Project Final Report
Challenges of setting up a scientific environment with Cassandra
Gabriela Concolin Schimidt – gc2658@columbia.edu
COMSE6156 - Topics on Software Engineering

1. Introduction

This proposed project is a follow-up on the paper earlier produced for this course. In summary, the paper addressed storage of scientific data, considering the necessities this type of data has and how it works on a scientific environment. The paper discussed the CAP theorem and how scientific data should favor availability, as well as many software qualities previous studies evaluated on NoSQL databases. Finally, Cassandra was proposed as an interesting way of storing SD, considering its features and a successful study on biological data storage. In light of the challenges scientists may face while dealing with SD presented on the paper, this project will narrate the process of replicating a scientific database environment on Cassandra.

The goal of this project is reporting the setup of a scientific environment, narrating all the steps throughout the machine setting, database setting and data filtering. This experiment will consider the complexity of those tasks and how maintainable such environment may be for someone who has tech knowledge, however, may have never dealt with databases and clustering before. The progress report aimed to describe the architectural choices and the setting of the clusters. The steps that followed the progress report will be addressed in this final report, which will consist in data filtering, as well as Cassandra deployment and system evaluations. All the steps proposed on the previous project proposal have been completed successfully.

1. Cassandra Deployment

Cassandra 2.2.6 was installed on each instance following the tutorial on [4], with minor changes to address this version of the database, which was chosen because it is stable and lighter than the others are. Cassandra 3.0.5 was used in the first attempt of deployment, however, the nodes of the cluster did not listened to each other’s requests (as in Cassandra jargon, “handshake”), whereas the cluster worked perfectly fine from the start when it was downgraded to version 2.2.6. The final architecture of the cluster is as follows.

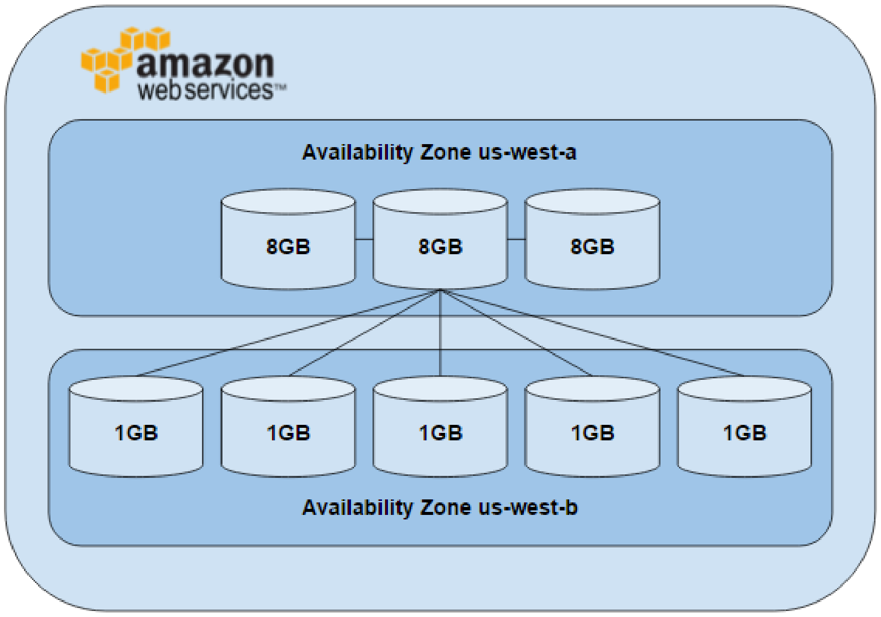
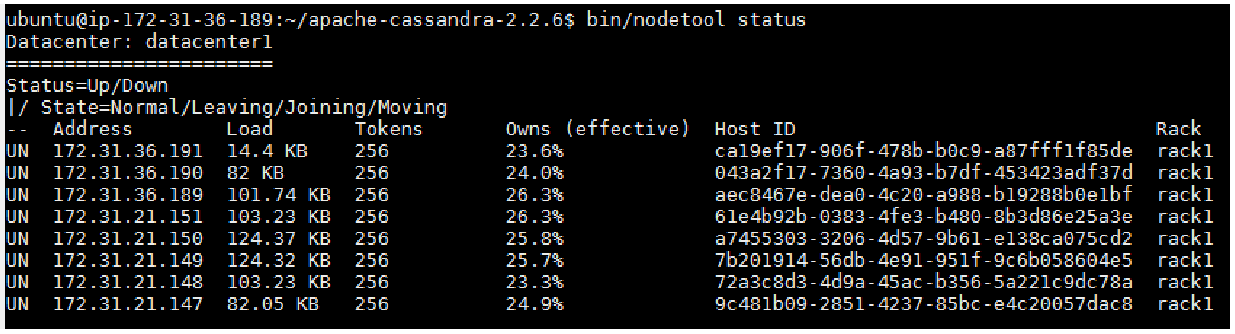


Figure 1 - Cluster Architecture

The cluster is composed by eight nodes, which are instances of AWS. There are two groups, each on a different Availability Zone, to ensure availability in failure. The Cassandra Cluster has a main seed node, which listens to the other nodes requests. The seed is represented by the center 8GB node on the picture. For the sake of availability, it would have been interesting to have more than one node as seed. However, Cassandra configuration has shown itself as very unstable and difficult to manage, and the nodes of the cluster did not "handshake" when more than one seed was involved. So, this simpler version has been used throughout the experiment.



```
ubuntu@ip-172-31-36-189:~/apache-cassandra-2.2.6$ bin/nodetool status
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address        Load       Tokens       Owns (effective)  Host ID                               Rack
UN  172.31.36.191  14.4 KB    256          23.6%             ca19ef17-906f-478b-b0c9-a87fff1f85de  rack1
UN  172.31.36.190  82 KB      256          24.0%             043a2f17-7360-4a93-b7df-453423adf37d  rack1
UN  172.31.36.189  101.74 KB  256          26.3%             aec8467e-dea0-4c20-a988-b19288b0e1bf  rack1
UN  172.31.21.151  103.23 KB  256          26.3%             61e4b92b-0383-4fe3-b480-8b3d86e25a3e  rack1
UN  172.31.21.150  124.37 KB  256          25.8%             a7455303-3206-4d57-9b61-e138ca075cd2  rack1
UN  172.31.21.149  124.32 KB  256          25.7%             7b201914-56db-4e91-951f-9c6b058604e5  rack1
UN  172.31.21.148  103.23 KB  256          23.3%             72a3c8d3-4d9a-45ac-b356-5a221c9dc78a  rack1
UN  172.31.21.147  82.05 KB   256          24.9%             9c481b09-2851-4237-85bc-e4c20057dac8  rack1
```

Figure 2 - Cluster Status

A repository with the directory each node must have after deployment is available on https://github.com/gabrielacschimidt/6156-cassandra.

2. Modelling data into Cassandra

2.1.Data

The data chosen for the tests in this study was a complete mapping of male human genome. The data was provided by an organization called GET-Evidence, which is a collaborative research tool for genetics study. It provides a platform for user discussion on literature already published and encourages the public to test and use its data for analysis and discussion. All the data is under no copyright and anyone can share and collaborate. The data provided by this community can stimulate independent research and testing, and it has been chosen to replicate scientific study environment.

The raw file contained 1.42GB and had tsv format [5]. It underwent the following bash command to become csv format:

```
tr "//t" "," < genome.tsv > gen.csv
```

This file has the following columns of information: locus, ploidy, allele, chromosome, begin, end, varType, reference, alleleSeq, varScoreVAF, varScoreEAF, varQuality, hapLink, xRef. The first 9 columns are mandatory, whereas the last 5 can be empty. As "being" and "end" are reserved words, the sufix "All" for allele has been appended to their column names on Cassandra. The file has been stored on a node on us-west-a Availability Zone.

2.2.Modelling

Cassandra has a different architectural concept for the data stored on it. The concept for schemas is now substituted to Keyspace (also known as keystores). In Cassandra, it is possible to chose from Tables or ColumnFamily to store data. For this project, ColumnFamily was chosen. These variations are very simple and do not jeopardize the productivity of a researcher who has basic SQL database knowledge. The keyspaces “Keyspace1” and “system” are created when the application is installed. Finally, the query language CQL that Cassandra provides has a low learning curve for those familiar with SQL concepts.

Based on the genomic data mined, the following code would create the keyspace necessary for aggregating the information:

```
CREATE KEYSPACE GENOME
WITH REPLICATION = {'CLASS':'SIMPLESTRATEGY',
'REPLICATION_FACTOR' : 3};
```

The class chosen for the keyspace was SimpleStrategy because this cluster is hosted on a single data center; otherwise, it should have been NetworkTopologyStrategy [6]. Most scientific environments would choose the later, considering research information may be divided in multiple data centers. Even though the cluster is spread in two availability zones, it does not configure different data centers in this context. They would be considered distinct data centers if the cluster was formed by nodes on different hosting services [1]. The replication factor is the number of replicas of the rows across the cluster [6]. A replication factor of 3 means that there are three copies of each row, each copy on a different node and each replica is worth the same; there is no primary or master replica.

The following code created the ColumnFamily for the Specimen.

```
CREATE COLUMNFAMILY SPECIMEN(
LOCUS VARCHAR PRIMARY KEY,
PLOIDY VARCHAR,
ALLELE VARCHAR,
CHROMOSOME VARCHAR,
BEGINALL INT,
ENDALL INT,
VARTYPE VARCHAR,
REFERENCE VARCHAR,
ALLELESEQ VARCHAR,
VARSCOREVAF VARCHAR,
VARSCOREEAF VARCHAR,
VARQUALITY VARCHAR,
HAPLINK VARCHAR,
XREF VARCHAR);
```

After the execution, Cassandra structure would look like Figure 3.

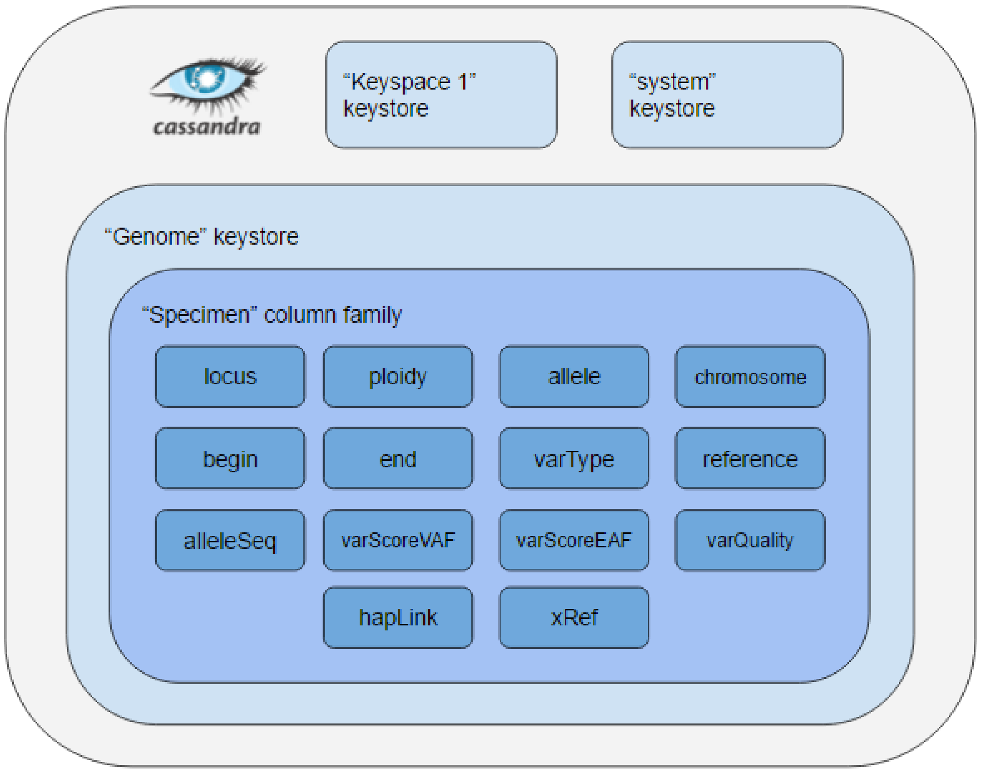


Figure 3 - Cassandra Keystore structures

The following command was used to insert the mined data into the database:

```
COPY SPECIMEN (LOCUS, PLOIDY, ALLELE, CHROMOSOME, BEGINALL,
ENDALL, VARTYPE, REFERENCE, ALLELESEQ, VARSCOREVAF, VARSCOREEAF,
VARQUALITY, HAPLINK, XREF) FROM '../GEN.CSV' WITH HEADER = TRUE;
```

This command imported 18898540 rows from the file in 1 hour and 34 minutes. After each inclusion through the "COPY" command, a row named after the primary key will be created containing the values of each column. The data becomes shaped in a nested manner, maintaining a pair appearance. This type of nested data is useful for scientific data, because it is easier to attach metadata, track its evolution and change the format of data without jeopardizing old rows. This column-oriented storing factor was chosen for this project because the pair characteristic is more important for scientists than the "spreadsheet" characteristics of row-oriented tables. As the genome is mapped and deformities are traced, other rows with slightly different metadata can be created but the structure will still maintain the hierarchy of data, keeping it in a key-value manner.

Despite the fact that Cassandra provides column-oriented storage, it is accessible in a row-oriented manner like any SQL store when one is using CQL. It is possible to use JSON files and access data through different ways in the database, however, it was assumed that most scientists would benefit from the row-oriented access because it is more likely that they understand SQL and structured data analysis.

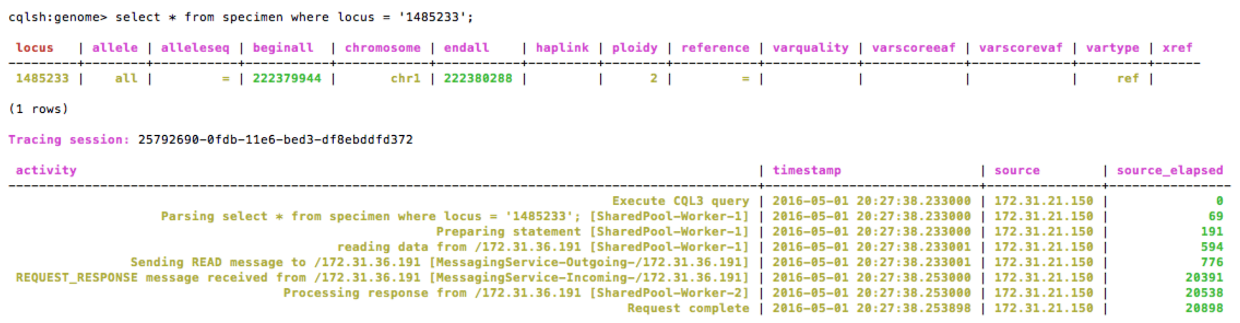
In addition, it is interesting to point that all these settings can be changed at any time, since Cassandra is very flexible. It would be possible to include a column for timestamps to the column family so updates could be easier to track. "In Cassandra, although the column families are defined, the columns are not. You can freely add any column to any column family at any time" [7]. Likewise, one could alter the replication factor of the keyspace. This adaptability is very important for an independent scientist to fully experiment on his data structure, as well as change and correct infrastructure choices that may not suit their dataset anymore.

3. Evaluations

While testing queries, Cassandra query language CQL showed itself very easy to learn, considering that it is almost identical to SQL.

```
SELECT * FROM SPECIMEN WHERE LOCUS = '14852335';
```

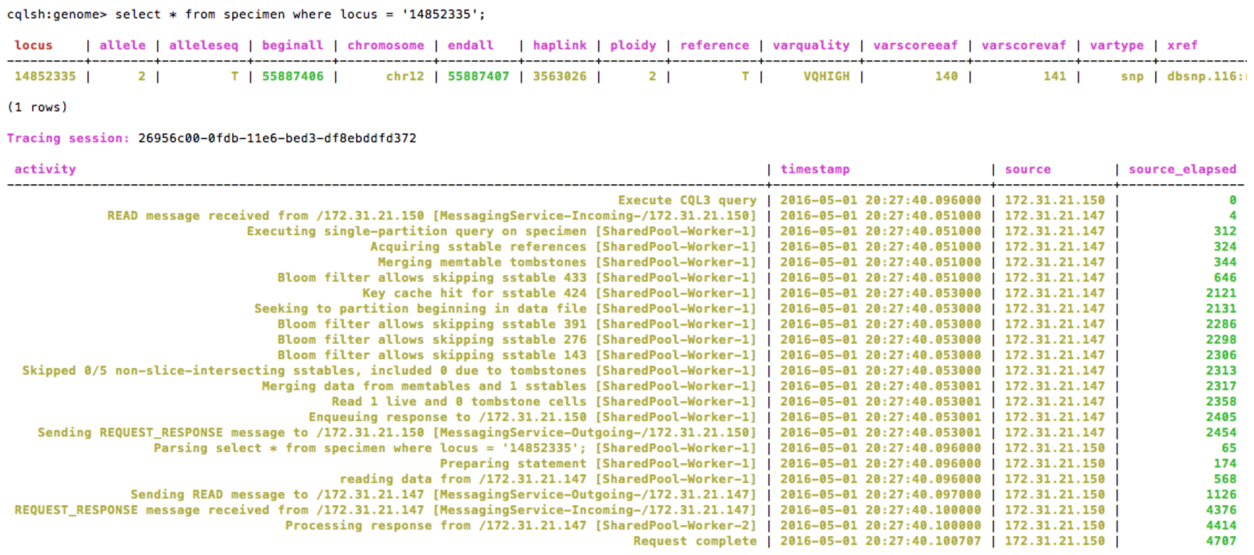
Using the tracing tool CQL offers, this query was processed in the node the data was initially stored, on us-west-a Availability Zone (AZ), within 0.31 seconds, which is impressive considering 18 million rows. When the same query was processed in a different AZ in a node that had less memory, it took 0.5 seconds, which is a good result as well.



```
cqlsh:genome> select * from specimen where locus = '1485233';

 locus   | allele | alleleseq | beginall  | chromosome | endall    | haplink | ploidy | reference | varquality | varscoreeaf | varscorevaf | vartype | xref
---------+--------+-----------+-----------+------------+-----------+---------+--------+-----------+------------+-------------+-------------+---------+------
 1485233 |    all |         = | 222379944 |       chr1 | 222380288 |         |      2 |         = |            |             |             |     ref |

(1 rows)

Tracing session: 25792690-0fdb-11e6-bed3-df8ebddfd372

 activity                                                                                         | timestamp                  | source       | source_elapsed
--------------------------------------------------------------------------------------------------+----------------------------+--------------+----------------
                                                                               Execute CQL3 query | 2016-05-01 20:27:38.233000 | 172.31.21.150 |              0
                        Parsing select * from specimen where locus = '1485233'; [SharedPool-Worker-1] | 2016-05-01 20:27:38.233000 | 172.31.21.150 |             69
                                                           Preparing statement [SharedPool-Worker-1] | 2016-05-01 20:27:38.233000 | 172.31.21.150 |            191
                                               reading data from /172.31.36.191 [SharedPool-Worker-1] | 2016-05-01 20:27:38.233001 | 172.31.21.150 |            594
                     Sending READ message to /172.31.36.191 [MessagingService-Outgoing-/172.31.36.191] | 2016-05-01 20:27:38.233001 | 172.31.21.150 |            776
 REQUEST_RESPONSE message received from /172.31.36.191 [MessagingService-Incoming-/172.31.36.191] | 2016-05-01 20:27:38.253000 | 172.31.21.150 |          20391
                                        Processing response from /172.31.36.191 [SharedPool-Worker-2] | 2016-05-01 20:27:38.253000 | 172.31.21.150 |          20538
                                                                                 Request complete | 2016-05-01 20:27:38.253898 | 172.31.21.150 |          20898
```

Figure 4 - When a query was run in a stable cluster circumstance, it retrieves data from where it is initially stored, the node "172.31.36.191".

While executing this example query on an instance, I stopped Cassandra on another node. The execution was shortly switched to a single-partition search for this query and the response was just even faster than before, taking about 0.2 seconds. Considering the information on the tracing tool, the results came from a different node, which probably hold one of the replicas of that row. After the cluster was stable again, the execution continued without the dropped node, the query was then run using all the available instances and the results appeared in 0.2 seconds as well. Figures 4 and 5 show the complete tracing.



```
cqlsh:genome> select * from specimen where locus = '14852335';

 locus     | allele | alleleseq | beginall | chromosome | endall    | haplink | ploidy | reference | varquality | varscoreeaf | varscorevaf | vartype | xref
-----------+--------+-----------+----------+------------+-----------+---------+--------+-----------+------------+-------------+-------------+---------+-----------
 14852335 |      2 |         T | 55887406 |      chr12 | 55887407 | 3563026 |      2 |         T |     VQHIGH |         140 |         141 |     snp | dbsnp.116:

(1 rows)

Tracing session: 26956c00-0fdb-11e6-bed3-df8ebddfd372

 activity                                                                                         | timestamp                  | source        | source_elapsed
--------------------------------------------------------------------------------------------------+----------------------------+---------------+----------------
                                                                               Execute CQL3 query | 2016-05-01 20:27:40.096000 | 172.31.21.150 |              0
             READ message received from /172.31.21.150 [MessagingService-Incoming-/172.31.21.150] | 2016-05-01 20:27:40.051000 | 172.31.21.147 |              4
                               Executing single-partition query on specimen [SharedPool-Worker-1] | 2016-05-01 20:27:40.051000 | 172.31.21.147 |            312
                                               Acquiring sstable references [SharedPool-Worker-1] | 2016-05-01 20:27:40.051000 | 172.31.21.147 |            324
                                                Merging memtable tombstones [SharedPool-Worker-1] | 2016-05-01 20:27:40.051000 | 172.31.21.147 |            344
                                   Bloom filter allows skipping sstable 433 [SharedPool-Worker-1] | 2016-05-01 20:27:40.051000 | 172.31.21.147 |            646
                                              Key cache hit for sstable 424 [SharedPool-Worker-1] | 2016-05-01 20:27:40.053000 | 172.31.21.147 |           2121
                                Seeking to partition beginning in data file [SharedPool-Worker-1] | 2016-05-01 20:27:40.053000 | 172.31.21.147 |           2131
                                   Bloom filter allows skipping sstable 391 [SharedPool-Worker-1] | 2016-05-01 20:27:40.053000 | 172.31.21.147 |           2286
                                   Bloom filter allows skipping sstable 276 [SharedPool-Worker-1] | 2016-05-01 20:27:40.053000 | 172.31.21.147 |           2298
                                   Bloom filter allows skipping sstable 143 [SharedPool-Worker-1] | 2016-05-01 20:27:40.053000 | 172.31.21.147 |           2306
  Skipped 0/5 non-slice-intersecting sstables, included 0 due to tombstones [SharedPool-Worker-1] | 2016-05-01 20:27:40.053000 | 172.31.21.147 |           2313
                                 Merging data from memtables and 1 sstables [SharedPool-Worker-1] | 2016-05-01 20:27:40.053001 | 172.31.21.147 |           2317
                                          Read 1 live and 0 tombstone cells [SharedPool-Worker-1] | 2016-05-01 20:27:40.053001 | 172.31.21.147 |           2358
                                       Enqueuing response to /172.31.21.150 [SharedPool-Worker-1] | 2016-05-01 20:27:40.053001 | 172.31.21.147 |           2405
    Sending REQUEST_RESPONSE message to /172.31.21.150 [MessagingService-Outgoing-/172.31.21.150] | 2016-05-01 20:27:40.053001 | 172.31.21.147 |           2454
                   Parsing select * from specimen where locus = '14852335'; [SharedPool-Worker-1] | 2016-05-01 20:27:40.096000 | 172.31.21.150 |             65
                                                        Preparing statement [SharedPool-Worker-1] | 2016-05-01 20:27:40.096000 | 172.31.21.150 |            174
                                           reading data from /172.31.21.147 [SharedPool-Worker-1] | 2016-05-01 20:27:40.096000 | 172.31.21.150 |            568
                Sending READ message to /172.31.21.147 [MessagingService-Outgoing-/172.31.21.147] | 2016-05-01 20:27:40.097000 | 172.31.21.150 |           1126
 REQUEST_RESPONSE message received from /172.31.21.147 [MessagingService-Incoming-/172.31.21.147] | 2016-05-01 20:27:40.100000 | 172.31.21.150 |           4376
                                    Processing response from /172.31.21.147 [SharedPool-Worker-2] | 2016-05-01 20:27:40.100000 | 172.31.21.150 |           4414
                                                                                 Request complete | 2016-05-01 20:27:40.100707 | 172.31.21.150 |           4707
```

Figure 5 - When a node is down during the request and the cluster is unstable, the execution is changed to "single-partition" and the current node "172.31.21.150" sends the request to a node that holds a replica of the row, in this case, "172.31.21.147".

These episodes show that, even nodes with little memory have a good request response time. It also shows how Cassandra is available when nodes are down and the user can barely notice any difference on the service. Considering that the cluster is formed by eight nodes and that the replication factor is 3, this system can survive the loss of 2 nodes [8]. Complex queries were tested, like:

```
SELECT * FROM SPECIMEN WHERE LOCUS IN
(SELECT LOCUS FROM SPECIMEN WHERE BEGINALL > 13000);
```

The initial attempts received a timeout from CQL after only 10 seconds. The configuration to change the timeout was somewhat complicated, by writing on a .yaml script that could or could not exist on a hidden folder. People who are not acquainted to UNIX systems may find this configuration complicated, reiterating that Cassandra may not be amiable to beginners.

4. Conclusions

As the progress report previously demonstrated, Amazon Web Services is a well-designed platform that offers many features for independent researchers. It has a good interface and all the steps towards virtual machines settings are well explained, something that favors less experienced professionals. The user can make decisions on infrastructure settings, and independent researchers may design the environment favoring availability by building instances in different Availability Zones.

The NoSQL database chosen, Cassandra, also copes with the availability choices. By scaling horizontally, the database assures consistency and the smallest latency possible. The querying language CQL is very close to standard SQL, therefore making it easier to learn and design keystores. However, the primary conclusion of the project was that, for scientists that want to independently maintain their servers, the worst part is deploying the cluster. Considering Cassandra, there are whitepapers instructing how to configure it, how to choose the best settings to it, but command line cluster configuration is hard for those in the industry, and can be even worse for those who do not have it as their main responsibility. Nonetheless, after deployment is ready, creating the database with mined data is not difficult for those who have SQL knowledge and comprehend the concepts on Cassandra. Overall, the evaluations have shown that the cluster responds promptly to requests, even when one node presents failure. Finally, an independent scientist would likely succeed if these steps were followed and his scientific application would benefit from this environment. All the steps proposed on the previous project proposal have been completed successfully.

This course has been, without a doubt, very challenging and extremely interesting for me. I have learned in depth about NoSQL databases and its characteristics. Nonetheless, the most important thing that I have learned was to manage and organize my tasks during a project process. As an exchange student, this course has immersed me in the Columbia research process, and I have accomplished way more than I expected from myself.

References

[1] Amazon Web Services. "Apache Cassandra on AWS." Jan. 2016. Web. 26 Mar. 2016. <https://d0.awsstatic.com/whitepapers/AWS_Cassandra_Whitepaper.pdf>.

[2] Lucky, David. "Guidelines And Best Practices For Using Apache Cassandra On AWS - Datapipe Blog." Datapipe Blog. 12 Feb. 2016. Web. 26 Mar. 2016. <https://www.datapipe.com/blog/2016/02/12/guidelines-and-best-practices-for-using-apache-cassandra-on-aws/>.

[3] Hutson, Adam. "How To Create a Cassandra Cluster in AWS Part 1." Datascaleio. 23 Dec. 2015. Web. 14 Apr. 2016. <http://datascale.io/how-to-create-a-cassandra-cluster-in-aws/>.

[4] Hutson, Adam. "How To Create a Cassandra Cluster in AWS Part 2." Datascaleio. 8 Jan. 2016. Web. 14 Apr. 2016. <http://datascale.io/how-to-create-a-cassandra-cluster-in-aws-part-2/>

[5] "Variant Report for HuAD8C77 (23anMe)." GET-Evidence: Genomes. 18 Apr. 2016. Web. 26 Apr. 2016. <http://evidence.pgp-hms.org/genomes?display_genome_id=1433b74b2c0f17ddb3367ac0f03270ea71beda61>.

[6] "Data Replication." *DataStax Apache Cassandra™ 2.0*. Web. 02 May 2016. <https://docs.datastax.com/en/cassandra/2.0/cassandra/architecture/architectureDataDistributeReplication_c.html>.

[7] "Cassandra Data Model." *Www.tutorialspoint.com*. Web. 01 May 2016. <http://www.tutorialspoint.com/cassandra/cassandra_data_model.htm>.

[8] “Cassandra Parameters for Dummies”. Web. 02 May 2 2016. <http://www.ecyrd.com/cassandracalculator/>.