Abrahm Rohwer, Thomas Nguyen
Professor Supreeth Shastri
CS:5630
18 November 2022

Project 2 Report

[VM Specs]

[Program A Analysis Between 1 to 3 Nodes]

[Program B Analysis Between 1 to 3 nodes/ Program A & Program B Analysis between 1 to 3 nodes]

[Program A Analysis between EMR configs]

[Program B Analysis between EMR Configs / Program A & Program B Analysis between EMR configs]

Part A

a. **Mapper & Reducer functions for both of MapReduce Programs – attached as pythonFiles.rar - note: We had to add the python prefix to our hadoop command inputs as follows to avoid errors:** hadoop jar /usr/lib/hadoop/hadoop-streaming.jar —files mapper.py,reducer.py —mapper 'python mapper.py' —reducer 'python reducer.py' —input Tweet.csv —output results
b. **Output summary from Hadoop runs**

Program A using single node – pseudo distributed setup

File System Counters	
FILE: Number of bytes read	1.54E+08
FILE: Number of bytes written	3.87E+08
FILE: Number of read operations	0
FILE: Number of large read operations	0
FILE: Number of write operations	0
HDFS: Number of bytes read	2.66E+09
HDFS: Number of bytes written	239752
HDFS: Number of read operations	63
HDFS: Number of large read operations	0
HDFS: Number of write operations	8
HDFS: Number of bytes read erasure-coded	0
Map-Reduce Framework	
Map input records	3717965
Map output records	7935782
Map output bytes	60708315
Map output materialized bytes	76579909
Input split bytes	540
Combine input records	0
Combine output records	0

Reduce input groups	27907
Reduce shuffle bytes	76579909
Reduce input records	7935782
Reduce output records	27907
Spilled Records	15871564
Shuffled Maps	5
Failed Shuffles	0
Merged Map outputs	5
GC time elapsed (ms)	369
Total committed heap usage (bytes)	3.51E+09
Shuffle Errors	
BAD_ID	0
CONNECTION	0
IO_ERROR	0
WRONG_LENGTH	0
WRONG_MAP	0
WRONG_REDUCE	0
File Input Format Counters	
Bytes Read	6.58E+08
File Output Format Counters	
Bytes Written	239752
Job Duration (Grabbed using https://unix.stackexchange.com/a/314372, our ResourceManager/JobHistory was not working so we manually timed our execution time)	55 Seconds
Cost ($0.10 an hour for m4.large instance), multiplying by job duration only to compare with cost of multi-node setups	$0.10*(55/60/60)= $0.00152777778

Program B

File System Counters	
FILE: Number of bytes read	49490332
FILE: Number of bytes written	1.19E+08
FILE: Number of read operations	0
FILE: Number of large read operations	0
FILE: Number of write operations	0
HDFS: Number of bytes read	2.66E+09
HDFS: Number of bytes written	1333050
HDFS: Number of read operations	63
HDFS: Number of large read operations	0

HDFS: Number of write operations	8
HDFS: Number of bytes read erasure-coded	0
Map-Reduce Framework	
Map input records	3717965
Map output records	1487600
Map output bytes	21337233
Map output materialized bytes	24312463
Input split bytes	540
Combine input records	0
Combine output records	0
Reduce input groups	94430
Reduce shuffle bytes	24312463
Reduce input records	1487600
Reduce output records	94430
Spilled Records	2975200
Shuffled Maps	5
Failed Shuffles	0
Merged Map outputs	5
GC time elapsed (ms)	368
Total committed heap usage (bytes)	3.46E+09
Shuffle Errors	
BAD_ID	0
CONNECTION	0
IO_ERROR	0
WRONG_LENGTH	0
WRONG_MAP	0
WRONG_REDUCE	0
File Input Format Counters	
Bytes Read	6.58E+08
File Output Format Counters	
Bytes Written	1333050
Job Duration (Grabbed using https://unix.stackexchange.com/a/314372, our ResourceManager/JobHistory was not working so we manually timed our execution time)	26 Seconds
Cost ($0.10 an hour for m4.large instance), multiplying by job duration only to compare with cost of multi-node setups	$0.10*(26/60/60)= $0.000722222222

c. **Resources/Notes:**
   a. We had to run dos2unix on every python file and removed comments just in case for encoding issues, if you would like us to explain our code feel free to reach out.
   b. https://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/ - reducer.py & reducer1.py are the same reduce function as written in this article
   c. https://docs.python.org/3/library/re.html & https://stackoverflow.com/questions/41786385/regular-expression-for-stock-tickers-python - using re.findall() for mapper.py to find stock tickers using regex - r'[$][A-Z]{1,5}\s*' was our attempt, r' to consume raw string, [A-Z] to consume capital alphabetical letters only given capital body of a tweet, {1,5} to contain 1-5 capital alphabetical letters, and \s* to end with 0+ white spaces
   d. https://stackoverflow.com/questions/2613800/how-to-convert-dos-windows-newline-crlf-to-unix-newline-lf - fixed windows encoding issues with dos2unix
   e. https://stackoverflow.com/questions/24796541/how-can-i-skip-first-line-reading-from-stdin#:~:text=You%20can%20use%20the%20enumerate,first%20line)%20 - for mapper1.py to skip column headings of csv file
   f. https://stackoverflow.com/questions/42756555/permission-denied-error-while-running-start-dfs-sh & https://stackoverflow.com/questions/11889261/datanode-process-not-running-in-hadoop - debugging start-dfs.sh/nodes not appearing using jps
   g. https://www.youtube.com/watch?v=Sos1QKaZySo&ab_channel=UnfoldDataScience - used for guidance for our environmental variables and where to find our hadoop .xml files
   h. https://stackoverflow.com/a/54253954 - to fix our streaming command

Part B

a. **Baseline comparison between single-node and multi-node cluster:** Compare the performance of your two programs from the single-node setup of part-A to the three-node cluster of partB. Please include the VM specs such as CPU, RAM, and disks for both set up. Was it cheaper to run your programs on single-node or cluster?
   a. Part A VM Specs:

Instance	vCPU*	Mem (GiB)	Storage	Dedicated EBS Bandwidth (Mbps)	Network Performance***
m4.large	2	8	32 GiB GP2 EBS	450	Moderate

   b. Part B VM Specs:

Instance	vCPU*	Mem (GiB)	Storage	Dedicated EBS Bandwidth (Mbps)	Network Performance***
m4.large	2	8	32 GiB GP2 EBS	450	Moderate

c. Compare performance of two programs from single-node setup of part A to three-node cluster of part B.

Program A

1 Node		3 Nodes	
File System Counters		File System Counters	
FILE: Number of bytes read	1.54E+08	FILE: Number of bytes read	3,791,780
FILE: Number of bytes written	3.87E+08	FILE: Number of bytes written	10,875,920
FILE: Number of read operations	0	FILE: Number of read operations	0
FILE: Number of large read operations	0	FILE: Number of large read operations	0
FILE: Number of write operations	0	FILE: Number of write operations	0
HDFS: Number of bytes read	2.66E+09	HDFS: Number of bytes read	880
HDFS: Number of bytes written	239752	HDFS: Number of bytes written	0
HDFS: Number of read operations	63	HDFS: Number of read operations	10
HDFS: Number of large read operations	0	HDFS: Number of large read operations	0
HDFS: Number of write operations	8	HDFS: Number of write operations	0
HDFS: Number of bytes read erasure-coded	0	S3: Number of bytes read	6.58E+08
Map-Reduce Framework		S3: Number of bytes written	239,752
Map input records	3717965	S3: Number of read operations	0
Map output records	7935782	S3: Number of large read operations	0
Map output bytes	60708315	S3: Number of write operations	0
Map output materialized bytes	76579909	Job Counters	
Input split bytes	540	Killed map tasks	2
Combine input records	0	Killed reduce tasks	1
Combine output records	0	Launched map tasks	10
Reduce input groups	27907	Launched reduce tasks	4
Reduce shuffle bytes	76579909	Data-local map tasks	10
Reduce input records	7935782	Total time spent by all maps in occupied slots (ms)	18,710,976

Reduce output records	27907	Total time spent by all reduces in occupied slots (ms)	8,213,184
Spilled Records	15871564	Total time spent by all map tasks (ms)	389,812
Shuffled Maps	5	Total time spent by all reduce tasks (ms)	85,554
Failed Shuffles	0	Total vcore-milliseconds taken by all map tasks	389,812
Merged Map outputs	5	Total vcore-milliseconds taken by all reduce tasks	85,554
GC time elapsed (ms)	369	Total megabyte-milliseconds taken by all map tasks	5.99E+08
Total committed heap usage (bytes)	3.51E+09	Total megabyte-milliseconds taken by all reduce tasks	2.63E+08
Shuffle Errors		Map-Reduce Framework	
BAD_ID	0	Map input records	3,717,965
CONNECTION	0	Map output records	7,935,782
IO_ERROR	0	Map output bytes	60,708,315
WRONG_LENGTH	0	Map output materialized bytes	4,169,173
WRONG_MAP	0	Input split bytes	880
WRONG_REDUCE	0	Combine input records	0
File Input Format Counters		Combine output records	0
Bytes Read	6.58E+08	Reduce input groups	27,907
File Output Format Counters		Reduce shuffle bytes	4,169,173
Bytes Written	239752	Reduce input records	7,935,782
Job Duration (Grabbed using https://unix.stackexchange.com/a/314372, our ResourceManager/JobHistory was not working so we manually timed our execution time)	55 Seconds	Reduce output records	27,907
Cost ($0.10 an hour for m4.large instance), multiplying by job duration only to compare with cost of multi-node setups	$0.10*(55/60/60)= $0.00152777778	Spilled Records	15,871,564
		Shuffled Maps	30
		Failed Shuffles	0
		Merged Map outputs	30
		GC time elapsed (ms)	8,799

		CPU time spent (ms)	143,250
		Physical memory (bytes) snapshot	7.41E+09
		Virtual memory (bytes) snapshot	4.73E+10
		Total committed heap usage (bytes)	6.19E+09
		Shuffle Errors	
		BAD_ID	0
		CONNECTION	0
		IO_ERROR	0
		WRONG_LENGTH	0
		WRONG_MAP	0
		WRONG_REDUCE	0
		File Input Format Counters	
Bytes Read 6.58E+08			
		File Output Format Counters	
		Bytes Written	239,752
		Job Duration	1.9 min
		Cost (Amazon EC2 Price + Amazon EMR Price) $0.10 per hour for m4.large instances, $0.03 per hour for EMR. We are going to multiply the EC2 price by number of nodes.	(1.9/60) * (($0.10 * 3) + $0.03) = $0.01045

Program A Analysis Between 1 to 3 Nodes

Interestingly, 1 node yielded better performance for read and write operations across all metrics than 3 nodes. For example, for file system counter metrics, the single node cluster read 1.54E+08 bytes while the 3-node cluster read only 3.8 million bytes and 1 node took 55 seconds to run Program A while 3 nodes took 1.9 minutes. We believe the reason for this is because our MapReduce key is only being sent to only a single node in the program and additionally, the 3-node cluster has more network overhead/data movement. Furthermore, running Hadoop on a single file that is smaller than the Hadoop block size (128 MB) results in wasted resources with the added cost of the network overhead. For instance, our file was only 124 MB which is 4 MB than the standard HDFS block size. This resulted in a waste of using HDFS' 128 MB block size. Additionally, the 3-node cluster was more expensive to run this program on. Our single node cluster cost only $0.001 while our 3-node cluster cost $0.01. The added network overhead and resource utilization resulted in higher costs, despite us not yielding any performance benefits. We believe that we could make the cost more worth it if we were able to configure YARN container sizes to get more benefit and have less wasted resources.

Program B

1 Node		3 Nodes	
File System Counters		File System Counters	
FILE: Number of bytes read	49,490,332	FILE: Number of bytes read	4,700,537
FILE: Number of bytes written	1.19E+08	FILE: Number of bytes written	13,482,161
FILE: Number of read operations	0	FILE: Number of read operations	0
FILE: Number of large read operations	0	FILE: Number of large read operations	0
FILE: Number of write operations	0	FILE: Number of write operations	0
HDFS: Number of bytes read	2.66E+09	HDFS: Number of bytes read	880
HDFS: Number of bytes written	1,333,050	HDFS: Number of bytes written	0
HDFS: Number of read operations	63	HDFS: Number of read operations	10
HDFS: Number of large read operations	0	HDFS: Number of large read operations	0
HDFS: Number of write operations	8	HDFS: Number of write operations	0
HDFS: Number of bytes read erasure-coded	0	S3: Number of bytes read	6.58E+08
Map-Reduce Framework		S3: Number of bytes written	13,33,050
Map input records	3,717,965	S3: Number of read operations	0
Map output records	1,487,600	S3: Number of large read operations	0
Map output bytes	21,337,233	S3: Number of write operations	0
Map output materialized bytes	24,312,463	Job Counters	
Input split bytes	540	Killed map tasks	2
Combine input records	0	Killed reduce tasks	1
Combine output records	0	Launched map tasks	10
Reduce input groups	94430	Launched reduce tasks	3
Reduce shuffle bytes	24,312,463	Data-local map tasks	10
Reduce input records	1,487,600	Total time spent by all maps in occupied slots (ms)	17,208,480

Reduce output records	94,430	Total time spent by all reduces in occupied slots (ms)	6,461,472
Spilled Records	2,975,200	Total time spent by all map tasks (ms)	358,510
Shuffled Maps	5	Total time spent by all reduce tasks (ms)	67,307
Failed Shuffles	0	Total vcore-milliseconds taken by all map tasks	358,510
Merged Map outputs	5	Total vcore-milliseconds taken by all reduce tasks	67,307
GC time elapsed (ms)	368	Total megabyte-milliseconds taken by all map tasks	5.51E+08
Total committed heap usage (bytes)	3.46E+09	Total megabyte-milliseconds taken by all reduce tasks	2.07E+08
Shuffle Errors		Map-Reduce Framework	
BAD_ID	0	Map input records	3,717,965
CONNECTION	0	Map output records	1,487,600
IO_ERROR	0	Map output bytes	21,337,233
WRONG_LENGTH	0	Map output materialized bytes	5,866,605
WRONG_MAP	0	Input split bytes	880
WRONG_REDUCE	0	Combine input records	0
File Input Format Counters		Combine output records	0
Bytes Read	6.58E+08	Reduce input groups	94,430
File Output Format Counters		Reduce shuffle bytes	5,866,605
Bytes Written	1,333,050	Reduce input records	1,487,600
Job Duration (Grabbed using https://unix.stackexchange.com/a/314372, our ResourceManager/JobHistory was not working so we manually timed our execution time)	26 Seconds	Reduce output records	94,430
Cost ($0.10 an hour for m4.large instance), multiplying by job duration only to compare with cost of multi-node setups	$0.10*(26/60/60)= $0.000722222222	Spilled Records	2,975,200

		Shuffled Maps	30
		Failed Shuffles	0
		Merged Map outputs	30
		GC time elapsed (ms)	9,104
		CPU time spent (ms)	112,050
		Physical memory (bytes) snapshot	7.22E+09
		Virtual memory (bytes) snapshot	4.73E+10
		Total committed heap usage (bytes)	6.23E+09
		Shuffle Errors	
		BAD_ID	0
		CONNECTION	0
		IO_ERROR	0
		WRONG_LENGTH	0
		WRONG_MAP	0
		WRONG_REDUCE	0
		File Input Format Counters	
		Bytes Read	6.58E+08
		File Output Format Counters	
		Bytes Written	1,333,050
		Job Duration	1.7min
		Cost (Amazon EC2 Price + Amazon EMR Price) $0.10 per hour for m4.large instances, $0.03 per hour for EMR. We are going to multiply the EC2 price by number of nodes.	(1.7/60) *(($0.10*3)+ $0.03) = $0.00935

Program B Analysis Between 1 to 3 nodes

Again, as with the first MapReduce program, 1 node has better performance than 3 nodes. 1 node took 26 seconds to run Program B while 3 nodes took 1.7 minutes. We believe this is because of the same reasons as the first. That is, our file was smaller than the HDFS block size needed to yield significant benefits from the multi-node cluster that would make up for the additional network/data overhead that comes with more nodes. Furthermore, something interesting to note is that while 3 nodes resulted in less bytes read/written, 3 nodes were also more expensive. So, despite the decreased performance, 3 nodes cost more. We believe that if we configured YARN to change the container sizes, we could have gotten much more utilization out of the multi-node cluster.

Program A & Program B Analysis Between 1 to 3 nodes

Overall, the results between the two jobs were very similar. The places were 1 node shined while 3 nodes lacked was identical across the jobs. 1 node proved to have better performance across both MapReduce jobs. As noted before, we believe this is due to the added latency associated with an increased infrastructural overhead caused by using 3 nodes. If our file size was bigger, our job would've been vastly improved by the use of 3 nodes when compared to one. Simply put, our job was too small to see the vast benefits of parallelization among other things but it was really interesting to see just how fast our jobs took using one node, only using 55 seconds for Program A and 26 seconds for Program B.

b. **List any references you used in performing these tasks (just a list of URLs would do). Pease remember that using any resources without citation constitutes plagiarism.**
    a. https://pdfs.semanticscholar.org/f906/6110f233bbf4f1096e1c078c54a2dab6d6f5.pdf
        i. Used this as the main insight into our understanding of single node vs multi-node clusters as they relate to HDFS
    b. https://www.projectpro.io/article/hadoop-cluster-overview-what-it-is-and-how-to-setup-one/356#:~:text=As%20the%20name%20says%2C%20Single,on%20the%20same%20machine%2Fhost.
        i. Used this to understand the differences between using 1 node vs multiple and some of the affects of each
    c. https://datascience.stackexchange.com/questions/8380/is-there-a-benefit-to-using-hadoop-with-only-one-node
        i. Used this as a supplemental reading into why using 1 node yielded better results than multiple nodes

c. Benchmarking different EMR configurations: Compare the performance of your two programs across the three configurations described in benchmarking. Make a table, where columns represent the three configs and rows represent the two programs. Be sure to compare both completion time and cost. Briefly explain your findings. **These completion times are pulled from 'High-level application history' for our MapReduce jobs.**

According to https://aws.amazon.com/emr/pricing/?nc=sn&loc=4, cost is calculated by adding the Amazon EMR price to the Amazon EC2 price, we will multiply the Amazon EC2 price by the number of nodes added to the Amazon EMR Price. We will be calculating cost for the execution duration of our jobs, given by the High-Level Application History provided by EMR logs since the output logs provide CPU time which is not reflective of real execution time.

Program A Chart

	2 Nodes	3 Nodes	4 Nodes
File System Counters			

FILE: Number of bytes read	3778597	3791780	3797659
FILE: Number of bytes written	10372657	10875920	11852621
FILE: Number of read operations	0	0	0
FILE: Number of large read operations	0	0	0
FILE: Number of write operations	0	0	0
HDFS: Number of bytes read	880	880	1056
HDFS: Number of bytes written	0	0	0
HDFS: Number of read operations	10	10	12
HDFS: Number of large read operations	0	0	0
HDFS: Number of write operations	0	0	0
S3: Number of bytes read	6.58E+08	6.58E+08	6.58E+08
S3: Number of bytes written	239752	239752	239752
S3: Number of read operations	0	0	0
S3: Number of large read operations	0	0	0
S3: Number of write operations	0	0	0
Job Counters			
Killed map tasks	1	2	1
Killed reduce tasks	0	1	1
Launched map tasks	10	10	12
Launched reduce tasks	1	4	6
Data-local map tasks	10	10	12
Total time spent by all maps in occupied slots (ms)	12764880	18710976	26999232
Total time spent by all reduces in occupied slots (ms)	2375616	8213184	10579968
Total time spent by all map tasks (ms)	265935	389812	562484
Total time spent by all reduce tasks (ms)	24746	85554	110208
Total vcore-milliseconds taken by all map tasks	265935	389812	562484
Total vcore-milliseconds taken by all reduce tasks	24746	85554	110208
Total megabyte-milliseconds taken by all map tasks	4.08E+08	5.99E+08	8.64E+08
Total megabyte-milliseconds taken by all reduce tasks	76019712	2.63E+08	3.39E+08
Map-Reduce Framework			
Map input records	3717965	3717965	3717965
Map output records	7935782	7935782	7935782
Map output bytes	60708315	60708315	60708315
Map output materialized bytes	4127871	4169173	4242904
Input split bytes	880	880	1056
Combine input records	0	0	0
Combine output records	0	0	0
Reduce input groups	27907	27907	27907
Reduce shuffle bytes	4127871	4169173	4242904
Reduce input records	7935782	7935782	7935782

Reduce output records	27907	27907	27907
Spilled Records	15871564	15871564	15871564
Shuffled Maps	10	30	60
Failed Shuffles	0	0	0
Merged Map outputs	10	30	60
GC time elapsed (ms)	6320	8799	12349
CPU time spent (ms)	132490	143250	176380
Physical memory (bytes) snapshot	6.52E+09	7.41E+09	9.1E+09
Virtual memory (bytes) snapshot	3.79E+10	4.73E+10	6.33E+10
Total committed heap usage (bytes)	5.58E+09	6.19E+09	7.58E+09
Shuffle Errors			
BAD_ID	0	0	0
CONNECTION	0	0	0
IO_ERROR	0	0	0
WRONG_LENGTH	0	0	0
WRONG_MAP	0	0	0
WRONG_REDUCE	0	0	0
File Input Format Counters			
Bytes Read	6.58E+08	6.58E+08	6.58E+08
File Output Format Counters			
Bytes Written	239752	239752	239752
High-Level Application History Duration	3.1 min	1.9 min	1.7 min
Cost (Amazon EC2 Price + Amazon EMR Price) $0.10 per hour for m4.large instances, $0.03 per hour for EMR. We are going to multiply the EC2 price by number of nodes.	(3.1/60) * (($0.10 * 2) + $0.03) = $0.0118833333	(1.9/60) * (($0.10 * 3) + $0.03) = $0.01045	(1.7/60) * (($0.10 * 4) + $0.03) = $0.0121833333

Program A Analysis between EMR configs

In our MapReduce jobs for Program A, where we compute the total number of Twitter mentions per stock ticker, as we go from a 2-node configuration to a 3 node and 4 node configuration, the duration of our job in real time decreases from 3.1 minutes for 2 nodes, to 1.9 minutes for 3 nodes, and 1.7 minutes for 4 nodes. We assume the execution time for our job decreases due to the nature of Hadoop, where the master node partitions our input into blocks, our 3 million records of tweets, and distributes these blocks to our worker nodes for processing. Since we increase our number of resources/worker nodes to process our computation, it only makes sense that our computation gets finished much faster, but it seems that our runtime improves a lot from 2 nodes to 3 nodes, but there is a much smaller improvement from 3 nodes to 4 nodes. Although the duration of our program decreases as we increase the number of nodes, we end up spending more money when we use 2 nodes or 4 nodes versus when we use 3 nodes. 2 nodes cost us about $0.0119, 3 nodes cost us about $0.0105, and 4 nodes cost us about $0.0122, using 3 nodes was most cost effective for running Program A if we only consider the time elapsed of running the job. We can also see that, when we add more nodes, our CPU time spent increases. This is also explained by adding more nodes, since the more nodes we have, the more CPU utilization due to more resources being available, and so the more time spent by CPUs processing instructions.

Program B Chart

	2 Nodes	3 Nodes	4 Nodes
File System Counters			
FILE: Number of bytes read	4652163	4700537	4726425
FILE: Number of bytes written	12890344	13482161	14604318
FILE: Number of read operations	0	0	0
FILE: Number of large read operations	0	0	0
FILE: Number of write operations	0	0	0
HDFS: Number of bytes read	880	880	1056
HDFS: Number of bytes written	0	0	0
HDFS: Number of read operations	10	10	12
HDFS: Number of large read operations	0	0	0
HDFS: Number of write operations	0	0	0
S3: Number of bytes read	6.58E+08	6.58E+08	6.58E+08
S3: Number of bytes written	1333050	1333050	1333050
S3: Number of read operations	0	0	0
S3: Number of large read operations	0	0	0
S3: Number of write operations	0	0	0
Job Counters			
Killed map tasks	1	2	2
Killed reduce tasks	0	1	1
Launched map tasks	10	10	12
Launched reduce tasks	1	3	5
Data-local map tasks	10	10	12
Total time spent by all maps in occupied slots (ms)	10883760	17208480	23011920
Total time spent by all reduces in occupied slots (ms)	1140768	6461472	7631424
Total time spent by all map tasks (ms)	226745	358510	479415
Total time spent by all reduce tasks (ms)	11883	67307	79494
Total vcore-milliseconds taken by all map tasks	226745	358510	479415
Total vcore-milliseconds taken by all reduce tasks	11883	67307	79494
Total megabyte-milliseconds taken by all map tasks	3.48E+08	5.51E+08	7.36E+08
Total megabyte-milliseconds taken by all reduce tasks	36504576	2.07E+08	2.44E+08
Map-Reduce Framework			
Map input records	3717965	3717965	3717965
Map output records	1487600	1487600	1487598
Map output bytes	21337233	21337233	21337209
Map output materialized bytes	5772135	5866605	6065767

Input split bytes	880	880	1056
Combine input records	0	0	0
Combine output records	0	0	0
Reduce input groups	94430	94430	94430
Reduce shuffle bytes	5772135	5866605	6065767
Reduce input records	1487600	1487600	1487598
Reduce output records	94430	94430	94430
Spilled Records	2975200	2975200	2975196
Shuffled Maps	10	30	60
Failed Shuffles	0	0	0
Merged Map outputs	10	30	60
GC time elapsed (ms)	5198	9104	11969
CPU time spent (ms)	90640	112050	128860
Physical memory (bytes) snapshot	6.49E+09	7.22E+09	9.02E+09
Virtual memory (bytes) snapshot	3.8E+10	4.73E+10	6.33E+10
Total committed heap usage (bytes)	5.51E+09	6.23E+09	7.52E+09
Shuffle Errors			
BAD_ID	0	0	0
CONNECTION	0	0	0
IO_ERROR	0	0	0
WRONG_LENGTH	0	0	0
WRONG_MAP	0	0	0
WRONG_REDUCE	0	0	0
File Input Format Counters			
Bytes Read	6.58E+08	6.58E+08	6.58E+08
File Output Format Counters			
Bytes Written	1333050	1333050	1333050
High-Level Application History Duration	2.5 min	1.7 min	1.4 min
Cost (Amazon EC2 Price + Amazon EMR Price) $0.10 per hour for m4.large instances, $0.03 per hour for EMR. We are going to multiply the EC2 price by number of nodes.	(2.5/60) *(($0.10 *2) + $0.03) = $0.00958333333	(1.7/60) *(($0.10*3)+ $0.03) = $0.00935	(1.4/60) * (($0.10 *4) + $0.03) = $0.0100333333

Program B Analysis between EMR Configs

In our MapReduce jobs for Program B, where we aggregate engagement metrics per twitter user, as we go from a 2-node configuration to 3 nodes and 4 nodes, the duration of our job in real time decreases from 2.5 minutes for 2 nodes, to 1.7 minutes for 3 nodes, and 1.4 minutes for 4 nodes. We again assume the execution time for our job decreases due to the nature of Hadoop, where the master node partitions our input into blocks, our 3 million records of tweets, and distributes these blocks to our worker nodes for processing. Since we increase our number of resources/worker nodes to process our computation, it only makes sense that our computation gets finished much faster, but it seems that our runtime improves a lot from 2 nodes to 3 nodes, but there is a much smaller improvement from 3 nodes to 4 nodes. Although the duration of our program decreases as we increase the number of nodes, we end up spending more money when we use 2 nodes or 4 nodes versus when we use 3 nodes. 2 nodes cost us about $0.0096, 3 nodes cost us about $0.0094, and 4 nodes cost us about $0.0100, using 3 nodes was most cost effective for running Program B if we only consider the time elapsed of running the job. We can also see that, when we add more nodes, our CPU time spent increases. This is also explained by adding more nodes, since the more nodes we have, the more CPU utilization due to more resources being available, and so the more time spent by CPUs processing instructions.

Program A & Program B Analysis between EMR configs

Between Program A, where we count total Twitter mentions of stock tickers, and Program B, where we aggregate engagement metrics per Twitter user, they exhibited similar characteristics regarding our Hadoop execution metrics. Both programs were executed faster as we increased the number of nodes, and both programs seemed to be most cost effective when ran using 3 nodes. Both programs also had their CPU time increase as we went from 2 nodes to 3 nodes to 4 nodes, which makes sense as since we had more resources, we had more CPUs processing our computations, increasing CPU time. But, one difference between the two was, Program A took longer than Program B in all node configurations. This is likely due to the runtime complexity of our implementations of Program A versus Program B, where Program B has a much faster runtime than Program A.

d. **List any references you used in performing these tasks (just a list of URLs would do). Pease remember that using any resources without citation constitutes plagiarism.**
    a. https://www.youtube.com/watch?v=p_l_Rltb7WU&ab_channel=SoumilShah - used this to use the step execution mode, we verified that our setup worked in cluster mode for emr before using this. This saved us a lot of time versus using ssh with the master node since we had to repeatedly pull our mapreduce programs/input file, make sure these files were readable/executable, and then run a series of commands, whereas the step execution mode integrated seamlessly with our S3 buckets containing our files and automatically generated the same outputs.
    b. https://levelup.gitconnected.com/map-reduce-with-python-hadoop-on-aws-emr-341bdd07b804 - used this guide initially to figure out how to make use of our master node and run jobs through emr