Cassandra Assignment 2 - SWEN 432

Zoltan Debre - 300360191

Original repository and progress history: https://github.com/zoltan-nz/cassandra-exercise

Question 1

(2 marks)

- Use `ccm` to make a single data center Cassandra cluster having 5 nodes call it `single_dc`
- Start the cluster
- Run the `ccm node1 ring` command
- Save the output of the ring command for future use and show it in the answer to the question.

```
$ ccm create single_dc --nodes=5
$ ccm start
```

```
$ ccm status -v

Cluster: 'single_dc'
--------------------
node1: UP
       auto_bootstrap=False
       thrift=('127.0.0.1', 9160)
       binary=('127.0.0.1', 9042)
       storage=('127.0.0.1', 7000)
       jmx_port=7100
       remote_debug_port=0
       byteman_port=0
       initial_token=-9223372036854775808
       pid=41519

node3: UP
       auto_bootstrap=False
       thrift=('127.0.0.3', 9160)
       binary=('127.0.0.3', 9042)
       storage=('127.0.0.3', 7000)
       jmx_port=7300
       remote_debug_port=0
       byteman_port=0
       initial_token=-1844674407370955162
       pid=41547

node2: UP
       auto_bootstrap=False
       thrift=('127.0.0.2', 9160)
       binary=('127.0.0.2', 9042)
       storage=('127.0.0.2', 7000)
       jmx_port=7200
       remote_debug_port=0
       byteman_port=0
       initial_token=-5534023222112865485
       pid=41588

node5: UP
       auto_bootstrap=False
       thrift=('127.0.0.5', 9160)
       binary=('127.0.0.5', 9042)
       storage=('127.0.0.5', 7000)
       jmx_port=7500
       remote_debug_port=0
       byteman_port=0
       initial_token=5534023222112865484
       pid=41629
```

```
node4: UP
       auto_bootstrap=False
       thrift=('127.0.0.4', 9160)
       binary=('127.0.0.4', 9042)
       storage=('127.0.0.4', 7000)
       jmx_port=7400
       remote_debug_port=0
       byteman_port=0
       initial_token=1844674407370955161
       pid=41665
```

```
$ ccm node1 ring

Datacenter: datacenter1
==========
Address    Rack        Status State   Load            Owns                Token
                                                                          5534023222112865484
127.0.0.1  rack1       Up     Normal  98.97 KiB       40.00%              -9223372036854775808
127.0.0.2  rack1       Up     Normal  98.98 KiB       40.00%              -5534023222112865485
127.0.0.3  rack1       Up     Normal  98.98 KiB       40.00%              -1844674407370955162
127.0.0.4  rack1       Up     Normal  98.98 KiB       40.00%              1844674407370955161
127.0.0.5  rack1       Up     Normal  98.98 KiB       40.00%              5534023222112865484
```

Question 2

(14 marks)

- a) (2 marks) What is the setting of the `endpoint_snitch` property?

Location of `node1`'s `cassandra.yaml`: `~/.ccm/single_dc/node1/conf/cassandra.yaml`
(You can find a copy of `cassandra.yaml` in `node1` sub-folder.)

```
endpoint_snitch: SimpleSnitch
```

- b) (6 Marks) What is the value of the `initial_token` property?

```
initial_token: -9223372036854775808
```

- Which Cassandra component has calculated it?

default partitioner:

```
Murmur3Partitioner
```

- Is there any relationship between `initial_token` property value and the output of the `ccm node1 ring` command?

Yes.

The first node from the ccm node1 ring output:

```
127.0.0.1  rack1       Up     Normal  98.97 KiB       40.00%              -9223372036854775808
```

The hash value of the first node is the same as the `initial_token`.

- c) (2 marks) What is the setting of the `partitioner` property?

```
partitioner: org.apache.cassandra.dht.Murmur3Partitioner
```

- d) (4 marks) What is the setting of the `rpc_address` property?

```
rpc_address: 127.0.0.1
```

- Is there any relationship between `rpc_address` property value and the output of the `ccm node1 ring` command?

Yes, this value is the ip address of the first node

```
127.0.0.1  rack1       Up     Normal  98.97 KiB       40.00%              -9223372036854775808
```

Question 3

(2 marks) Consider the `casssandra.topology.properties` file of `node1` and comment on the relationship between file's content and the output of the `ccm node1 ring` command.

Location of the file: `~/.ccm/single_dc/node1/conf/cassandra-topology.properties`
(Copy saved in `node1` sub-folder.)

The main content of the property file:

```
# Cassandra Node IP=Data Center:Rack
192.168.1.100=DC1:RAC1
192.168.2.200=DC2:RAC2

10.0.0.10=DC1:RAC1
10.0.0.11=DC1:RAC1
10.0.0.12=DC1:RAC2

10.20.114.10=DC2:RAC1
10.20.114.11=DC2:RAC1

10.21.119.13=DC3:RAC1
10.21.119.10=DC3:RAC1

10.0.0.13=DC1:RAC2
10.21.119.14=DC3:RAC2
10.20.114.15=DC2:RAC2

# default for unknown nodes
default=DC1:r1
```

Because of using a single data center (Simple Snitch) we don't see any relation between this file and our `ccm node1 ring` output. Simple Snitch does not recognize data center or rack information. Our property file content is just dummy data, examples, not related to our node.

Question 4

(8 marks)

a) (3marks) Connect to `cqlsh` prompt and create a keyspace with the name `ass2`. Replication strategy should be `simple`, and the replication factor equal `3`. In your answer, show your keyspace declaration.

```
$ ccm node1 cqlsh
cqlsh> CREATE KEYSPACE IF NOT EXISTS ass2 with replication = { 'class' : 'SimpleStrategy',
'replication_factor' : 3 };
```

b) (5marks)

The following files:

```
table_declarations.cql
data_point_data.txt
driver_data_txt
time_table_data.txt
vehicle_data.txt
```

are given on the course Assignments page. The file `table_declarations.cql` contains create table statements, while the other files contain comma separated table data. Use these files, and `SOURCE` and `COPY` cqlsh commands to implement a version of the train time table data base. In your answer show the results of running the cqlsh command `describe tables` and of running `select` statements on each table for a row of your choice.

```
$ ccm node1 cqlsh
cqlsh> SOURCE './table_declarations.cql'
cqlsh> DESCRIBE TABLES;

Keyspace system_schema
----------------------
tables     triggers    views    keyspaces  dropped_columns
functions  aggregates  indexes  types      columns

Keyspace system_auth
--------------------
resource_role_permissons_index  role_permissions  role_members  roles

Keyspace system
---------------
available_ranges          peers               batchlog        transferred_ranges
batches                   compaction_history  size_estimates  hints
prepared_statements       sstable_activity    built_views
"IndexInfo"               peer_events         range_xfers
views_builds_in_progress  paxos               local

Keyspace system_distributed
---------------------------
repair_history  view_build_status  parent_repair_history

Keyspace system_traces
----------------------
events  sessions

Keyspace ass2
-------------
time_table  data_point  driver  vehicle
```

```
cqlsh> COPY ass2.time_table (line_name, service_no, time, distance, latitude, longitude, stop)
FROM './time_table_data.txt';
Using 7 child processes

Starting copy of ass2.time_table with columns [line_name, service_no, time, distance, latitude,
longitude, stop].
Processed: 30 rows; Rate:      44 rows/s; Avg. rate:      66 rows/s
30 rows imported from 1 files in 0.458 seconds (0 skipped).
cqlsh> SELECT * FROM ass2.time_table;

 line_name        | service_no | time | distance | latitude | longitude | stop
------------------+------------+------+----------+----------+-----------+--------------
          Melling |          3 |  807 |     13.7 | -41.2036 |  174.9054 |      Melling
          Melling |          3 |  801 |     11.4 | -41.2118 |    174.89 | Western Hutt
          Melling |          3 |  754 |      8.3 |  -41.227 |  174.8851 |       Petone
          Melling |          3 |  741 |        0 | -41.2865 |  174.7762 |   Wellington
 Hutt Valley Line |          1 |  650 |     34.3 | -41.1244 |  175.0708 |   Upper Hutt
 Hutt Valley Line |          1 |  642 |     26.5 | -41.1479 |  175.0122 | Silverstream
 Hutt Valley Line |          1 |  634 |       19 | -41.1798 |  174.9608 |        Taita
 Hutt Valley Line |          1 |  629 |     15.8 | -41.2024 |  174.9423 |       Naenae
 Hutt Valley Line |          1 |  625 |     13.3 | -41.2092 |  174.9081 |     Waterloo
 Hutt Valley Line |          1 |  622 |       11 | -41.2204 |  174.9081 |       Woburn
 Hutt Valley Line |          1 |  617 |      8.3 |  -41.227 |  174.8851 |       Petone
 Hutt Valley Line |          1 |  605 |        0 | -41.2865 |  174.7762 |   Wellington
         Waikanae |          5 | 1139 |     62.8 | -40.8755 |  175.0668 |     Waikanae
         Waikanae |          5 | 1118 |     51.3 | -40.9142 |  175.0084 |  Paraparaumu
         Waikanae |          5 | 1059 |     33.1 | -40.9881 |   174.951 |  Paekakariki
         Waikanae |          5 | 1042 |     15.9 | -41.1339 |  174.8406 |      Porirua
         Waikanae |          5 | 1025 |        0 | -41.2865 |  174.7762 |   Wellington
 Hutt Valley Line |         11 | 2025 |     34.3 | -41.1244 |  175.0708 |   Upper Hutt
 Hutt Valley Line |         11 | 2019 |     26.5 | -41.1479 |  175.0122 | Silverstream
 Hutt Valley Line |         11 | 2010 |       19 | -41.1798 |  174.9608 |        Taita
 Hutt Valley Line |         11 | 2001 |     15.8 | -41.2024 |  174.9423 |       Naenae
 Hutt Valley Line |         11 | 1955 |     13.3 | -41.2092 |  174.9081 |     Waterloo
 Hutt Valley Line |         11 | 1952 |       11 | -41.2204 |  174.9081 |       Woburn
 Hutt Valley Line |         11 | 1947 |      8.3 |  -41.227 |  174.8851 |       Petone
 Hutt Valley Line |         11 | 1935 |        0 | -41.2865 |  174.7762 |   Wellington
 Hutt Valley Line |          2 | 1045 |     34.3 | -41.2865 |  174.7762 |   Wellington
 Hutt Valley Line |          2 | 1033 |       26 |  -41.227 |  174.8851 |       Petone
```

```
 Hutt Valley Line |          2 | 1025 |       21 | -41.2092 |  174.9081 |     Waterloo
 Hutt Valley Line |          2 | 1015 |     15.3 | -41.1798 |  174.9608 |        Taita
 Hutt Valley Line |          2 | 1000 |        0 | -41.1244 |  175.0708 |   Upper Hutt

(30 rows)
```

```
cqlsh> COPY ass2.data_point FROM './data_point_data.txt';
Using 7 child processes

Starting copy of ass2.data_point with columns [line_name, service_no, date, sequence, latitude,
longitude, speed].
Processed: 5 rows; Rate:       8 rows/s; Avg. rate:      11 rows/s
5 rows imported from 1 files in 0.437 seconds (0 skipped).
cqlsh> SELECT * FROM ass2.data_point;

 line_name       | service_no | date     | sequence                | latitude | longitude | speed
-----------------+------------+----------+-------------------------+----------+-----------+-------
 Hutt Valey Line |          2 | 20160326 | 2016-03-25 21:07:40.000 | -41.2012 |       175 |  70.1
 Hutt Valey Line |          2 | 20160326 | 2016-03-25 21:02:10.000 | -41.1255 |    175.07 |  40.5
 Hutt Valey Line |          2 | 20160326 | 2016-03-24 21:27:10.000 | -41.2262 |    174.77 |  29.1
 Hutt Valey Line |          2 | 20150322 | 2015-03-21 21:44:10.000 | -41.2862 |  174.7759 |   9.1
 Hutt Valey Line |          2 | 20160322 | 2016-03-21 21:37:50.000 | -41.2272 |    174.77 |  29.1

(5 rows)
```

```
cqlsh> COPY ass2.driver FROM './driver_data.txt';
Using 7 child processes

Starting copy of ass2.driver with columns [driver_name, current_position, email, mobile, password,
skill].
Processed: 6 rows; Rate:      10 rows/s; Avg. rate:      14 rows/s
6 rows imported from 1 files in 0.430 seconds (0 skipped).
cqlsh> SELECT * FROM ass2.driver;

 driver_name | current_position | email                 | mobile   | password | skill
-------------+------------------+-----------------------+----------+----------+------------------------------------
        fred |            Taita |   fred@ecs.vuw.ac.nz |  2799797 |     f00f | {'Ganz Mavag', 'Guliver'}
        jane |         Waikanae |   jane@ecs.vuw.ac.nz |  2131131 |     jj77 | {'Matangi'}
         ann |    not available |    ann@ecs.vuw.ac.nz | 21998877 |     aaaa | {'Matangi'}
       milan |       Upper Hutt |  milan@ecs.vuw.ac.nz |   211111 |     mm77 | {'Matangi'}
       pondy |       Wellington |  pondy@ecs.vuw.ac.nz |   214455 |     pd66 | {'Guliver', 'Matangi'}
       pavle |       Upper Hutt | pmogin@ecs.vuw.ac.nz |   213344 |     pm33 | {'Ganz Mavag', 'Guliver', 'Matangi

(6 rows)
```

```
cqlsh> COPY ass2.vehicle FROM './vehicle_data.txt';
Using 7 child processes

Starting copy of ass2.vehicle with columns [vehicle_id, status, type].
Processed: 6 rows; Rate:      10 rows/s; Avg. rate:      14 rows/s
6 rows imported from 1 files in 0.433 seconds (0 skipped).
cqlsh> SELECT * FROM ass2.vehicle;

 vehicle_id | status       | type
------------+--------------+------------
     KW3300 |   Wellington |    Matangi
     FP3003 | out of order |    Guliver
     FA3456 |       in_use |    Matangi
     FP8899 |   Upper Hutt |    Matangi
     FA4864 |  maintenance |    Matangi
     FA1122 |   Upper Hutt | Ganz Mavag

(6 rows)
```

Question 5

(10 marks) To answer this question, you will need to use the `getendpoints nodetool` command.

a) (1 mark) Find the nodes storing data of driver `pavle`. In your answer, show the output of the getendpoints nodetool command. Let us call these nodes `node_a`, `node_b`, and `node_c`.

```
$ ccm node1 nodetool getendpoints ass2 driver pavle

127.0.0.1
127.0.0.2
127.0.0.3
```

b) (3 marks)

Connect to `cqlsh` prompt using a node that is not in the set `{node_a, node_b, node_c}`.

```
$ ccm node4 cqlsh
```

Set the consistency level to ALL and read data of the driver pavle.

```
cqlsh> CONSISTENCY;
Current consistency level is ONE.
cqlsh> CONSISTENCY ALL;
Consistency level set to ALL.
cqlsh> SELECT * FROM ass2.driver WHERE driver_name='pavle';

 driver_name | current_position | email                | mobile | password | skill
-------------+------------------+----------------------+--------+----------+-------------------------------------
       pavle |       Upper Hutt | pmogin@ecs.vuw.ac.nz | 213344 |     pm33 | {'Ganz Mavag', 'Guliver', 'Matangi'}

(1 rows)
```

Stop `node_a`, connect to cqlsh, set the consistency level to ALL and read pavle's data again. What have you learned?

```
$ ccm node1 stop
$ ccm node4 cqlsh
cqlsh> CONSISTENCY ALL;
Consistency level set to ALL.
cqlsh> SELECT * FROM ass2.driver WHERE driver_name='pavle';
NoHostAvailable:
```

`CONSISTENCY ALL` in Read Consistency Levels means that Cassandra returns the record after all replicas have responded. The read operation will fail if a replica does not respond. Exactly this happened in our case.

c) (3 marks)

With `node_a` still being stopped, set the consistency level to QOURUM and read pavle's data.

```
cqlsh> CONSISTENCY QUORUM ;
Consistency level set to QUORUM.
cqlsh> SELECT * FROM ass2.driver WHERE driver_name='pavle';

 driver_name | current_position | email                | mobile | password | skill
-------------+------------------+----------------------+--------+----------+-------------------------------------
       pavle |       Upper Hutt | pmogin@ecs.vuw.ac.nz | 213344 |     pm33 | {'Ganz Mavag', 'Guliver', 'Matangi'}

(1 rows)
```

Stop node_b, connect to cqlsh, set the consistency level to QUORUM and read pavle's data again. What have you learned

```
$ ccm node2 stop
$ ccm node4 cqlsh
cqlsh> CONSISTENCY QUORUM;
Consistency level set to QUORUM.
cqlsh> SELECT * FROM ass2.driver WHERE driver_name='pavle';
NoHostAvailable:
```

The `CONSISTENCY QUORUM` means that Cassandra will return the record after a quorum of replicas from all datacenters has responded. We have one datacenter at this stage, the replaction factor is 3, so at least 2 should respond. We got results when only one node was dead, but no responses when 2 were dead from 3 nodes.

d) (3 marks)

With node_a and node_b still being stopped, set the consistency level to ONE and read pavle's data.

```
cqlsh> CONSISTENCY ONE ;
Consistency level set to ONE.
cqlsh> SELECT * FROM ass2.driver WHERE driver_name='pavle';

 driver_name | current_position | email                | mobile | password | skill
-------------+------------------+----------------------+--------+----------+--------------------------------------
       pavle |       Upper Hutt | pmogin@ecs.vuw.ac.nz | 213344 |     pm33 | {'Ganz Mavag', 'Guliver', 'Matangi'}

(1 rows)
```

Stop node_c, connect to cqlsh, and read pavle's data again. What have you learned

```
$ ccm node3 stop
$ ccm node4 cqlsh
cqlsh> CONSISTENCY ONE;
Consistency level set to ONE.
cqlsh> SELECT * FROM ass2.driver WHERE driver_name='pavle';
NoHostAvailable:
```

`CONSISTENCY ONE` : Returns a response from the closest replica, as determined by the snitch. We have one data center and 3 replicas. In the first case we still had one node available, so one replica was still existed. But after stopping node_c does not left any live replica, so no response.

Question 6

(15 marks)

You are asked to find those nodes of the `single_dc` Cassandra cluster that store replicas of driver `eileen` . Very soon you realized that all `ccm` commands and `nodetool` commands, including `ccm start` , `ccm stop` , `ccm status` , `ccm nodei cqlsh` and so on, work properly except the command `ccm nodei nodetool getendpoints ass2 driver eileen` . Despite that, you have devised a procedure to find the nodes requested. In your answer, describe the procedure and show how you have applied it.

First of all, we don't have driver with `eileen` in our database. We can check it with the following query.

```
$ ccm start
$ ccm switch single_dc
$ ccm status

Cluster: 'single_dc'
--------------------
node1: UP
node3: UP
node2: UP
node5: UP
node4: UP

$ ccm node1 cqlsh -e "use ass2; select * from driver where driver_name = 'eileen';"

   driver_name | current_position | email | mobile | password | skill
  -------------+------------------+-------+--------+----------+-------

  (0 rows)

$ ccm node1 cqlsh -e "INSERT INTO ass2.driver (driver_name, current_position, email, mobile,
password, skill) VALUES ('eileen', 'Wellington', 'eileen@ecs.vuw.ac.nz', 555444, 'abcd123',
{'Guliver', 'Matangi'}) IF NOT EXISTS;"

 [applied]
-----------
      True

ccm node1 cqlsh -e "use ass2; select * from driver where driver_name = 'eileen';"

 driver_name | current_position | email                | mobile | password | skill
-------------+------------------+----------------------+--------+----------+------------------------
```

```
      eileen |       Wellington | eileen@ecs.vuw.ac.nz | 555444 |  abcd123 | {'Guliver', 'Matangi'}

(1 rows)
```

Without using `nodetool`, we can find nodes which stores our record, if we close all other nodes except one. Using `CONSISTENCY ONE`, we get back our record if that node stores our requested data.

The following bash script can help us to iterate through on all nodes and run the query. (`q6-node-finder.sh`)

```
nodes=('node1' 'node2' 'node3' 'node4' 'node5');
for node in "${nodes[@]}"; do
  ccm switch single_dc
  ccm stop
  ccm ${node} start
  echo "Active node: ${node}"
  ccm status
  ccm ${node} cqlsh -e "use ass2; consistency one; select * from driver where driver_name='eileen';"
done
```

Or we can run manually this one line for each node:

```
$ ccm switch single_dc; ccm stop; ccm node1 start; ccm status; ccm node1 cqlsh -e "use ass2;
consistency one; select * from driver where driver_name='eileen';"

Cluster: 'single_dc'
--------------------
node1: UP
node3: DOWN
node2: DOWN
node5: DOWN
node4: DOWN

 driver_name | current_position | email                | mobile | password | skill
-------------+------------------+----------------------+--------+----------+------------------------
      eileen |       Wellington | eileen@ecs.vuw.ac.nz | 555444 |  abcd123 | {'Guliver', 'Matangi'}

(1 rows)

$ ccm switch single_dc; ccm stop; ccm node2 start; ccm status; ccm node2 cqlsh -e "use ass2;
consistency one; select * from driver where driver_name='eileen';"

Cluster: 'single_dc'
--------------------
node1: DOWN
node3: DOWN
node2: UP
node5: DOWN
node4: DOWN

Consistency level set to ONE.

 driver_name | current_position | email                | mobile | password | skill
-------------+------------------+----------------------+--------+----------+------------------------
      eileen |       Wellington | eileen@ecs.vuw.ac.nz | 555444 |  abcd123 | {'Guliver', 'Matangi'}

(1 rows)

$ ccm switch single_dc; ccm stop; ccm node3 start; ccm status; ccm node3 cqlsh -e "use ass2;
consistency one; select * from driver where driver_name='eileen';"

Cluster: 'single_dc'
--------------------
node1: DOWN
node3: UP
node2: DOWN
node5: DOWN
node4: DOWN

Consistency level set to ONE.

NoHostAvailable:

$ ccm switch single_dc; ccm stop; ccm node4 start; ccm status; ccm node4 cqlsh -e "use ass2;
consistency one; select * from driver where driver_name='eileen';"
```

```
Cluster: 'single_dc'
--------------------
node1: DOWN
node3: DOWN
node2: DOWN
node5: DOWN
node4: UP

Consistency level set to ONE.

NoHostAvailable:

$ ccm switch single_dc; ccm stop; ccm node5 start; ccm status; ccm node5 cqlsh -e "use ass2;
consistency one; select * from driver where driver_name='eileen';"

Cluster: 'single_dc'
--------------------
node1: DOWN
node3: DOWN
node2: DOWN
node5: UP
node4: DOWN

Consistency level set to ONE.

 driver_name | current_position | email                 | mobile | password | skill
-------------+------------------+-----------------------+--------+----------+------------------------
      eileen |       Wellington | eileen@ecs.vuw.ac.nz | 555444 |  abcd123 | {'Guliver', 'Matangi'}

(1 rows)
```

We can see, that our record exists on `node1`, `node2` and `node5`.

We can test this with our restricted command:

```
$ ccm start; ccm node1 nodetool getendpoints ass2 driver eileen
127.0.0.5
127.0.0.1
127.0.0.2
```

Update:

I found a more elegant solution when I solved Question 15. Using the token function.

```
$ ccm start
$ ccm node1 cqlsh -e "USE ass2; SELECT driver_name, TOKEN(driver_name) FROM driver WHERE
driver_name='eileen'"

   driver_name | system.token(driver_name)
  -------------+---------------------------
        eileen |       2694043365177161046

  (1 rows)
```

Comparing our above token number with previously printed ring token thresholds, we can see, that `eileen`'s token is above `node4`'s threshold but smaller than `node5`. It means, that the primary node is `node5`. In the ring the following nodes, which will store replicas, are `node1` and `node2`. Again, the answer is `node1`, `node2` and `node5`.

```
Datacenter: datacenter1
==========
Address    Rack        Status State   Load            Owns                Token
...
127.0.0.4  rack1       Up     Normal  98.98 KiB       40.00%              1844674407370955161
127.0.0.5  rack1       Up     Normal  98.98 KiB       40.00%              5534023222112865484
```

Question 7

(15 marks)

Assume the following situation:

The data of the driver `james` should be stored on `node4`, `node5`, and `node1`.

A client (say c0) connected to node3 and sent a request to write james's data.

In the moment of running the statement `insert into driver (driver_name, password) values ('james', '7007');` `node4` was down.

Writing succeeded.

In the next moment `node5` and `node1` went down and the `node4` started.

A client (say c1) connected to `cqlsh` prompt via `node3` and sent the following read statement: `select driver_name, password from driver where driver_name = 'james';`

The read result was:

```
 driver_name | password |
-------------+----------+
       james |     7007 |
```

Repeat the experiment described above. Name and briefly explain Cassandra mechanism that made succeeding of the select statement above possible.

We can see where cassandra would like to store our record. Because the primary key in `driver` table is the `driver_name`, we can list our nodes. And it is really the `node1`, `node4` and `node5`.

```
$ ccm start
$ ccm node1 nodetool getendpoints ass2 driver james

127.0.0.4
127.0.0.5
127.0.0.1

$ ccm status

Cluster: 'single_dc'
--------------------
node1: UP
node3: UP
node2: UP
node5: UP
node4: UP
```

Simulate `node4` is down.

```
$ ccm node4 stop
$ ccm status
Cluster: 'single_dc'
--------------------
node1: UP
node3: UP
node2: UP
node5: UP
node4: DOWN
```

We may insert our new row with the following command.

```
$ ccm node3 cqlsh -e "use ass2; insert into driver (driver_name, password) values ('james', '7007');"
```

However, if we don't change our consistency level than our experiment will fail. The default consistency level `ONE`. In this case Cassandra will not replicate our record to `node4` when it starts. We have to switch at least to `CONSISTENCY QUORUM`.

We have to use the following command to insert our data.

```
$ ccm node3 cqlsh -e "use ass2; consistency quorum; insert into driver (driver_name, password)
values ('james', '7007');"
```

Simulate `node1`, `node5` are down and `node4` is back.

```
$ ccm node1 stop; ccm node5 stop; ccm node4 start
$ ccm status
Cluster: 'single_dc'
--------------------
node1: DOWN
node3: UP
node2: UP
node5: DOWN
node4: UP

$ ccm node3 cqlsh -e "use ass2; select driver_name, password from driver where driver_name='james';"

 driver_name | password
-------------+----------
       james |     7007

(1 rows)
```

When we run our insert command with quorum consistency, Cassandra write our record at least in two nodes, plus it will write in log and in the memtable. When our `node1` and `node5` is stoped and node4 came back, Cassandra inserted our record from memtable in `node4`.

Cassandra uses gossip process to track states of nodes. It helps to determine which node is up or down and when it can replicate a missing data.

Question 8

Lost...

Question 9

(3 marks)

Use ccm to make a Cassandra cluster spanning two datacenters. The cluster name should be `multi_dc`.
Cassandra will automatically assign default names `dc1` and `dc2` to datacenters. The cluster `multi_dc` uses 5 nodes in `dc1` and 4 nodes in `dc2`. Start the cluster and run the `ccm ring` command. Save the output of the ring command for future use and show it in the answer to the question.

```
$ ccm create -n 5:4 -s multi_dc
$ ccm switch multi_dc
$ ccm start
$ ccm status

Cluster: 'multi_dc'
-------------------
node9: UP
node8: UP
node1: UP
node3: UP
node2: UP
node5: UP
node4: UP
node7: UP
node6: UP

$ ccm node1 ring

Datacenter: dc1
==========
Address    Rack        Status State   Load            Owns                Token
                                                                          5534023222112865484
127.0.0.1  r1          Up     Normal  98.97 KiB       25.00%              -9223372036854775808
127.0.0.2  r1          Up     Normal  98.96 KiB       20.00%              -5534023222112865485
127.0.0.3  r1          Up     Normal  98.97 KiB       20.00%              -1844674407370955162
```

```
127.0.0.4  r1          Up     Normal  98.97 KiB       20.00%              1844674407370955161
127.0.0.5  r1          Up     Normal  98.96 KiB       20.00%              5534023222112865484


Datacenter: dc2
==========
Address    Rack        Status State   Load            Owns                Token
                                                                          4611686018427388004
127.0.0.6  r1          Up     Normal  98.96 KiB       20.00%              -9223372036854775708
127.0.0.7  r1          Up     Normal  98.97 KiB       25.00%              -4611686018427387804
127.0.0.8  r1          Up     Normal  98.94 KiB       25.00%              100
127.0.0.9  r1          Up     Normal  98.97 KiB       25.00%              4611686018427388004
```

Question 10

(4 marks)

Consider the `casssandra.yaml` file of `node1`. What is the setting of the `endpoint_snitch` property? If you find it different to the setting in the case of the `single_dc` cluster, explain briefly why it is different.

(Please find the `cassandra.yaml` in the `multi_dc` folder.)

```
endpoint_snitch: org.apache.cassandra.locator.PropertyFileSnitch
```

In case of `single_dc` our `endpoint_snitch` value was `SimpleSnitch`. In case of `multi_dc` Cassandra uses `PropertyFileSnitch` protocol.

The `SimpleSnitch` is used only for single-datacenter deployments. `PropertyFileSnitch` determines the location of nodes by rack and datacenter. It uses the network details located in the `cassandra-topology.properties` file (copied in `multi_dc` folder).

Question 11

(4 marks)

Consider the `cassandra.topology.properties` file of `node1` and comment on the relationship between file's content and the output of the `ccm node1 ring` command.

Content of `cassandra.topology.properties`:

```
default=dc1:r1
127.0.0.1=dc1:r1
127.0.0.2=dc1:r1
127.0.0.3=dc1:r1
127.0.0.4=dc1:r1
127.0.0.5=dc1:r1
127.0.0.6=dc2:r1
127.0.0.7=dc2:r1
127.0.0.8=dc2:r1
127.0.0.9=dc2:r1
```

`cassandra.topology.properties` file contains the same mapping as we can list with `ccm node1 ring`. We see in both cases that we have two data centers and which IP address, which node belongs to `dc1` or to `dc2` datacenter.

Both list the rack numbers also. In our case, we use only one-one rack.

Question 12

(2 marks)

Create a keyspace with the name `ass2` having network topology replication strategy and a replication factor of 3 for both `dc1` and `dc2` datacenters. In your answer, show your keyspace declaration.

```
cqlsh> CREATE KEYSPACE IF NOT EXISTS ass2 WITH replication = {'class': 'NetworkTopologyStrategy',
'dc1': 3, 'dc2': 3 };
```

Question 13

(3 marks)

Use `SOURCE` and `COPY` `cqlsh` commands and the following files:

```
table_declarations.cql
driver_data_txt
time_table_data.txt
```

to implement a version of the train time table data base. You need to populate only `driver` and `time_table` tables by data. In your answer show the results of running the `cqlsh` command `describe tables` and of running CQL `select` statements on `driver` and `time_table` for a row of your choice.

```
$ ccm node1 cqlsh -e "use ass2; SOURCE './table_declarations.cql';"
$ ccm node1 cqlsh
cqlsh> DESCRIBE tables;

Keyspace system_schema
----------------------
tables     triggers    views    keyspaces  dropped_columns
functions  aggregates  indexes  types      columns

Keyspace system_auth
--------------------
resource_role_permissons_index  role_permissions  role_members  roles

Keyspace system
---------------
available_ranges          peers               batchlog        transferred_ranges
batches                   compaction_history  size_estimates  hints
prepared_statements       sstable_activity    built_views
"IndexInfo"               peer_events         range_xfers
views_builds_in_progress  paxos               local

Keyspace system_distributed
---------------------------
repair_history  view_build_status  parent_repair_history

Keyspace system_traces
----------------------
events  sessions

Keyspace ass2
-------------
time_table  data_point  driver  vehicle
```

```
cqlsh> USE ass2;
cqlsh:ass2> COPY driver FROM './driver_data.txt';
Using 7 child processes

Starting copy of ass2.driver with columns [driver_name, current_position, email, mobile,
password, skill].
Processed: 6 rows; Rate:       6 rows/s; Avg. rate:       9 rows/s
6 rows imported from 1 files in 0.664 seconds (0 skipped).
cqlsh:ass2> SELECT * FROM driver;

 driver_name | current_position | email                 | mobile   | password | skill
-------------+------------------+-----------------------+----------+----------+------------------------------------
        fred |            Taita |   fred@ecs.vuw.ac.nz |  2799797 |     f00f | {'Ganz Mavag', 'Guliver'}
        jane |         Waikanae |   jane@ecs.vuw.ac.nz |  2131131 |     jj77 | {'Matangi'}
         ann |    not available |    ann@ecs.vuw.ac.nz | 21998877 |     aaaa | {'Matangi'}
       milan |       Upper Hutt |  milan@ecs.vuw.ac.nz |   211111 |     mm77 | {'Matangi'}
       pondy |       Wellington |  pondy@ecs.vuw.ac.nz |   214455 |     pd66 | {'Guliver', 'Matangi'}
       pavle |       Upper Hutt | pmogin@ecs.vuw.ac.nz |   213344 |     pm33 | {'Ganz Mavag', 'Guliver', 'Matangi

(6 rows)
```

```
cqlsh:ass2> COPY time_table FROM './time_table_data.txt';
Using 7 child processes

Starting copy of ass2.time_table with columns [line_name, service_no, time, distance, latitude,
longitude, stop].
Processed: 30 rows; Rate:      47 rows/s; Avg. rate:      86 rows/s
30 rows imported from 1 files in 0.348 seconds (0 skipped).
cqlsh:ass2> SELECT * FROM time_table;

 line_name         | service_no | time | distance | latitude | longitude | stop
-------------------+------------+------+----------+----------+-----------+--------------
           Melling |          3 |  807 |     13.7 | -41.2036 |  174.9054 |      Melling
           Melling |          3 |  801 |     11.4 | -41.2118 |    174.89 | Western Hutt
           Melling |          3 |  754 |      8.3 |  -41.227 |  174.8851 |       Petone
           Melling |          3 |  741 |        0 | -41.2865 |  174.7762 |   Wellington
 Hutt Valley Line |          1 |  650 |     34.3 | -41.1244 |  175.0708 |   Upper Hutt
 Hutt Valley Line |          1 |  642 |     26.5 | -41.1479 |  175.0122 | Silverstream
 Hutt Valley Line |          1 |  634 |       19 | -41.1798 |  174.9608 |        Taita
 Hutt Valley Line |          1 |  629 |     15.8 | -41.2024 |  174.9423 |       Naenae
 Hutt Valley Line |          1 |  625 |     13.3 | -41.2092 |  174.9081 |     Waterloo
 Hutt Valley Line |          1 |  622 |       11 | -41.2204 |  174.9081 |       Woburn
 Hutt Valley Line |          1 |  617 |      8.3 |  -41.227 |  174.8851 |       Petone
 Hutt Valley Line |          1 |  605 |        0 | -41.2865 |  174.7762 |   Wellington
          Waikanae |          5 | 1139 |     62.8 | -40.8755 |  175.0668 |     Waikanae
          Waikanae |          5 | 1118 |     51.3 | -40.9142 |  175.0084 |  Paraparaumu
          Waikanae |          5 | 1059 |     33.1 | -40.9881 |   174.951 |  Paekakariki
          Waikanae |          5 | 1042 |     15.9 | -41.1339 |  174.8406 |      Porirua
          Waikanae |          5 | 1025 |        0 | -41.2865 |  174.7762 |   Wellington
 Hutt Valley Line |         11 | 2025 |     34.3 | -41.1244 |  175.0708 |   Upper Hutt
 Hutt Valley Line |         11 | 2019 |     26.5 | -41.1479 |  175.0122 | Silverstream
 Hutt Valley Line |         11 | 2010 |       19 | -41.1798 |  174.9608 |        Taita
 Hutt Valley Line |         11 | 2001 |     15.8 | -41.2024 |  174.9423 |       Naenae
 Hutt Valley Line |         11 | 1955 |     13.3 | -41.2092 |  174.9081 |     Waterloo
 Hutt Valley Line |         11 | 1952 |       11 | -41.2204 |  174.9081 |       Woburn
 Hutt Valley Line |         11 | 1947 |      8.3 |  -41.227 |  174.8851 |       Petone
 Hutt Valley Line |         11 | 1935 |        0 | -41.2865 |  174.7762 |   Wellington
 Hutt Valley Line |          2 | 1045 |     34.3 | -41.2865 |  174.7762 |   Wellington
 Hutt Valley Line |          2 | 1033 |       26 |  -41.227 |  174.8851 |       Petone
 Hutt Valley Line |          2 | 1025 |       21 | -41.2092 |  174.9081 |     Waterloo
 Hutt Valley Line |          2 | 1015 |     15.3 | -41.1798 |  174.9608 |        Taita
 Hutt Valley Line |          2 | 1000 |        0 | -41.1244 |  175.0708 |   Upper Hutt

(30 rows)
```

Question 14

(8 marks)

Find nodes storing data of the driver `pavle`. Let these nodes be `node_a`, `node_b`, `node_c`, `node_d`, `node_e`, and `node_f`, where a < b < c < d < e < f.

```
$ ccm node1 nodetool getendpoints ass2 driver pavle

127.0.0.1
127.0.0.2
127.0.0.3
127.0.0.6
127.0.0.7
127.0.0.8
```

I. (4 marks)

Connect to ass2 keyspace.

```
$ ccm node1 cqlsh
Connected to multi_dc at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.10 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> USE ass2;
cqlsh:ass2>
```

Run the statement: `select driver_name, password from driver where driver_name = 'pavle';`
under consistency levels:

- `quorum`

```
cqlsh:ass2> CONSISTENCY QUORUM;
Consistency level set to QUORUM.
cqlsh:ass2> SELECT driver_name, password FROM driver WHERE driver_name = 'pavle';

 driver_name | password
-------------+----------
       pavle |     pm33

(1 rows)
```

- `each_qourum`

```
cqlsh:ass2> CONSISTENCY EACH_QUORUM;
Consistency level set to EACH_QUORUM.
cqlsh:ass2> SELECT driver_name, password FROM driver WHERE driver_name = 'pavle';

 driver_name | password
-------------+----------
       pavle |     pm33

(1 rows)
```

Run the select statement under consistency level local_quorum once for dc1 being local, and once for dc2 being local.

- `local_quorum` (on node1)

```
$ ccm node1 cqlsh -e "USE ass2; CONSISTENCY local_quorum; SELECT driver_name, password FROM
driver WHERE driver_name = 'pavle';"
Consistency level set to LOCAL_QUORUM.

 driver_name | password
-------------+----------
       pavle |     pm33

(1 rows)
```

```
$ ccm node6 cqlsh -e "USE ass2; CONSISTENCY local_quorum; SELECT driver_name, password FROM
driver WHERE driver_name = 'pavle';"
Consistency level set to LOCAL_QUORUM.

 driver_name | password
-------------+----------
       pavle |     pm33

(1 rows)
```

Our all nodes are live. We have at least 2 nodes from 3 in each datacenter. Quorum means that Cassandra returns the record after a quorum of replicas has responded from any data center. Based on the documentation `each_quorum` is not supported for reads, however our query worked and it will be clear effect when we stop nodes in the next task. In case of `local_quorum` Cassandra returns the record after a quorum of replicas in the current data center. We can avoid latency of inter-data center communication.

II. (4 marks)

Use `ccm` to stop `node_e` and `node_f`. Connect to `ass2` keyspace.

```
$ ccm node7 stop
$ ccm node8 stop
$ ccm status

Cluster: 'multi_dc'
-------------------
node9: UP
```

```
node8: DOWN
node1: UP
node3: UP
node2: UP
node5: UP
node4: UP
node7: DOWN
node6: UP
```

Run the statement `select driver_name, password from driver where driver_name = 'pavle';` under consistency levels: `quorum`, `each_quorum`, and `local_quorum`. Run the select statement under consistency level `local_quorum` once for `dc1` being local, and once for `dc2` being local. In your answer to the question, show results of your experiments and describe briefly what you have learned.

```
$ ccm node1 cqlsh -e "USE ass2; CONSISTENCY quorum; SELECT driver_name, password FROM driver
WHERE driver_name = 'pavle';"
Consistency level set to QUORUM.

 driver_name | password
-------------+----------
       pavle |     pm33

(1 rows)

$ ccm node6 cqlsh -e "USE ass2; CONSISTENCY quorum; SELECT driver_name, password FROM driver
WHERE driver_name = 'pavle';"
Consistency level set to QUORUM.

 driver_name | password
-------------+----------
       pavle |     pm33

(1 rows)
```

We got proper respond, because at least 2 nodes live on one of the cluster. Doesn't matter to which cluster the client connects.

```
$ ccm node1 cqlsh -e "USE ass2; CONSISTENCY each_quorum; SELECT driver_name, password FROM driver
WHERE driver_name = 'pavle';"
Consistency level set to EACH_QUORUM.
<stdin>:1:NoHostAvailable:

$ ccm node6 cqlsh -e "USE ass2; CONSISTENCY each_quorum; SELECT driver_name, password FROM driver
WHERE driver_name = 'pavle';"
Consistency level set to EACH_QUORUM.
<stdin>:1:NoHostAvailable:
```

Our request will fail, because `each_quorum` consistency expect that at least 2 nodes are active in each data center. It forces strong consistency. We should use this level in multiple datacenter clusters to strictly maintain consistency at the same level in each datacenter and if we want a read to fail when a datacenter is down and the QUORUM cannot be reached on that datacenter. Exactly this happened with us in this case.

```
$ ccm node1 cqlsh -e "USE ass2; CONSISTENCY local_quorum; SELECT driver_name, password FROM driver
WHERE driver_name = 'pavle';"
Consistency level set to LOCAL_QUORUM.

 driver_name | password
-------------+----------
       pavle |     pm33

(1 rows)

$ ccm node6 cqlsh -e "USE ass2; CONSISTENCY local_quorum; SELECT driver_name, password FROM driver
WHERE driver_name = 'pavle';"
Consistency level set to LOCAL_QUORUM.
<stdin>:1:NoHostAvailable:
```

We can see, that `local_quorum` expects that we have at least 2 active nodes with replica from the connected datacenter. In the first case, when we connected to `node1` we had enough nodes active in `dc1`. In the second case, when we connected `node6` which is in `dc2`, there was not more live replica, so the quorum is not satisfied.

Question 15

(10 marks)

You are asked to find those nodes of the `multi_dc` Cassandra cluster that store replicas of the `time_table` table row

```
line_name         | service_no | time | distance | latitude | longitude | stop
------------------+------------+------+----------+----------+-----------+------------
Hutt Valley Line  | 2          | 1045 | 34.3     | -41.2865 | 174.7762  | Wellington
```

Very soon you realized that all `ccm` and `nodetool` commands, except `ccm nodei cqlsh`, do not work. So, you are unable to use: `ccm stop`, `ccm status`, `ccm start`, `ccm nodei ring` and so on, including the command `ccm nodei nodetool getendpoints ass2 time_table <key>`.

Despite that, you have devised a procedure to find the nodes requested. In your answer, describe the procedure and show how you have applied it.

Hint: Luckily, you have saved the output of the ccm nodei ring command and cqlsh prompt is still working.

We can use the primary key for finding the default node. Using the token function with primary keys of `time_table` we can list the token number of each row. The token number determines the default node of a record. Because our replication level is 3, the default node and the following 2 nodes will store our record.

```
$ ccm node1 cqlsh -e "USE ass2; SELECT line_name, service_no, time, token(line_name, service_no)
as t FROM time_table"

line_name        | service_no | time | t
------------------+------------+------+----------------------
          Melling |          3 |  807 | -7474942320664480980
          Melling |          3 |  801 | -7474942320664480980
          Melling |          3 |  754 | -7474942320664480980
          Melling |          3 |  741 | -7474942320664480980
 Hutt Valley Line |          1 |  650 | -6012480106752428297
 Hutt Valley Line |          1 |  642 | -6012480106752428297
 Hutt Valley Line |          1 |  634 | -6012480106752428297
 Hutt Valley Line |          1 |  629 | -6012480106752428297
 Hutt Valley Line |          1 |  625 | -6012480106752428297
 Hutt Valley Line |          1 |  622 | -6012480106752428297
 Hutt Valley Line |          1 |  617 | -6012480106752428297
 Hutt Valley Line |          1 |  605 | -6012480106752428297
         Waikanae |          5 | 1139 | -5905794062536720418
         Waikanae |          5 | 1118 | -5905794062536720418
         Waikanae |          5 | 1059 | -5905794062536720418
         Waikanae |          5 | 1042 | -5905794062536720418
         Waikanae |          5 | 1025 | -5905794062536720418
 Hutt Valley Line |         11 | 2025 | -2183064056535108044
 Hutt Valley Line |         11 | 2019 | -2183064056535108044
 Hutt Valley Line |         11 | 2010 | -2183064056535108044
 Hutt Valley Line |         11 | 2001 | -2183064056535108044
 Hutt Valley Line |         11 | 1955 | -2183064056535108044
 Hutt Valley Line |         11 | 1952 | -2183064056535108044
 Hutt Valley Line |         11 | 1947 | -2183064056535108044
 Hutt Valley Line |         11 | 1935 | -2183064056535108044
 Hutt Valley Line |          2 | 1045 |  2322329569350831795
 Hutt Valley Line |          2 | 1033 |  2322329569350831795
 Hutt Valley Line |          2 | 1025 |  2322329569350831795
 Hutt Valley Line |          2 | 1015 |  2322329569350831795
 Hutt Valley Line |          2 | 1000 |  2322329569350831795

(30 rows)
```

With filter for our specific data.

```
$ ccm node1 cqlsh -e "USE ass2; SELECT line_name, service_no, time, token(line_name, service_no)
as t FROM time_table WHERE line_name='Hutt Valley Line' AND service_no=2 AND time=1045"
```

```
 line_name        | service_no | time | t
------------------+------------+------+---------------------
 Hutt Valley Line |          2 | 1045 | 2322329569350831795

(1 rows)
```

Because we still have our node ring details, we can find the first node where this token belongs.

```
dc1:
127.0.0.4  r1          Up     Normal  98.97 KiB       20.00%              1844674407370955161
127.0.0.5  r1          Up     Normal  98.96 KiB       20.00%              5534023222112865484

dc2:
127.0.0.8  r1          Up     Normal  98.94 KiB       25.00%              100
127.0.0.9  r1          Up     Normal  98.97 KiB       25.00%              4611686018427388004
```

As we can see in node ring list our "Hutt Valley Line" token is smaller then the threshold of `node5` of `dc1`. So our record default node is `node5`. Replicas are the following in the ring: `node1`, `node2`
In `dc2` token number is smaller than the threshold of `node9`, the primary node will be `node9` and replicas will be stored on `node6` and `node7`, because they are the following nodes in the ring of `dc2`.

Of course, we have to test our solution, so when we get back our tools, we can test above numbers with a "brute force" way, as we did in Question 6. Stop nodes, use `consistency one` to determine which node responds and which not.

```
$ ccm stop
$ ccm node9 start
$ ccm status
Cluster: 'multi_dc'
-------------------
node9: UP
node8: DOWN
node1: DOWN
node3: DOWN
node2: DOWN
node5: DOWN
node4: DOWN
node7: DOWN
node6: DOWN
$ ccm node9 cqlsh -e "CONSISTENCY LOCAL_ONE; SELECT * FROM ass2.time_table WHERE line_name='Hutt
Valley Line' AND service_no=2 AND time=1045 ;"
  Consistency level set to LOCAL_ONE.

   line_name        | service_no | time | distance | latitude | longitude | stop
  ------------------+------------+------+----------+----------+-----------+------------
   Hutt Valley Line |          2 | 1045 |     34.3 | -41.2865 |  174.7762 | Wellington

  (1 rows)
```

It shows, that `node9` stores our requested record.
Repeating the above steps for other nodes, I got respond from `node6` and `node7`, but not from `node8`.
And testing `dc1`, I got respond from `node1`, `node2` and `node5`, but not from `node3` and `node4`.

So our token solution was right. (Automated bash script is saved in `q15-brute-force.sh`)