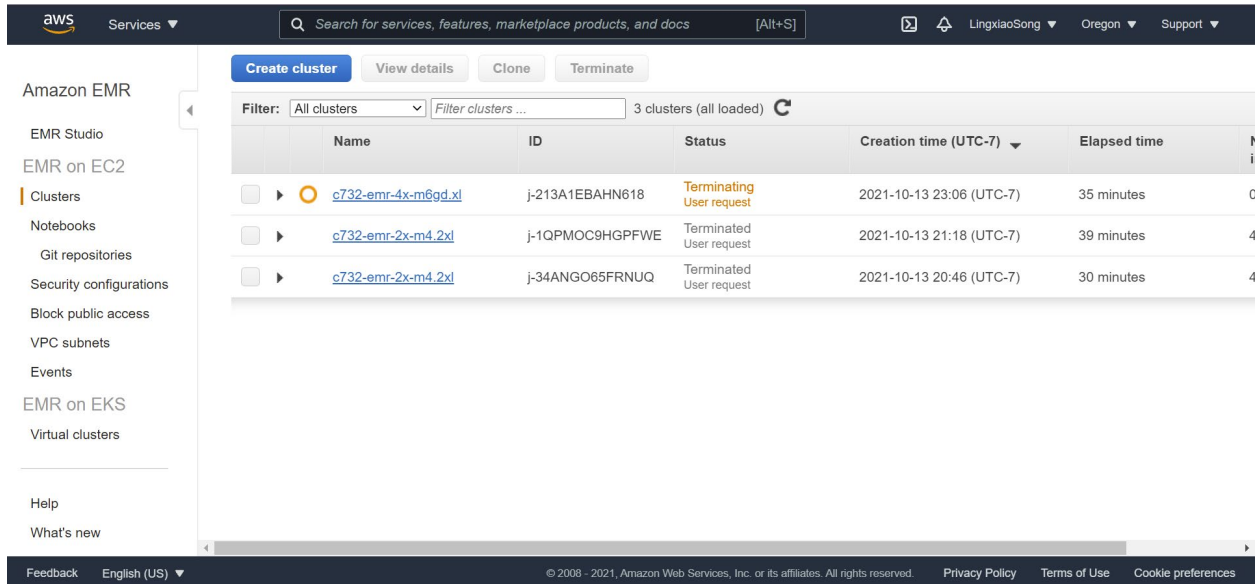


**Q1. Take a screen shot of your list of EMR clusters (if more than one page, only the page with the most recent), showing that all have Terminated status.**



The screenshot shows the AWS Management Console for Amazon EMR. The left sidebar lists navigation options like EMR Studio, EMR on EC2, Clusters, Notebooks, etc. The main content area shows a list of EMR clusters. The filter is set to 'All clusters', and 3 clusters are loaded. All three clusters are in a 'Terminated' status.

	Name	ID	Status	Creation time (UTC-7)	Elapsed time	
<input type="checkbox"/>	<a href="#">c732-emr-4x-m6gd.xl</a>	j-213A1EBAHN618	Terminating User request	2021-10-13 23:06 (UTC-7)	35 minutes	0
<input type="checkbox"/>	<a href="#">c732-emr-2x-m4.2xl</a>	j-1QPMOC9HGPFWE	Terminated User request	2021-10-13 21:18 (UTC-7)	39 minutes	4
<input type="checkbox"/>	<a href="#">c732-emr-2x-m4.2xl</a>	j-34ANGO65FRNUQ	Terminated User request	2021-10-13 20:46 (UTC-7)	30 minutes	4

**For Section 2:**

**Q2. What fraction of the input file was prefiltered by S3 before it was sent to Spark?**

The screenshot of the regular version is as below:



The screenshot shows the Databricks Jobs page. The 'Stages' tab is selected, showing details for a job named 'weather etl application'. The job is in a 'Completed' state.

### Details for Stage 0 (Attempt 0)

Total Time Across All Tasks: 13 s  
 Locality Level Summary: Rack local: 4  
 Input Size / Records: 2.6 MiB / 3245  
 Output Size / Records: 27.2 KiB / 3245  
 Associated Job Ids: 0

[DAG Visualization](#)  
[Show Additional Metrics](#)  
[Event Timeline](#)

#### Summary Metrics for 4 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	3 s	3 s	3 s	3 s	3 s
GC Time	0.2 s	0.2 s	0.2 s	0.2 s	0.2 s
Input Size / Records	674 KiB / 790	674.4 KiB / 794	674.9 KiB / 814	675.2 KiB / 847	675.2 KiB / 847
Output Size / Records	6.7 KiB / 790	6.7 KiB / 794	6.8 KiB / 814	7.1 KiB / 847	7.1 KiB / 847

Showing 1 to 4 of 4 entries

The screenshot of the prefiltered version is as below:

## Details for Stage 0 (Attempt 0)

Total Time Across All Tasks: 12 s  
Locality Level Summary: Rack local: 4  
Input Size / Records: 97.7 KiB / 3245  
Output Size / Records: 27.2 KiB / 3245  
Associated Job Ids: 0

- ▶ DAG Visualization
- ▶ Show Additional Metrics
- ▶ Event Timeline

### Summary Metrics for 4 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	3 s	3 s	3 s	3 s	3 s
GC Time	0.2 s	0.2 s	0.2 s	0.2 s	0.2 s
Input Size / Records	23.8 KiB / 790	23.9 KiB / 794	24.5 KiB / 814	25.5 KiB / 847	25.5 KiB / 847
Output Size / Records	6.7 KiB / 790	6.7 KiB / 794	6.8 KiB / 814	7.1 KiB / 847	7.1 KiB / 847

Showing 1 to 4 of 4 entries

We can find that the input size of the regular version is 2.6MiB; while the input size of the prefiltered version is only 97.7KiB. Thus,  $(2.6 - 97.7/1024) = 2.50\text{MiB}$  data was prefiltered by S3 before it was sent to Spark, that about 96.3% of the original dataset.

**Q3. 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?**

The reader, three filter functions, and the select function were performed by S3; and the column value computing operation were performed in Spark.

## For Section 3:

**Q4. Reviewing the job times in the Spark history, which operations took the most time? Is the application IO-bound or compute-bound?**

The screenshot of running on SFU cluster:

## Spark Jobs (?)

User: lsa108  
Total Uptime: 21 min  
Scheduling Mode: FIFO  
Completed Jobs: 4

- ▶ Event Timeline

### ▼ Completed Jobs (4)

Page: 1

1 Pages. Jump to 1. Show 100 items in a page. Go

Job Id ▼	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
3	runJob at SparkHadoopWriter.scala:83 runJob at SparkHadoopWriter.scala:83	2021/10/14 06:08:28	8.0 min	2/2	32/32
2	sortBy at /home/lsa108/A4/relative_score_bcast.py:39 sortBy at /home/lsa108/A4/relative_score_bcast.py:39	2021/10/14 06:04:06	4.4 min	1/1	16/16
1	sortBy at /home/lsa108/A4/relative_score_bcast.py:39 sortBy at /home/lsa108/A4/relative_score_bcast.py:39	2021/10/14 05:59:45	4.3 min	1/1	16/16
0	collect at /home/lsa108/A4/relative_score_bcast.py:33 collect at /home/lsa108/A4/relative_score_bcast.py:33	2021/10/14 05:55:23	4.4 min	2/2	32/32

## Stages for All Jobs

Completed Stages: 6

### Completed Stages (6)

Page: 1 1 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
5	runJob at SparkHadoopWriter.scala:83	2021/10/14 06:13:23	3.1 min	16/16		1578.0 MiB	566.2 MiB	
4	sortBy at /home/lsa108/A4/relative_score_bcast.py:39	2021/10/14 06:08:28	4.9 min	16/16	8.5 GiB			566.2 MiB
3	sortBy at /home/lsa108/A4/relative_score_bcast.py:39	2021/10/14 06:04:06	4.4 min	16/16	8.5 GiB			
2	sortBy at /home/lsa108/A4/relative_score_bcast.py:39	2021/10/14 05:59:45	4.3 min	16/16	8.5 GiB			
1	collect at /home/lsa108/A4/relative_score_bcast.py:33	2021/10/14 05:59:45	0.5 s	16/16			7.1 KiB	
0	reduceByKey at /home/lsa108/A4/relative_score_bcast.py:28	2021/10/14 05:55:23	4.4 min	16/16	8.5 GiB			7.1 KiB

Page: 1 1 Pages. Jump to 1 . Show 100 items in a page. Go

The screenshot of running on AWS:

## Spark Jobs (?)

User: hadoop

Total Uptime: 4.5 min

Scheduling Mode: FIFO

Completed Jobs: 4

Event Timeline

### Completed Jobs (4)

Page: 1 1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
3	runJob at SparkHadoopWriter.scala:83	2021/10/14 06:32:14	1.8 min	2/2	32/32
2	sortBy at /mnt/tmp/spark-e2d23de0-553c-47c2-b6ed-32a50bf7d4c5/relative_score_bcast.py:38	2021/10/14 06:31:52	22 s	1/1	16/16
1	sortBy at /mnt/tmp/spark-e2d23de0-553c-47c2-b6ed-32a50bf7d4c5/relative_score_bcast.py:38	2021/10/14 06:31:31	21 s	1/1	16/16
0	collect at /mnt/tmp/spark-e2d23de0-553c-47c2-b6ed-32a50bf7d4c5/relative_score_bcast.py:32	2021/10/14 06:29:42	1.8 min	2/2	32/32

## Stages for All Jobs

Completed Stages: 6

### Completed Stages (6)

Page: 1 1 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
5	runJob at SparkHadoopWriter.scala:83	2021/10/14 06:32:46	1.3 min	16/16		1578.0 MiB	566.2 MiB	
4	sortBy at /mnt/tmp/spark-e2d23de0-553c-47c2-b6ed-32a50bf7d4c5/relative_score_bcast.py:38	2021/10/14 06:32:14	32 s	16/16	8.6 GiB			566.2 MiB
3	sortBy at /mnt/tmp/spark-e2d23de0-553c-47c2-b6ed-32a50bf7d4c5/relative_score_bcast.py:38	2021/10/14 06:31:52	22 s	16/16	8.6 GiB			
2	sortBy at /mnt/tmp/spark-e2d23de0-553c-47c2-b6ed-32a50bf7d4c5/relative_score_bcast.py:38	2021/10/14 06:31:31	21 s	16/16	8.6 GiB			
1	collect at /mnt/tmp/spark-e2d23de0-553c-47c2-b6ed-32a50bf7d4c5/relative_score_bcast.py:32	2021/10/14 06:31:31	0.3 s	16/16			7.1 KiB	
0	reduceByKey at /mnt/tmp/spark-e2d23de0-553c-47c2-b6ed-32a50bf7d4c5/relative_score_bcast.py:27	2021/10/14 06:29:42	1.8 min	16/16	8.5 GiB			7.1 KiB

As shown in the screenshots, the compute-bound (reduceByKey, sortBy) time in the SFU cluster is around  $(4.4+4.3+4.4+4.9) = 18$  min, and the IO operation (writer) time is around 3.1min; in the AWS, the compute-bound time is around  $(1.8+(21+22+32)/60) = 2.9$ min, and the IO operation (writer) time is

around 1.3min. Thus, seems the compute-bound took more time in this case. But on the other hand, within the reduceByKey stage, the IO operation and the compute operation can exist at the same time, and we cannot find the exact time-cost separately, so it's hard to say whether the compute-bound or IO bound would take more time. Besides, the time cost also depends on the size of dataset, the compute-related code amount, etc., not just the cluster setting.

**Q5. 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 screenshot of EC2 on-demand pricing page of m6gd.xlarge is as following:

Viewing 378 of 378 available instances						
<input type="text" value="m6gd.xlarge"/>						
Instance name ▲	On-Demand hourly rate ▼	vCPU ▼	Memory ▼	Storage ▼	Network performance ▼	
m6gd.xlarge	\$0.1808	4	16 GiB	1 x 237 NVMe SSD	Up to 10 Gigabit	

Thus, we can estimate the cost of processing a dataset ten times as large as reddit-5 using 4 instances:  
 $0.1808 * 4 * (4.5/60) * 10 = 0.5424$  dollars

If we use 16 instances, then we would have 64 cores to process data. According to our prior experiment results, we can get a better performance when the partitions are around two or three times of cores. Thus, we can organize the dataset into 128-192 files before inputting it, or we can use the repartition function to help reorganize the RDD.