NoSQL Database: CASSANDRA

Anonymous

ABSTRACT

This paper entails the working of a NoSQL database named Cassandra. This database by Apache works differently than a Relational database like MySQL. We explain the structure and working of Cassandra, focusing on what makes it better than a Relational database. Cassandra is a more viable option for enhancing efficiency related to data storage, indexing, query processing, and transaction management. Security support features are also demonstrated. Finally, we list in detail how our application using Cassandra was developed. We further explain why we chose this database, how it applies to our data, and thus give a sense of Cassandra working in the real world. After going through this paper, a reader may be able to understand the differences between Cassandra and a traditional database, with the help of our developed application as an assisting example.

Keywords

NoSQL, Cassandra, database, query

1. OVERVIEW

A NoSQL database is a database which uses a different mechanism for storing and retrieving data than the traditional relational databases. NoSQL databases are schema-free, data consistent and they support easy replication. The main objective of a NoSQL database is to have simple designs, allow for scaling horizontally and have a better and finer control over data availability. NoSQL systems are faster than relational databases as they use different data structures altogether. Choosing among these two types is a complex question with no easy answer. One should select the type of database depending on the application's purpose behind solving some problem. There are many differences among Relational and NoSQL databases. NoSQL has no fixed schema, whereas Relational Databases have fixed schemas. Relational database follows ACID properties (Atomicity, Consistency, Isolation, and Durability). NoSQL, on the other hand, is only "eventually consistent".

NoSQL was developed based on Dynamo [datastax 2012b]. Dynamo is type data storage which combines properties of both databases and hash tables. It uses a key-value structured storage system and was created to solve Amazon's scalability issues faced during 2004's holiday season. By 2007, it was implemented in Amazon Web Services. Many NoSQL projects were based on this dynamo system principle published by Amazon. One such database which became the world's 9th most used database is Cassandra.

CASSANDRA

Cassandra is a NoSQL database that is highly scalable, boasts of being fault-tolerant and data consistent. In comparison to traditional relational databases, Cassandra is a column-oriented database. It adapts its distribution design from Amazon's Dynamo while using data model similar to Google's BigTable. This database was developed at Facebook, for inbox search.

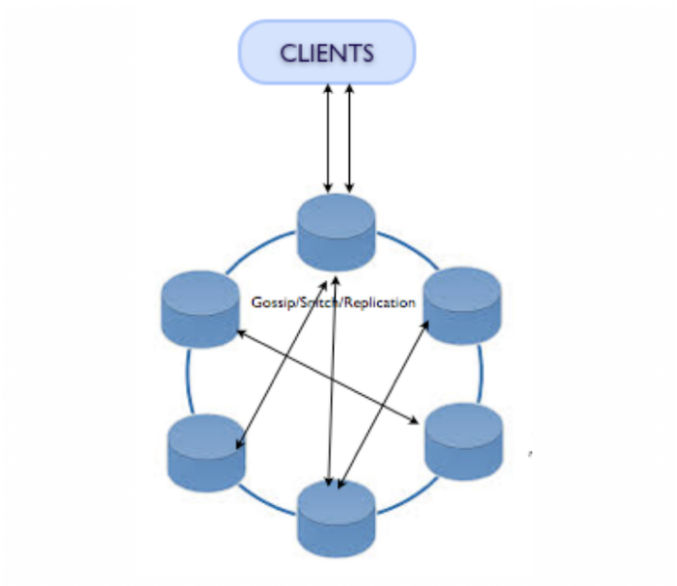

Figure 1: Cassandra ring topology

Cassandra has a peer-to-peer distributed system , that spans across multiple nodes in a cluster. Data is divided among these nodes. Its aim is to manage big data workloads across its various nodes, without any single point of

failure. All nodes are independent of each other while being connected among themselves. This enables read and write requests being accepted despite where the data is actually stored in the cluster. While recovering the system if a node goes down, read and write requests can be passed on to adjacent nodes. Cassandra is being used by some of the biggest companies such as Facebook, Netflix, SoundCloud, Uber, Cisco's WebEx, Reddit, OpenWave and many more [Quora 2010]. Apple has claimed to be using 100,000 Cassandra nodes. Cisco's WebEx stores user feeds and activities in realtime using this database. Openwave utilizes it for its messaging platform's distributed storage needs. Soundcloud stored user's dashboards. Uber uses Cassandra to store around 10,000 features in their daily updated company-wide Feature Store for low-latency access during live model predictions. We will study how Netflix implements Cassandra in their system in the next sections. To justify this wide acceptance, here are some of its unique features:

- Flexible scalability: Cassandra has a master-less design where all nodes are the same. It is easier to scale it horizontally. It also allows to add more hardware to accommodate more customers and more data as per requirement.
- Linear/Horizontal scale performance: Since Cassandra helps one increase throughput as one increases the nodes in a cluster, it maintains a quick response time.
- High Availability: There is an inbuilt failure detection algorithm called Gossip protocol to identify states of all nodes in a cluster Also there is no master-slave related concern in this kind of system.
- Transaction support: Cassandra supports atomicity and isolation at the row-level, but trades transactional isolation and atomicity for high availability and fast write performance.
- Faster writes: Cassandra's performance is blazing fast even on cheap commodity hardware. It is thus apt for Big Data applications.

Following is the roadmap that can be used to parse through this paper's layout. Section 2 discusses Cassandra's architecture and data model. Next we see how a Cassandra application like Netflix uses indexing and data storage to implement their system. Section 3 delves into Query processing and its optimization. Cassandra's transaction management and security support system is explained in Section 4. Section 5 introduces our team's Cassandra based application.

2. DATA STORAGE AND INDEXING

2.1 Architecture

Cassandra's architecture is a peer to peer system distributed across multiple nodes in a cluster. Data is assigned to these nodes. Every node is independent while being interconnected. Data is replicated among nodes to increase availability [Mohammadinasab 2016]. When a node goes down, other nodes in the cluster serve the read/write requests.

One or multiple nodes represent replicas for a given set of data. Time-stamp is used to store all values as seen in figure 3. If a stale or out-of-date value is retrieved from some of the nodes, the most recent value is sent to the requester client. Thus maintaining consistency is of utmost importance. Cassandra then to handle such situation, performs a **read-repair** in the background, which updates all outdated values and replaces them with the latest value.

The basic components in Cassandra are:

- Node: Data is stored here.
- Data Center: Collection of related nodes.
- Cluster: Contains one or more data centers.
- Commit log: A crash-recovery mechanism. Every write operation leads to data being written onto commit log first. This ensures durability. Once all the data from commit log has been flushed to SSTables, it can be recycled, deleted or archived for later use.
- Mem-table: This is a memory-resident data structure. After data is written onto commit log, it is then written onto the mem-table. A single-column family may require multiple mem-tables in some cases.
- SSTable: SSTable(Sorted String Table) is an immutable data file. Mem-tables are written on SSTables periodically. They are append-only and are stored on disk in sequence. Each Cassandra table has a maintained SSTable. Its contents are flushed from Mem-table when they reach a certain threshold.
- Bloom filter: They are algorithms that are fast and nondeterministic. These are used to test whether an element is a member of a set. It is a special kind of cache. Bloom filters are accessed after every query.

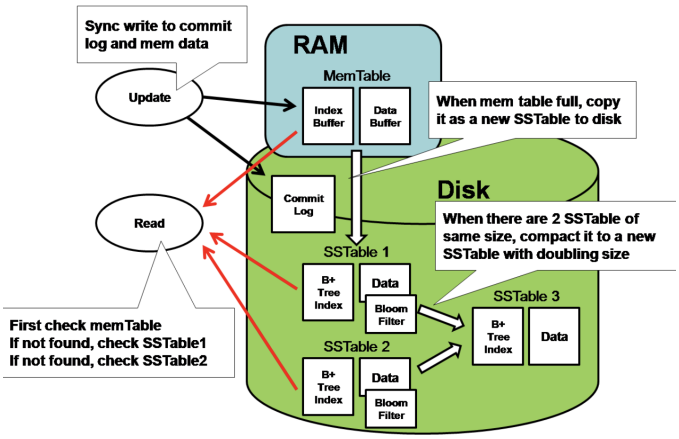

Figure 2: Read-Write Overview

CQL is Cassandra Query Language which a very simple SQL like language. CQL shell is used to work with CQL. **Write operations** are captured by the commit logs and are written in the nodes. This data is then flushed to mem-tables. When the mem-table gets full, data is transferred and written onto SStable. All writes are partitioned on their own, then replicated across the cluster. Data is checked periodically and any unnecessary records are discarded. When **Read Operations** are performed, Cassandra reads values from the mem-table, then looks up the bloom filter for finding the appropriate SSTable that holds the required data.

2.2 Data model

Distributed over multiple machines, Cassandra ensures Data Consistency. The Data model is explained as follows. Outermost container called Cluster contain several nodes. Each node has a replica, in situations of node failure, where the replica would take charge. These nodes are placed in a ring format, and data is assigned to them. Data has its outermost wrapper called a **Keyspace**, which is equivalent to database space for a relational model. It takes three parameters that can be changed as per requirement.

- **Replication factor** is the number of nodes in a cluster that should receive copies of the same data.
- **Replica placement strategy** is the algorithm used for determining where to place these replicas in the ring topology. These are Simple-Strategy(rack-aware), Old Network topology strategy(rack-aware) and Network topology strategy(datacenter-shared).
- **Column family(ies)** can be single or more than one in a keyspace. These are a set of rows, each row containing ordered columns.

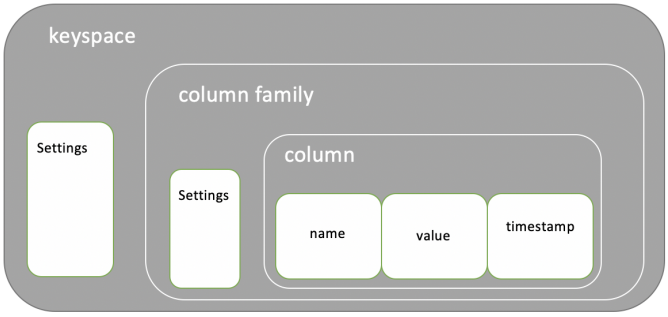

Figure 3: Cassandra's Data Model

Comparing **Column Family** and table of RDBMS. A column family contains a set of ordered rows. Every row is collection of columns. Relational model has fixed schema, where inserting data in a new row needs null or some values for all columns in the schema. On the other hand, in Cassandra, column families are defined but columns are not. One can easily add a new row with additional columns or use lesser columns than specified initially. Relational tables define columns only and the user has to fill in values in these columns. However, in Cassandra, a table may be defined as a super column family or may contain some columns. Data in columns is physically sorted, thus making querying a set of columns easier when one is spanning rows.

Columns have three attributes in its data structure namely:

- Key or Name (byte[])
- Value (byte[])
- Timestamp (clock[])

A super column stores a map of all its sub-columns. Since it itself is a column, it has to maintain a key-value pair. Usually column families are stored in separate files on disk. Hence it is recommended to store columns that will be queried together in the same column family. This ensures performance optimization.

Super columns have:

- Name (byte[])
- Cols (map<byte[],col)

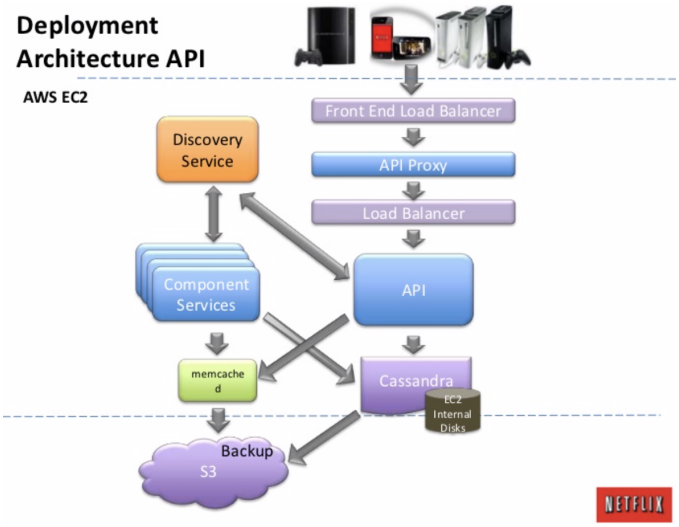

Figure 4: Netflix deployment architecture

Considering the key differences between RDBMS and NoSQL Cassandra: we know that RDBMS deals with structured data. Cassandra deals with unstructured data. No fixed schema in Cassandra unlike RDBMS. In RDBMS, a table is laid out as an array of arrays with size of (Rows x Columns). In Cassandra, a table is a list of nested key-value pairs of size (Row x Column_key x Column_value). Row being an individual record in RDBMS, it is a unit of replication in Cassandra. Attributes in a relation are represented as Columns in RDBMS, while Column is a unit of storage in Cassandra.

Consider Netflix as an example to further learn about data storage and indexing. Netflix uses Cassandra to store enormous amounts of data on an hourly fashion. Cassandra is used in its backend with Astyanax. Astuanax is an algorithm that writes rows from a single region. This causes multiple zones being available using the idea of a variable named Token. Observing the figure 4, the deployment architecture of Netflix has Cassandra accessing a central memory. From figure 5, we can see the clients are Token aware. The system can be spanned across multiple regions. Cassandra is a very useful NoSQL database for this purpose. As we can have widely placed interlinked nodes. Using Local and Remote writers who request and release writes from clients, central database can be accessed. Cassandra's indexing comes into play here. Indexing sorts the Astyanax, making it faster as index is auto-configured[TutorialsPoint 2015].

3. QUERY PROCESSING AND OPTIMIZATION

Apache Cassandra is a distributed database and provides scalability of nodes to store the data. Apache Cassandra can

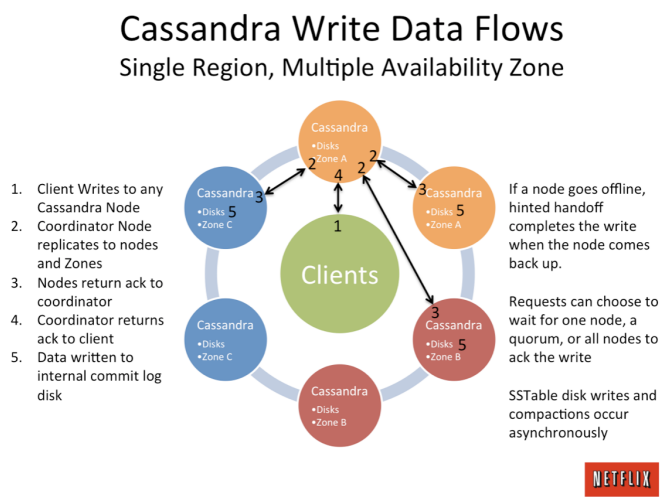

Figure 5: Write flow (Netflix)

store the data in tables and each table has rows/records and columns/attributes. Each table must have only one primary key. But the primary key could be a tuple of attributes making it ideal for easy access in case of more complex queries. The primary key declared for a table is given as input to hash function to find the node in which the data will be stored. Once the node is decided, Cassandra processes data at several stages such as logging data in commit logs, writing data to memtable, flushing the data from memtable, storing data on disk in SSTables.

Cassandra Query Language (CQL) is used to execute queries in Cassandra. Though it is similar to SQL, it has some underlying differences. One can run queries using the 'cqlsh' or from source code and using Cassandra drivers to talk to an DB instance. Cassandra uses Netty as its network protocol. Below we go through how Apache Cassandra executes CQL query as shown in figure 6:

- The service is initiated by the server and is a byproduct of CassandraDaemon being started where the results will be catered to by the client. This is followed by the JVM being invoked which is for the purposes of running the Main method when running the './bin/cassandra'. This helps with the initial set up of the NativeTransport Service by conquering and beginning the network services, and also helps with the initialization of the Cassandra Service. All classes below are placed under the path 'org.apache.cassandra'

- transport.Server
  A socket server is initialized using this class. Using 'childHandler' for setting up and initializing each newly connected channel. Class Initializer in Cassandra receives a CQL query request, and delegates the processing and sending of response.

- transport.Server.Initializer
  This class configures pipeline for incoming and outgoing messages.

- transport.Message.Dispatcher
  Requests in Cassandra are all based on transport.Request

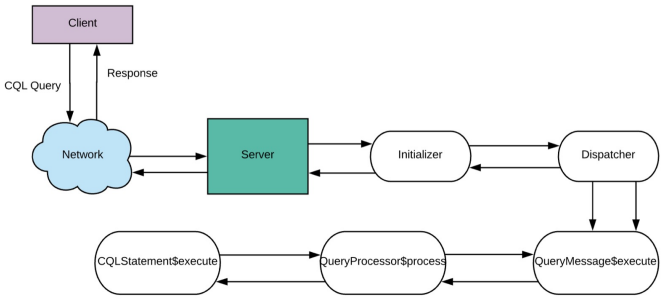

Figure 6: CQL's Query Processing

abstract class. For each different type of request, there is a separate concrete implementation of this class. The most important of which for this post is transport.messages.QueryMessage. Method 'channelRead0' calls execute method of the received request. The output is a Response given back to the client.

- transport.Message.Request$execute
  Abstract method for different types of messages. We are considering QueryMessage.

- transport.messages.QueryMessage.execute
  This method delegates the processing to QueryProcessor.process method. This works on tracing and exception handling

- cql3.QueryProcessor$process
  This first calls 'getStatement' to parse the query string, creates an instance of 'ParsedStatement.Prepared'. Here an attribute called 'statement' is specified which is of CQLStatement type. This type is an interface having all query types' implementations. For instance, cql3.statements.UpdateStatement handles INSERT / UPDATE statements. After getting an instance of CQLStatement, its 'execute' method is invoked to do the processing and returning the response.

Let us consider our example Cassandra application of Netflix and see how it applies query optimization techniques to improve efficiency. Netflix handles around 5 TB of data on everyday basis. There is a two-tier system for storing all queries accessed by a user with login credentials. Other tier is for a new set of queries that can be taken in. This system has been optimized by depending on low latency performance and yet utilizing RAM to store new and old queries together. Using GRANT/REVOKE system, these queries are transacted from client to the cloud for storage. When load to access cloud increased overnight, the engineers at Netflix started relying on Cassandra's transaction management and security support.

4. TRANSACTION MANAGEMENT AND SECURITY SUPPORT

Transaction in a database can be defined as a unit of work performed. It can be an update, insert or delete operation. Transaction management is the process of managing all transactions so that there is no duplicates records or loss

of data. Generally, every transaction management has ACID properties (atomicity, consistency, isolation, durability) with rollbacks or locking mechanism. However, Cassandra being different from other databases, support AID properties. That is it supports atomicity, isolation and durability on any transaction but there is no concept of consistency in terms of referential integrity. It does users and option to control how strong or eventual they want the consistency to be.
Atomicity: In Cassandra, the atomicity on write operation is defined on partition level i.e two or more write operations associated with common node are treated as a single operation. Delete operations follow the same case. Client side timestamps are used to figure out the most recent updates on the data. So, in case of multiple client updates on a common data, only the most recent update is shown in read operation.
Isolation: Cassandra provides row level isolation for write and delete operations. The log structure of Cassandra makes it easier to provide row level isolation. Row level isolation means that any operation being done on a row within a partition on a node is visible to the user performing it. However if operation involves different partitions then that operation is not isolated
Durability: Write operation in Casandra are durable. Each write to a node is written both in memory and in a commit log on disk. In case of a crash or server failure before memtables are flushed to disk, the commit logs are replayed to recover any lost data. The data replication which happens on other nodes adds to the durability feature of Cassandra.
Cassandra mainly provides three main security features which are different from other databases. It provides client to node and inter node communication security through SSL encryption. Using encryption makes the data secure during transmission so that we can depend on data integrity. To manage inter-node and client to node security, settings are changed in the cassandra.yaml file. Authentication security is a plugin feature in Cassandra and is considered in the same cassandra.yaml file using the authenticator setting. Authorization too is a plugin feature configured by authorizer setting in the yaml file. The default authorizer setting allows all authorization to all roles however settings can be changed to grant or evoke access to users as per need [datastax 2012a].

5. NOSQL DATABASE APPLICATION

For the term paper, we have developed sample application to demonstrate the performance,scalability and reliability of Cassandra database. The application provides information about the sales of video games in the world over a certain period of time. Instead of creating a primary key-foreign key relation like in relational data base, a composite primary key if necessary would be created for each table created in Cassandra database. Since the tables in RDBMS are normalized, there might be situations where we would require data from different tables. For example, we might need the number of games that are sold in North America region for a particular year. The data is not present in a single table and might need to retrieve data from multiple tables in case of RDBMS. But in Cassandra database, the table is created with composite primary key and the index that refers to primary key can be used to locate the data quickly.

For the demonstration of scalability and reliability, we have created 1 datacenter containing 2 nodes with SimpleStrategy and replication factor as 2. One node is the seed node and the other node is non-seed node. We have created keyspace named "finalproject" and data tables to store the data. This would mean that if the seed node is down for some reason, non seed node would become the seed node to provide availability of data and we have demonstrated the same in our project.

Nodetool is useful to check the load/ownership of data across the nodes in that cluster. We used the nodetool to track the status of the nodes ar frequent intervals. First we have modified the "cassandra.yaml" file at "/etc/cassandra/" in ubunutu machine at each node so that each node could understand which node to connect. After that, we brought up the nodes, created keyspace, created tables,loaded data from random nodes at each process just to verify the connectivity of the nodes.We then brought down one of the node and queried another node and checked for the available data. We found that all the data is available even after one node went down.

6. COMPARATIVE STUDY

Here we draw a comparison between Cassandra and other databases.

6.1 Cassandra versus Relational Databases

Relational databases are the most commonly used type of database. They have been in practice for years and continue to be a great option for database implementation. They are applicable for a wide variety of uses, from scientific research setups to business applications and commercial operations. Web applications are famous for using Relational Databases that use SQL such as MySQL, and PostgreSQL. Limitations of Relational Databases:

1. Some requirements are tough to meet when using relational databases. For instance, when an application needs to write large volumes of data quickly, or when an application needs fastest query response time possible.

2. High availability is another aspect where relational databases need to compromise on other functions. For example, when such databases are achieving high read and write rates, other features like consistency of maintaining correct data becomes a huge task. Tracking multiple users concurrently, along with reading and writing data is a significant overhead to the database operations that can be improved with NoSQL databases.

NoSQL databases like Cassandra do not guarantee the same level of consistency that a relational database does, but if one is willing to comprise on that, Cassandra can yield a significantly faster database. Cassandra a wide-column, NoSQL database has many **similarities** to relational database such as basic structure which is a table in both cases. Cassandra uses same datatypes as Relational database such as Integer, Fair Care, and Date. Cassandra tables also have primary keys. Primary keys are used to access data in Cassandra. Primary keys are not enough to find rows in Cassandra, because Cassandra is designed to run on a cluster of servers. Cassandra uses many commands similar to the ones in SQL, like commands to select, update data, define structures, etc.

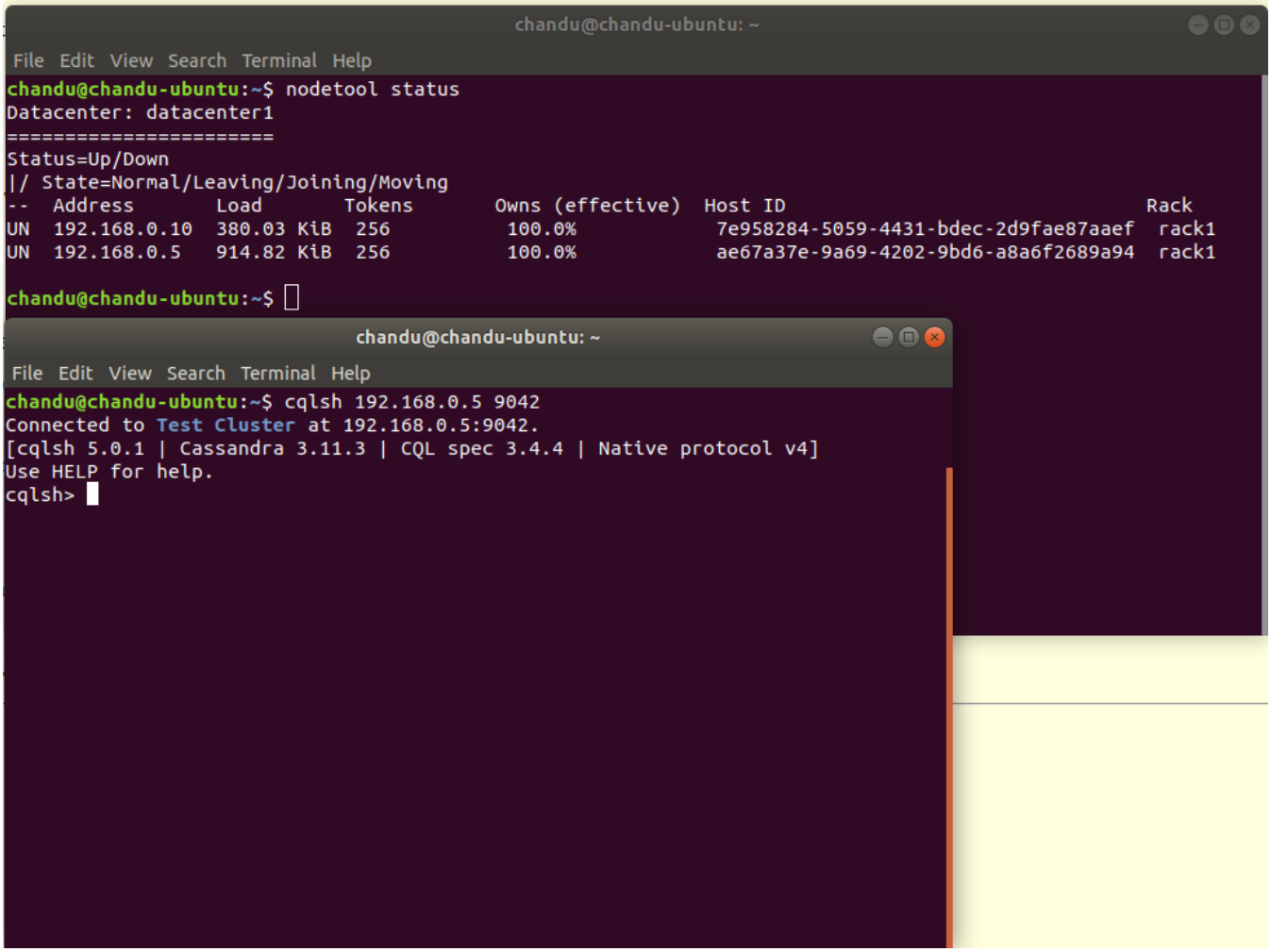

Figure 7: Status of all the nodes

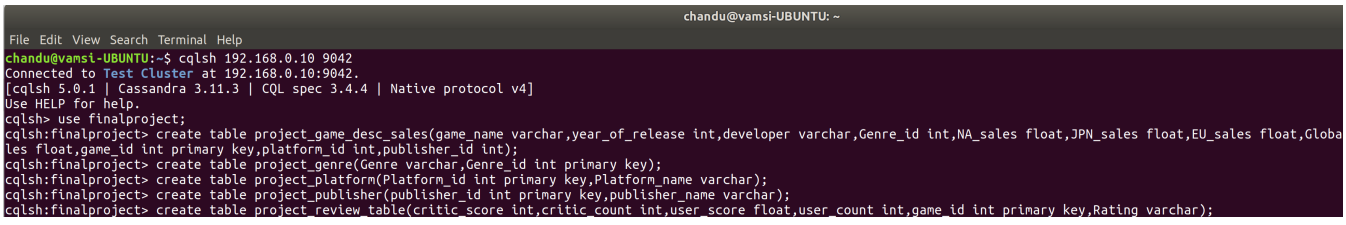

Figure 8: Creating tables at node 2

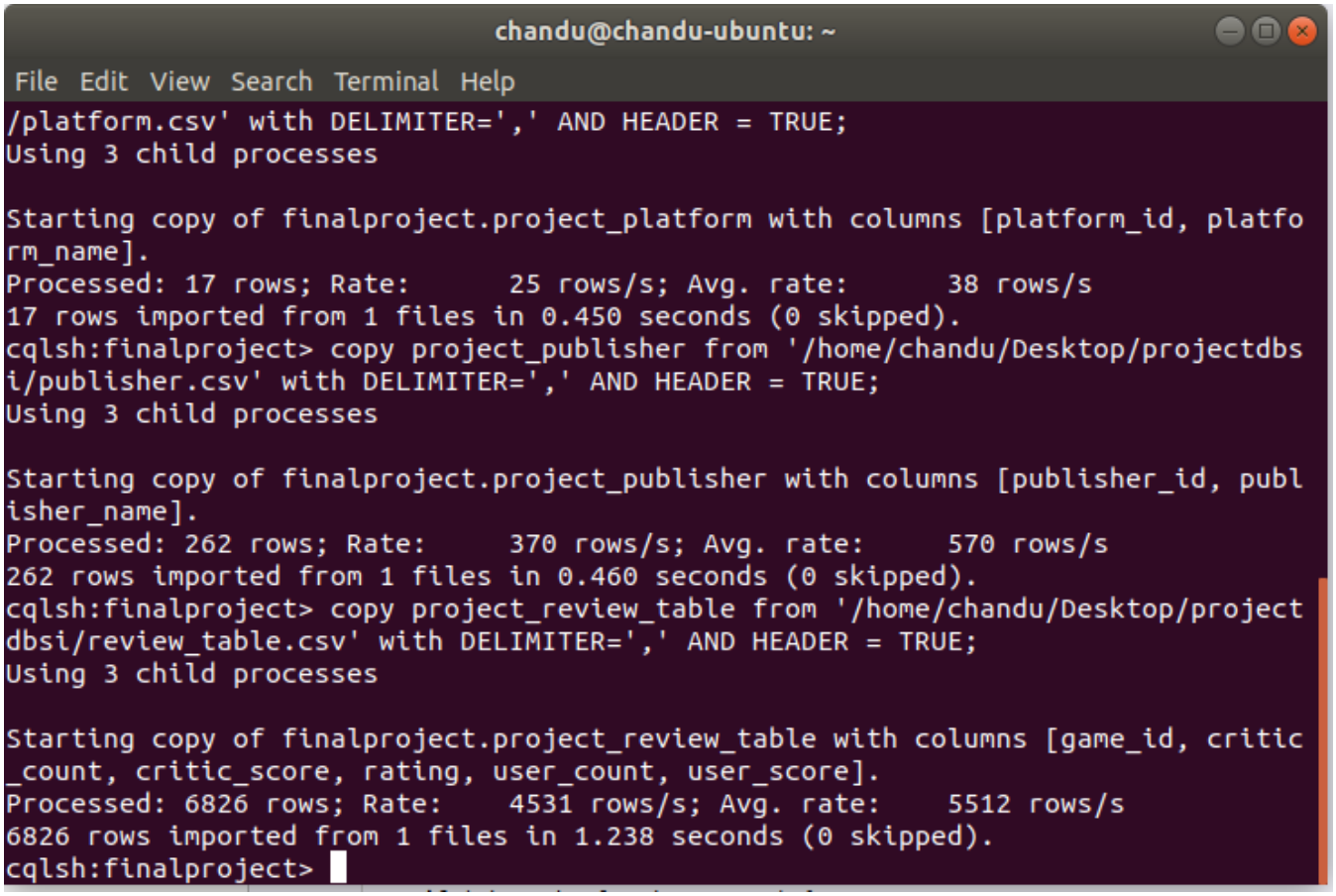

Figure 9: Loading data to tables at node 1

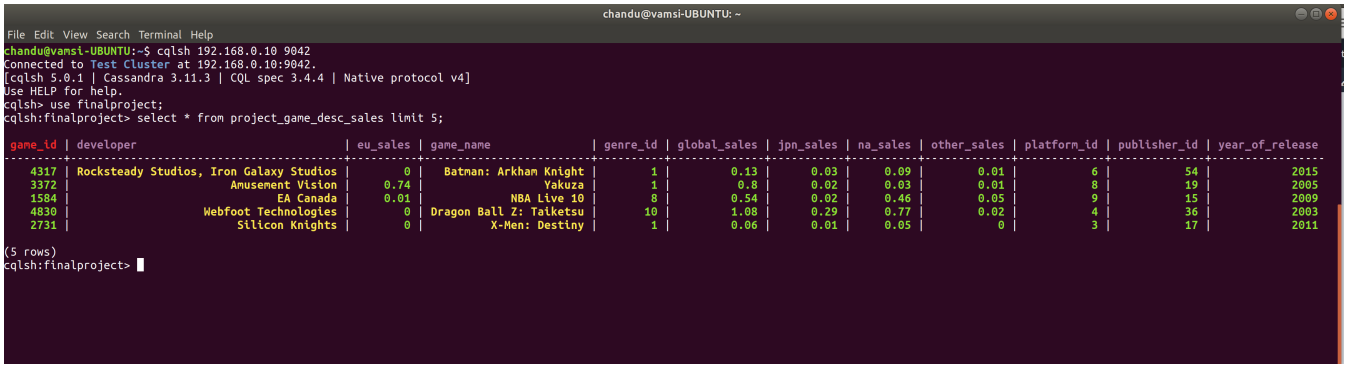

Figure 10: Sample Data at node 2

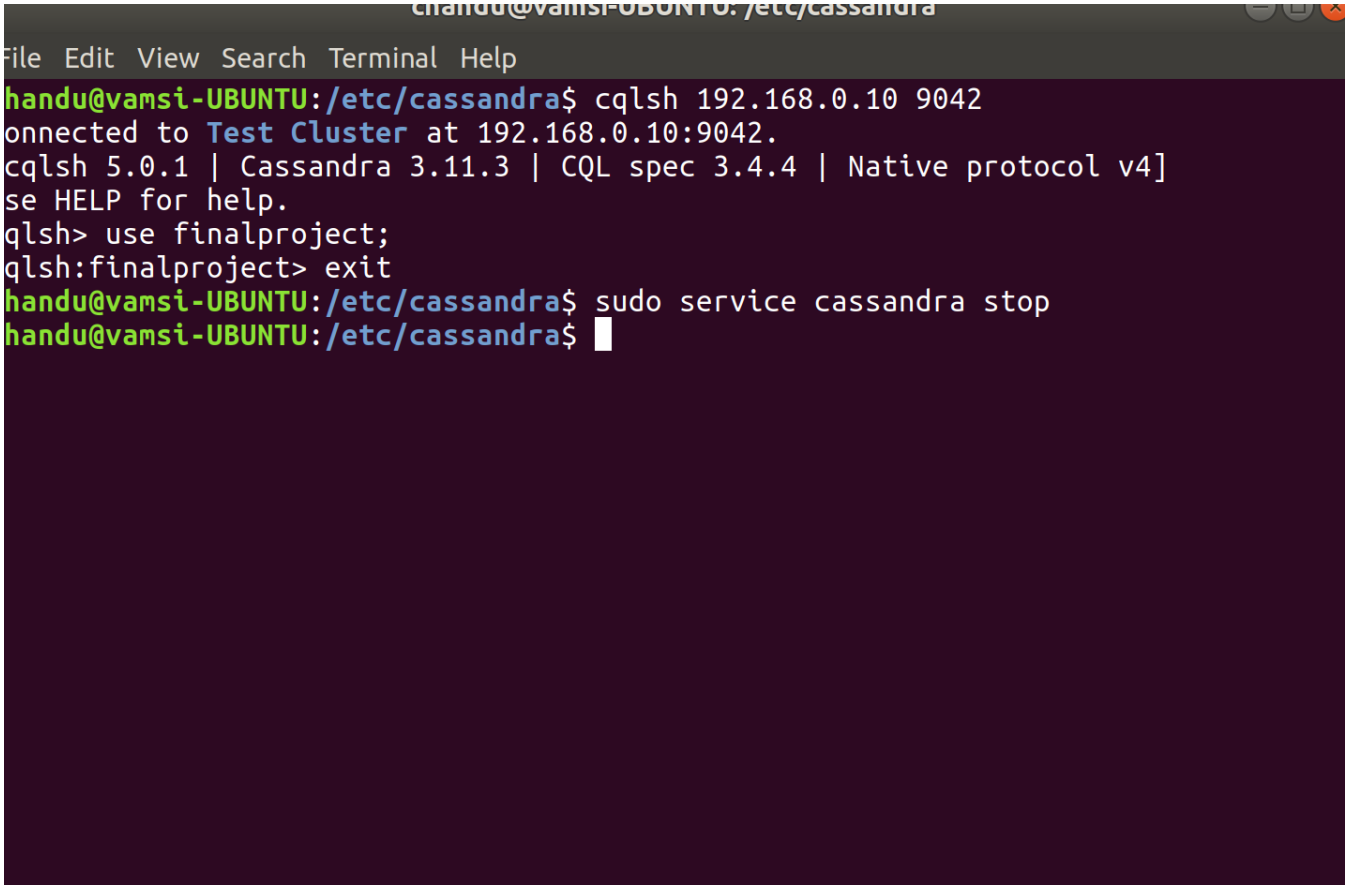

Figure 11: Bringing down node 2

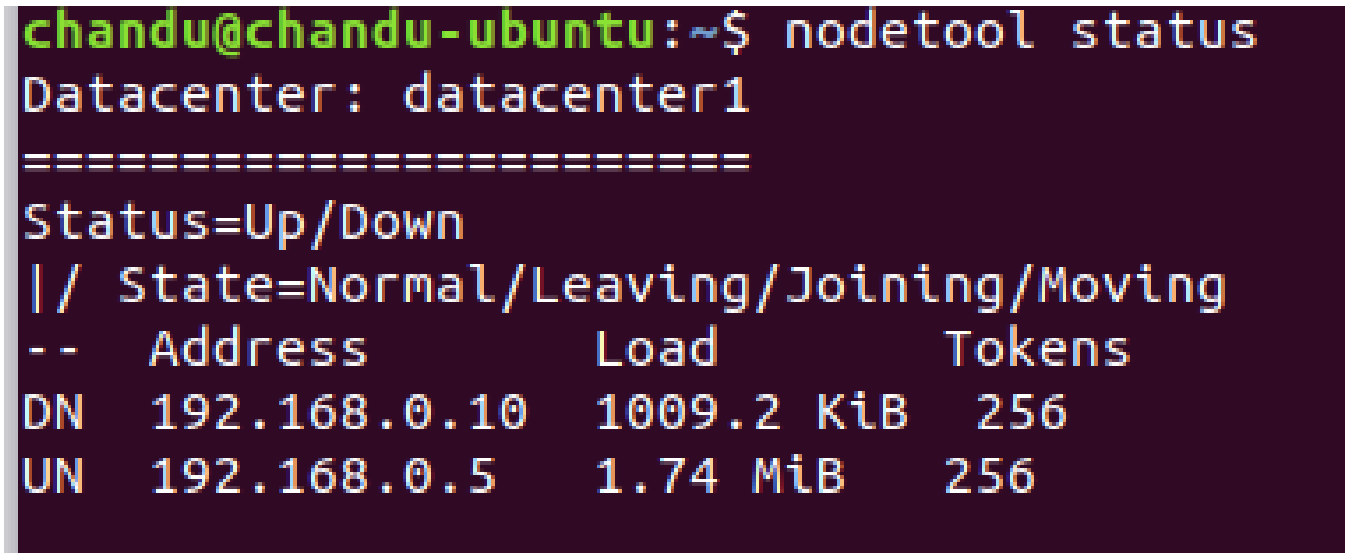

Figure 12: Status of nodes after bringing node 2 down

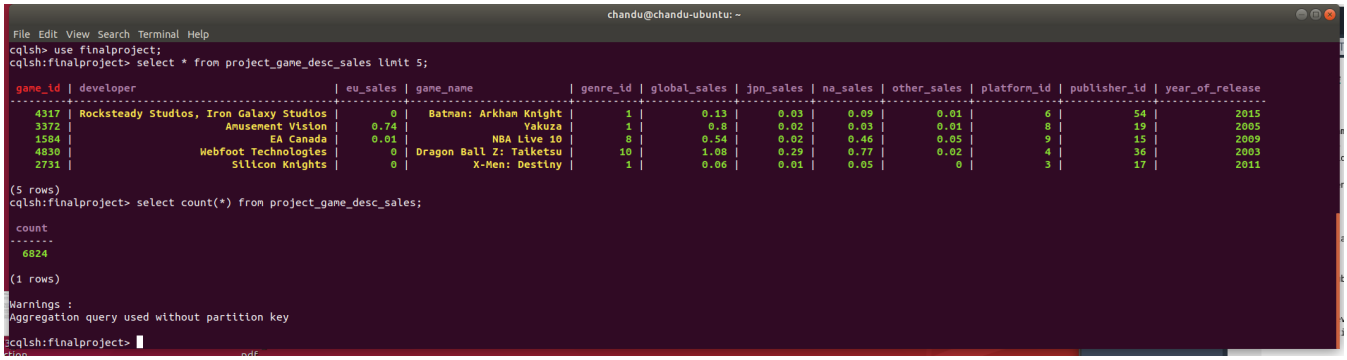

Figure 13: Querying data at node 1 after bringing down node 2

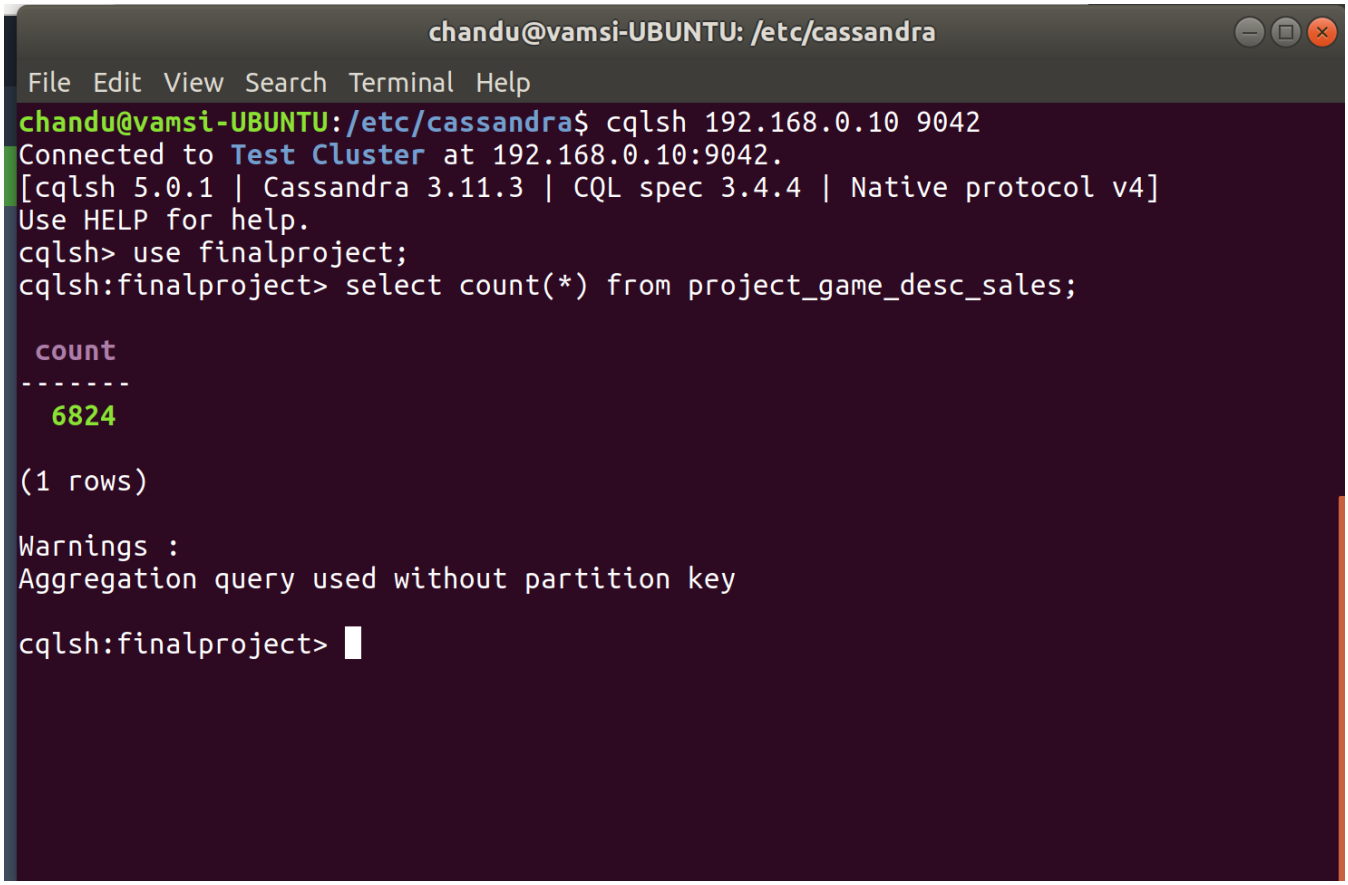

Figure 14: Querying at node 2

Differences are as follows. Cassandra runs on a cluster of servers unlike Relational databases that run on single server. Production databases that use Cassandra are structured across multiple nodes of servers to enable faster access. Cassandra uses multiple keys to enable quick row access. A partition key is used to determine which node in the cluster to store a row in. A clustering column defines the order in which rows are stored. The main difference is that Cassandra uses CQL instead of SQL.It uses a select statement for querying. Cassandra does not have a fixed schema. Some rows may have different columns than other rows.

Another difference is that Cassandra is **'eventually' consistent** unlike relational databases. This implies there are brief periods of time where replicas of same rows have inconsistent data. This is because Cassandra keeps multiple copies of data on different nodes. It takes a minuscule amount of time to update all records. User when finds different versions of data is nothing but a scenario of being inconsistent which will eventually become consistent. Cassandra is compatible with Multi-cloud, multi-directional availability. High availability, dynamic data model are some reasons where a NoSQL database like Cassandra gets bonus marks compared to relational databases.

6.2 Cassandra versus Other NoSQL databases

There are many NoSQL databases like MongoDB, Neo4j, Couchbase, among others. Following are why Cassandra is different from them.

1. MongoDB has a very rich object model. In MongoDB we use objects which can be interleaved inside one another. Whereas Cassandra follows a very traditional table structure. Secondary indexes work better in MongoDB than Cassandra. MongoDB has a single master multiple slave architecture, so it has a single point of failure. Whereas, Cassandra has multiple masters due to it's distributed nature thus there is no single point of failure.

2. Neo4j is a graph database, whereas Cassandra has traditional table structure. Neo4j has clustering properties which provides higher queries per second but it's not distributed as compared to Cassandra.

3. Couchbase has a JSON based storage for documents whereas Cassandra has column based storage. For security features in Cassandra, the access rights to users can be defined per object, whereas Couchbase has LDAP integrated authentication.

7. FINAL REMARKS

Researching NoSQL as choice instead of a traditional Relational Database, we learned the differences and benefits of both. NoSQL is an efficient database system that is flexible and scalable. The comparison between them was supported with a case study of Netflix that uses Cassandra in their system implementation. Out of various NoSQL database, Cassandra was the one that we decided to study and implement. Cassandra is known best for its features like scalability and reliability. It is a schema free database that uses CQL. It uses a different than RDBMS keyspace environment and stores data in nodes that may contain multiple replicas for high availability. Since the data retrieval depends only on the primary key, the data can be easily found. But there are few disadvantages of using Cassandra. One of the biggest drawbacks would be the amount of space required to store the data. Since the tables created are not normalized as in RDBMS, a lot of space is required to store data. There could be out of sync index issues and latency in write operations under heavy load. Also, we have learned that Cassandra is schema optional database, unlike RDBMS. We have implemented our application to demonstrate the usefulness of the Cassandra database system. We have created multiple nodes in multiple PCs. Cassandra may not the solution for every scenario, but when used for the right reasons(good scalability, low operation cost and advanced query models), it does a great job. After our project, we understood how minimal effort is needed to manage the data in a distributed network. We have understood the reliability of the data along with availability of the data even if some of the nodes in the network goes down.

8. APPENDIX

Trisha P. Malhotra: Abstract, Overview, Data Storage and Indexing(Architecture, Data Model), Query Processing for CQL with Apache classes, Cassandra versus Relational Databases, Flowcharts, Final Remarks, presentation and completing the final draft of this paper.
Vamsi Chandu Manne: Query Processing, Development of application, Final remarks of this paper.
Mayank Pandey: Transaction Management and Security Support, Comparative Study, presentation.

9. REFERENCES

[datastax 2012a] datastax. 2012a. Cassandra Security. https://docs.datastax.com/en/cassandra/. (June 2012).

[datastax 2012b] datastax. 2012b. How are Cassandra transactions different from RDBMS transactions. https://docs.datastax.com/en/cassandra/3.0. (June 2012).

[Mohammadinasab 2016] Mahdi Mohammadinasab. 2016. How Apache Cassandra executes CQL queries? http://www.mahdix.com/blog/2016/06/01/how-apache-cassandra-executes-cql-queries/. (June 2016).

[Quora 2010] Quora. 2010. Quora Cassandra. https://www.quora.com/Which-companies-use-Cassandra. (June 2010).

[TutorialsPoint 2015] TutorialsPoint. 2015. Cassandra Architecture. https://www.tutorialspoint.com/cassandra. (June 2015).