INFSCI 2750 - Mini Project 03

Yichi Zhang (yiz141@pitt.edu)
Quan Zhou (quz3@pitt.edu)

Part 1: Setting up Cassandra

Setting up Cassandra on a cluster with Debian package installation is relatively easy compared to manually setting up Hadoop or Spark. Here is how we did it.

```
# ssh on to master(159.89.43.89) and slave(159.89.43.152) and run the following commands:
echo "deb http://www.apache.org/dist/cassandra/debian 311x main" \
 | sudo tee -a /etc/apt/sources.list.d/cassandra.sources.list
curl https://www.apache.org/dist/cassandra/KEYS | sudo apt-key add -
sudo apt-get update
sudo apt-get install cassandra
sudo service cassandra stop
```

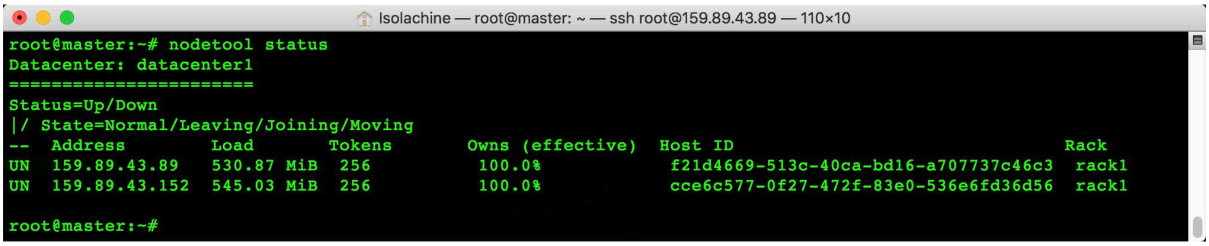
Modify the configuration file `cassandra.yaml` on each node:

```
vim /etc/cassandra/cassandra.yaml
# on both nodes, set
 - seeds: "master,slave"
 # setting the read timeout to a larger number to make sure UDF wouldn't timeout
 read_request_timeout_in_ms: 600000
# on master node, set
 listen_address: master
 rpc_address: master
# on slave node, set
 listen_address: slave
 rpc_address: slave
```

With all the preparation steps done, we can start the Cassandra cluster by running the following command on each node:

```
cassandra -Rf
```

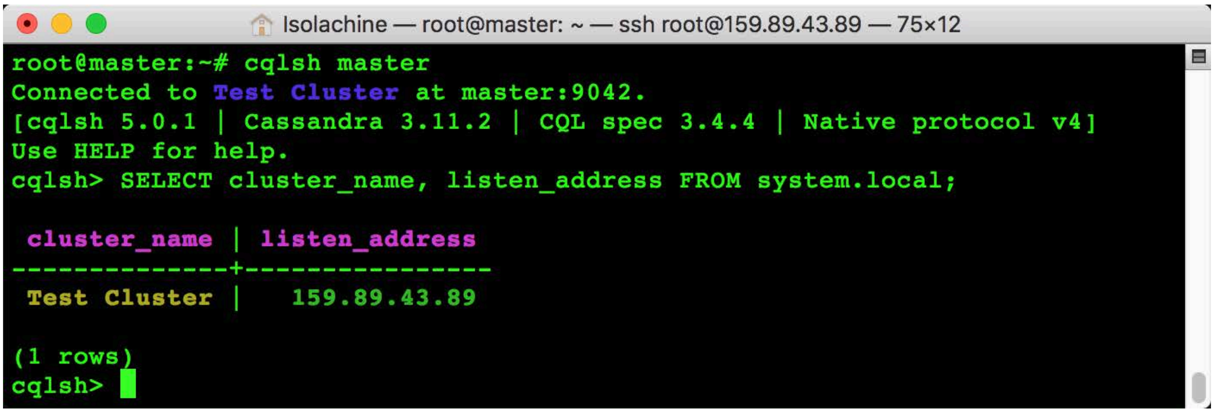
On a new ssh session, run `nodetool status` to check the status of the Cassandra cluster.



> Screenshot of the cluster up and running

To start a Cassandra CQL Shell on the cluster, simply run the following command. Note we specified the request timeout (in seconds) for the shell to match the timeout we set in the configuration file.

```
cqlsh master --request-timeout=600
```



> Screenshot of CQL Shell and result of a test CQL

The project's source code was written in JAVA and used Maven as dependency management and build tool. The JAVA code and `pom.xml` file for the project is in the `source_code` folder. The `mini-project-03-1.0.0-jar-with-dependencies.jar` file was built locally with maven to include all the source code provided and was uploaded to the VM server for running. Note here we need to build the jar file with dependencies in order to successfully run it.

```
mvn package
scp target/mini-project-03-1.0.0-jar-with-dependencies.jar root@159.89.43.89:~/
```

Other than the JAVA code and jar file, we also provided all the CQL commands used in this project in `source_code/cql.txt`

Part 2 : Import Data into Cassandra

The `ImportData.java` file is the source code for importing the `access_log` file into Cassandra. The import process can be launched by:

```
java -cp mini-project-03-1.0.0-jar-with-dependencies.jar log.ImportData
```

The program first run setup CQL for the `project_03` keyspace synchronously:

```
# drop existing data
DROP TABLE IF EXISTS project_03.log;
DROP TABLE IF EXISTS project_03.ip;
DROP TABLE IF EXISTS project_03.path;
DROP KEYSPACE IF EXISTS project_03;

# create keyspace
CREATE KEYSPACE project_03
 WITH replication = {
  'class': 'SimpleStrategy',
  'replication_factor' : 2};

# create log table and index on columns ip and path
CREATE TABLE project_03.log (
 id int,
 ip text,
 identity text,
 username text,
 time text,
 method text,
 path text,
 protocol text,
 status int,
 size int,
 PRIMARY KEY ((id), ip, path));
CREATE INDEX ip_index ON project_03.log (ip);
CREATE INDEX path_index ON project_03.log (path);

# create counter table for ip and path
CREATE TABLE project_03.ip (ip text PRIMARY KEY, count counter);
CREATE TABLE project_03.path (path text PRIMARY KEY, count counter);

# create UDF for retrieving max group count on ip and path columns directly from log table
CREATE FUNCTION project_03.state_group_and_count(state map<text, int>, type text)
 CALLED ON NULL INPUT
 RETURNS map<text, int>
 LANGUAGE java
 AS $$
```

```
  Integer count = (Integer) state.get(type);
  if (count == null)
   count = 1;
  else
   count++;
  state.put(type, count);
  return state;
 $$;

CREATE FUNCTION project_03.ccmapmax(input map<text, int>)
    RETURNS NULL ON NULL INPUT
    RETURNS map<text, int>
    LANGUAGE java
    AS $$
     Integer max = Integer.MIN_VALUE;
     String data="";
     for (String k : input.keySet()) {
         Integer tmp = input.get(k);
         if (tmp > max) { max = tmp; data = k; }
     }
     Map<String,Integer> mm = new HashMap<String,Integer>();
     mm.put(data,max);
     return mm;
 $$;

CREATE OR REPLACE AGGREGATE group_and_count_q34(text)
    SFUNC state_group_and_count
    STYPE map<text, int>
    FINALFUNC ccmapmax
    INITCOND {};
```

After the setup is done, we will read the log file row by row and insert it into the database. Here, for each row of the file, we insert the the raw data pre-processed by regular expression into the `log` table.

```
INSERT INTO project_03.log
 (id, ip, identity, username, time, method, path, protocol, status, size)
 VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
```

For the accessing IP address and the resource path being accessed, we use the `UPDATE` CQL to increment the `count` column by 1 in the corresponding tables, namely `ip` and `path`. This is feasible by using the `UPDATE` CQL alone since the `count` columns in the tables have the data type of counter. Upon invocation of a `UPDATE` CQL on a non-existing key, Cassandra automatically creates a row with the key and set the counter to 0. By setting `count = count + 1`, the value of the `count` column will be 1 for the key's appearance.

```
UPDATE project_03.ip SET count = count + 1 WHERE ip = ?
UPDATE project_03.path SET count = count + 1 WHERE path = ?
```

We used asynchronous execution on inserts. The JAVA class `Semaphore` was used to limit the number of existing asynchronous requests. No more than 256 requests can exist in the request pool, matching the configuration of the Cassandra database.

Isolachine — root@master: ~ — ssh root@159.89.43.89 — 100×12

```
root@master:~# java -cp mini-project-03-1.0.0-jar-with-dependencies.jar log.ImportData
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
inserted 20000
inserted 40000
inserted 60000
inserted 80000
inserted 100000
inserted 120000
inserted 140000
inserted 160000
```

Isolachine — root@master: ~ — ssh root@159.89.43.89 — 100×12

```
inserted 4280000
inserted 4300000
inserted 4320000
inserted 4340000
inserted 4360000
inserted 4380000
inserted 4400000
inserted 4420000
inserted 4440000
inserted 4460000
inserted 4477813
Total running time: 1433 seconds
```

The two figures above shows the process of inserting the log file. A total of `4477813 * 3 = 13433439` `INSERT`s and `UPDATE`s was performed in 1433 seconds, resulting in an average of about `9400 iops`. We can view the resulting tables in CQLSH. Results are shown in figures below.

Isolachine — root@master: ~ — ssh root@159.89.43.89 — 176×17
~ — root@master: ~ — ssh root@159.89.43.89 | ~/Desktop/2750/mini_projects/2750_project_03 — -bash

```
cqlsh:project_03> SELECT * FROM log LIMIT 10;

 id      | ip             | path                                                        | identity | method | protocol | size    | status | time                       | username
---------+----------------+-------------------------------------------------------------+----------+--------+----------+---------+--------+----------------------------+----------
 1792034 | 10.167.188.164 |              /images/filmpics/0000/0975/ShinjukuDVD2D.jpg |     null |    GET | HTTP/1.1 | 3107362 |    200 | 07/Jan/2011:09:49:37 -0800 |     null
 3607449 | 10.198.238.249 |       /images/newspics/0000/0373/Atrociousweb_thumb.jpg |     null |    GET | HTTP/1.1 |    null |    304 | 24/Aug/2011:02:25:04 -0700 |     null
  302602 |  10.163.155.84 |                                     /download.php?id=90 |     null |    GET | HTTP/1.1 | 1089536 |    200 | 08/Mar/2010:18:51:09 -0800 |     null
 3819940 | 10.142.203.173 |                                  /assets/css/combined.css |     null |    GET | HTTP/1.0 |    6112 |    200 | 16/Sep/2011:07:42:45 -0700 |     null
 2301876 |  10.205.174.82 |                                                /index.php |     null |    GET | HTTP/1.1 |   18931 |    200 | 12/Mar/2011:07:12:14 -0800 |     null
  531141 |     10.219.4.6 |         /images/filmmediablock/290/HelgiBj%C3%AErns.jpg |     null |    GET | HTTP/1.1 |  502868 |    200 | 05/May/2010:08:04:43 -0700 |     null
 3472067 | 10.115.218.237 |                     /assets/js/javascript_combined.js |     null |    GET | HTTP/1.1 |   20404 |    200 | 11/Aug/2011:15:17:44 -0700 |     null
 2119753 |   10.122.217.3 |          /images/filmpics/0000/4291/Monsters6_thumb.jpg |     null |    GET | HTTP/1.1 |   26737 |    200 | 21/Feb/2011:08:19:32 -0800 |     null
 1416569 |   10.82.64.235 | /images/filmpics/0000/2563/deadcert_20091114_0151crop.jpg |     null |    GET | HTTP/1.1 |  105006 |    200 | 24/Oct/2010:19:30:32 -0700 |     null
 1817764 | 10.103.214.246 |                                  /assets/css/combined.css |     null |    GET | HTTP/1.1 |    6112 |    200 | 12/Jan/2011:11:01:43 -0800 |     null

(10 rows)
cqlsh:project_03>
```

Isolachine — root@master: ~ — ssh root@159.89.43.89 — 176×17
~ — root@master: ~ — ssh root@159.89.43.89 | ~/Desktop/2750/mini_projects/2750_project_03 — -bash

```
cqlsh:project_03> SELECT * FROM ip LIMIT 10;

 ip             | count
----------------+-------
 10.226.129.213 |     2
  10.207.147.18 |    13
  10.217.21.189 |    14
 10.142.189.149 |     1
 10.126.208.138 |     1
  10.232.73.246 |     1
   10.68.57.243 |    24
  10.140.232.61 |     1
  10.140.203.33 |     1
  10.10.191.185 |     1

(10 rows)
cqlsh:project_03>
```

Isolachine — root@master: ~ — ssh root@159.89.43.89 — 176×17
~ — root@master: ~ — ssh root@159.89.43.89 | ~/Desktop/2750/mini_projects/2750_project_03 — -bash

```
cqlsh:project_03> SELECT * FROM path LIMIT 10;

 path                                                                    | count
-------------------------------------------------------------------------+-------
 /database/fullDetails.php?height=600&modal=true&id=163&random=1306336880267 |     1
                                   /downloadSingle.php?id=2085&fid=345 |    48
                 /SH/shanghai/360_bid/3_etid/28_did/15_ps/1_stid/ |     1
         /images/filmpics/0000/2155/SBX481_InvisibleTarget_DVD_lge.jpg |    71
 /database/fullDetails.php?height=600&modal=true&id=134&random=1314117502102 |     1
                                  /release-schedule/?p=28&l=&rpp=10 |     1
                                       /assets/img/about-us-logo.png |  3157
                  /displaytitle.php?id=546%27%20aND%20%278%27%3D%278 |     1
               /images/filmmediablock/295/TaiChiMaster_2DSleeve.jpg |    76
                 /2010/02/dead-wizard-always-wins/?replytocom=113343 |     2

(10 rows)
cqlsh:project_03>
```

Part 3: Operate Data in Cassandra

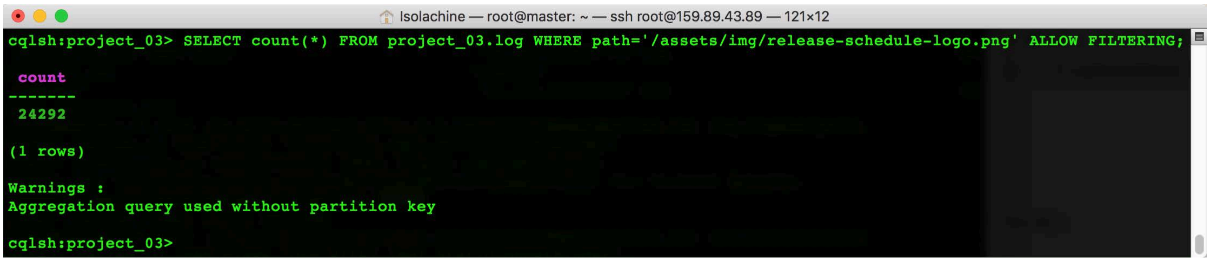
Problem 1

The `LogAnalysis1.java` file is the source code for problem 1.
The program can be launched by the following command at any where, either on the cluster or a local machine.

```
java -cp mini-project-03-1.0.0-jar-with-dependencies.jar log.LogAnalysis1
```

We can also simply get the result by running the CQL

```
SELECT count(*)
 FROM project_03.log
 WHERE path='/assets/img/release-schedule-logo.png'
 ALLOW FILTERING;
```



As the screenshots show, `/assets/img/release-schedule-logo.png` was accessed 24292 times.

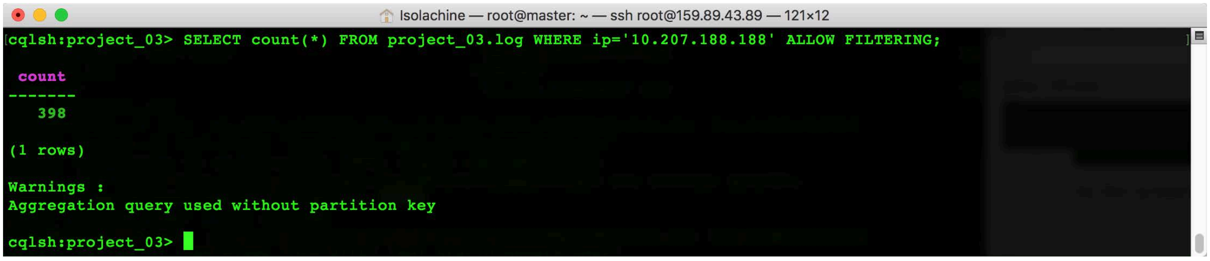
Problem 2

The `LogAnalysis2.java` file is the source code for problem 2.
The program can be launched by the following command at any where, either on the cluster or a local machine.

```
java -cp mini-project-03-1.0.0-jar-with-dependencies.jar log.LogAnalysis2
```

We can also simply get the result by running the CQL

```
SELECT count(*)
 FROM project_03.log
 WHERE ip='10.207.188.188'
 ALLOW FILTERING;
```



As the screenshots show, the IP address `10.207.188.188` accessed the website 398 times.

Problem 3

The `LogAnalysis3.java` file is the source code for problem 3.
The program can be launched by the following command at any where, either on the cluster or a local machine.

```
java -cp mini-project-03-1.0.0-jar-with-dependencies.jar log.LogAnalysis3
```

The JAVA program actually retrieves the whole `path` table and find the max count row internally. We can also get the same result from running the two following CQL in CQLSH.

```
cqlsh:project_03> SELECT max(count) FROM path;

 system.max(count)
-------------------
            117348

(1 rows)

Warnings :
Aggregation query used without partition key

cqlsh:project_03> SELECT * FROM path WHERE count = 117348 ALLOW FILTERING ;

 path                       | count
----------------------------+--------
 /assets/css/combined.css   | 117348

(1 rows)
cqlsh:project_03>
```

Since Cassandra doesn't support subqueries in the latest version, we have to do it with two separate queries instead of a single nested query.

It's also possible to get the result from the `log` table with the UDF we defined earlier. Due to the fact the system has to query and aggregate over the big table of 4.7 million rows, this query may take a few minutes to run.

```
cqlsh:project_03> SELECT group_and_count_q34(path) FROM log;

 project_03.group_and_count_q34(path)
--------------------------------------
 {'/assets/css/combined.css': 117348}

(1 rows)

Warnings :
Aggregation query used without partition key
```

As the screenshots show, `/assets/css/combined.css` was the most accessed resource, with 117348 times.

Problem 4

The `LogAnalysis4.java` file is the source code for problem 4.
The program can be launched by the following command at any where, either on the cluster or a local machine.

```
java -cp mini-project-03-1.0.0-jar-with-dependencies.jar log.LogAnalysis4
```

```
root@master:~# java -cp mini-project-03-1.0.0-jar-with-dependencies.jar log.LogAnalysis4
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
10.216.113.172 158614
Total running time: 6.978 seconds
root@master:~#
```

The JAVA program actually retrieves the whole `ip` table and find the max count row internally. We can also get the same result from running the two following CQL in CQLSH.

```
cqlsh:project_03> SELECT max(count) FROM ip;

 system.max(count)
-------------------
            158614

(1 rows)

Warnings :
Aggregation query used without partition key

cqlsh:project_03> SELECT * FROM ip WHERE count = 158614 ALLOW FILTERING ;

 ip             | count
----------------+--------
 10.216.113.172 | 158614

(1 rows)
cqlsh:project_03>
```

Since Cassandra doesn't support subqueries in the latest version, we have to do it with two separate queries instead of a single nested query.

It's also possible to get the result from the `log` table with the UDF we defined earlier. Due to the fact the system has to query and aggregate over the big table of 4.7 million rows, this query may take a few minutes to run.

```
cqlsh:project_03> SELECT group_and_count_q34(ip) FROM log;

 project_03.group_and_count_q34(ip)
------------------------------------
         {'10.216.113.172': 158614}

(1 rows)

Warnings :
Aggregation query used without partition key

cqlsh:project_03>
```

As the screenshots show, `10.216.113.172` accessed the website the most, with 158614 times.