CS553 Programming Assignment 2

Mazen Ataya- Alaa Ayach

A20317925 - A20317680

1. Virtual Cluster

For virtual cluster we used EC2 c3.large instance, we used Ubuntu OS.

First we used spot instances as they are cheaper, however when some time passed, the node was revoked and we used on-demand instances instead

2. Shared Memory:

For shared memory, we implemented a java multithreading program that perform word counts.

The number of the threads is an input for the program

We divided the input between these threads by a the shell script:

split -d -b 10m wiki10gb split-

We did the following to write the code (***following one of the best practices tutorials for using Java for shared memory***):

- each thread will take its part and perform the word count for that part;
- We kept tracking parts in the sense no part will be taken twice;
- each thread will have local hash table for the words that finds, the value if the hash table will be the word count;

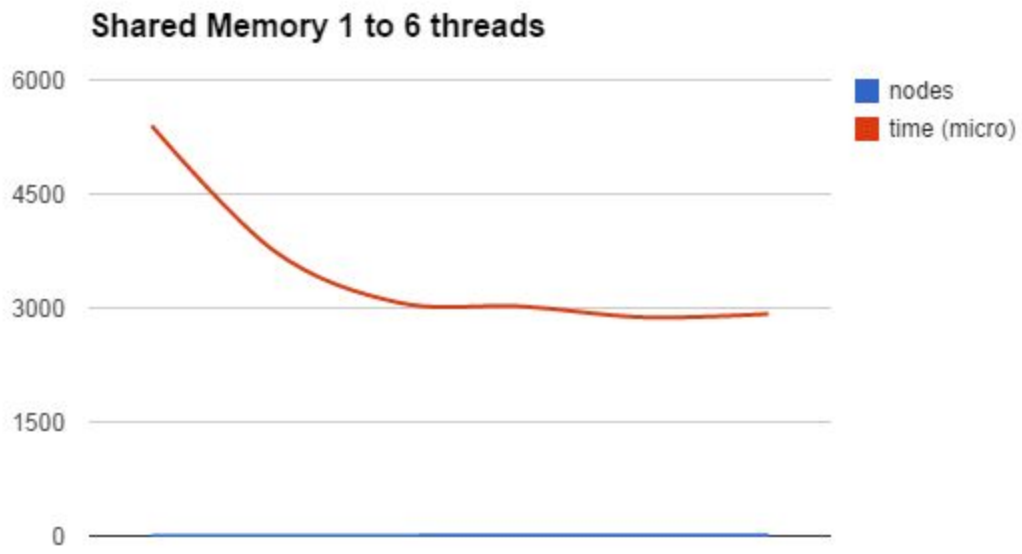
- We will have a one sync data structure shared between all threads, we should use the sync keyword for accessing this data structure from all threads when they merge the counts into it
- Merging will be when we have the same index and the value will be the addition

.

<u>Java program is inclosed and instructions are put in the manual file</u>

Experiments (we stopped at 6 as the time is fixed from 6 to 8 nodes):

Threads	time (micro)
1	5400
2	3743
3	3071
4	3021
5	2877
6	2922



This chart is for the small dataset.

We notice that there is an improvement when increasing the number of the threads, The performance jumped greatly from one thread to two threads as the node has two processors, and began to be fixed when increasing for more than two threads

3. Virtual Cluster of 17 nodes:

We set up a virtual cluster of 17 nodes on amazon EC2. We made one node for control and the other 16 nodes for computing. We used the instance C3.large as mentioned in the write up. Each node of them runs Ubuntu server 14.04. We created the cluster using our preconfigured AMIs. We also created a S3FS shared file system using S3 bucket.

You can find screeshots of our cluster attached to the project.

4. Hadoop

We used hadoop 1.1.2 for the configuration

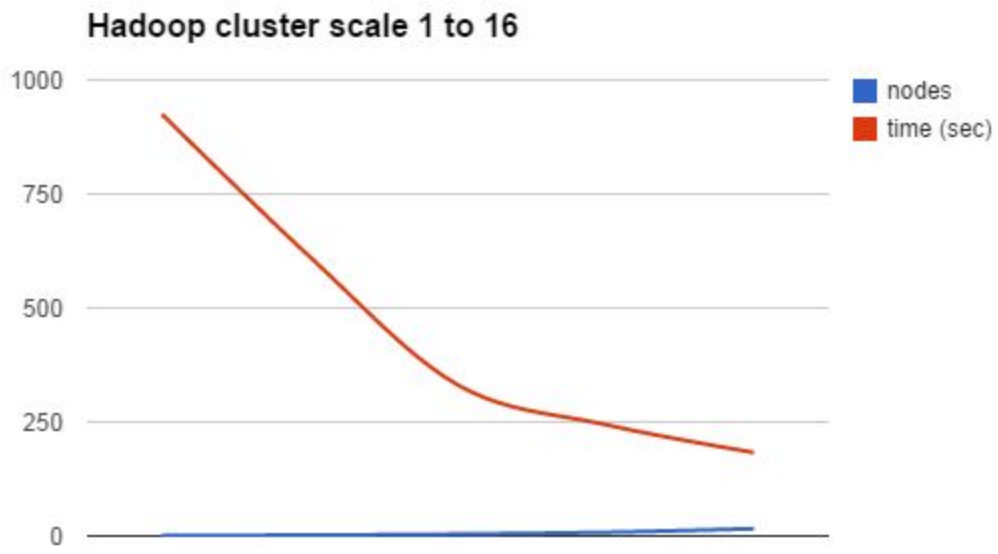
We configured dataNodes and two kinds of master nodes: nameNodes and SecondaryNodses

The dataset will be distributed between all nodes in the cluster and the namenode will be responsible for knowing how to distribute the data by having all IPs for the slaves

A jobtracker is configured in the nameNode and for each datanode there is a tasktracker

Results:

nodes	time (sec)
1	925
2	612
4	331
8	244
16	183



We see increasing in the performance, however , we don't notice a fully scalable result as the time taken to distribute the data

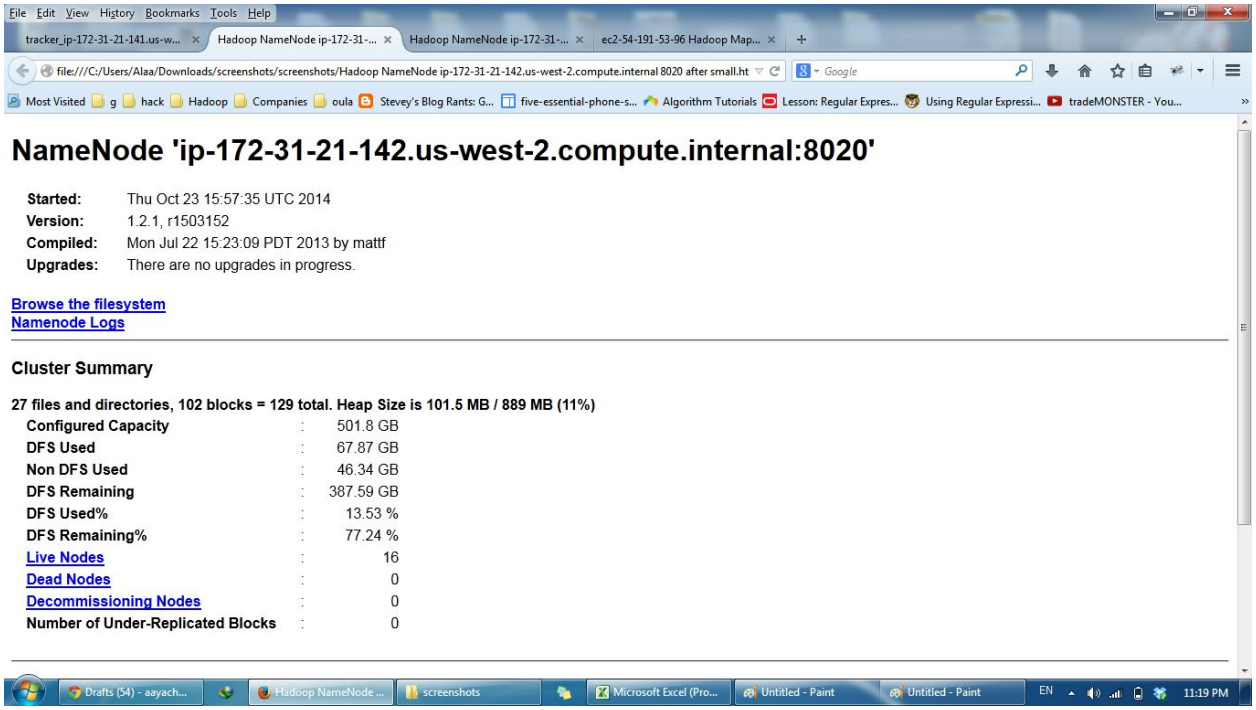
To change the number of the workers, you should put in slaves file only the workers that you need them to participate, and in hdfs file we should put the new number for replication property

I did the configuration as it mentioned in this tutorial which I put it in this Piazza post, the tutorial is providing scripts for configuring all nodes , I put the reorganized scripts in bash files for the complete configuration,

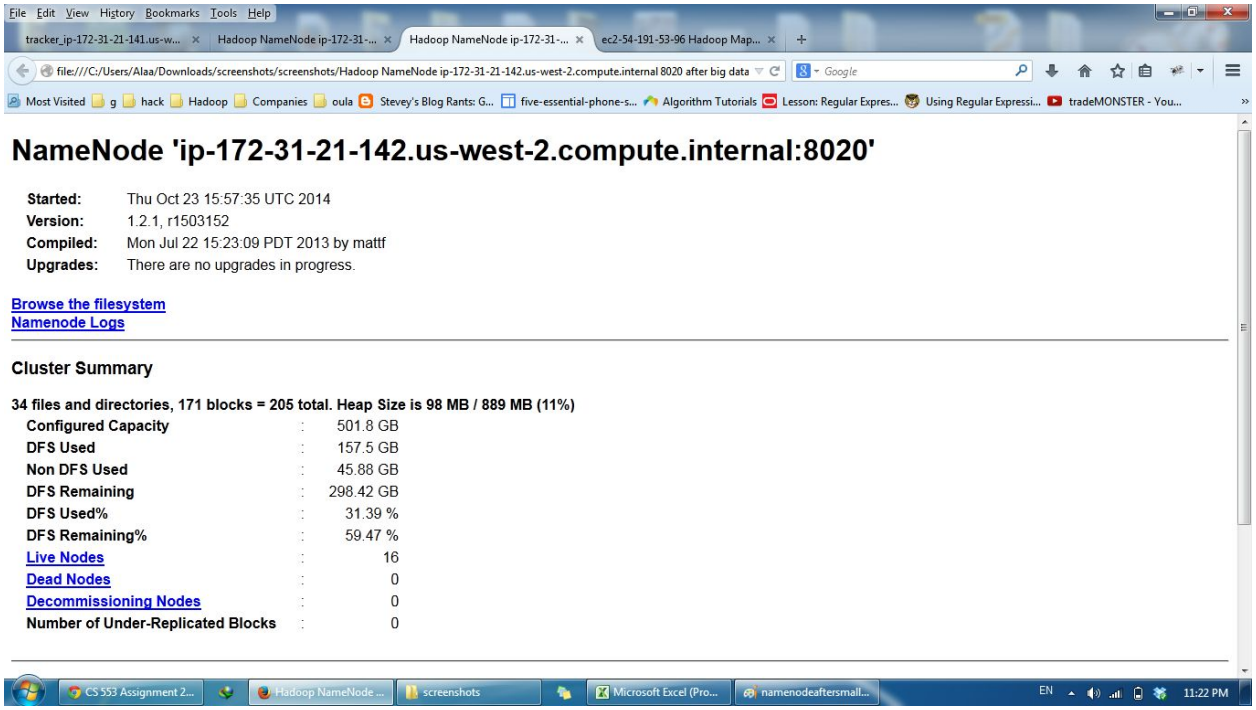
See manual for the files

Screen shots

Namenode after the small dataset



after the big dataset. Notice the usage for the DFS increasing



NameNode 'ip-172-31-21-142.us-west-2.compute.internal:8020'

Started: Thu Oct 23 15:57:35 UTC 2014
Version: 1.2.1, r1503152
Compiled: Mon Jul 22 15:23:09 PDT 2013 by mattf
Upgrades: There are no upgrades in progress.

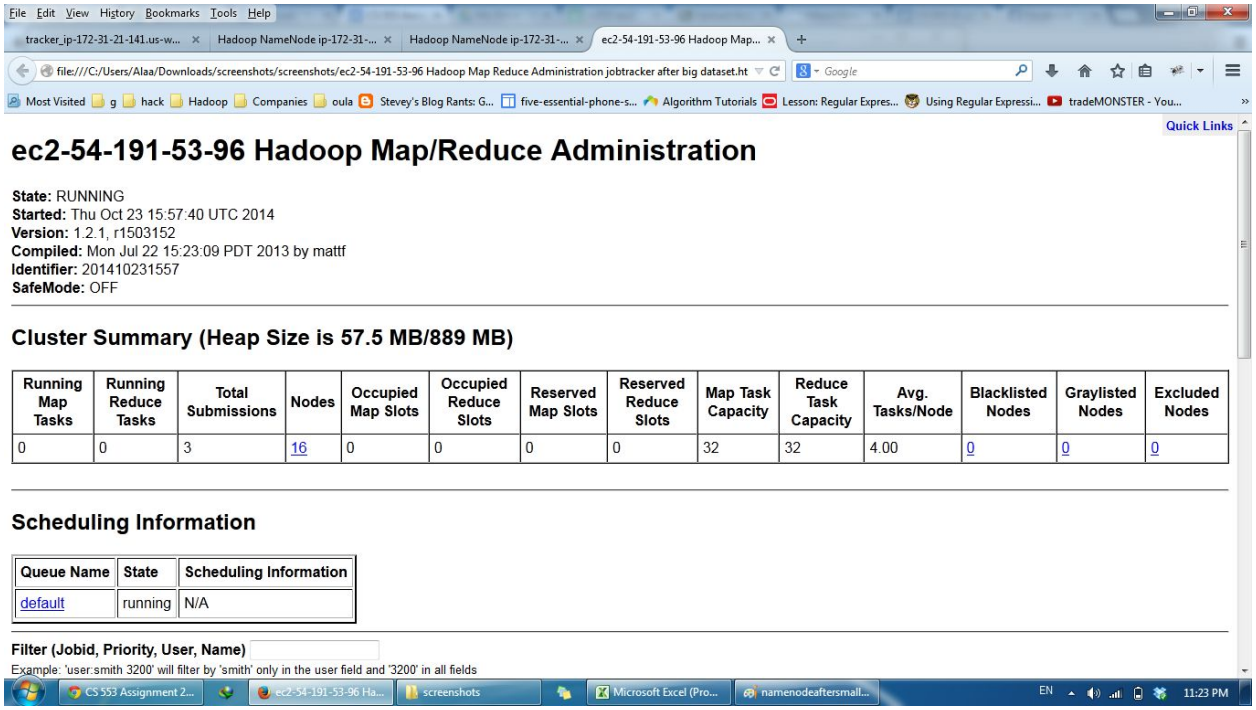
Browse the filesystem
Namenode Logs

Cluster Summary

34 files and directories, 171 blocks = 205 total. Heap Size is 98 MB / 889 MB (11%)

Configured Capacity	:	501.8 GB
DFS Used	:	157.5 GB
Non DFS Used	:	45.88 GB
DFS Remaining	:	298.42 GB
DFS Used%	:	31.39 %
DFS Remaining%	:	59.47 %
Live Nodes	:	16
Dead Nodes	:	0
Decommissioning Nodes	:	0
Number of Under-Replicated Blocks	:	0

JobTracker



ec2-54-191-53-96 Hadoop Map/Reduce Administration

State: RUNNING
Started: Thu Oct 23 15:57:40 UTC 2014
Version: 1.2.1, r1503152
Compiled: Mon Jul 22 15:23:09 PDT 2013 by mattf
Identifier: 201410231557
SafeMode: OFF

Cluster Summary (Heap Size is 57.5 MB/889 MB)

Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Occupied Map Slots	Occupied Reduce Slots	Reserved Map Slots	Reserved Reduce Slots	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes	Graylisted Nodes	Excluded Nodes
0	0	3	16	0	0	0	0	32	32	4.00	0	0	0

Scheduling Information

Queue Name	State	Scheduling Information
default	running	N/A

Filter (Jobid, Priority, User, Name)
Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

5. Hadoop Wordcount

We used ***two versions*** or wordcount, the first one implemented by hadoop and we put results from it, and we reimplement the wordcount to get better result as for the separator of words in calling split function.

The results are shown in the the previous graphs.

A java file is inclosed in the submission.

6. Swift

We installed swift on a cluster of 17 nodes. One Headnode and the others are 16 worker nodes. All the nodes run Ubuntu Server 14.04 LTS. The Headnode is of type M3.large and the worker nodes are of type C3.large.

We followed the tutorial that was on the write up of the assignment, We had some difficulties especially when we tried to attach extra volume to the nodes because they only come with 8 GB of Disk memory.

We created a shared file system using S3 bucket and automatically mounted it to the worker nodes using a start-up script that we created, “s3fs_mount.sh” and we also mounted the shared file system to the Headnode.

When the shared file system was ready we started to test some swift programs to make sure that everything was working. When we ran swift on my local machine (e.g Headnode), it was working properly, but when we run swift on my cloud, the headnode was able to submit the jobs but there was no response from any worker. After some troubleshooting and debugging, it turned out that there was some issue with the firewall, after fixing it we were able to run all the swift

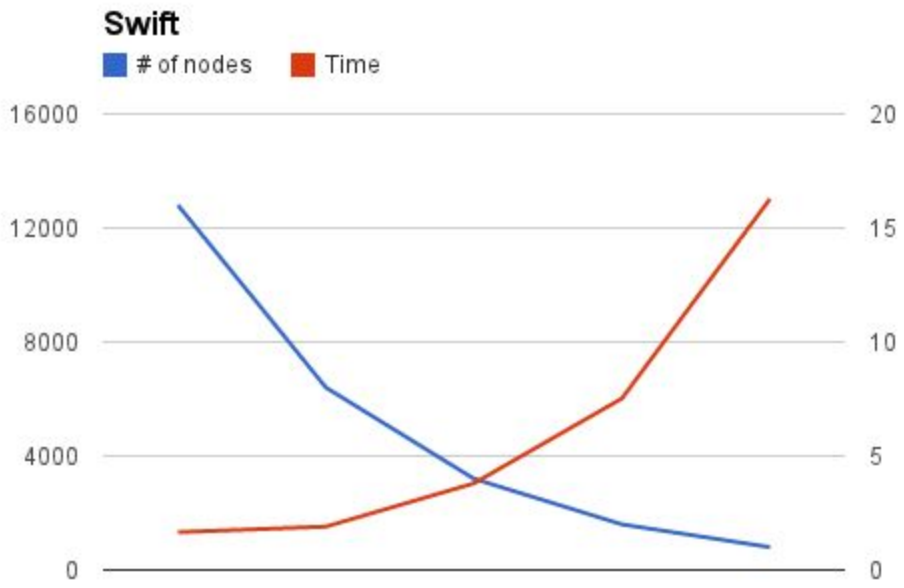
programs that come with the tutorial successfully and you can find some screenshots attached to the project.

7. Swift Word Count:

We implemented the word count using swift and We saved the output of the program in a file called wordcount-swift.txt. You can find the first MB of the file attached to the project.

We also evaluated our program against strong scaling of 1,2,4,8, and 16 nodes.

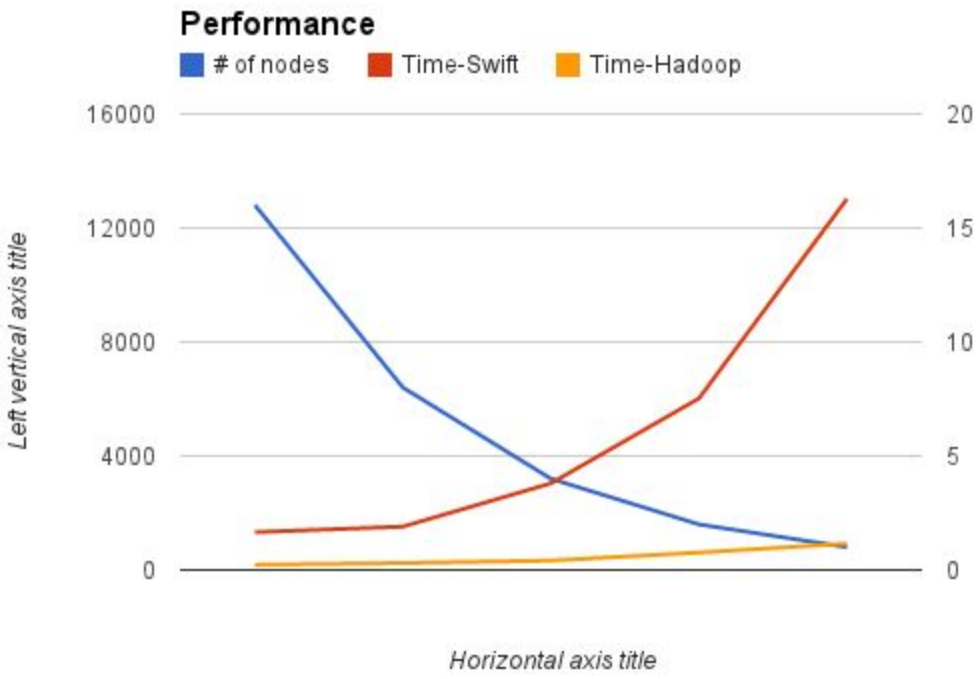
# of nodes	Time(s)
16	1326
8	1522
4	3040
2	6020
1	13021



8. Performance:

Our data shows that the best for a cluster of one node is the shared memory with a number of threads that is equal to the number of cpu cores. Our results also show that hadoop is much better at solving these kinds of problems than swift. And this is because the problem is a map/reduce by nature and hadoop is specialized at solving these kinds of problems.

Because hadoop is always better than swift in our results then we conclude that hadoop is going to be better at all scales including a 100 and 1000 nods scale.



9. Sorting on shared memory

The same program is used to implement the sorting, however, without using the counts.

So we used the dictionaries in this case to match the words (merge sort like)

We output the final file using the final dictionary.

10. Sorting with Hadoop

As Hadoop itself sort the words, when we used the same file for wordcount with deleting the line of omitting the count, we will get an output file with the whole words sorted by Hadoop.

A file is enclosed.

11. Sorting in Swift

You can find the output and the source code attached to the project.