CASSANDRA OPTIONAL BIG DATA PROJECT REPORT

CMPUT 391 LEC H1

Prepared by Steven Myers, Michael Kmicik, & Divyank Katira

NoSQL System Overview

NOSQL SYSTEM

The Cassandra NoSQL big data system was chosen for use in our implementation of the project specifications. Version 2.1.3 was used instead of version 2.0.5, which was the specifications used during the lab lecture.

JUSTIFICATION

The Cassandra NoSQL big data system was chosen over the other options due to the following reasons.

1. **Support and Documentation:** Early review of the available options suggested that Cassandra had sufficient documentation and support readily available to assist in learning the system and to troubleshoot any problems and issues that may arise.
2. **Familiarity:** Cassandra 2.0.5 was demonstrated by Bing Xu during the second lab session, and as such was the system that our group was most familiar with.
3. **Stability:** After some research, it was decided that Cassandra had a history of being a stable and reliable NoSQL system.
4. **Industry Applicability:** Cassandra is highly ranked in popularity[1] and utilized by many large and successful organizations in their operations. Notable organizations include: AOL, CERN, Cisco, EBay, FedEx, IBM, Microsoft, NASA, Netflix, Reddit, and Twitter[2].
5. **CQL:** Cassandra supports a query language that is similar to SQL.

ACCESSING CODE BASE

The code base and related resources are available on GitHub:
https://github.com/SMD-Cassandra-391/project/tree/master

To download the git project on your local machine one should use the following command in terminal

```
git clone https://github.com/SMD-Cassandra-391/project.git
```

[1] http://db-engines.com/en/ranking (April 4, 2015).
[2] http://planetcassandra.org/companies/ (April 4, 2015).

Overview of Cassandra Data Model

Partition Key

Cassandra utilizes the idea of a partition key in place of the SQL concept of primary key. The value of the partition key is used to generate a token via the Murmur3 hashing algorithm[3]. This newly created hash is used to index the column values in a Sorted String Table.

Clustering Key

There are times where one will want to keep multiple sets of data for a single partition key. To allow for this Cassandra uses a pattern called row partitioning. This allows for one to partition a row into multiple sets of data.

Compound Key

Combining these two concepts leads us to the Compound key. A compound key allows us to create a data structure in which each data partition is clustered in a manner of our choosing. Patrick McFadin has written an excellent tutorial on the subject matter[4].

Overview of Data Generation and Loading

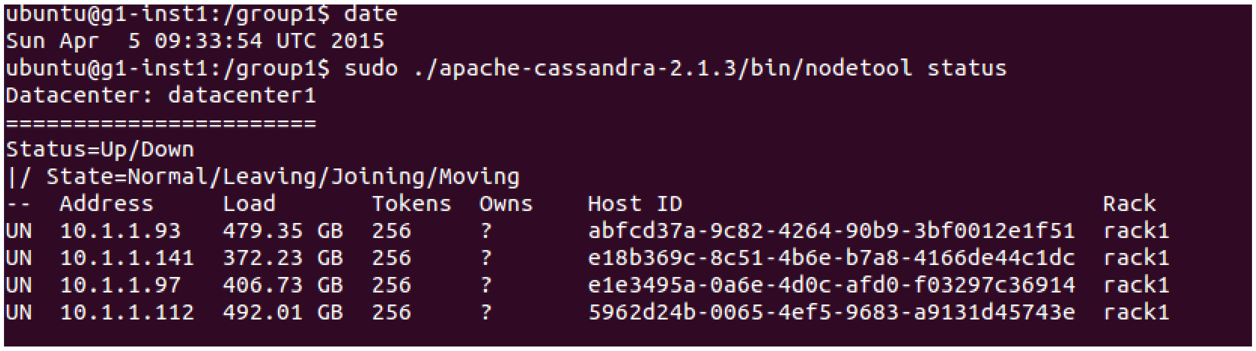
Since Cassandra data is stored in Sorted String Tables (a set of key-value pairs[5]) it only makes sense to utilize the CQLSSTableWriter Class in Java to generate the data. The data generation program utilizes four threads -- four threads were used because there are four processors on the strongest instance provided -- in order to write multiple sets of tables concurrently. However, the CQLSSTableWriter class cannot be instantiated with multiple schemas, so the data generation program writes four sets of the same table per iteration. Data loading is done in place using the JMX StorageService > bulkload() call. This allows for uploading the newly written SSTables into the appropriate directory in the local node. This technique is well outlined by Patrick Callaghan[6].

[3] http://spyced.blogspot.ca/2009/01/all-you-ever-wanted-to-know-about.html (April 4, 2015).
[4] http://planetcassandra.org/blog/getting-started-with-time-series-data-modeling/ (April 4, 2015).
[5] http://www.quora.com/What-is-an-SSTable-in-Googles-internal-infrastructure (April 4, 2015).
[6] https://github.com/SMD-Cassandra-391/datastax-bulkloader-writer-example (April 4, 2015).

EXPERIMENTAL SETUP

The experiment was run on four Ubuntu instances hosted by Cybera[7]. Instance one had 32GB of RAM and 4 CPUs while the other three instances had 4GB RAM and 2 CPUs. Each node was associated with a 2TB volume. These instances were clustered into one data center in the manner described in the User Documentation.

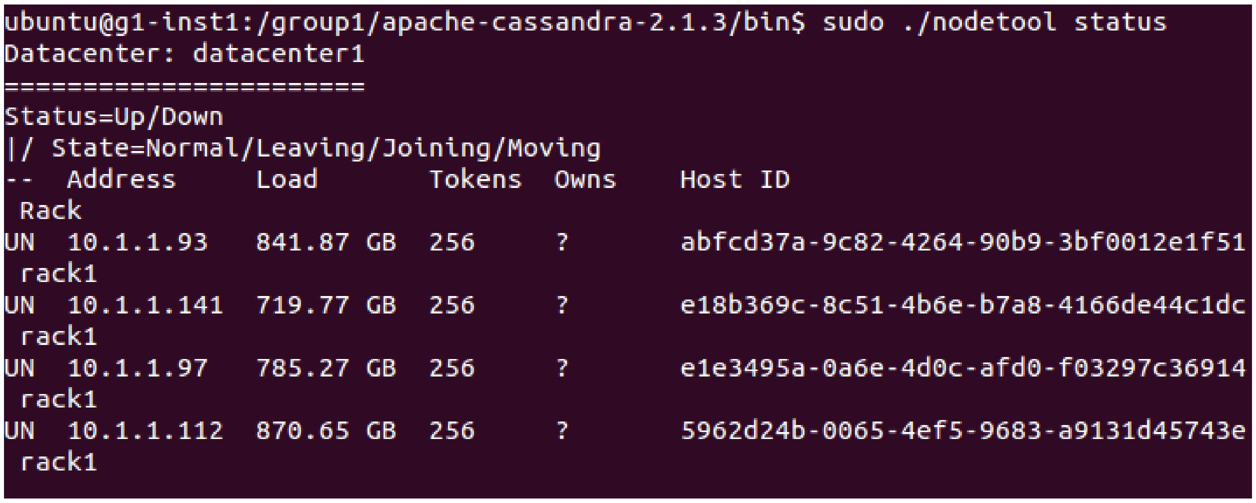
The first run of the experiment was executed with the following load:



```
ubuntu@g1-inst1:/group1$ date
Sun Apr  5 09:33:54 UTC 2015
ubuntu@g1-inst1:/group1$ sudo ./apache-cassandra-2.1.3/bin/nodetool status
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address     Load       Tokens  Owns    Host ID                               Rack
UN  10.1.1.93   479.35 GB  256     ?       abfcd37a-9c82-4264-90b9-3bf0012e1f51  rack1
UN  10.1.1.141  372.23 GB  256     ?       e18b369c-8c51-4b6e-b7a8-4166de44c1dc  rack1
UN  10.1.1.97   406.73 GB  256     ?       e1e3495a-0a6e-4d0c-afd0-f03297c36914  rack1
UN  10.1.1.112  492.01 GB  256     ?       5962d24b-0065-4ef5-9683-a9131d45743e  rack1
```

The cluster contains a total of 1.75 TB of data.

The second run of the experiment was executed with the following load:



```
ubuntu@g1-inst1:/group1/apache-cassandra-2.1.3/bin$ sudo ./nodetool status
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address     Load       Tokens  Owns    Host ID
 Rack
UN  10.1.1.93   841.87 GB  256     ?       abfcd37a-9c82-4264-90b9-3bf0012e1f51
 rack1
UN  10.1.1.141  719.77 GB  256     ?       e18b369c-8c51-4b6e-b7a8-4166de44c1dc
 rack1
UN  10.1.1.97   785.27 GB  256     ?       e1e3495a-0a6e-4d0c-afd0-f03297c36914
 rack1
UN  10.1.1.112  870.65 GB  256     ?       5962d24b-0065-4ef5-9683-a9131d45743e
 rack1
```

The cluster contains a total of 3.22TB of data of which 1.47TB is the modified partitions.

DATA SET SPECIFICATIONS AND SCHEMAS

[7] http://www.cybera.ca/ (April 4, 2015).

A sample table that was given in the assignment specifications (setup2.sql) was partitioned into three separate tables -- t1, t2 and t3. Furthermore, two more tables (t4 and t5) were created in order to write 'group by' and 'order by' queries. These tables are found in the CQL folder. Note that aside from the keyspace the demo and project tables are identical. See Appendix A for the Create Table statements for the project keyspace.

Query Evaluation

In order to effectively query Cassandra for tables with large numbers of rows one must specify the partition key. Otherwise all rows will be scanned and in the case of a table that contains billions of rows a timeout will undoubtedly occur. An effective giveaway that your query will not scale is that it contains the '**ALLOW FILTERING**' command. This command essentially tells Cassandra to scan every row in a table.

Query 1:

```
SELECT      huntpilotdn
FROM        project.t1
WHERE       num = 36681769924989
AND         (currentroutingreason, orignodeid, origspan, origvideocap_bandwidth,
            datetimeorigination)
            > (1,1,1,1,1)
AND         (currentroutingreason, orignodeid, origspan, origvideocap_bandwidth,
            datetimeorigination)
            < (5000,5000,5000,5000,5000);
```

Time elapsed run 1: 76ms

Time elapsed run 2: 1552ms

Query 2:

```
SELECT      destcause_location, destipaddr
FROM        project.t2
WHERE       datetimeorigination = 110
AND         num >= 10030531462782
AND         num <= 10043787914378;
```

Time elapsed run 1: 1040ms

Time elapsed run 2: 938ms

Query 3:

```
SELECT      origdevicename, duration
FROM        project.t3
WHERE       num = 17702306473948
AND         destdevicename IN ('a','f', 'g', 'h');
```

Time elapsed run 1: 664ms

Time elapsed run 2: 183ms

Query 4:

```
SELECT      count, duration
FROM        project.t4
WHERE       duration IN (10,20,30,40,50,60,70,80,90,100)
ORDER BY    count DESC
LIMIT 10;
```

Time elapsed run 1: 14ms

Time elapsed run 2: 7ms

Query 5:

```
SELECT      callingnumber, callcount
FROM        project.t5
WHERE       callingNumber IN ('6214789','7350819', '1255452', '9151967', '9657707',
            '9998888')
ORDER BY    callcount ASC
LIMIT 6;
```

Time elapsed run 1: 53ms
Time elapsed run 2: 22ms

Partition Analysis

NoSQL systems are generally used for OLTP or web applications where the queries to be performed are known in advance. Keeping this in mind, the dataset is partitioned to conform to the queries that are performed on it. Here is a brief description our partitions:

Table One (t1)

PRIMARY KEY ((num), currentroutingreason, orignodeid, origspan, origvideocap_bandwidth, datetimeorigination)

Table one's partition key is a bigint and as a result the Index SStable for this table is sparse. The partition key is clustered by five int values, so as to allow for the required ten condition query. When the query 1 is executed each set of cluster keys within the num partition is compared against the inequality operators to find a set of columns where all the cluster keys are in the range (0, 5000).

Table Two(t2)

PRIMARY KEY ((datetimeorigination), num)

This query took more time than any other query. While the project specification require two recorded runs, multiple runs were executed, but not recorded. In a total of four runs this query timed out twice. To re-partition this table the num value will become the partition key and datetimeorigination will be the clustering key. It is expected that times similar to query one will result.

modification to t2:

PRIMARY KEY ((num), datetimeorigination)

Table Three(t3)

PRIMARY KEY ((num), destdevicename)

In this case the table is partitioned by a big int value meaning there will be a sparse index table. The accompanying query for this table contains an **"in"** statement containing ten values. An optimization similar to table two can be made here where the clustering is ordered by destdevicename with ascending values.

modification to t3:

PRIMARY KEY ((num), destdevicename)
WITH CLUSTERING ORDER BY (destdevicename ASC);

Table Four(t4)

PRIMARY KEY (duration, count)

Table four is partitioned by duration and clustered by count. The goal of this table is to imitate a group by query. Because the accompanying query compares the duration key (the partition key) against a set of values, there are no optimizations to be made with respect to clustering.

Table Five(t5)

PRIMARY KEY (callingnumber, callcount)

The final table is used by a "group by" query, and because the comparisons made are against the partition key we are again in a situation where an optimization can not be made with respect to clustering.

Query Reevaluation

Query 1:

```
SELECT      huntpilotdn
FROM        project.t1
WHERE       num = 36681769924989
AND         (currentroutingreason, orignodeid, origspan, origvideocap_bandwidth,
            datetimeorigination)
            > (1,1,1,1,1)
AND         (currentroutingreason, orignodeid, origspan, origvideocap_bandwidth,
            datetimeorigination)
            < (5000,5000,5000,5000,5000);
```

Time elapsed run 1: 231ms

Time elapsed run 2: 52ms

Query 2:

```
SELECT      destcause_location, destipaddr
FROM        modified_project.t2
WHERE       num = 17306447014892
AND         datetimeorigination > 1
AND         datetimeorigination < 5000;
```

Time elapsed run 1: 1041ms

Time elapsed run 2: 68ms

Query 3:

```
SELECT      origdevicename, duration
FROM        modified_project.t3
WHERE       num = 17702306473948
AND         destdevicename IN ('a','f', 'g', 'h');
```

Time elapsed run 1: 72ms

Time elapsed run 2: 26ms

Query 4:

```
SELECT      count, duration
FROM        project.t4
WHERE       duration IN (10,20,30,40,50,60,70,80,90,100)
ORDER BY    count DESC
LIMIT 10;
```

Time elapsed run 1: 104ms

Time elapsed run 2: 16ms

Query 5:

```
SELECT      callingnumber, callcount
FROM        project.t5
WHERE       callingNumber IN ('6214789','7350819', '1255452', '9151967', '9657707',
            '9998888')
ORDER BY    callcount ASC
LIMIT 6;
```

Time elapsed run 1: 122ms
Time elapsed run 2: 10ms

Results of Partitioning

As expected the performance of both queries 3 and 2 were improved substantially.

Improvements

Various improvements can be made to the data generation and loading process. Firstly, currently the loading and generating of data is a non-terminating loop, so to load the desired amount of data requires one to carefully observe the load state of the cluster. It would be desirable for the application to query the cluster to determine the load state, and if the user requested amount of data has been generated, to self terminate.

Secondly, it is possible to stream SSTables from multiple nodes, therefore, an avenue to improve total generation and loading time would be to run multiple instances of the database generation program on different nodes. However, it is necessary that one streams all the generated SSTables to the coordinator node. Otherwise concurrency exceptions will occur.

Thirdly, during execution of the data generation application an ExceptionInInitializerError is thrown on the third and fourth iteration. This is due to the Static schema strings being uninitialized when the SSTwriter class is being instantiated. A solution to this bug is to ensure that all schema strings are initialized at application launch.

Applications

The skills learned acquired during the course of this project can be applied to rapidly modelling NoSQL databases. Because of the non-relational nature of NoSQL systems, and the nature of Big-Data, proper table schema is important to ensure that your database application will run smoothly during all stages of data load. By being able to rapidly inject test data into an initial design one is able to evaluate table partitioning and evaluate effectiveness of queries before launching the application. In this manner one is able to encounter and solve problems in schema design during testing and avoid such issues in production.

Appendix A: Original Tables

```
CREATE TABLE IF NOT EXISTS project.t1  (
        num bigint,
        datetimeorigination int,
        callingpartynumber int,
        callingpartyunicodeloginuserid text,
        currentroutingreason int,
        huntpilotdn text,
        huntpilotpartition text,
        lastredirectingroutingreason int,
        origcause_location float,
        origcause_value float,
        origipaddr int,
        origmediacap_maxframesperpacket int,
        origmediacap_payloadcapability int,
        origmediatransportaddress_ip text,
        origmediatransportaddress_port int,
        orignodeid int,
        origprecedencelevel float,
        origroutingreason int,
        origspan int,
        origvideocap_bandwidth int,
        origvideocap_codec float,
        origvideocap_resolution float,
        origvideotransportaddress_ip int,
        PRIMARY KEY ((num), currentroutingreason, orignodeid, origspan,
        origvideocap_bandwidth, datetimeorigination)
);
```

```
CREATE TABLE IF NOT EXISTS project.t2 (
        num bigint,
        datetimeorigination int,
        cdrrecordtype float,
        destcause_location float,
        destcause_value float,
        destipaddr text,
        destlegcallidentifier int,
        destmediacap_maxframesperpacket int,
        destmediacap_payloadcapability int,
        destmediatransportaddress_ip text,
        destmediatransportaddress_port int,
        destnodeid int,
        destprecedencelevel float,
        destspan int,
        finalcalledpartynumber int,
        finalcalledpartyunicodeloginuserid text,
        globalcallid_callid int,
        globalcallid_callmanagerid int,
        incomingprotocolcallref text,
        incomingprotocolid int,
        originalcalledpartynumber text,
        origlegcallidentifier int,
        origrsvpaudiostat float,
        origrsvpvideostat float,
        origvideotransportaddress_port int,
        outgoingprotocolcallref text,
        outgoingprotocolid int,
        PRIMARY KEY ((datetimeorigination), num)
);
```

```
CREATE TABLE IF NOT EXISTS project.t3 (
        num bigint,
        datetimeorigination timestamp,
        datetimeconnect timestamp,
        datetimedisconnect timestamp,
        destcallterminationonbehalfof int,
        destdevicename text,
        destrsvpaudiostat float,
        destrsvpvideostat float,
        destvideocap_bandwidth int,
        destvideocap_bandwidth_channel2 int,
        destvideocap_codec float,
        destvideocap_codec_channel2 float,
        destvideocap_resolution float,
        destvideocap_resolution_channel2 float,
        destvideochannel_role_channel2 int,
        destvideotransportaddress_ip text,
        destvideotransportaddress_ip_channel2 text,
        destvideotransportaddress_port int,
        destvideotransportaddress_port_channel2 int,
        duration int,
        finalcalledpartynumberpartition text,
        lastredirectdn text,
        lastredirectdnpartition text,
        numberpartition text,
        origcalledpartyredirectonbehalfof int,
        origcallterminationonbehalfof int,
        origdevicename text,
        originalcalledpartynumberpartition text,
        pkid text,
        PRIMARY KEY ((num), destdevicename)
);

CREATE TABLE IF NOT EXISTS project.t4 (
   duration int,
   count int,
   PRIMARY KEY (duration, count)
);
```

```
CREATE TABLE IF NOT EXISTS project.t5 (
   callingnumber text,
   callcount int,
   PRIMARY KEY (callingnumber, callcount)
);
```

APPENDIX B: MODIFIED TABLES

```
REATE TABLE IF NOT EXISTS modified_project.t2 (
   num bigint,
        datetimeorigination int,
        cdrrecordtype float,
        destcause_location float,
        destcause_value float,
        destipaddr text,
        destlegcallidentifier int,
        destmediacap_maxframesperpacket int,
        destmediacap_payloadcapability int,
        destmediatransportaddress_ip text,
        destmediatransportaddress_port int,
        destnodeid int,
        destprecedencelevel float,
        destspan int,
        finalcalledpartynumber int,
        finalcalledpartyunicodeloginuserid text,
        globalcallid_callid int,
        globalcallid_callmanagerid int,
        incomingprotocolcallref text,
        incomingprotocolid int,
        originalcalledpartynumber text,
        origlegcallidentifier int,
        origrsvpaudiostat float,
        origrsvpvideostat float,
        origvideotransportaddress_port int,
        outgoingprotocolcallref text,
        outgoingprotocolid int,
        PRIMARY KEY ((num), datetimeorigination)
);
```

```
CREATE TABLE IF NOT EXISTS modified_project.t3 (
   num bigint,
   datetimeorigination timestamp,
   datetimeconnect timestamp,
         datetimedisconnect timestamp,
         destcallterminationonbehalfof int,
         destdevicename text,
         destrsvpaudiostat float,
         destrsvpvideostat float,
         destvideocap_bandwidth int,
         destvideocap_bandwidth_channel2 int,
         destvideocap_codec float,
         destvideocap_codec_channel2 float,
         destvideocap_resolution float,
         destvideocap_resolution_channel2 float,
         destvideochannel_role_channel2 int,
         destvideotransportaddress_ip text,
         destvideotransportaddress_ip_channel2 text,
         destvideotransportaddress_port int,
         destvideotransportaddress_port_channel2 int,
         duration int,
         finalcalledpartynumberpartition text,
         lastredirectdn text,
         lastredirectdnpartition text,
         numberpartition text,
         origcalledpartyredirectonbehalfof int,
         origcallterminationonbehalfof int,
         origdevicename text,
         originalcalledpartynumberpartition text,
         pkid text,
         PRIMARY KEY ((num), destdevicename)
)
WITH CLUSTERING ORDER BY (destdevicename ASC);
```