**Notes Regarding Cassandra**

**Background** – Just a few notes since you know most of this information, this open source database management system has been an Apache production project since 2010 which is an indicator of the immaturity of this system. It has had six major releases since then. The latest stable release is 1.2.4 which is what we have installed over 2 nodes (soon to be three with the provisioning of our Gordon node).

**Installation** – Installation was relatively pain free. Each node gets the same software deployed and started/stopped independently. Cassandra has no notion of master/slave nodes. Each node installation is independent and becomes aware of other nodes on the cluster via a configuration file called cassandra.yaml and a protocol called Gossip. Apache has a relationship with a third party packager called DataStax. This company provides a number of technologies beyond Cassandra like Hadoop, Solr, Mahout, Pig, etc. integrated for easy deployment. I have installed the enterprise version of this software stack on the nodes.

**Deployment** – Cassandra is installed on a token ring network topology. Row IDs are the key part of a key/value pair that makes up a record in Cassandra. The keys are tokenized when loaded and this token determines where on the ring data is stored. Determination about how this will be done has to happen during the deployment phase. The deployment decision will center around concepts in Cassandra known as a partitioner and a tokenizing protocol.

**ACID** – NoSQL databases do not follow all the conventions of atomicity, consistency, isolation and durability, primarily in the area of consistency. Cassandra is referred to as an eventually consistent database. The implication is that data might not be available on all nodes as soon as it is inserted. This would be problematic for an OLTP repository but since this is a more analytical framework it shouldn’t be an issue. The consistency level is tunable however if this ever becomes an issue. Also because of this feature there is no commit or rollback functionality with Cassandra.

**Scalability and Durability** – Cassandra should be infinitely scalable in a linear fashion simply by adding nodes to the cluster. It is also considered durable since if a single node becomes unavailable for any reason the data on that node is still available on the other active nodes. The bigonc-db server went offline this weekend leaving only the bigonc node available. This proved to work exactly as advertised. We should be able to test the scalability claims when we add the Gordon node to our current cluster

**Partitioner** - The partitioner decides how data is stored on the ring. There are basically two types a random partitioner and a Byte Order. The random paritioner is the most efficient for storage on the ring in that it automatically balances data on the ring and can using either an MD5 or Murmur3 type hash to encode the row ID. The Byte Order paritioner places the data on the ring in order based on the row ID value. This provides the ability to query the database and get sorted results and utilize functionality like comparison operators (i.e. greater than, less than, etc.). This arrangement can lead to hotspots (a lopsided distribution of the data) on the cluster.

I have used the recommended random partitioner using a murmur3 hash function. This makes for efficient data management and storage but creates challenges when extracting data from the repository.

Development Shells – There are two different shells that come with the Cassandra tarball. They are the Cassandra-CLI and CQLSH. Further CQLSH has two versions, 2 and 3. All of these shells, I have found, are necessary to do development. Cassandra-CLI show how the data actually looks in a hash representation (figure 1). CQLSH is an attempt to provide an shell interface similar to SQLPlus (Oracle) or PGSQL (PostgreSQL). They work very similar but due to the immaturity of this software the only deliver a fraction of the toolkit that SQL provides. CQLSH -2 has support for some of the original Cassandra design patterns like wide row Column Families (figure 2). This functionality for some reason is not apparent in CQLSH -3 but this version of the shell provides a more table like appearance (figure 3) for big row tables.

***Figure 1 – Cassandra-CLI***

[default@gene\_data] list af\_lookup;

Using default limit of 100

Using default column limit of 100

-------------------

RowKey: 0.89

=> (column=1-113959089-A-ATTT, value=, timestamp=1368213935285000)

=> (column=1-150055739-GAT-G, value=, timestamp=1368213935287000)

=> (column=1-155579029-TATATA-T, value=, timestamp=1368213935288000)

=> (column=1-163434552-TAC-T, value=, timestamp=1368213935290000)

=> (column=1-165498918-C-T, value=, timestamp=1368213935292000)

=> (column=1-195540954-T-TATC, value=, timestamp=1368213935293000)

=> (column=1-197683035-A-AC, value=, timestamp=1368213935294000)

=> (column=1-201870748-ATGTG-A, value=, timestamp=1368213935296000)

=> (column=1-21701168-TACACAC-T, value=, timestamp=1368213935277000)

=> (column=1-225992563-TA-T, value=, timestamp=1368213935298000)

=> (column=1-225992564-A-AC, value=, timestamp=1368213935300000)

=> (column=1-238790794-A-ATAG, value=, timestamp=1368213935301000)

***Figure 2 – CQLSH -2***

[wwest@bigonc dse]$ bin/cqlsh -2

Connected to bigonc-cass at localhost:9160.

[cqlsh 2.3.0 | Cassandra 0.0.0 | CQL spec 2.0.0 | Thrift protocol 19.35.0]

Use HELP for help.

cqlsh> use gene\_data;

cqlsh:gene\_data> select \* from af\_lookup limit 10;

af\_id,0.89 | '\x00\x121-113959089-A-ATTT\x00', | '\x00\x111-150055739-GAT-G\x00', | '\x00\x141-155579029-TATATA-T\x00', | '\x00\x111-163434552-TAC-T\x00', | '\x00\x0f1-165498918-C-T\x00', | '\x00\x121-195540954-T-TATC\x00', | '\x00\x101-197683035-A-AC\x00', | '\x00\x131-201870748-ATGTG-A\x00', | '\x00\x141-21701168-TACACAC-T\x00', | '\x00\x101-225992563-TA-T\x00', | '\x00\x101-225992564-A-AC\x00', | '\x00\x121-238790794-A-ATAG\x00', | '\x00\x111-24287812-GTTA-G\x00', | '\x00\x101-24300902-CAA-C\x00', | '\x00\x0f1-246683624-A-C\x00', | '\x00\x141-248818938-C-CAGGAG\x00', | '\x00\x0f1-56834941-T-TA\x00', | '\x00\x0f1-83634036-A-AG\x00', | '\x00\x1410-116070711-G-GCCAC\x00', | '\x00$10-118203309-GTGTGCATATATATATATATA-G\x00', | '\x00\x1410-118203310-TGTGC-T\x00', | '\x00\x1210-24113577-AAAT-A\x00', | '\x00\x1110-24113578-AAT-A\x00', | '\x00\x1010-24113579-AT-A\x00', | '\x00!10-29042558-TATATATATATATATATAC-T\x00', | '\x00\x1010-32965138-C-CA\x00', | '\x00\x1210-4538085-A-ATATT\x00', | '\x00\x1010-47279917-T-TA\x00', | '\x00\x0e10-524319-G-GC\x00', | '\x00\x1310-53926554-TATAC-T\x00', | '\x00\x0e10-724609-A-AT\x00', | '\x00\x1411-104662021-A-ATACT\x00', | '\x00\x1211-104662023-A-ACT\x00', | '\x00211-106793388-CATATATATATATATATATATATATATATATATAT-C\x00', | '\x00\x1111-116446983-T-TA\x00', | '\x00\x1911-121284861-AAATAATAAT-A\x00', | '\x00\x1411-12980725-A-AGGCAG\x00', | '\x00\x1b11-39809182-GTGTGTATATATA-G\x00', | '\x00\x1511-49563841-TACACAC-T\x00', | '\x00\x1011-91892088-C-CA\x00', | '\x00\x0f11-93102816-G-T\x00', | '\x00\x1211-99515993-C-CAAA\x00', | '\x00\x1212-112378284-T-TAC\x00', | '\x00\x1112-121952855-C-CT\x00', | '\x00\x1012-126706712-G-A\x00', | '\x00\x1012-132044517-T-C\x00', | '\x00\x1012-132133879-G-A\x00', | '\x00\x1212-132134341-CCT-C\x00', | '\x00\x1012-132135288-G-C\x00', | '\x00\x1012-132135296-T-C\x00', | '\x00\x1012-132136592-G-A\x00', | '\x00\x1012-132136918-A-C\x00', | '\x00\x1012-132136919-C-T\x00', | '\x00\x1512-19891705-C-CAAAAAA\x00', | '\x00\x1f12-21065235-TTTTATTTATTTATTTA-T\x00', | '\x00\x1312-21278446-T-TTAAG\x00', | '\x00\x1312-60412832-C-CCTTT\x00', | '\x00\x1b12-95757426-ATGTATATATATG-A\x00', | '\x00\x1b12-95757428-GTATATATATGTG-G\x00', | '\x00\x1212-96302411-GGGA-G\x00', | '\x00\x0e12-9716126-C-T\x00', | '\x00\x0f12-97220768-C-A\x00', | '\x00\x1013-114539823-G-A\x00', | '\x00\x0f13-22597915-A-G\x00', | '\x00\x0f13-29207791-A-G\x00', | '\x00\x0f13-29207858-T-C\x00', | '\x00713-56936842-AATATATATATATATATATATATATATATATATATATATAT-A\x00', | '\x00\x0f13-85151546-G-A\x00', | '\x00\x1513-98369613-TATATAA-T\x00', | '\x00\x1014-106102204-T-C\x00', | '\x00\x0f14-20405514-T-C\x00', | '\x00\x1014-52594962-C-CA\x00', | '\x00\x0f14-70039587-C-T\x00', | '\x00\x1914-77736137-TCCCCCCCCCC-T\x00', | '\x00\x0f14-83900411-A-C\x00', | '\x00\x1014-86772484-AT-A\x00', | '\x00\x0f14-90767229-G-A\x00', | '\x00%15-29147054-ATG-ATGTGTGTGTGTGTGTGTGTG\x00', | '\x00/15-48683934-TATATATATATATATATATATATATATATATAC-T\x00', | '\x00\x1715-58859536-ATGTATATG-A\x00', | '\x00\x0f16-11593152-C-G\x00

***Figure 3 – CQLSH -3***

[wwest@bigonc dse]$ bin/cqlsh -3

Connected to bigonc-cass at localhost:9160.

[cqlsh 2.3.0 | Cassandra 1.2.4 | CQL spec 3.0.0 | Thrift protocol 19.35.0]

Use HELP for help.

cqlsh> use gene\_data;

cqlsh:gene\_data> select \* from af\_lookup limit 10;

af\_id | column1 | column2 | value

-------+----------------------+---------+-------

0.89 | 1-113959089-A-ATTT | null |

0.89 | 1-150055739-GAT-G | null |

0.89 | 1-155579029-TATATA-T | null |

0.89 | 1-163434552-TAC-T | null |

0.89 | 1-165498918-C-T | null |

0.89 | 1-195540954-T-TATC | null |

0.89 | 1-197683035-A-AC | null |

0.89 | 1-201870748-ATGTG-A | null |

0.89 | 1-21701168-TACACAC-T | null |

0.89 | 1-225992563-TA-T | null |

**Drivers and Clients** – There are database drivers for most major programming languages. I am using the drivers for CQL for Python for the ETL work and for Java for the RESTFul web services. There are also a number of clients for both Java and Python. The two main clients are Hector and Astyanax. These clients are needed to perform range querying operations on Cassandra. This is important because the strategy to query data from a NoSQL database is much different than an RDBMS. Cassandra is almost designed to prevent long running queries and especially full table scans which are tolerated in the RDBMS world. These clients abstract some of this strategy away from the programmer to make the job more approachable. The problem with this is that the clients are behind the major and minor releases of the database itself. So all of the new functionality of Cassandra may not be exposed through these clients. Finding this missing functionality is a matter of trial and error.

**Inserting Data** – Getting data into Cassandra is fairly easy and follows those strategies employed by RDBMS’s. There is a copy command that can be used for uploads of not more than a couple of million records. I have used this on the VCF files and it can load files of 10 to 15 million records in between 2 and 3 hours (without using MapReduce). There is also an export/import functionality which reads and/or writes data to/from a JSON formatted file. I have not used this functionality yet.

**Querying Data** – This by far is the most challenging aspect of Cassandra. NoSQL databases no matter how much they make CQL resemble SQL are fundamentally different and those differences emerge mostly in this area. Since there are no joins or full table scans and since CQL is definitely a work in progress, other strategies need to be employed and thus the need for client libraries on top of the CQL drivers.

Using a randomizing partitioner also prevents you from using relational operators like <,>, <=, >= in queries. This forces some of the processing typically done by the database SQL engine into the application layer.

Also since Cassandra tables are basically partitioned by row ID, you have to query these objects in a similar way to partitioned tables in an RDBMS. That is you need to specify the primary key in the filter clause of a CQL query (in the event of a composite primary key, at least the first element of the key must be specified). This leads to a design pattern called Wide Row tables. These are essentially lookup tables or home made indexes. Querying this table allows you to find the appropriate primary keys to do lookups in the Big Data table based on the query you are trying to solve. Primary keys just like in RDBMSs must be unique. Due to the presence of horizontal partitioning this means that the queries will need some way to identify the primary key in the target table prior to querying the data. This design pattern fills that requirement. This is where most of the development time and effort will be focused for the time being until CQL or Cassandra itself develops more robust searching techniques. This is were the bulk of my time is currently being absorbed.

Authentication and Backup Strategy – Although these things have nothing to do with each other I am grouping them together because I have not research them yet.

**Keywords and Definitions**

|  |  |
| --- | --- |
| Keyword | Definition |
| ColumnFamily | Similar to RDBMS table but in a hash format. This term is being deprecated in favor of ‘table’ but it is still a useful descriptor given the differences in the layout on disk. |
| Schema | Similar to schema in RDBMS |
| Validator | Data Type |
| Comparator | Column Names |
| Static vs Dynamic Tables | Cassandra can have tables defined at design time just like an RDBMS. However due to the hash like layout of the table structure columns can be added at anytime in the application layer. Cassandra support both of these types and also hybrids of the two. |
| Partitioner | A design consideration when creating a Cassandra instant on whether to store data on disk in a random or ordered fashion. |
| Row ID | Equivalent to the primary key in an RDBMS. Also acts as a partition key. |

**Conclusions**

Cassandra now referred to in writing as C\* for short is in it’s infancy from a production standpoint. While the application itself is very stable the supporting infrastructure of software (drivers and clients) struggle to keep up with the current release schedule. I also look for much greater development in the area of the CQL language which is even newer than C\*.