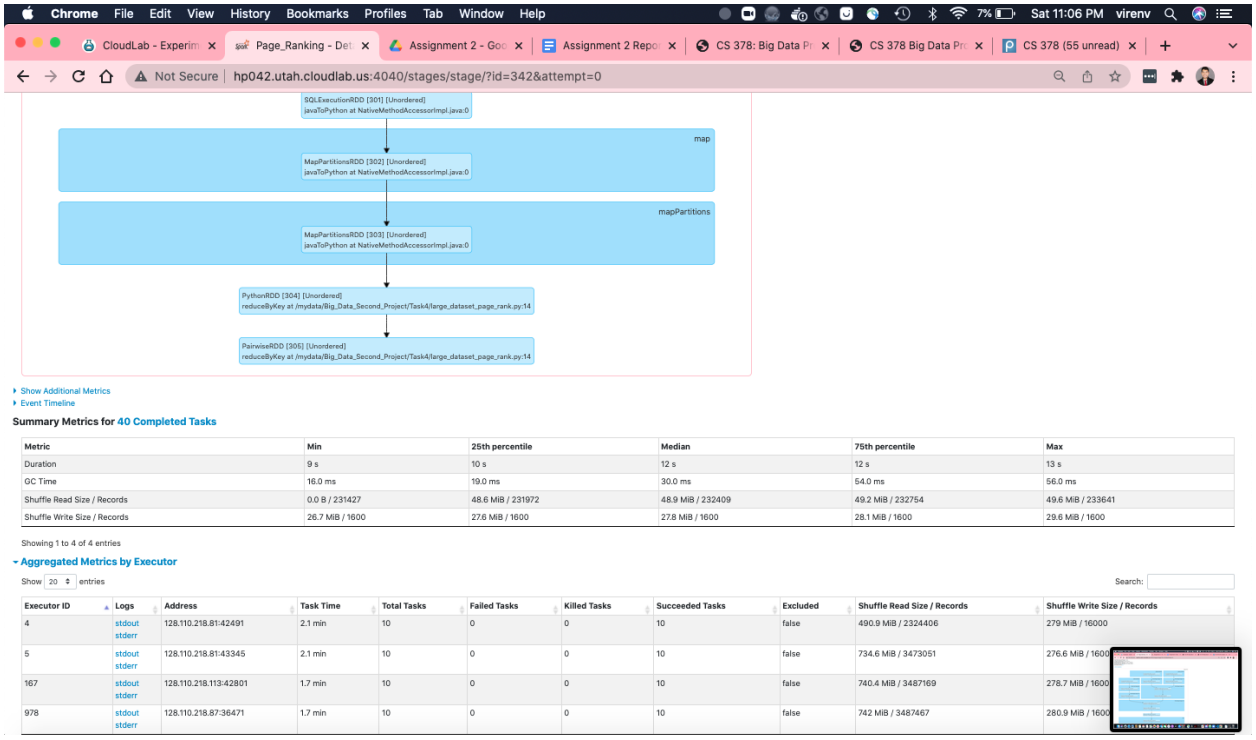
(Timeline at 15 minutes when second worker is killed)



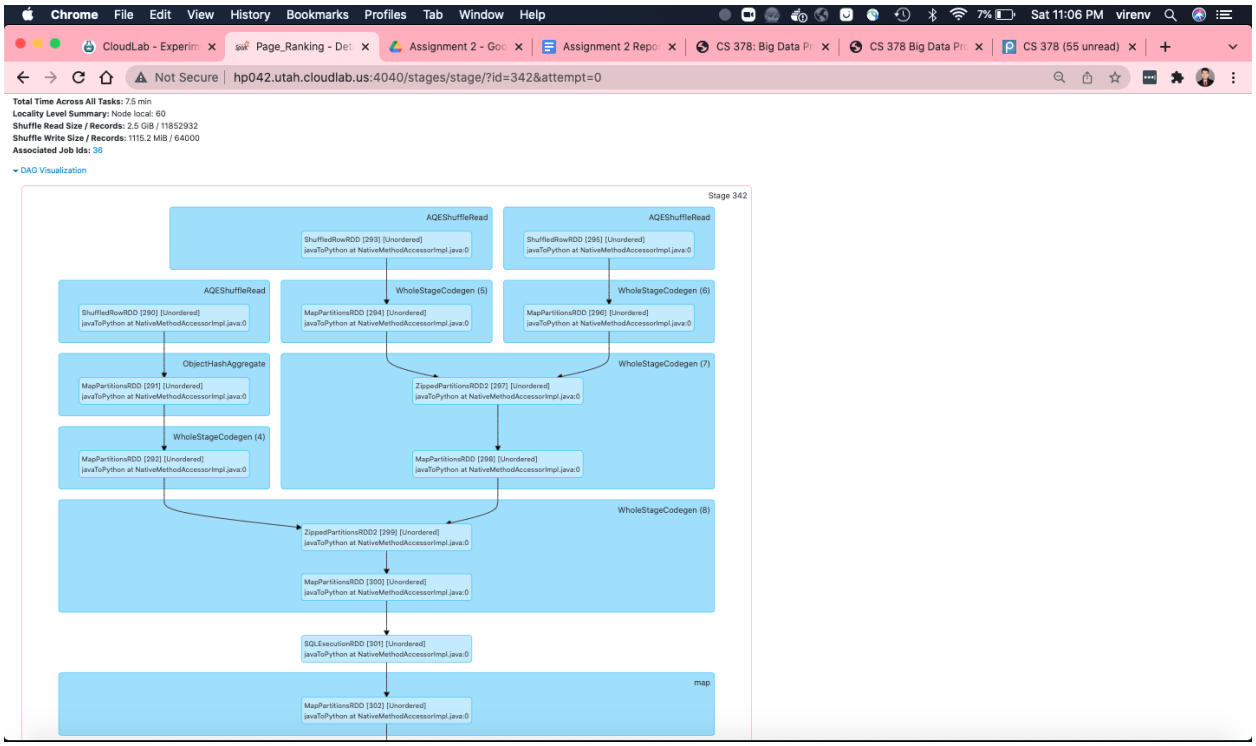
(Overall Timeline)



(Executor Statistics)



(Summary Statistics)



(Interesting Lineage graph right after 25% mark)

Takeaway: Like for the small dataset, the loss of workers led to a larger application runtime. However, for the large dataset it was even worse as the application completion time was 31 minutes, a 55% increase compared to the baseline. As seen from the executor statistics, there were even more failed tasks that took place. The jobs themselves seemed larger and more complex as seen by the above DAG visualization for one of the jobs. Like for the small dataset, the number of jobs completed was the same as the baseline application at a little over 40 jobs, but it just took longer.

Specific Contributions of each Group Member

Note: In general, all of the members worked together on all aspects and would really only work on the project when everyone was present. However, each member specialized in driving some aspect of the project.

Pranav: Pranav was mainly in charge of implementing the actual page ranking algorithm and determining the logic of what needed to be done. There was pseudo code given in a prior lecture written in Scala and not handling preprocessing. Since this project was done in Python, some effort needed to be made in terms of looking at the PySpark APIs and figuring out how the code should be written to meet the requirements of the algorithm as detailed on the assignment page. Pranav had the most experience with Scala and Spark so it was easier for him to lead the coding aspect due to his prior experience. However, coding was still largely a group effort.

Viren: Viren was the one primarily in charge of setting up Hadoop and Spark and running the experiment (s). Most of the time when we met up, his screen was shared as he drove most of the time. He also set up the repository on Github and helped in organizing a lot of the things, such as the run scripts for the different tasks as well as the README files.

Sameer: Sameer was responsible primarily for gathering and compiling the necessary evidence (including screenshots) and organizing them within the report. He assisted with communicating various trends and interesting features from each of the tasks to the rest of the team and helped drive some of the discussions surrounding the causes of these events and how to move forward with any further tests.