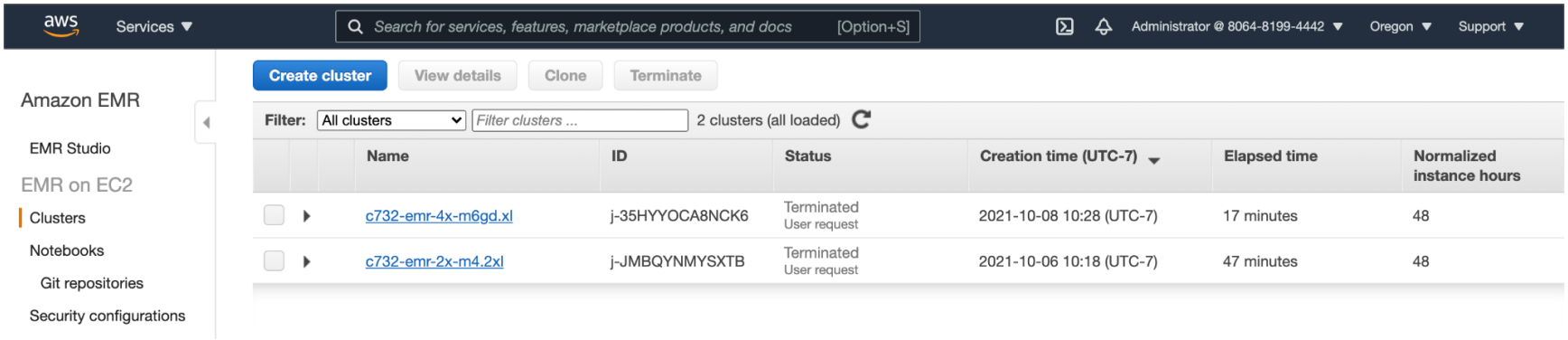
Brendan Artley

1. Terminated EMR clusters.



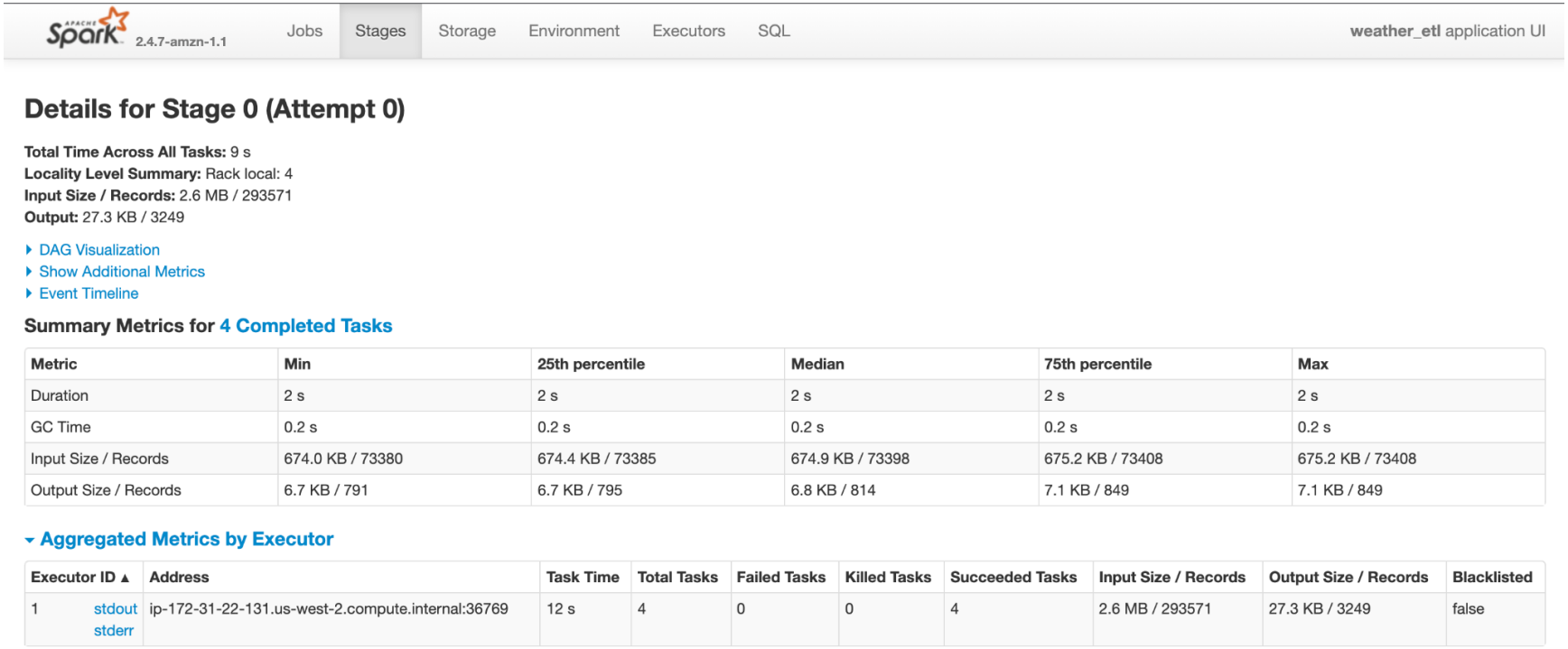
2. Section 2:
    a. What fraction of the input file was prefiltered by S3 before it was sent to Spark?

        - There was a significant portion of the input file that got prefiltered by S3 before getting sent to Spark. In the tests that I ran, I found that without using S3 Select the input size was 2.6MB and 293571 records, whereas when using S3 Select the input size was 97.9kb and 3249 records. This means that the number of records in the input was reduced by roughly 99%. As we have learned in past assignments, reducing the amount of data that is transferred across a network is important in maintaining quick and efficient applications. We can leverage cloud resources all we want but at the end of the day, a more logical implementation will save both time and computational resources needed to run a job.

    b. Comparing the different input numbers for the regular version versus the prefiltered one, what operations were performed by S3 and which ones performed in Spark?

        - When comparing the two application versions which ran with S3 Select or without S3 Select, it seems S3 performs all the SQL-like filtering operations that it can. Running the program locally in a pyspark shell, I found that after I filter the weather data for weather['mflag'].isNull(), station.startswith('CA'), and weather['observation'] == 'TMAX', the total number of records remaining is 3249. This matches the number of input records when we

used S3 Select and therefore this would be the point at which S3 operations end. Spark then creates an additional column that displays the temperature in celsius, selects the relevant columns, and writes the output to a gzipped file. This combination of using S3 and spark was very easy to use and resulted in lower costs and an increase in performance.



Tasks (4)

Index ▲	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	GC Time	Input Size / Records	Output Size / Records	Errors
0	0	0	SUCCESS	RACK_LOCAL	1	ip-172-31-22-131.us-west-2.compute.internal stdout stderr	2021/10/06 17:51:19	2 s	0.2 s	675.2 KB / 73408	7.1 KB / 849	
1	1	0	SUCCESS	RACK_LOCAL	1	ip-172-31-22-131.us-west-2.compute.internal stdout stderr	2021/10/06 17:51:19	2 s	0.2 s	674.9 KB / 73398	6.7 KB / 791	
2	2	0	SUCCESS	RACK_LOCAL	1	ip-172-31-22-131.us-west-2.compute.internal stdout stderr	2021/10/06 17:51:19	2 s	0.2 s	674.4 KB / 73380	6.8 KB / 814	
3	3	0	SUCCESS	RACK_LOCAL	1	ip-172-31-22-131.us-west-2.compute.internal stdout stderr	2021/10/06 17:51:19	2 s	0.2 s	674.0 KB / 73385	6.7 KB / 795	

Details for Stage 0 (Attempt 0)

Total Time Across All Tasks: 8 s
Locality Level Summary: Rack local: 4
Input Size / Records: 97.9 KB / 3249
Output: 27.3 KB / 3249

- DAG Visualization
- Show Additional Metrics
- Event Timeline

Summary Metrics for 4 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	2 s	2 s	2 s	2 s	2 s
GC Time	0.1 s	0.1 s	0.1 s	0.1 s	0.1 s
Input Size / Records	23.8 KB / 791	23.9 KB / 795	24.5 KB / 814	25.6 KB / 849	25.6 KB / 849
Output Size / Records	6.7 KB / 791	6.7 KB / 795	6.8 KB / 814	7.1 KB / 849	7.1 KB / 849

Aggregated Metrics by Executor

Executor ID ▲	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Input Size / Records	Output Size / Records	Blacklisted
1 stdout stderr	ip-172-31-22-131.us-west-2.compute.internal:45071	11 s	4	0	0	4	97.9 KB / 3249	27.3 KB / 3249	false

Tasks (4)

Index ▲	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	GC Time	Input Size / Records	Output Size / Records	Errors
0	0	0	SUCCESS	RACK_LOCAL	1	ip-172-31-22-131.us-west-2.compute.internal stdout	2021/10/06 18:02:19	2 s	0.1 s	25.6 KB / 849	7.1 KB / 849	

Tasks (4)

Index ▲	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	GC Time	Input Size / Records	Output Size / Records	Errors
0	0	0	SUCCESS	RACK_LOCAL	1	ip-172-31-22-131.us-west-2.compute.internal stdout stderr	2021/10/06 18:02:19	2 s	0.1 s	25.6 KB / 849	7.1 KB / 849	
1	1	0	SUCCESS	RACK_LOCAL	1	ip-172-31-22-131.us-west-2.compute.internal stdout stderr	2021/10/06 18:02:19	2 s	0.1 s	23.8 KB / 791	6.7 KB / 791	
2	2	0	SUCCESS	RACK_LOCAL	1	ip-172-31-22-131.us-west-2.compute.internal stdout stderr	2021/10/06 18:02:19	2 s	0.1 s	24.5 KB / 814	6.8 KB / 814	
3	3	0	SUCCESS	RACK_LOCAL	1	ip-172-31-22-131.us-west-2.compute.internal stdout stderr	2021/10/06 18:02:19	2 s	0.1 s	23.9 KB / 795	6.7 KB / 795	

3. Section 3:
    a. Reviewing the job times in the Spark history, which operations took the most time? Is the application IO-bound or compute-bound?

        - 5.5 minutes out of the total 7 minutes it took to run the program were spent on sortby and reducebykey operations (Note that one of these sortBy operations is included in runJob and is roughly 1.2mins). I looked further into this as I would have thought that when using pyspark the operations that take the most time are transferring data and shuffling across the network.

        - Furthermore, when analyzing the stages of the application I found that only ⅓ of the major sortBy operations contained any shuffling at all even though all the times of the sortBy operations were very similar (1.1-1.3 minutes). In the sortBy operations where no shuffle occurs, we are still reading the cached dataframe. The data frame is small enough to not overflow the cache, and therefore the application is still compute-bound. Taking this into account, roughly 4 out of the 7 minutes were spent on computational steps rather than I/O steps. Since we are reaching somewhat of a bottleneck in the program during CPU operations we can say that the application is compute-bound. An increase in computational capability will reduce the execution time of this specific program more than if we increased the speed of I/O bound operations.

        - Spark does a lot of the work behind the scenes for the user and as we found in this example it is sometimes difficult to get all the information we want from the application history. Given the abstraction of some of the operations in spark, I think it is almost impossible to be sure if the application is I/O or compute bound. That being said, the ease of use and automatic optimization of spark in most cases outweighs this limitation.

Spark Jobs (?)

User: hadoop
Total Uptime: 7.0 min
Scheduling Mode: FIFO
Completed Jobs: 6

▸ Event Timeline

▾ **Completed Jobs (6)**

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
5	runJob at SparkHadoopWriter.scala:78 runJob at SparkHadoopWriter.scala:78	2021/10/08 17:39:16	2.6 min	2/2	32/32
4	sortBy at /mnt/tmp/spark-1cb92567-df27-4ee6-a631-88b41270593f/relative_score_bcast.py:47 sortBy at /mnt/tmp/spark-1cb92567-df27-4ee6-a631-88b41270593f/relative_score_bcast.py:47	2021/10/08 17:38:08	1.1 min	1/1	16/16
3	sortBy at /mnt/tmp/spark-1cb92567-df27-4ee6-a631-88b41270593f/relative_score_bcast.py:47 sortBy at /mnt/tmp/spark-1cb92567-df27-4ee6-a631-88b41270593f/relative_score_bcast.py:47	2021/10/08 17:36:55	1.2 min	1/1	16/16
2	collect at /mnt/tmp/spark-1cb92567-df27-4ee6-a631-88b41270593f/relative_score_bcast.py:45 collect at /mnt/tmp/spark-1cb92567-df27-4ee6-a631-88b41270593f/relative_score_bcast.py:45	2021/10/08 17:36:54	0.3 s	2/2 (1 skipped)	32/32 (16 skipped)
1	sortBy at /mnt/tmp/spark-1cb92567-df27-4ee6-a631-88b41270593f/relative_score_bcast.py:42 sortBy at /mnt/tmp/spark-1cb92567-df27-4ee6-a631-88b41270593f/relative_score_bcast.py:42	2021/10/08 17:36:54	0.1 s	1/1 (1 skipped)	16/16 (16 skipped)
0	sortBy at /mnt/tmp/spark-1cb92567-df27-4ee6-a631-88b41270593f/relative_score_bcast.py:42 sortBy at /mnt/tmp/spark-1cb92567-df27-4ee6-a631-88b41270593f/relative_score_bcast.py:42	2021/10/08 17:35:04	1.8 min	2/2	32/32

b. Look up the hourly costs of the m6gd.xlarge instance on the EC2 On-Demand Pricing page. Estimate the cost of processing a dataset ten times as large as reddit-5 using just those 4 instances. If you wanted instead to process this larger dataset making full use of 16 instances, how would it have to be organized?

- The hourly costs of the m6gd.xlarge instance is $0.1808 / hour for Amazon EC2 and $0.0452 / hour for Amazon EMR. The dataset Reddit-5 is roughly 15.7 GB so I will be estimating the cost of processing a dataset that is ~157GB using just those 4 processors. We also need to take into account that we are using an instance as our master node as well (I am going to assume this is an m6gd.xlarge instance). It took me ~7 minutes to run the application on Reddit-5 so I am going to estimate that the cost will be 70 minutes * the number of instances * prices. The cost of this calculation would come out to roughly $1.3108.
- 1.16 * (0.1808 + 0.0452) * 5 = 1.3108

- Time * (EC2 Charge + EMR Charge) * Number of Instances

- Another small but important thing to note is the cost of storing a dataset of that size on S3 and taxes. Using the AWS S3 standard pricing, for the first 50 TB of data you can store data at a rate of $0.023 per GB / month. It would cost 3.611$ a month to store a 157GB dataset on AWS, though I doubt we would need to store it there for that long in this case. Furthemore, on my billing dashboard I had 0.10$ tax charge with a total of 1.81$ for all charges. Including these marginal costs, a more accurate estimate may be in the range of $1.50 for the application.

- If I wanted to instead process a larger dataset and make full use of 16 instances, I would first have to ensure that there are an appropriate number of zipped files that could be processed in parallel by the instances and that they are of consistent sizes. I would have to have at least 64 partitions to process in parallel, and if need be I could split these files further but should try and maintain a number of files that is divisible by 64. ie (128, 256, etc.) This should reduce the time that processors are idle and waiting for other nodes on the cluster to finish their jobs.

Spark Jobs (?)

User: hadoop
Total Uptime: 7.0 min
Scheduling Mode: FIFO
Completed Jobs: 6

▸ Event Timeline

▾ Completed Jobs (6)

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
5	runJob at SparkHadoopWriter.scala:78 runJob at SparkHadoopWriter.scala:78	2021/10/08 17:39:16	2.6 min	2/2	32/32
4	sortBy at /mnt/tmp/spark-1cb92567-df27-4ee6-a631-88b41270593f/relative_score_bcast.py:47 sortBy at /mnt/tmp/spark-1cb92567-df27-4ee6-a631-88b41270593f/relative_score_bcast.py:47	2021/10/08 17:38:08	1.1 min	1/1	16/16
3	sortBy at /mnt/tmp/spark-1cb92567-df27-4ee6-a631-88b41270593f/relative_score_bcast.py:47 sortBy at /mnt/tmp/spark-1cb92567-df27-4ee6-a631-88b41270593f/relative_score_bcast.py:47	2021/10/08 17:36:55	1.2 min	1/1	16/16
2	collect at /mnt/tmp/spark-1cb92567-df27-4ee6-a631-88b41270593f/relative_score_bcast.py:45 collect at /mnt/tmp/spark-1cb92567-df27-4ee6-a631-88b41270593f/relative_score_bcast.py:45	2021/10/08 17:36:54	0.3 s	2/2 (1 skipped)	32/32 (16 skipped)
1	sortBy at /mnt/tmp/spark-1cb92567-df27-4ee6-a631-88b41270593f/relative_score_bcast.py:42 sortBy at /mnt/tmp/spark-1cb92567-df27-4ee6-a631-88b41270593f/relative_score_bcast.py:42	2021/10/08 17:36:54	0.1 s	1/1 (1 skipped)	16/16 (16 skipped)
0	sortBy at /mnt/tmp/spark-1cb92567-df27-4ee6-a631-88b41270593f/relative_score_bcast.py:42 sortBy at /mnt/tmp/spark-1cb92567-df27-4ee6-a631-88b41270593f/relative_score_bcast.py:42	2021/10/08 17:35:04	1.8 min	2/2	32/32