Cassandra Cluster Analysis.

Dongyao Wang (dowa4431@colorado.edu), Sandesh Dhawaskar Sathyanarayana (sadh0344@colorado.edu), Sananda Banerjee(Sananda.Banerjee@colorado.edu)

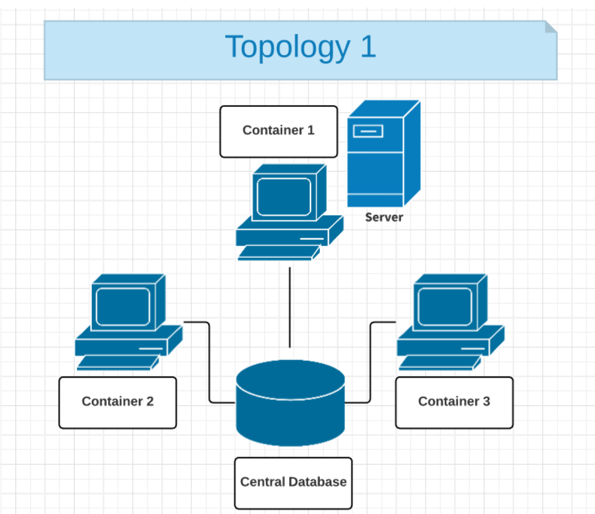
1. Introduction

Clock Synchronization is one of the major challenge and issue in the Distributed Data Bases like Cassandra. NTP is one of the main tool used for the clock synchronization. NTP measure the clock difference and use the same to either decrease or increase the clock speed in the individual node/machine. This project has two major phases with two major analysis. In the first phase we look at the analysis of the delay, offset of the clock with NTP servers and in the second phase we measure the read/write latencies of the distributed data base. Cassandra is a key-value store distributed database, essentially a giant hash table. Hence, we choose the Cassandra for our analysis. We also aim at identifying and developing tools and methodologies to facilitate running and testing of Cassandra Cluster, across various topologies and network conditions. We gained experience with the Apache Cassandra database. There are three main topologies that have been implemented and we will look at each one of them in detail in the following sections.

2. Topology 1

2.1 Description – setup, configuration

For the first topology we had to setup three

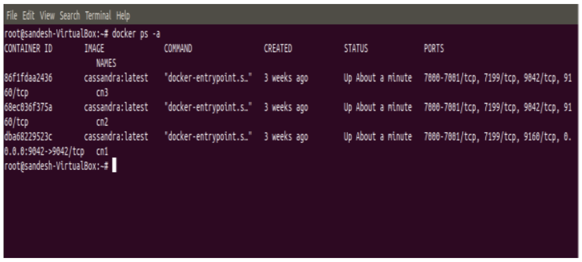


Fig(1)

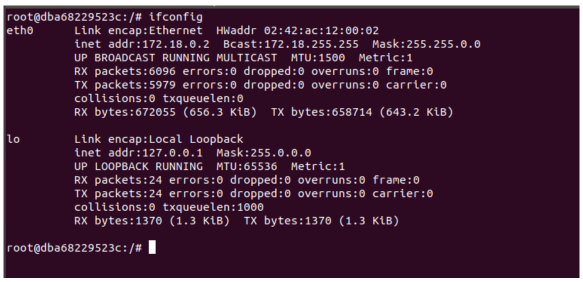
machines on the same device. So, we created three containers called Cassandra nodes and named them cn1, cn2 and cn3 as can be seen in the figure Fig (1). We established this arrangement using docker on the virtual machine of the same computer.

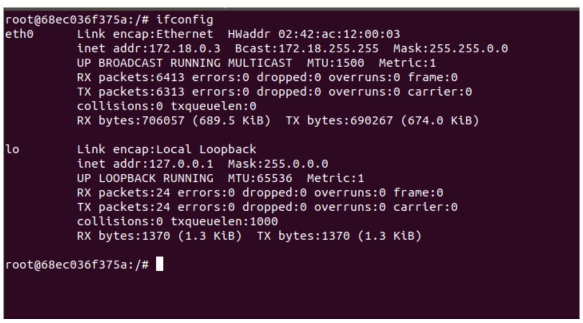
To test if the three instances are up and running, we used the command node tool status that shows the status of the machines (containers). Moreover, we opened three separate tabs for the three containers and tried pinging between them. We used the command ifconfig to determine the IP address of the current machine and accordingly pinged to the other two nodes. The nodes could successfully ping each other, sending and receiving packets. This confirms that Cassandra cluster has formed.

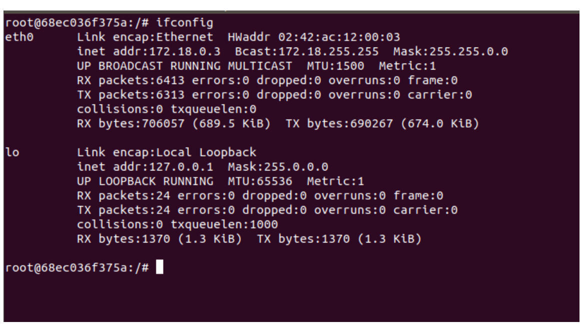
The below snapshot shows that three containers are up and running.



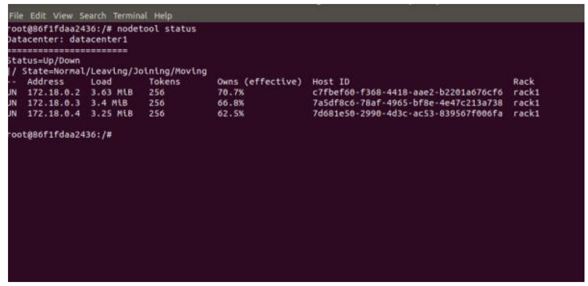
After this you should login to each container and see if each of them has got different IP address. Below snap shots give that information.



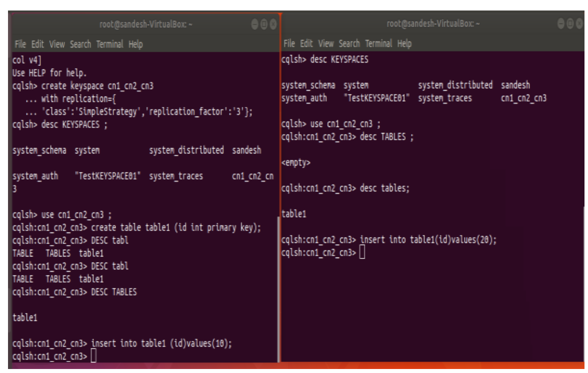


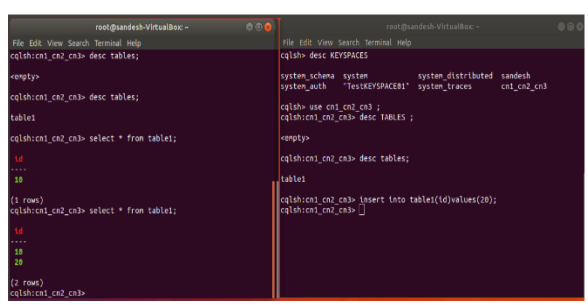


After this you should login to machine one of them and check if cluster has formed. Also, they will show the different IP's we showed above.



Next, we took the testing it a step forward, by exploring the Cassandra Query language, also known as the CQL. Inside cqlsh of one node (say cn1), we created a key space and inside the key space we created tables where we inserted and viewed values. Then, we looked up key spaces from a second node (say cn2) and we could view the same table from cn2 as well. We could insert and update values on the table and the changes would be reflected in the table and could be verified from the other nodes. Similarly, we tested with cn3 and obtained similar results. Following snapshots will describe the example explained above.



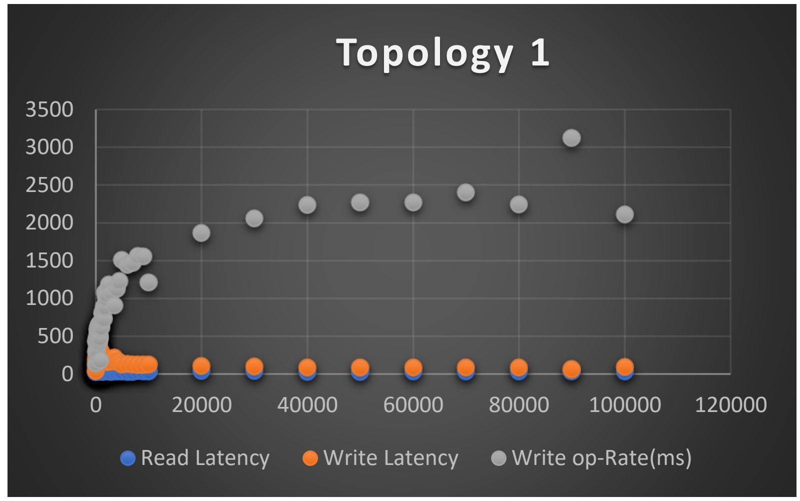


After this, we used the Cassandra stress tool to generate statistics to measure the latencies for read and write operations as can be viewed in the table below. We obtained interesting results, for various iterations of n. As we incremented the number of iterations, the latency would increase both for read and write as can be seen from the grey dots in the graph below.

2.2. Table:

Iterations	Read Latency	Write Latenc	Write op-Rate(ms)
10	28.54	35.9	141
50	26.28	109.3	293
100	46.73	194	429
150	62.8	218.3	444
200	68.68	342.7	317
250	42.75	281.1	501
300	81.13	264.3	597
350	90	270.9	565
400	75	320.2	482
450	97	295.1	561
500	83.17	344.3	631
550	78.04	233.1	565
600	70	382.5	420
650	78	261	653
700	68.76	255.5	452
750	88.13	325	503
800	92	190	477
850	44	267	182
900	64	220.6	679
950	57	206.9	624
1000	67	251.3	656
1100	58	286.3	618
1200	26	215.6	811
1300	49	229.6	797
1400	32	224.4	750
1500	40	230.7	727
1600	44	211.7	893
1700	37.42	201.7	887
1800	36	173.2	1074
1900	54	166.5	1009
2000	43	159.2	1054
2500	34.15	162.8	1179
3000	28.46	168.2	1069
3500	39	213.8	907
4000	33	163.8	1125
4500	36	159.7	1229
5000	34	128.4	1510
6000	26.25	132.9	1444
7000	31	127.6	1470
8000	36	120.3	1561
9000	33	122.4	1556
10000	32	127.8	1210
20000	38	106.3	1866
30000	37	96.4	2059
40000	31	88.9	2236
50000	31.23	87.8	2267
60000	33.2	87.9	2269
70000	34.5	82.9	2404
80000	31.33	88.8	2245
90000	33.7	63.6	3126
100000	31.62	94.7	2107

2.3. Graph:



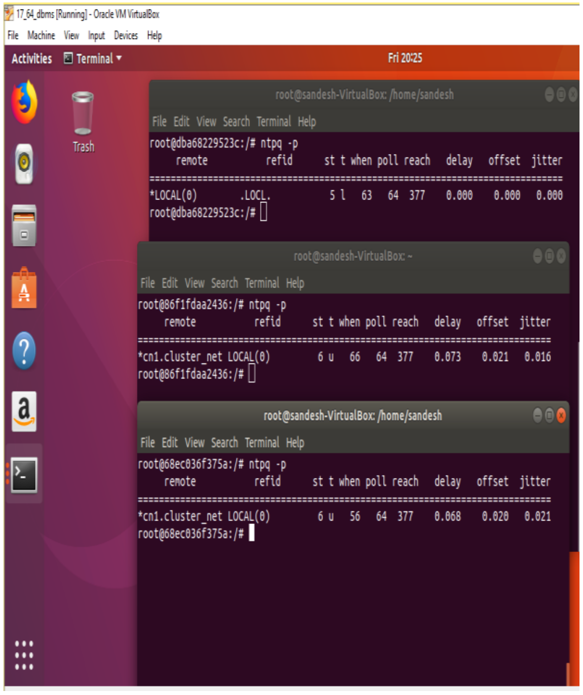
2.4 Observations:

As can be observed from the graph plotting and the values of read and write mean latencies in the table, the op rate keeps increasing linearly with increase in the number of iterations, as can be seen in the grey dots on the graph above. Another major observation being done here is, since all the three containers sit on one Linux machine, no matter how you change the date and time of each node they will always get set to underling Linux machine date and time.

2.5 Technical Challenges:

- Formation of the cluster with different IP's for each container
- NTP configuration of each node to make cn1 as master node and other two as slaves
- Figuring out how to login to cqlsh with the given IP.

Following snapshots shows the NTP configurations that was used in this topology. CN1 is the server and cn2, cn3 are the clients.

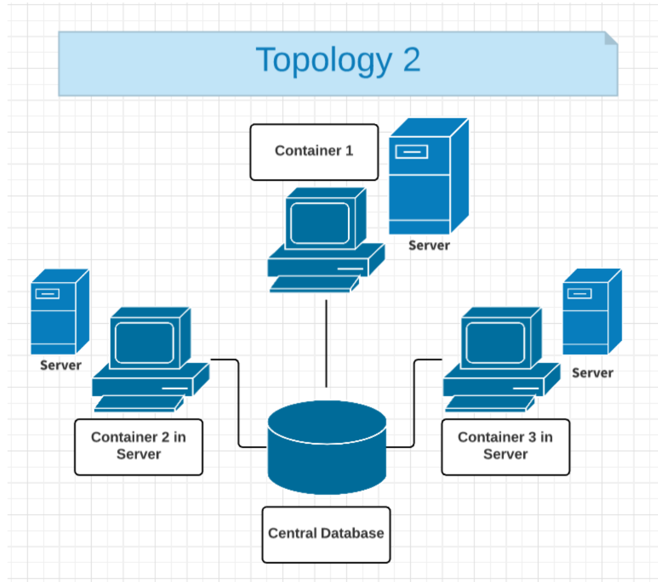


3. Topology 2

3.1. Description – setup, configuration

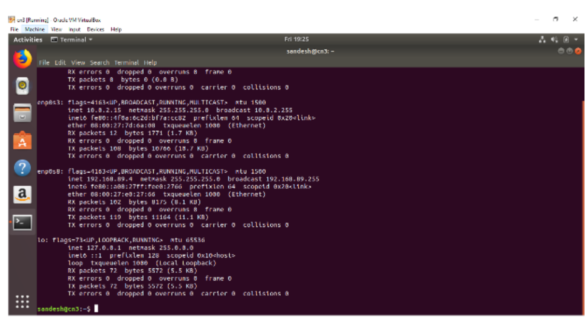
For the second topology, all the Cassandra nodes are locally on the same host, but in separate virtual machines and a single NTP server. In this case each VM runs a single instance of a Cassandra node, using docker containers. So, now we setup two individual containers using docker on the different instances of VM and formed a cluster with a single NTP server.

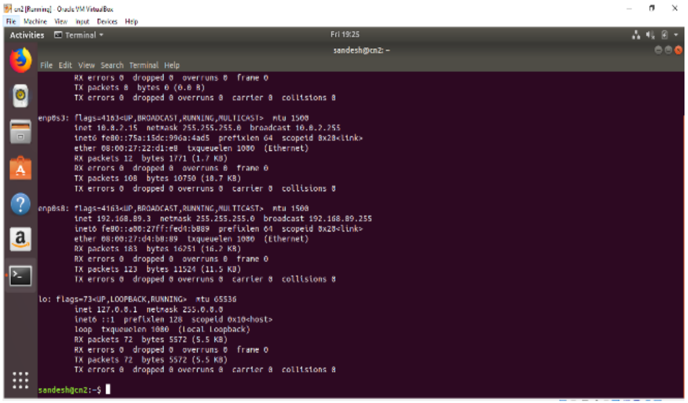
Then, using commands like nodetool status we tested the current state of all the nodes across the two instances of VM. Then tried pinging between them, sending and receiving packets. Once ping was successful, we tested the status using ntpq, by setting on node as the NTP server or master node, and the other two nodes in the other two VM instances as NTP clients or slave nodes.

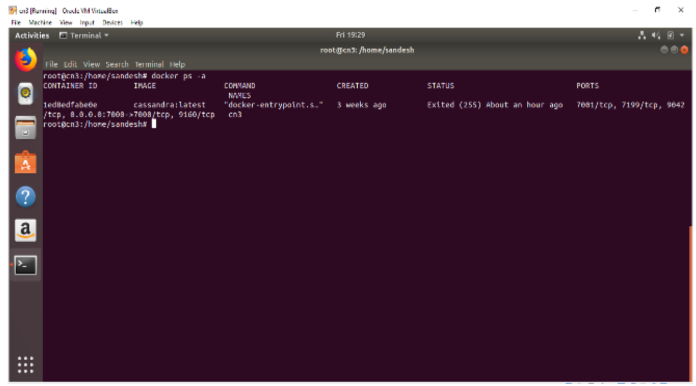


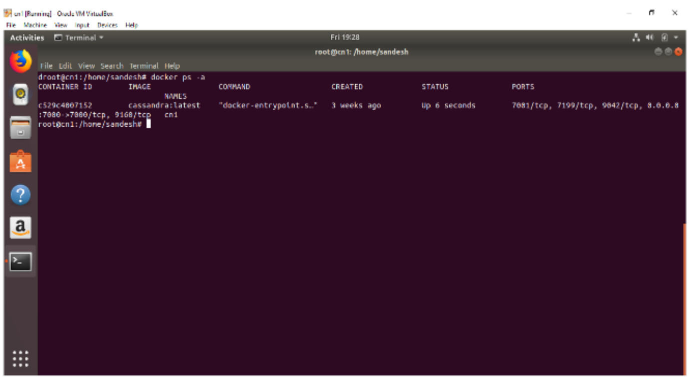
Following this, we carried out the Cassandra stress test, measuring read write latency values for incrementing number of iterations. We started testing for n = 10, where n is the number of iterations and went on until n = 100000. The result of the observations of the measurements is discussed in the subsequent sections.

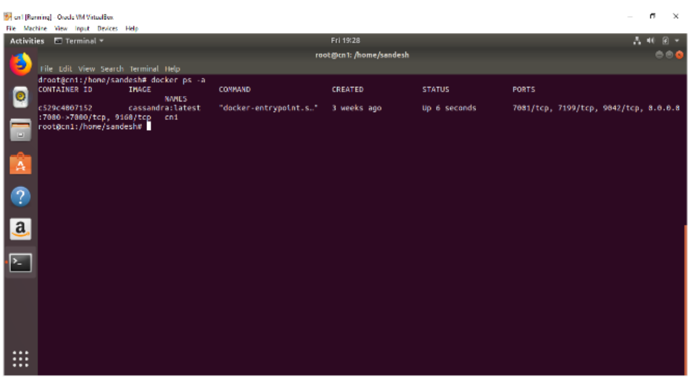
The following snapshot gives the IP's of each host and show that containers are running.

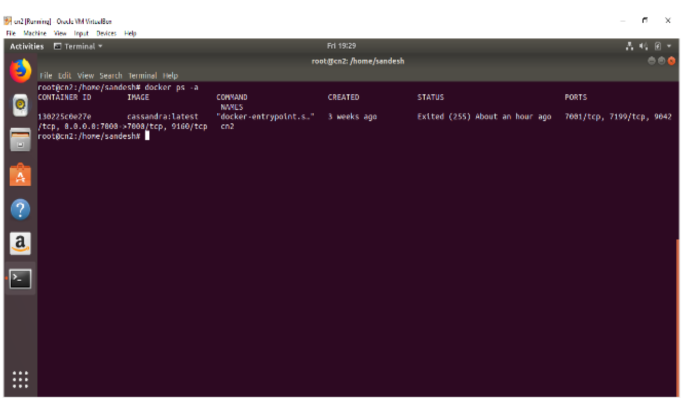






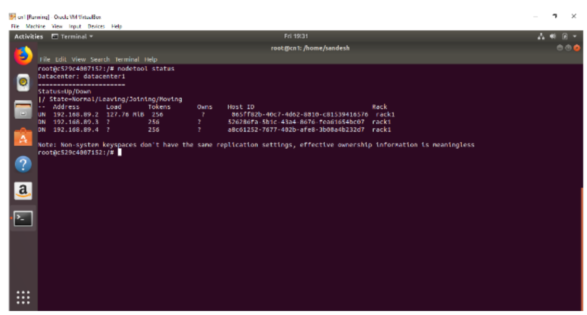




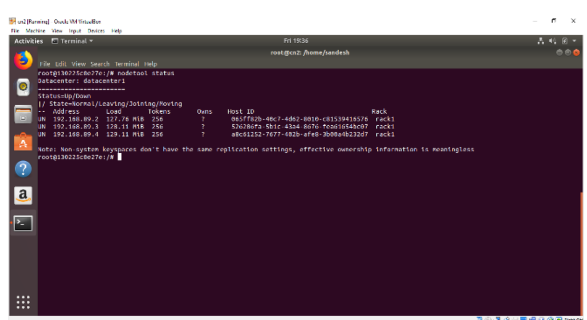


Now we have containers up and running and now lets login and check if cluster is formed well. When we login after bringing the container 1 , we see cluster node 2 and node 3 are down but after starting and running the nodes they are up

Before:



After all the nodes are up:



After this database was created and entry test was done as we did in topology 1.

3.2. Tables:

- Write

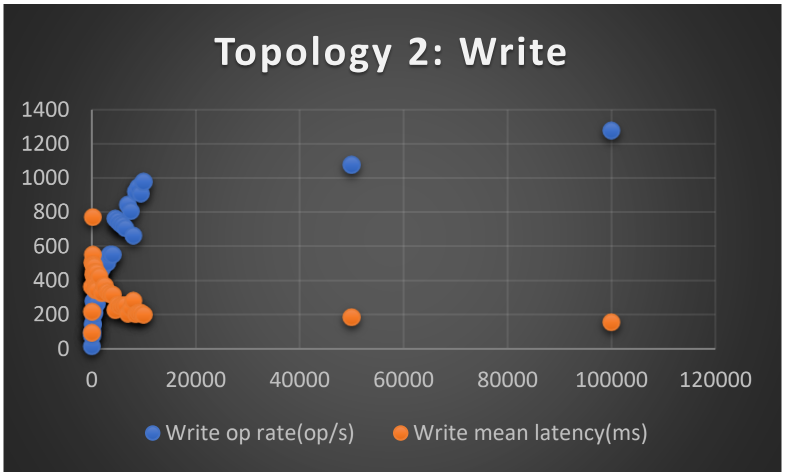
Iteration	Read op rate(op/s)	Read mean latency(ms)
10	12	122.4
50	46	109.4
100	119	235.2
150	201	133.5
200	170	159
250	180	418.6
300	472	606.4
350	247	243
400	293	93
450	335	371.8
500	381	135.8
550	441	436.3
600	385	82.8
650	523	157.3
700	627	177.8
750	794	134
800	688	193.2
850	619	188.2
900	699	680.4
950	498	103.7
1000	419	264.8
1500	656	133.5
2000	1284	909.4
2500	715	466.1
3000	1548	531.8
3500	1350	492.9
4000	1255	703.8
4500	1091	678.6
5000	825	909.1
5500	1171	735
6000	1427	500.7
6500	1271	550
7000	1470	625
7500	1090	505.4
100000	1681	532.6

-Read

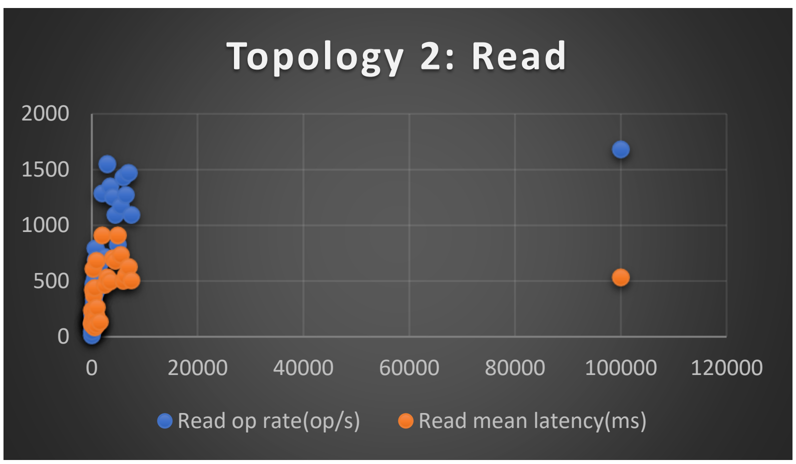
Iterations	Write op rate(op/s)	Write mean latency(ms)
10	13	93.5
50	76	215.6
100	94	363.1
150	100	503.7
200	146	551.6
250	140	770.2
300	278	492.6
350	196	433.9
400	216	454.2
450	210	437.1
500	211	482.2
550	262	404.4
600	255	448.3
650	387	416.8
700	421	342.5
750	357	433.1
800	294	399.1
850	309	381.1
900	396	354.2
950	264	447.9
1000	424	344.1
1500	368	426.3
2000	467	326.2
2500	517	368.8
3000	504	334.5
3500	549	320.8
4000	549	314.5
4500	762	227.7
5000	749	257.9
5500	731	247
6000	723	255.6
6500	706	255.4
7000	841	206.4
7500	804	237.3
8000	662	280.8
8500	919	201.1
9000	946	210.2
9500	907	211.8
10000	976	198.4
50000	1077	183
100000	1277	154.9

3.3. Graphs:

-Write



-Read



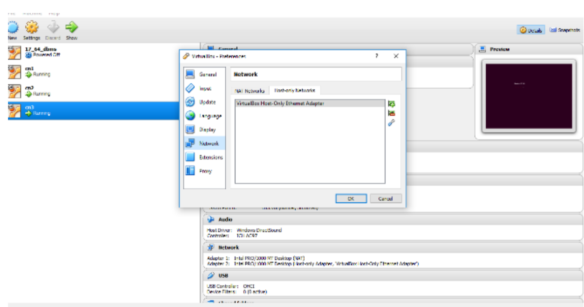
3.4 Observations:

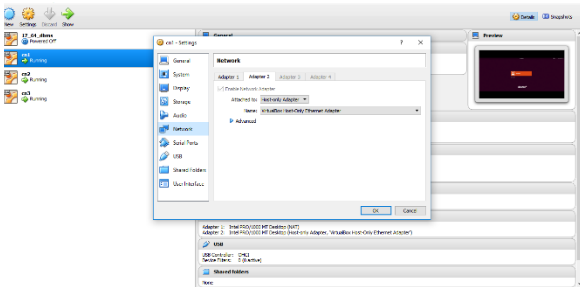
As can be observed from the tables and graphs in the previous sections, the curve rises exponentially with the increase in the number of iterations, much like the first case (topology 1). However, it is interesting to observer that. Second major observation that could be done is regarding the NTP. Now each docker is one each of the VM and hence their time and date would be set on basis of underlying Linux. Hence, we formed underlying Linux kernel as the NTP servers of first node and other two VM kernel machine as the clients of the master for NTP synchronization.

3.5 Technical challenge:

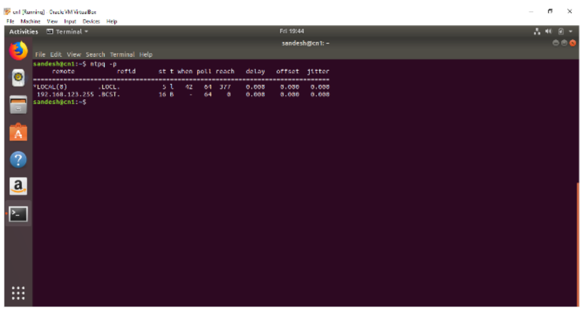
- Formation of the cluster is very difficult task when we have machine one different VM.
- We had to make VM to use host-only machine adaptor and then we had to assign the static IP to each of the machine.
- Once the IP was set we had to form the cluster using the docker.
- Then pinging each machine and forming the cluster.
- Formation of cluster is much harder than the one in topology 1.

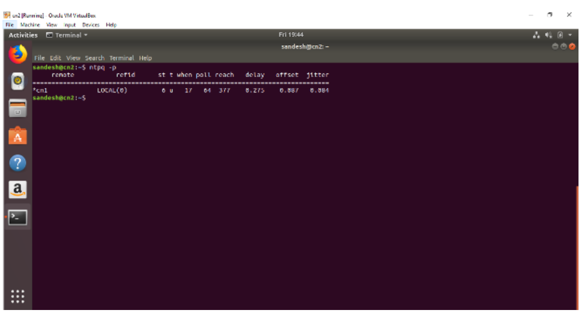
Below snapshots give the idea how we achieved the configurations.

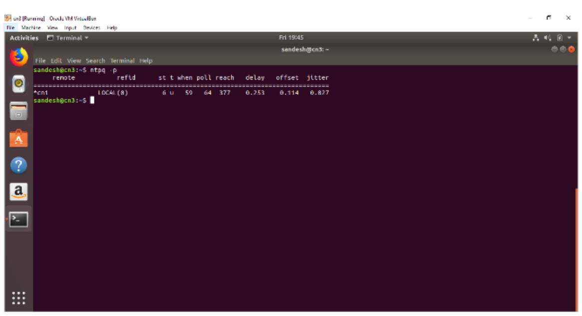




Below are some of the NTP configurations snapshots which shows that cn1 was the master and cn2, cn3 were the slaves.





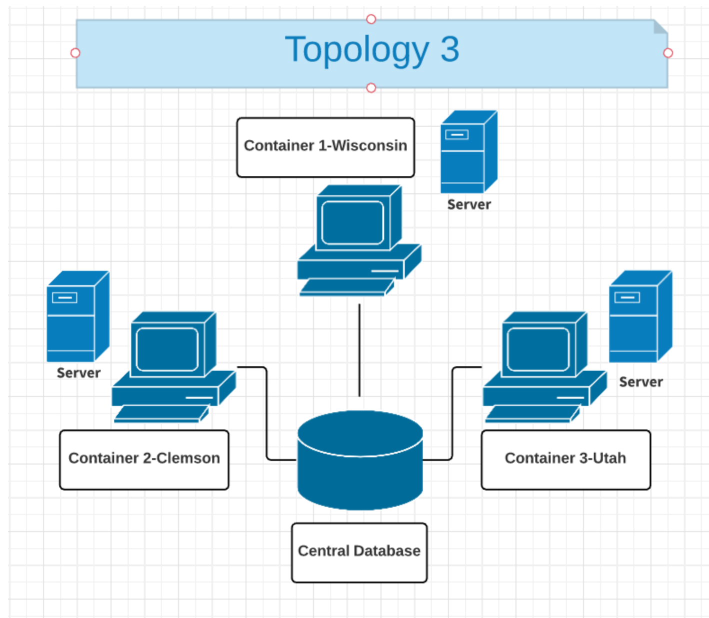


We have not added any database result snapshots as they are exactly as similar to one attached in topology 1.

4. Topology 3

4.1 Description – setup, configuration

For the third topology, we are required to create a cluster with geographically distant Cassandra nodes. To establish this, we used the CloudLab to set up nodes in the cloud and used our device as physical hosts (with nodes on instances of VM) to connect to the nodes in the cloud. As can be seen in the diagram below, all the three nodes are geographically distant from one another, at Wisconsin, Clemson and Utah respectively.



Like the previous two topologies, here also we used node tool status to check the current state of the machine. Then using Network Time Protocol, we set a node as the server and the other two as clients, using the Python codes for server and client, adjusting the IPs to establish the network.

After this, we used Cassandra stress test to overload the network with read and write operations and took measurements for various iterations of read and write operations.

4.2. Tables:

- Write

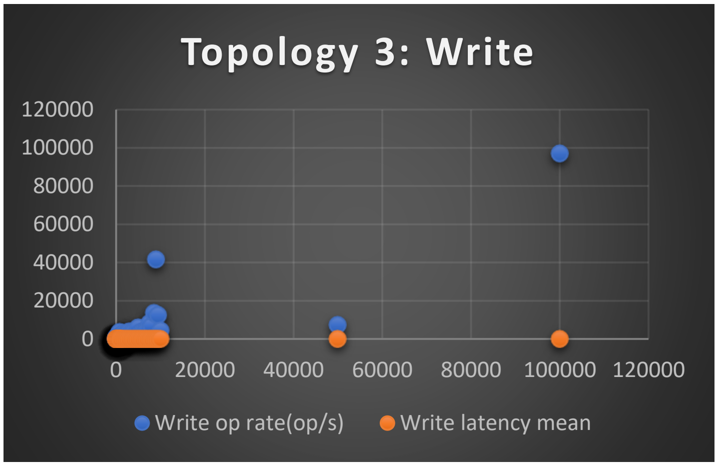
Iterations	Write op rate(op/s)	Write latency mean
10	78	12.1
50	258	16.2
100	943	37.5
150	291	48.1
200	712	34
250	797	30.8
300	1555	46.3
350	508	42.3
400	419	28.6
450	463	39.5
500	688	28.6
550	879	12.9
600	554	19.2
650	1019	41.4
700	957	12.3
750	1883	12.2
800	3663	27.8
850	813	38.2
900	1521	27.4
950	996	37.8
1000	831	37.6
1500	3103	27.1
2000	2605	7.8
2500	2582	37.4
3000	3918	26.4
3500	3552	36.9
4000	3560	37
4500	2875	36.8
5000	6083	26.2
5500	3335	26.2
6000	3829	37.4
6500	3989	26.2
7000	5050	26.3
7500	8172	4.2
8000	6964	26.8
8500	13632	2.6
9000	41467	2.6
9500	12358	2.5
10000	4440	26.6
50000	7363	26.4
100000	97056	1.5

- Read

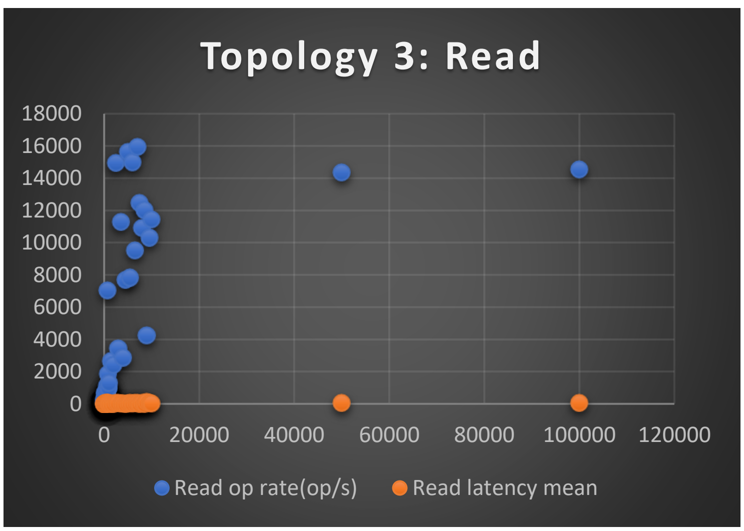
Iterations	Read op rate(op/s)	Read latency mean
10	286	4.5
50	74	4.6
100	248	2.7
150	652	1.4
200	251	2.8
250	650	1
300	434	37.5
350	323	26.5
400	497	1
450	534	26.5
500	505	29
550	788	38
600	657	36.9
650	1118	0.8
700	896	26.1
750	7029	39.1
800	991	38.9
850	1839	26.8
900	1004	26.1
950	967	31.3
1000	1295	29
1500	2643	0.5
2000	2407	0.5
2500	14934	41.8
3000	3440	39.6
3500	11287	28.9
4000	2852	27.2
4500	7682	0.4
5000	15618	28.2
5500	7827	37.7
6000	15001	26.9
6500	9505	37.3
7000	15938	38.9
7500	12447	0.3
8000	10901	37.2
8500	12006	0.2
9000	4222	113.4
9500	10304	39.9
10000	11440	27.9
50000	14361	57.4
100000	14543	53.7

4.3. Graphs:

-Write



-Read



4.4. Observations:

As can be seen in the subsequent sections, the mean latency and op-rate for both read and write operations keep increasing, when increasing the number of iterations of read and write operations. The graph shows an exponential increase in, from the op-rates for both read and write operations.

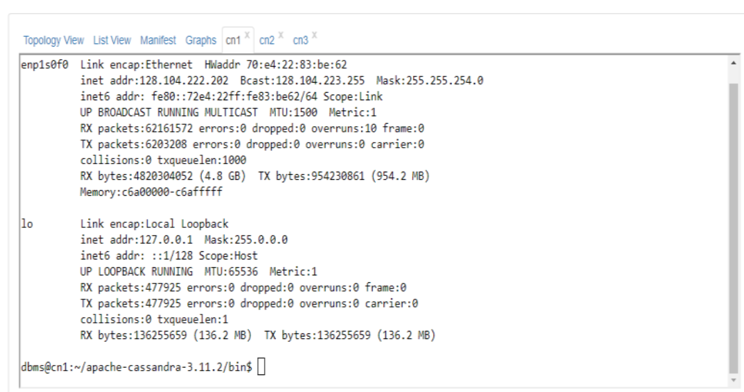
4.5. Technical Challenges:

Major challenge in this topology was as was in topology 2 but after this docker couldn't be used.

Manually Cassandra was installed on each node

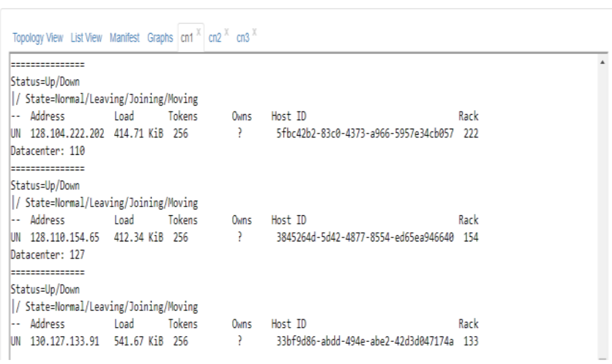
Configuration of Cassandra.yaml file was most challenging phase which consumed more than 20 hours to figure out and configure the node.

The cloud lab topology for the same is given in the below graph along with there IP's -




```
dbms@cn2:~$ ifconfig
eno1      Link encap:Ethernet  HWaddr ec:b1:d7:85:5a:32
          inet addr:128.110.154.65  Bcast:128.110.155.255  Mask:255.255.252.0
          inet6 addr: fe80::eeb1:d7ff:fe85:5a32/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6123767 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4921192 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:896657490 (896.6 MB)  TX bytes:963501571 (963.5 MB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:110101 errors:0 dropped:0 overruns:0 frame:0
          TX packets:110101 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:11712760 (11.7 MB)  TX bytes:11712760 (11.7 MB)

dbms@cn2:~$
```


```
eno1      Link encap:Ethernet  HWaddr 34:17:eb:e5:72:6e
          inet addr:130.127.133.91  Bcast:130.127.135.255  Mask:255.255.252.0
          inet6 addr: fe80::3617:ebff:fee5:726e/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:45646582 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4492581 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3220895237 (3.2 GB)  TX bytes:856314843 (856.3 MB)
          Memory:91820000-9183ffff

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:106806 errors:0 dropped:0 overruns:0 frame:0
          TX packets:106806 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:10837841 (10.8 MB)  TX bytes:10837841 (10.8 MB)

dbms@cn3:~$
```

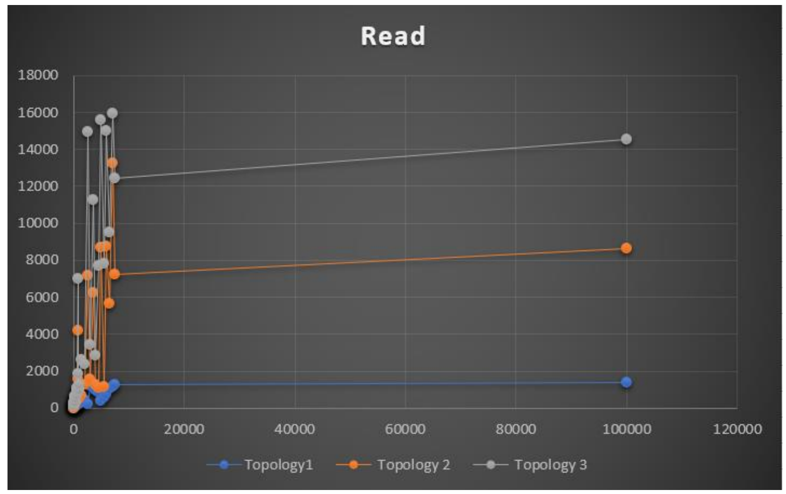



```
===============
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address         Load       Tokens       Owns    Host ID                               Rack
UN  128.104.222.202  414.71 KiB  256          ?       5fbc42b2-83c0-4373-a966-5957e34cb057  222
Datacenter: 110
===============
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address         Load       Tokens       Owns    Host ID                               Rack
UN  128.110.154.65   412.34 KiB  256          ?       3845264d-5d42-4877-8554-ed65ea946640  154
Datacenter: 127
===============
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address         Load       Tokens       Owns    Host ID                               Rack
UN  130.127.133.91   541.67 KiB  256          ?       33bf9d86-abdd-494e-abe2-42d3d047174a  133
```

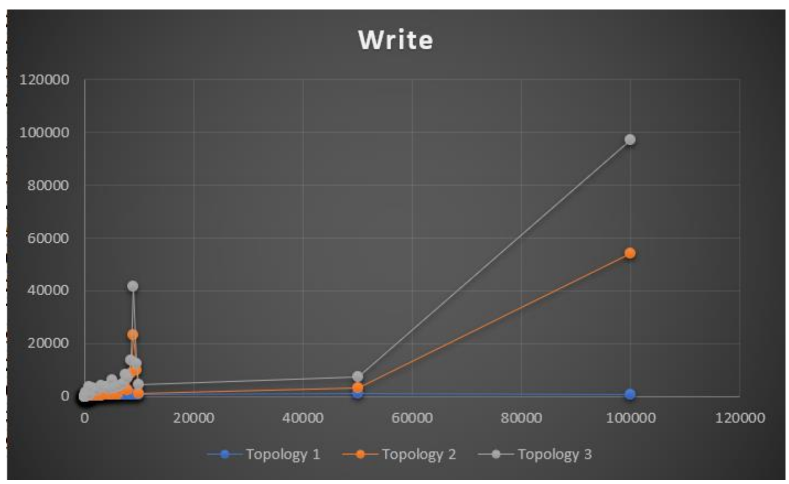
6. Conclusions:

We can infer from the below graph that the latency will increase as the distance of the physical nodes in the topologies increases. Hence the curve of the topology one lies below in the graph where in topology 2 line lies above the topology line1, but it comes below the topology 3 line. Hence in the distributed data bases we can say that latencies increase as the distance increases. This could be improved by using very good network infrastructure in the background. Future work for this project would be testing this scenario with that infrastructure.

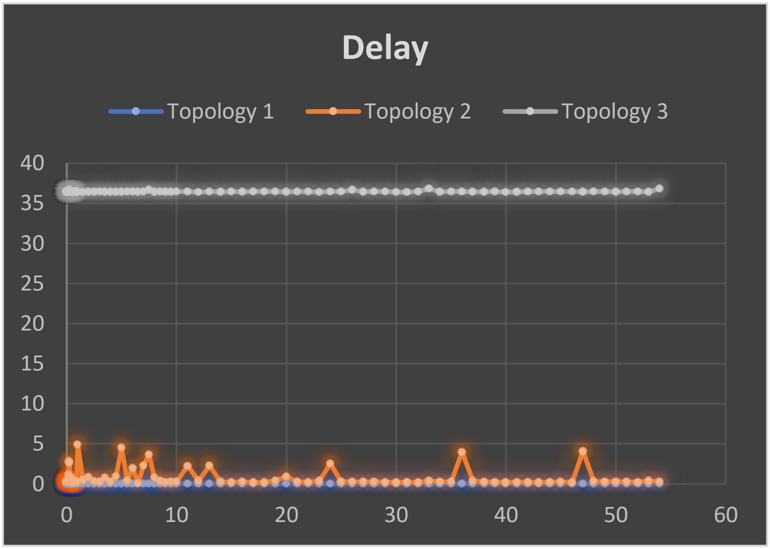
-Read



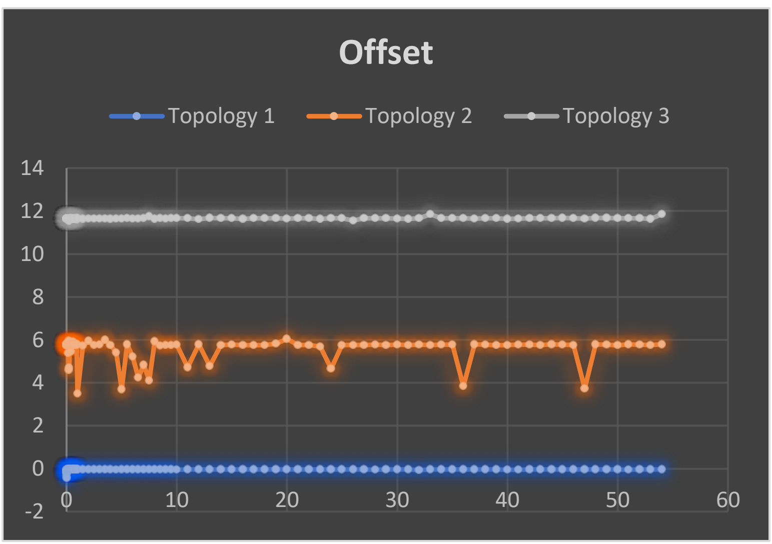
-Write



-Delay



-Offset



7. References

1. https://hub.docker.com/r/bitnami/cassandra/

2. http://cassandra.apache.org/

3. https:/help.ubuntu.com/lts/serverguide/NTP.html

4. https://blog.rapid7.com/2014/03/14/synchronizing-clocks-in-a-cassandra-cluster-pt-1-the-problem/

5. http://thelastpickle.com/blog/2017/02/08/Modeling-real-life-workloads-with-cassandra-stress.html

6. https://cloudlab.us/login.php

7. http://docs.cloudlab.us/users.html#%28part._join-project%29

8. Links

All our work is saved and added in below Git repository.

GitHub:

https://github.com/dhawaskar/Cassandra_cluster_analysis