**Purpose of the Cassandra project:**

The purpose of my project is to do the following:

* Create a Cassandra cluster of 4 nodes in a local machine using docker.
* Show which data is stored in which node (Partitioning).
* Show how replication works in Cassandra.
* Show how to tune consistency in Cassandra.
* Data Modelling in Cassandra(Explaining with application I have built)

**Cassandra Database type:**

There is a general misconception that Cassandra is a columnar database. However, Cassandra is a column family store oriented. Cassandra stores its data is the data structure as below: ***Map<RowKey, SortedMap<ColumnKey, ColumnValue>>.*** Each row is mapped to a node using partition key, and the value is a set of key value pairs sorted by clustering key. Partition key determines on which node in a Cassandra cluster data is going to be stored. Clustering key determines how data is sorted within a partition.

**CAP Theorem for Cassandra:**

CAP theorem states that for a distributed database, it cannot have consistency, availability and partition tolerant at the same time. Consistency means every node must return the same most recently written data. The case scenario where the nodes may not be consistent is when we write data to a node and it takes some time to replicate the data to other nodes. Hence, some nodes may return the old data. Every functional node must return the response to read/write

request. The database must always be available for read/write operation. In partition tolerant, the system will continue to function even during network partition or failure. Network partition is where a node can't communicate to other node.

Cassandra fits in between Availability(A) and Partition tolerant(P). However, we can tune the consistency level of Cassandra to gain high consistency level which will be explained below:

**Create a Cassandra cluster of 4 nodes in a local machine using docker/Installation notes:**

Assuming you are using linux and have docker installed, following are the commands to create a 4-node cluster in your local machine:

* sudo docker pull cassandra:3.11.2 #Fetch the cassandra image from the docker hub
* sudo docker run --name my-cassandra-1 -m 1g -d cassandra:3.11.2 #Runs the image to builds a container(node) named my-cassandra-1 and allocates 1 GB RAM to it.
* sudo docker inspect --format='{{ .NetworkSettings.IPAddress }}' my-cassandra-1 #Gets the IP address of the container my-cassandra-1

#Creates the second node and connects it to the network link of the first node.

* sudo docker run --name my-cassandra-2 -m 1g -d -e CASSANDRA\_SEEDS="$(docker inspect --format='{{ .NetworkSettings.IPAddress }}' my-cassandra-1)" cassandra:3.11.2

#Creates the third node and connects it to the network link of the first node.

* sudo docker run --name my-cassandra-3 -m 1g -d -e CASSANDRA\_SEEDS="$(docker inspect --format='{{ .NetworkSettings.IPAddress }}' my-cassandra-1)" cassandra:3.11.2

#Creates the fourth node and connect it to the network link of the first node.

* sudo docker run --name my-cassandra-4 -m 1g -d -e CASSANDRA\_SEEDS="$(docker inspect --format='{{ .NetworkSettings.IPAddress }}' my-cassandra-1)" cassandra:3.11.2
* sudo docker exec -i -t my-cassandra-1 bash -c 'nodetool status' # To verify the cluster status.

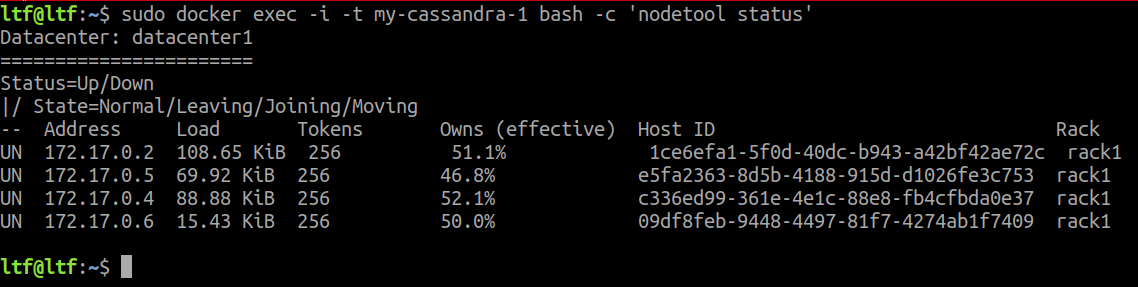


Fig (1): Cluster status

The figure 1 shows the setup of 4 node cluster in the same rack.

**Show which data is stored in which node(Partitioning):**

Assuming we have a keyspace(keyspace can be thought as of RDBMS database) and table structure as following:

CREATE KEYSPACE testing WITH replication = {'class':'SimpleStrategy', 'replication\_factor' : 2};

CREATE TABLE table\_test(

id int primary key,

name varchar);

Use the following command to see which data is stored in which node.

docker exec -i -t my-cassandra-1 bash -c "nodetool getendpoints testing table\_test '<partition\_key\_which\_is\_id\_in\_this\_case>'";

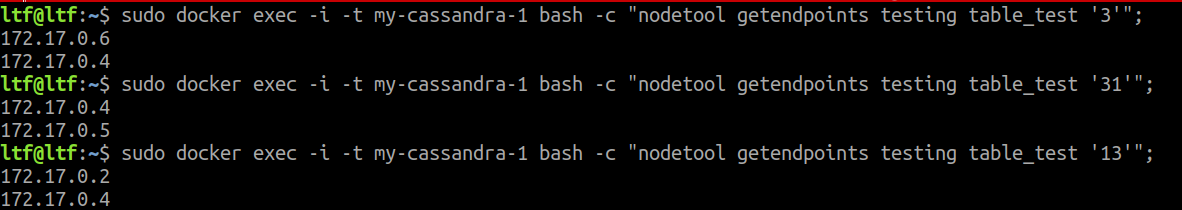


Fig 2: Getting node endpoint for data based on partition key

The above screenshot shows that for the table “table\_test” when partition key (id) is 3, then the data is stored in nodes 172.17.0.6 and 172.17.0.4. It displays two IP address because we have set the replication factor to 2, so any row containing id=3 is always stored in the node IP address as mentioned above.

Cassandra uses Murmur3Partitioner to uniformly distribute data amongst all the nodes. For more information on Murmur3Partitioner please visit:

<https://www.codota.com/code/java/classes/org.apache.cassandra.dht.Murmur3Partitioner>

**Show how replication works in Cassandra:**

There are two types of replication in Cassandra:

1. Simple Strategy: This uses single data center and one rack within which multiple nodes can exist. Simple Strategy places the first replica on a node determined by the Murmur3Partitioner. Additional replicas are placed on the next nodes clockwise in the ring without considering topology based on the replication factor.
2. Network Topology Strategy: It is used when the cluster expands multiple data center or multiple regions. It places replicas in the same data center by walking the ring clockwise until reaching the first node in another rack.

For the demonstration purpose, I have used Simple Strategy for the replication as my cluster is in same rack in a single data center.

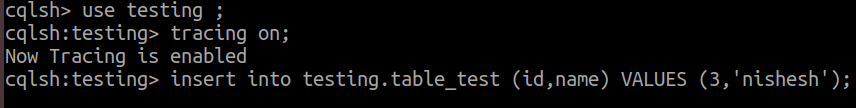
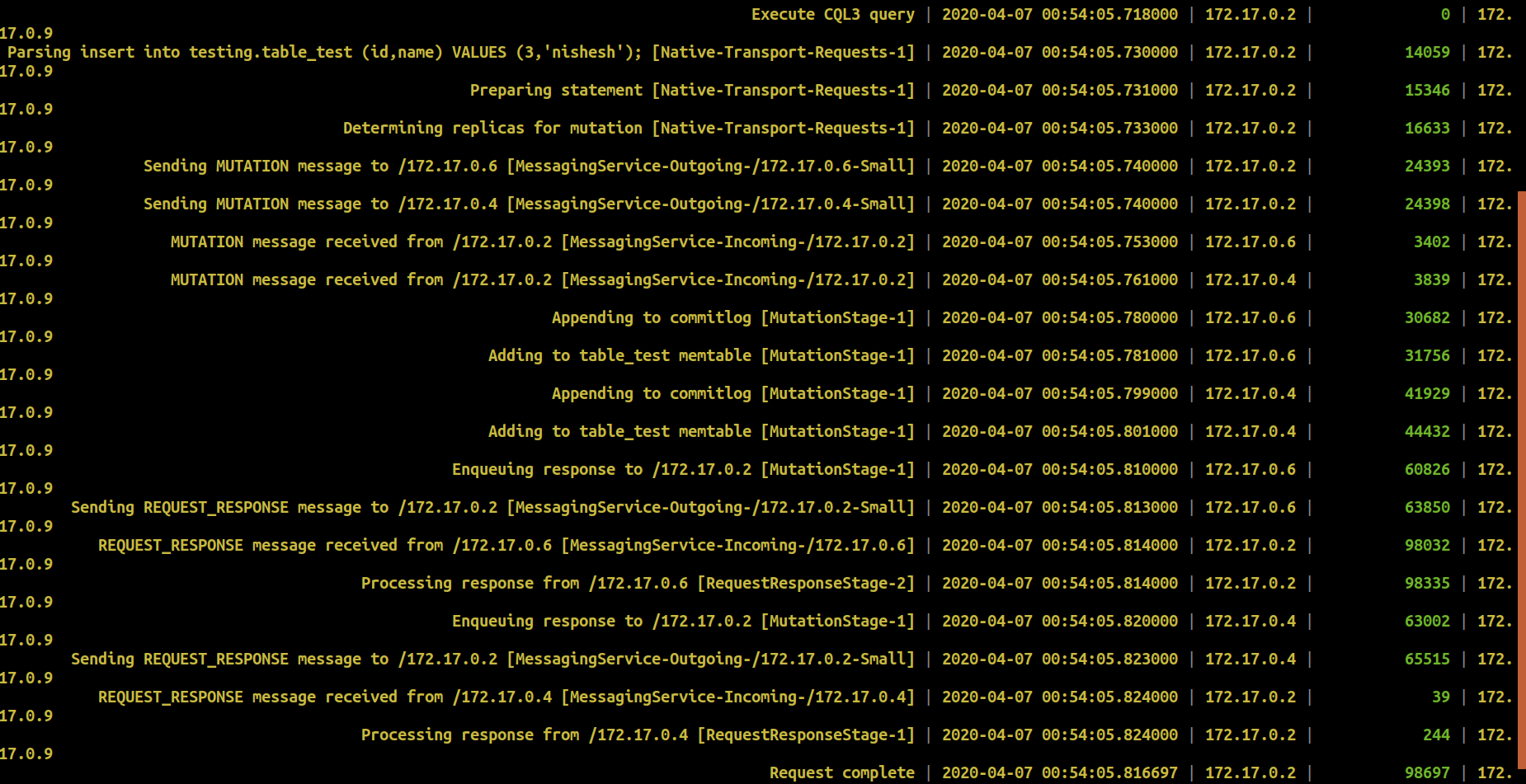


Fig 3: Insertion of record in Cassandra while enabling tracing

 Fig 4: Tracing of the insert statement

From figure(2) when partition key = 3, we got the endpoint of the node where the data should be stored, which was 172.17.0.6 and 172.17.0.4. This can be proved by figure 3 and 4. In figure 3, we are trying to insert a row in Cassandra table from 172.17.0.2 which is the request coordinator. Please note that there is no master and slave in Cassandra. The request coordinator then figures out where the data should be stored using the partitioner. In this case, it is 172.17.0.6. As we have set the replication factor to 2, which means the data must be stored in two different nodes. So, it must replicate the data in another one node in clockwise direction of 172.17.0.6 which happens to be 172.17.0.4.

**How to tune consistency in Cassandra:**

Several levels of consistency can be set in Cassandra which in turn affects the availability of database. If consistency is set to ALL, this means during a read operation, all the nodes must return the same data, which guarantees the highest level of consistency but affects the availability. And for the write operation, the data must be replicated to the n nodes (where n=replication factor) in the clockwise direction, then only the write operation is considered successful.

Similarly, if consistency is set to 1 and let us assume replication factor is 3. Then a write operation is successful, when the data is written to one node, and data is just replicated to one of the node, since this condition satisfies the consistency level. And for a read operation, if one of the replica returns the same value as the node where the data is stored, then the operation is successful. So, this is one of the least consistency levels in Cassandra.

We can set the consistency level in Cassandra using cqlsh. Cqlsh is command line interface which allows us to perform several actions on Cassandra database.

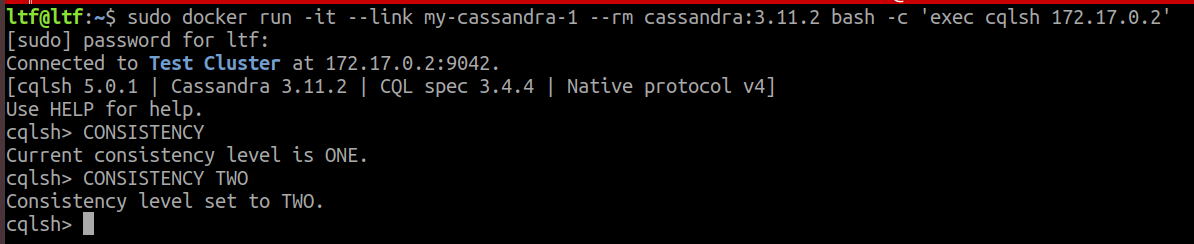


Fig 5: Setting consistency level in Cassandra using cqlsh

As shown in figure 5, when we enter the command “CONSISTENCY” in cqlsh, we get the current consistency level of the database. If we would like to change the consistency level that can be done using the command “CONSISTENCY <consistency\_level>”.

There are several consistency levels in Cassandra. For more information on consistency levels, please refer to this URL:

<https://docs.datastax.com/en/ddac/doc/datastax_enterprise/dbInternals/dbIntConfigConsistency.html>

**Data Modelling in Cassandra:**

* In RDBMS, we used to define entity along with its relationships. But data modelling in Cassandra is driven by our read operations.
* We need to design tables in such a way that our use cases are satisfied using minimum number of partitions (ideally one).
* The very important point to realize in data modelling using Cassandra is to optimize read operations, which means reading data from minimum number of partitions.
* This means there will be data duplications.

**Application Implementation:**

I have taken IMDB movie dataset from the link <https://www.imdb.com/interfaces/>. The data set has columns as mentioned below:

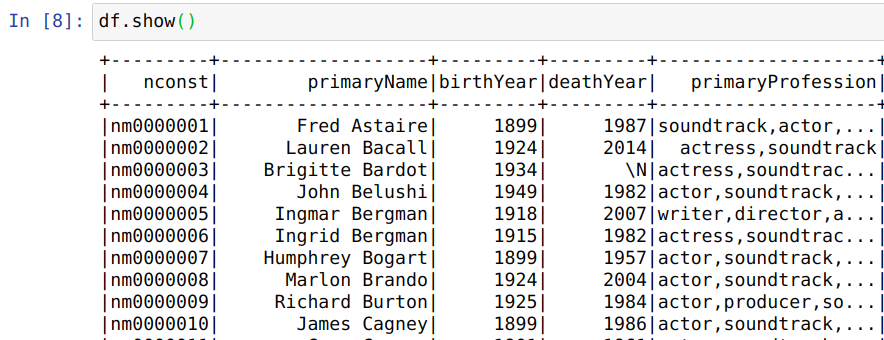


Fig 6: IMDB Dataset

**Columns description:**

nconst = Unique row identifier

PrimaryName = Name of the person involved in the field of movie

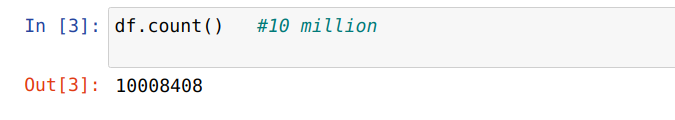
birthYear and deathYear of the person

PrimaryProfession = Top 3 professions of the person

**Goal from the data:**

* Given the profession, find the people in that profession.

**Number of records in dataset:**



**Data Modelling in RDBMS:**

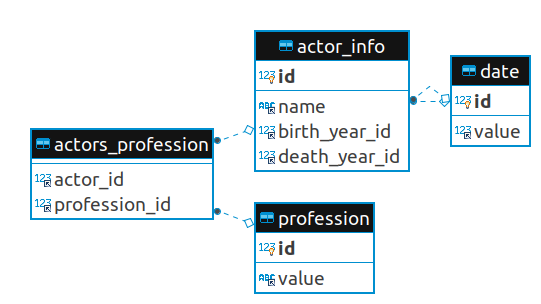


Fig 7: Data Modelling in RDBMS

I have used “LOAD DATA INFILE” approach provided by MySQL to load the data into a temporary table. More information about the same is present here. <https://dev.mysql.com/doc/refman/8.0/en/load-data.html>

Then, I used that temporary table to load the data into the different tables (as shown in fig 7) using SQL queries. Because I don’t want this report to be lengthy, so I am not including those queries here.

**RDBMS Query to fetch people by their profession:**

Fig 8 shows the query to find out the people whose profession is “actor”.

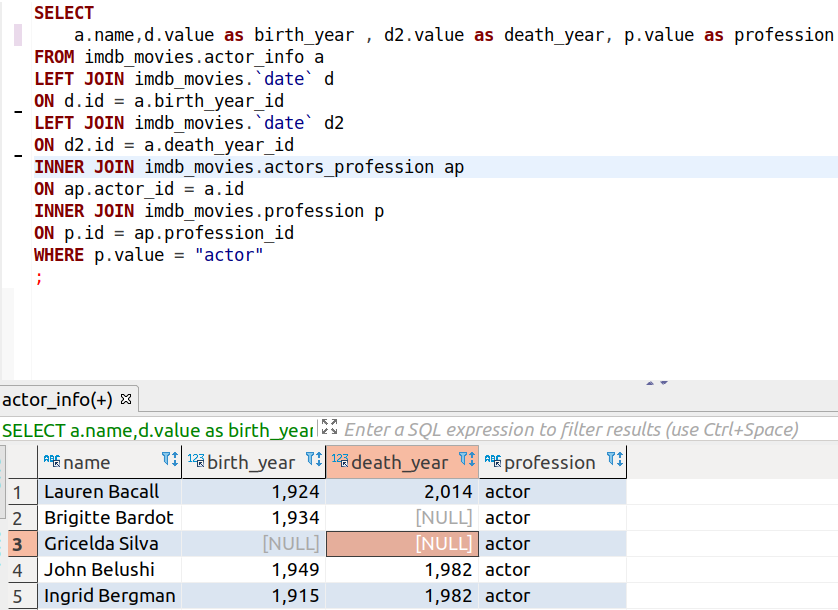


Fig 8: SQL query to fetch people by their profession

Please note that some of the actor’s birth date is missing in the original data; hence it shows null for those actors.

Also, fig 8 query is an expensive operation. Fig 9 shows the execution plan for this query:

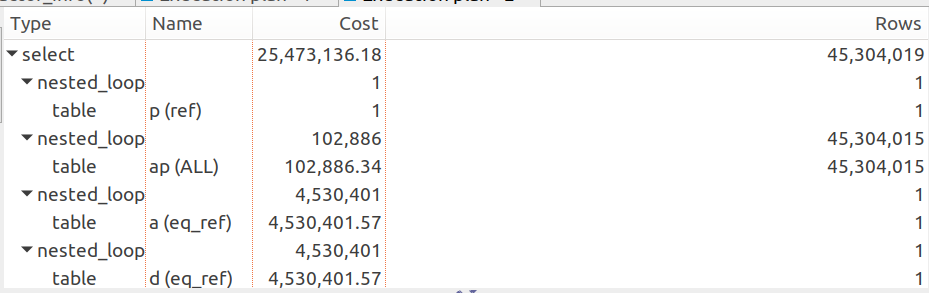


Fig 9: Execution plan for query of fig(8)

So, I had to use BTree indexes in several fields of the different table to optimize the read operation. The performance improved but still the data size being very huge around 45 millions, I could not retrieve all the query result. So, I had to limit the result to 200.

Next, I tried implementing the same in Cassandra with 4 node cluster. My expectation from cassandra is that it should execute the above query and fetch the result in least time.

As mentioned earlier, the data modelling is cassandra is query driven. Another goal is to read from the minimum number of partitions(ideally one). There are no joins in Cassandra, which means there is data duplication.

The query we would like to answer is “Get me the person information whose profession is ‘x’ ”, Where x is the profession name.

So, firstly I would create a keyspace (keyspace is like creating database in RDBMS) and table as shown in fig 10 and 11 respectively:



Fig 10: Creation of Keyspace

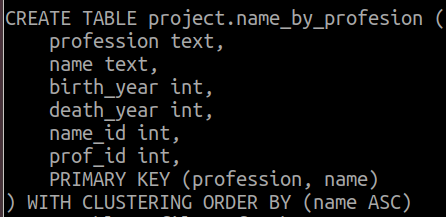


Fig 11: Table creation

I have decided to make combination of profession and name as the primary key, and profession is the partitioning key here. So, when we query the table by profession, we have to visit minimum number of partitions to fetch the result. From the above table structure, it is clear that the data is being duplicated, as an actor can have multiple role. But, that is the life of Cassandra. If we start applying JOINs then, we need to visit all the partitions to fetch the data.

To load the data into Cassandra, I used the SQL query result from fig 8 excluding the WHERE clause from the query. As the query was very expensive as shown in fig 9, I could only fetch 30000 records and write those into csv file which was finally imported into Cassandra.

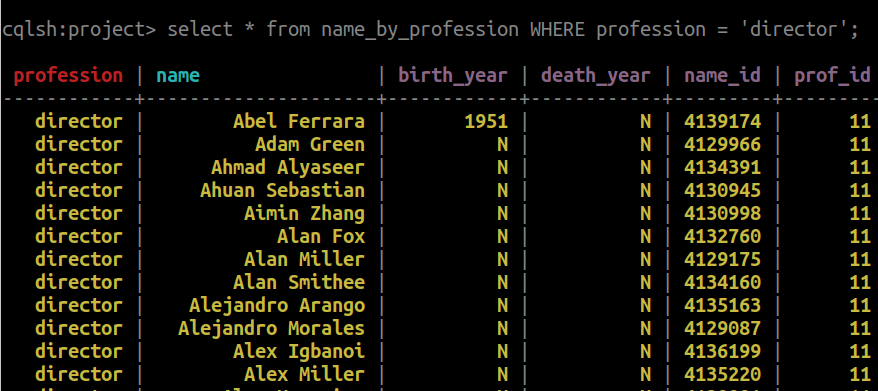


Fig 12: Filtering the result by profession as director

Fig 12 shows the table query by profession.

However, I could not reach my final goal which was comparing the read operation time difference between MySQL and Cassandra when querying against large datasets. The reason behind this was because I failed to run the query as shown in fig 8 due to hardware limitation.