# Understanding Distributed Databases: Measuring Latency and Clock Skew in Apache Cassandra

Soham Shah

CSCI 5817: Data Systems

Professor Frank Miller

Table of Contents

[Understanding Distributed Databases: Measuring Latency and Clock Skew in Apache Cassandra 1](#_Toc508633429)

[Project Overview 3](#_Toc508633430)

[Procedure 3](#_Toc508633431)

[Topology 1: 3](#_Toc508633432)

[Topology 2: 3](#_Toc508633433)

[Topology 3: 4](#_Toc508633434)

[Topology 4: 4](#_Toc508633435)

[Technical Details: 5](#_Toc508633436)

[Results and Analysis: 5](#_Toc508633437)

[Conclusion: 10](#_Toc508633438)

[Works Cited: 11](#_Toc508633439)

# Project Overview

The goal of this project was work with and understand Apache Cassandra which is a NoSQL key-value store and to analyze how different topologies affect the latency and clock skew.

# Procedure

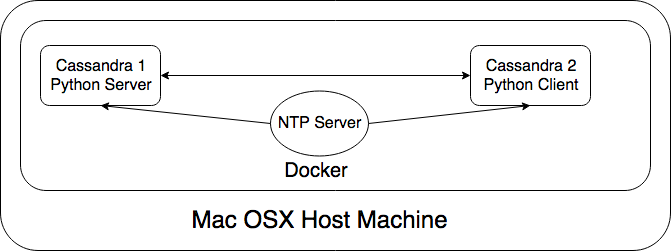
Because we wanted to measure clock skew and latency we utilized a combination of Network Time Protocol (NTP) to synchronize clocks and then a simple Cassandra cluster of two nodes. The NTP server was in charge of syncing the two Cassandra nodes clocks. This way, the clocks were coordinated when we started testing. We would run Cassandra-stress, which is a tool to stress test the Cassandra cluster and run a bunch of writes followed by a bunch of reads.

As the writes were occurring, we a pair of python programs to measure the difference in clock latency. The python server and client would each use the time library to get that system’s time. Then the client would compare the times to figure out the delay and the offset as shown in the following diagram.

We created and tested 3 different topologies. They are as follows:

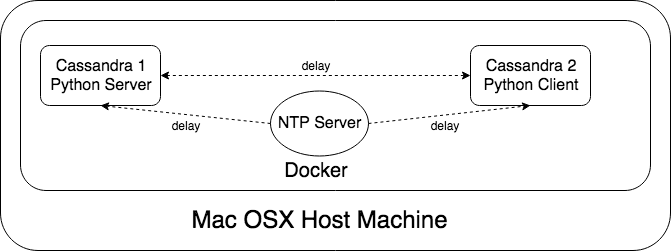
## Topology 1:

The first topology involved all the instances in Docker containers on a single host. There were 2 Cassandra nodes and a single NTP server. Here is a visualization of that topology



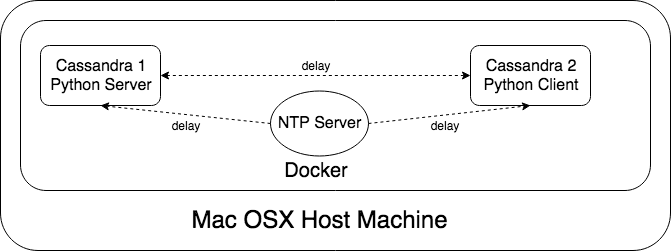
## Topology 2:

The second topology was built off the original topology but introduced a time delay. It still involved all the instances in Docker containers on a single host. However, the tool, Pumba was used. Pumba allows developers to simulate packet loss and a degradation in network quality. For this topology. A 3 second delay was added for each packet going between each Docker container.



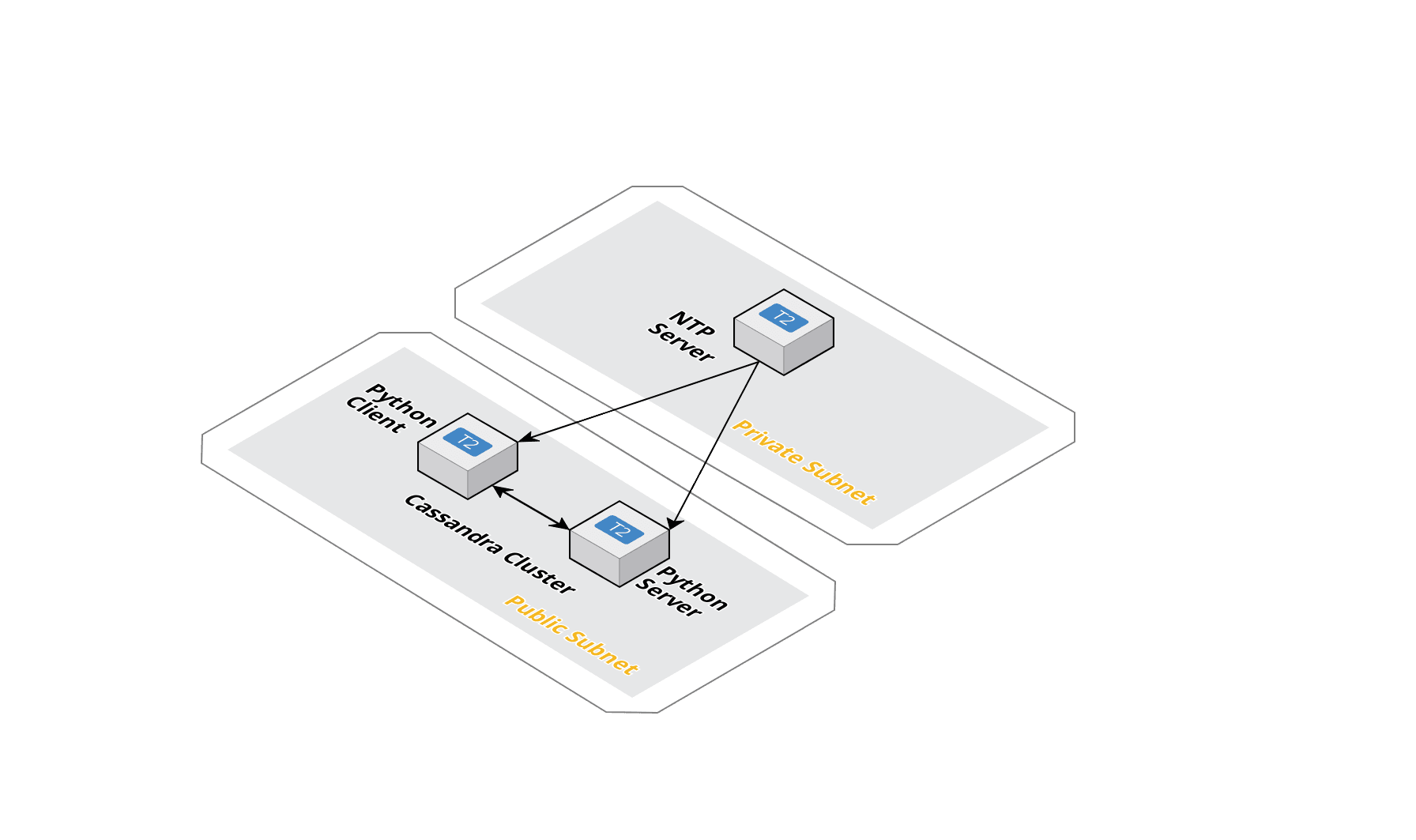
## Topology 3:

The third topology Is the same as the second but instead of a 3 second delay we used a 1 second delay.



## Topology 4:

For the fourth and final topology we deployed the system to the cloud. We ended up using Amazon Web Services (AWS) because that’s what I was familiar with. We used T2.Micro instances on the free tier and manually configured Cassandra to work on the Ubuntu AMI. The instances were all on a single availability zone (US East) . While Cassandra is able to scale to multiple availability zones, this is much more complicated to configure and we ended up running out of time.



# Technical Details:

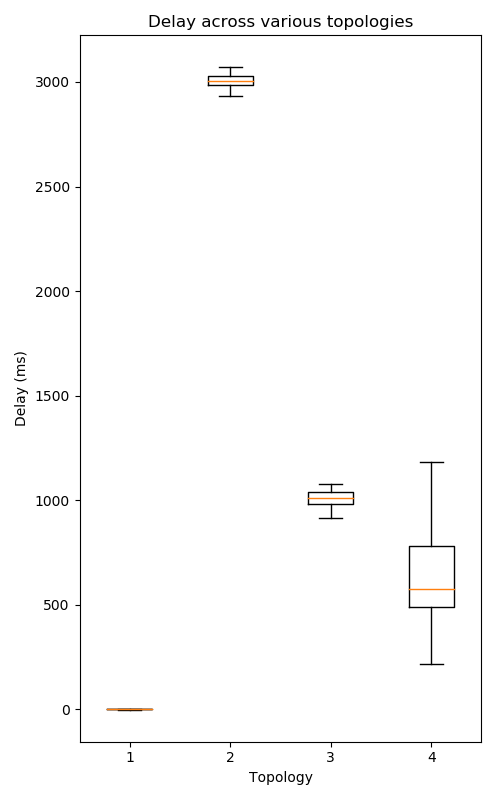
There was a lot of code written for this project including dockerfiles, docker-compose files, python files, and text files documenting the steps taken to sand up the instances in AWS. I’ve stored all the code onto my Github page which you can view at this link:

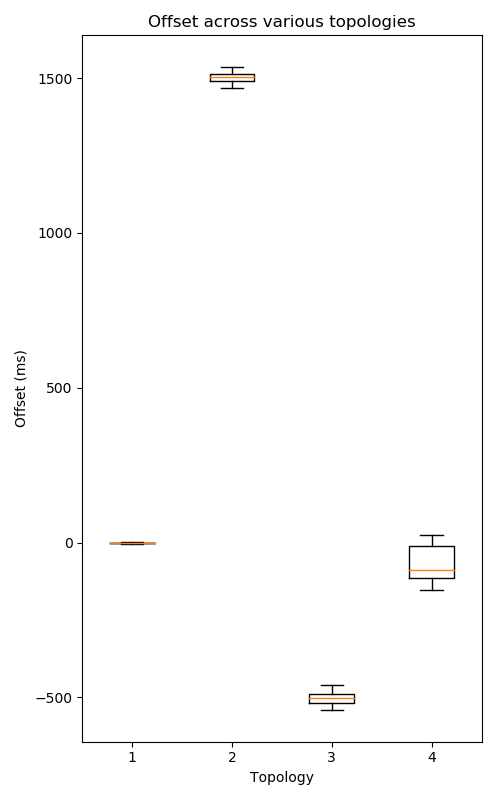
<https://github.com/soham-shah/CSCI-5817-Cassandra/tree/master>

I’ve also stored all the raw data that we generated and the python files used to parse this data and generate the visualizations you see below.

# Results and Analysis:

The results were interesting. Here are the box and whisker plots for the Time Offset and Delay across topologies



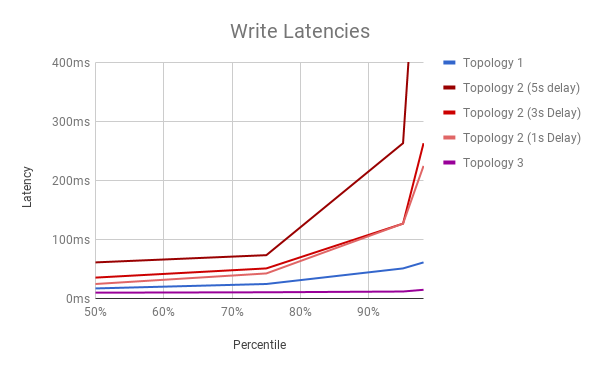


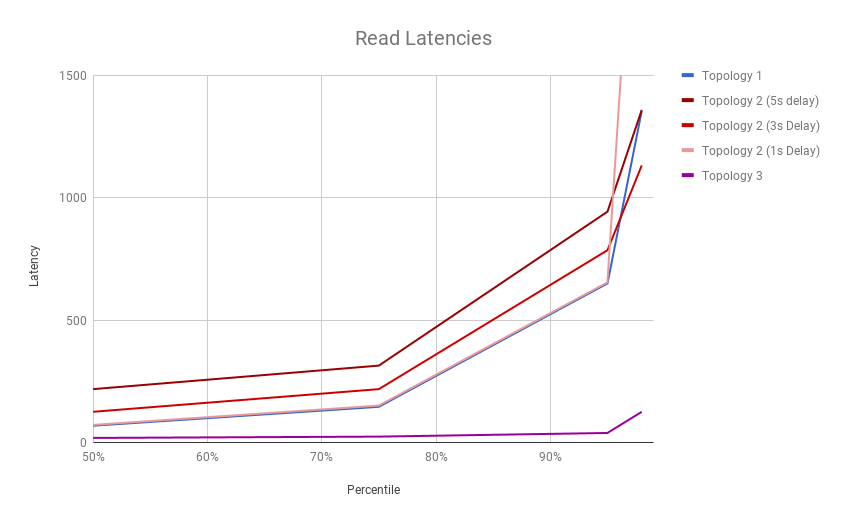
First let’s examine the offset and delay data. It’s interesting to note how tight the box and whisker plots are for the first, second, and third topologies. Note how for the first topology you can’t really see the difference between the first and third quartiles. When we added Pumba, which simulated packet loss and network latency the delay and offset jumped up by a lot. The delay jumped up by about 3 seconds in topology 1 which makes total sense as we added a 3 second delay. In topology 2 the delay is again 1 second because that’s what we’re simulating. But the other interesting thing is the widening of the range between the first and third quartiles. This is because of the packet loss adding some variation to the results. With the fourth topology, the most important thing to note is the widening of the variance for time delay and offset. This makes sense as the production implementation is going to have a much larger delay than a local implementation where Docker is managing everything. However, the delay and offset weren’t that bad. They were in fact, quite close to the results of the first topology and certainly not as bad as the second topology. This tells us that if you wanted to simulate what would happen in production, you can add some simulated packet loss and network latency but the real world had much more variance that a constant 3 second delay.

We also took measurements of the read and write latencies. Please note the different naming scheme here. All the delay tests are classified under Topology 2 and the AWS test is Topology 3. Here is the raw data for each topology:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Write Latency (Micros) | | | | | |
| Percentile | Topology 1 | Topology 2 (1s delay) | Topology 2 (3s delay) | Topology 2 (5s delay) | Topology 3 |
| 50% | 17.08 | 24.6 | 35.43 | 61.21 | 9.89 |
| 75% | 24.6 | 42.5 | 51.01 | 73.46 | 9.89 |
| 95% | 51.01 | 126.93 | 126.93 | 263.21 | 11.86 |
| 98% | 61.21 | 224.75 | 263.21 | 785.94 | 11.86 |
| 99% | 61.21 | 654.94 | 315.85 | 785.94 | 11.86 |
|  |  |  |  |  |  |
| Read Latency (micros) | | | | | |
| Percentile | Topology 1 | Topology 2 (1s delay) | Topology 2 (3s delay) | Topology 2 (5s delay) | Topology 3 |
| 50% | 73.46 | 73.46 | 126.93 | 219.34 | 20.34 |
| 75% | 152.32 | 152.32 | 219.34 | 315.34 | 25.67 |
| 95% | 654.95 | 654.95 | 785.94 | 943.13 | 40.56 |
| 98% | 1358.1 | 2816.16 | 1131.75 | 1358.1 | 126.78 |
| 99% | 2816.16 | 4866.32 | 5839.59 | 1358.1 | 356.78 |

Visualizing this data gives us the following result:





This data was interesting to see once visualized. We obtained it using the nodetool utility shipped with Cassandra. What jumps out clearly is that the second topologies had the longest read and write latency. Not only that, it spiked the highest at the end as well. This makes sense. We added a specific delay and also simulated packet loss. This means that some packets would have taken way longer to get to the other servers and onto disk than others. Also, it would take much longer to get a response after a read request was sent. Perhaps more interesting of note is that the first topology was slower on reads and writes than deploying to a cluster. This seems to be counterintuitive. Wouldn’t the read and write times be slower on the cloud? The answer is that because the instances were all on the same server farm, the packet loss and delay was minimal. Not as small as the first topology but not as bad as the second, as seen in the previous charts. The key is actually the hardware in this case. Our computer for the first topology was run on hard disks while the AWS cluster was run on solid state drives in a server rack optimized for Input output operations. This means that read and write were much faster in the server farm even after accounting for the network delay which is why our reads and writes were faster!

# Conclusion:

Overall, this was an interesting project. We were able to test programmatically what the clock skew and read write speeds were for a distributed cluster. The results clearly show how important it is to have a good connection, synchronized clocks and solid state drives for distributed database performance.

# Works Cited:

* [**https://www.digitalocean.com/community/tutorials/how-to-install-cassandra-and-run-a-single-node-cluster-on-ubuntu-14-04**](https://www.digitalocean.com/community/tutorials/how-to-install-cassandra-and-run-a-single-node-cluster-on-ubuntu-14-04)
* [**https://hub.docker.com/r/bitnami/cassandra/**](https://hub.docker.com/r/bitnami/cassandra/)
* [**https://www.rosehosting.com/blog/how-to-install-apache-cassandra-on-ubuntu-16-04/**](https://www.rosehosting.com/blog/how-to-install-apache-cassandra-on-ubuntu-16-04/)
* [**http://cassandra.apache.org/**](http://cassandra.apache.org/)
* [**https://help.ubuntu.com/lts/serverguide/NTP.html**](https://help.ubuntu.com/lts/serverguide/NTP.html)
* [**https://blog.rapid7.com/2014/03/14/synchronizing-clocks-in-a-cassandra-cluster-pt-1-the-problem/**](https://blog.rapid7.com/2014/03/14/synchronizing-clocks-in-a-cassandra-cluster-pt-1-the-problem/)
* [**https://blog.rapid7.com/2014/03/17/synchronizing-clocks-in-a-cassandra-cluster-pt-2-solutions/**](https://blog.rapid7.com/2014/03/17/synchronizing-clocks-in-a-cassandra-cluster-pt-2-solutions/)
* [**https://github.com/fwmiller/pylatency**](https://github.com/fwmiller/pylatency)
* [**https://stackoverflow.com/questions/23199796/detect-and-exclude-outliers-in-pandas-dataframe**](https://stackoverflow.com/questions/23199796/detect-and-exclude-outliers-in-pandas-dataframe)
* [**https://alexei-led.github.io/post/pumba\_docker\_netem/**](https://alexei-led.github.io/post/pumba_docker_netem/)