Cassandra

Anonymous

ABSTRACT

NoSQL databases were released in the early 2000s. Since then, they have been a topic of growing interest. Many highly scalable systems make use of NoSQL databases. NoSQL databases support replication, are schema free, are eventually consistent, have easy api for access, and are designed for handling lots of data. Cassandra databases are a popular NoSQL database available today. Cassandra provides high scalability, high performance and high availability to its users. Cassandra has features that are different from traditional and still-in-use RDBMS. Cassandra's features in data storage, indexing, query processing, query optimization and transaction management are presented. A typical Cassandra application is taken as an example and features are compared with RDBMS. Design and implementation of an actual Cassandra application is also described.

1 OVERVIEW

RDBMS were an interesting improvement over traditional databases like IMS(Information Management Systems). Although they were difficult to learn, they were able to solve a lot of problems and hence generate good revenue. Relational applications as it was seen over the years, had scalability issues with the growth of data. Joins inherent in relational databases were slow. Database locks resulted in the queuing up of user requests and were satisfied in turns. Although solutions like vertical scaling, improving existing indexes, denormalization were suggested , these weren't enough for the scenarios where data moved at a high velocity. NoSQL databases came at a time when highly scalable and high performance database management systems were needed.

Cassandra was developed by Facebook as they were faced with a problem that traditional RDBMS would find difficult to handle. Exciting features like being distributed and decentralized[17], being elastically scalable[15][17], having highly availability and fault tolerance[15][17], having tuneable consistency [17], being column oriented[17], schema free and having high performance[17] allowed a new model to emerge in Cassandra which is widely used by many big and small organizations today.

Distributed and Decentralized

Cassandra is distributed, meaning it can be run on many different machines. Its own code base shows that it is optimized for running across multiple different machines and across multiple data center racks.

RDBMS follows a master-slave strategy while trying to scale to multiple machines. Here one node is the master node and we have multiple slave nodes. This means that nodes can perform different operations. In typical RDBMS clusters, we can have a master-slave relationship to optimize performance as we can serve simultaneous requests. A master is more like a authoritative source of data which interacts with the slaves in a unidirectional manner. Slaves in turn try to synchronize the copies of the data they have. With multiple slaves and a single master, master node can become one point of failure. To avoid this scenario, complex measures like having "multiple masters" might need to be taken up.

In Cassandra, the master-slave model is not followed and we have multiple nodes that are equal and function the same("server symmetry"). This means there can't be a special host coordinating the activities unlike the master-slave model in many popular RDBMS systems. Cassandra uses P2P and gossip to keep the nodes in sync. Since all the nodes are equal there is no one point of failure too.

The decentralization is easier to maintain than the master-slave relationship model. Adding 50 nodes is almost same as adding 1.

Elastic Scalability

Scalability is associated with ability to serve more number of requests. In database systems there are two strategies for scalability - vertical scaling and horizontal scaling.

Vertical scaling involves adding more memory and hardware capacity to our existing systems to serve more requests. Horizontal scaling is concerned with adding more nodes with some or all of the data so that we do not burden a single system with processing requests. We do need an internal mechanism to keep the data in sync between the nodes.

Elastic scalability is a property of horizontal scaling that deals with adding or removing machines seamlessly. This means there is minimal to no disruption in service.

High Availability and Fault Tolerance

Computer systems are susceptible to corruption, failure of hardware components etc. A highly available system should be able to serve requests continuously with minimal to no disruption in its functioning. Cassandra is highly available. A failed node can be replaced in no time. Data can be replicated to local data centers to improve performance during a catastrophe.

Tuneable Consistency

Consistency is referred to getting the most recent written value. Cassandra trades consistency for availability. More the number of replicas of the data more difficult it is to maintain the most updated value. Replicas eventually have the most recent value but this tends to take some time. Availability factor(replication factor) can be tuned to provide a more strict consistency or a more weak consistency.

Column-oriented

Cassandra is column oriented . It has features separate from RDBMS and stores data similar to a sparse multidimensional hash table. Every row has a unique key that makes its data accessible. It is called sparse because not all rows will have the same number of columns as the rows like them.

Unlike RDBMS where we create a model and write queries around that model, we have the freedom to choose the queries first and then provide the data that answers them.

Schema-Free

Cassandra defines "keyspaces" which are outer level containers for storing column families and configuration properties. Column families are names for associated data and a sort order. Besides this, the data tables are sparse and new data and columns can be added easily. Columns need not be defined in advance. This shifts the focus on modelling the data and optimizing the query for the data to modelling the queries that are needed and then providing the data around them.

High Performance

Cassandra performs exceptionally well under heavy loads. It was designed to work with multiprocessor machines present across many data centers. It has a high throughput for writes / second.

We choose a hotel application as an example to explain the various data storage, indexing and query support features of Cassandra. This applications allows guests to search for a hotel in a city, query room availability at a hotel, query amenities provided in each rooms, get information about points of interest around a hotel,etc.

Section 2 talks about the data storage and indexing features of Cassandra. Section 3 talks about the query processing and optimization features of Cassandra. Section 4 talks about transaction management and security support that comes with Cassandra. Section 5 describes the design and implementation of the hotel app. We conclude and provide final remarks in Section 6.

2 DATA STORAGE AND INDEXING

Cassandra stores its data in a different form than RDMS. As Cassandra is a distributed system, it can spread its functions over various machines and can interact with every machine. Cassandra stores its data in a cluster which is created of multiple nodes. Every cluster in Cassandra is shaped in a ring and the data in stored in each node. Every node in the cluster has its duplicate which comes into use when the actual node fails. Therefore, in Cassandra, if a node fails due to any reason, Cassandra will still continue to work. Cassandra cannot perform operations such as joins, aggregation, group by, etc, and hence, the relations in Cassandra should be created in such a way that retrieving information from it is a simple, easy and fast. [14]

Keyspace

Cassandra has a way of replicating the data on the nodes using a namespace known as "keyspace". Keyspace in Cassandra is used on clusters and is the outermost container in Cassandra. Each node in Cassandra has a keyspace. The syntax for keyspace is [16]

CREATE KEYSPACE <identifier> WITH <properties> (1)

where "identifier" is the name of the keyspace. We can define "properties" for a keyspace such as "strategy"(Simple / NetworkTopology) and the "replication factor" which is the total number of duplicate copies generated. For example, if we are creating a hotel management application, then we define a keyspace as shown in Figure 1. [14] [16]

```
cqlsh> CREATE KEYSPACE hotels with
   ...  replication = {'class': 'SimpleStrategy',
   ...                 'replication_factor':3}
   ... ;
cqlsh> describe keyspaces

javafrm      hotel_management  system_distributed  system_schema
system_auth  system            system_traces       hotels
```

Figure 1: Keyspace example for hotel management [16]

We can define three kinds of strategies for a given keyspace. These include “Simple Strategy“ which uses a simple replication factor and all the duplicates are only shared within a single data center and is not aware of the placement of the data on the data racks, “Old Network Topological Strategy“ knows which racks the replicas are placed on, and “Network Topological strategy“ is an advanced version of the old network topological strategy, suggests how the replicas should be placed on different racks in the data center. [5] [16]

By defining the replication factor, we can create multiple duplicates of data (rows / tuples) and then spread it across different nodes.[14] [5] Every row in a relation is a unit of replication. We just have to make sure that the number of duplicate copies do not exceed the number of nodes in the cluster. If the total number of nodes is less than the replication factor, then the writes into these duplicate copies will get rejected. [5]

Column Family

One of the specialties of Cassandra is to store the data in a structured format in columns using a concept called as the "Column Families". A column family has many rows gathered together. In RDMS, every row, has the same number of columns, and every column which does not have a value is marked with a NULL value. Cassandra is flexible when it comes to number of columns for every row. Every row in the column family can have any number of columns. Hence, Casandra can handle unstructured data and store it in a structured form. Every value in relation can be accessed by accessing the row and the column family.[14]

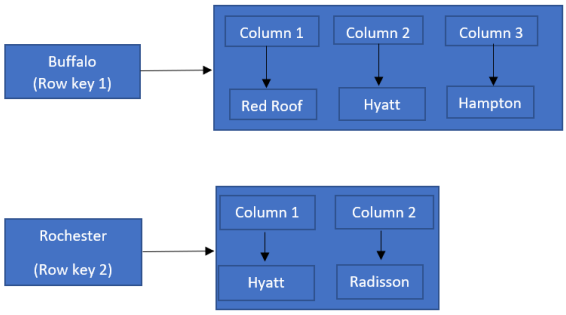

Figure 2: Column Family for City and Hotel [17]

Figure 2 is an example of how rows in a column family are stored in Cassandra. In this example, the rows store the city names and the columns store the hotels The rows of the relation are distributed across different nodes in Cassandra. [2] [17]

Indexing

Cassandra uses indexes based off on a non-primary key, as primary key is unique and is thus indexed. These non-primary key indexes are called secondary indexes. By having a secondary index, we speed up the query process. When an index is created on a column, the values of that column are stored in a different table (hidden column family). The secondary index has the data stored only in its relation, that is, the secondary indexes is not duplicated across different nodes. Hence, when we query on a certain column which has a secondary index, we have to take permission from the nodes to allow to parse through them, wait for a reply from them, parse through different nodes, get the values which match the query condition, combine the results together and then send the results back. Therefore, when there are a large number of nodes, querying using a secondary index takes time. Cassandra, does not have its indexes in a sorted fashion in the hidden column family table, hence, querying for a range using a secondary index cannot be done. Querying for equality using secondary indexes is possible and fast. It is best to create a secondary on a column which has multiple repeating values or which are highly required while querying.[4] For example, in the hotel management application, it would be wise to create an index on the column family called HotelByCity (which stores hotels for each city in the States) as there are more possibilities that the user will query hotels in a certain city. [4] [17]

3 QUERY PROCESSING AND OPTIMIZATION

The query language that is used by Cassandra is known as CQL (Cassandra Query Language). Even though it is a bit similar to SQL, it is quite different. To run CQL queries, 'cqlsh' tool is used in Cassandra.[11]

The query is sent through a network connection when it is submitted through 'cqlsh' tool. A daemon is waiting for the queries which generates a response for each of the received queries. After the response is generated, it is sent to the originator. [11]

CQL query is processed by calling different classes and methods. CassandraDaemon which is present on the server side, initializes the service and server where the requests are received from the clients. This call is made to the JVM which then runs the main method. Here the initialization for Cassandra service is done and NativeTransportService is set which starts the network connection. Org.apache.cassandra.transport Server class sets up a socket server and creates a 'childHandler' which creates and configures new network connections. It processes the CQL query which is requested by the client and acknowledges the response. After the connection is created with a client, org.apache.cassandra.transport.Server.Initializer configures the pipeline for encoding/decoding frames and messages and also performs compression/decompression. It also adds a message dispatcher to the pipeline which is responsible for responding to incoming messages and executing the required process. All the requests that are made in Cassandra are of one of the types of org.apache.cassandra.transport.Request abstract class. It has specific implementation for all the different types of requests. If there is a need to process any new message that is received then it is handled by org.apache.cassandra.transport.messages.QueryMessage which sends back a response of the message to the client. org.apache.cassandra.transport.messages.QueryMessage.execute checks for errors in the query and handles the exceptions. The query string is parsed using 'getStatement' and an instance of 'ParsedStatement.Prepared' is created which includes a 'statement' attribute of type CQLStatement using org.apache.cassandra.cql3.QueryProcessor$process method. Then using the 'execute' method, the query string is processed and the result is returned.[11]

Cassandra works quite well with read and write. It has clever and fast read repair process which solves the consistency issues and makes reading very efficient. It has great data availability. The data in Cassandra isn't stored on a single node but many nodes. So even if some nodes are not responding or down due to some reason, data can still be retrieved from another node. When there is some data which needs to be retrieved much often then Cassandra has a much better read process for that. Here, the data that is to be retrieved is stored in a row cache. To make the process easy, its address can be stored in the key cache. When all the nodes need to be accessed, then secondary indexes become much more useful. Cassandra does not allow any random input of data which optimizes the performance by avoiding unwanted data. Commit log is used while writing the data which makes sure that all the lost data which is not yet stored is restored even if a node goes down.[10]

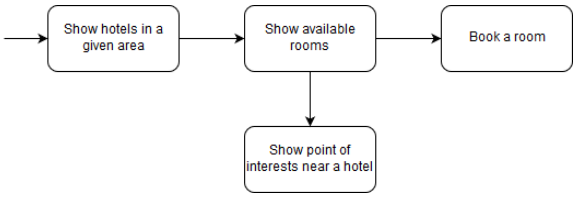

Figure 3: Query flow of hotel management [3]

Here, the flow of some of the queries that can be implemented in the application is shown below. Cassandra stores the data that is read more often in the row cache. The first query is to show display all the hotels in a particular area. This hotel information then can be used to show the rooms that are available in the hotel and also the point of interests that are near to the hotel. If this data is similar to the clients needs then the client can book the room. So the past queries in the flow are used by the other queries which makes Cassandra work faster and efficient. [3]

4 TRANSACTION MANAGEMENT AND SECURITY SUPPORT

The database Cassandra does not have the concept of ACID transactions, rollback, and blocking mechanisms. It mainly works on just atomicity, durability and isolation. Consistency works a little differently in Cassandra. Cassandra works on the CAP theorem which states one has to choose two of the three between, consistency availability and partition tolerance, where, Cassandra provides AP(availability and partitioning) or CP tolerance. [1] The user has control over how strong or eventual the data consistency should be. Cassandra does not have the concept of joins or foreign keys. The write Cassandra writes is durable. [8]

Transaction Property in Cassandra:

Atomicity: In Cassandra, at row/partition level atomicity is supported. i.e., any insertion, updating and delete is considered as a write operation. Many row updating is not supported for everything or none operations. The only time atomicity is ignored is during fast write and high availability operations. A batch log is used to ensure this property. To make sure we do not incur the performance penalty, the UNCLOGGED option is used, which ensures the batch is atomic in one partition. In this database, the most recent updation of a column is kept track of using time-stamp. I.e., if many clients are trying to update a row at the same time, only the latest update is visible to the users. [13]

Isolation: If one row was being updated by one user and another user was reading the same row it is possible to see the partial updates in older versions. The writes to a row are isolated to the user who is currently making a write operation. The other users will not be able to view this row until the operations are complete. All the delete operations are isolated. An update made in a batch operation for a partition key is also isolated. [13]

Durability: A write operation in Cassandra is durable. A write made to a replica node is saved in a commit log and memory. If this operation is complete, only then is it deemed as successful. In case of a crash where memtable is not flushed to a disk, the commit log is made to replay until all the lost writes are recovered. The durability of data is increased with the replication of data on different nodes. [13]

Consistency: Cassandra has 2 types of consistency: Tunable and Linearizable.

- Tunable consistency: To have appropriate levels of consistency, eventual consistency is introduced by providing tunable consistency. We can either set the consistency globally for a data-center/cluster or tune the consistency level for each operation. I.e., you can vary the consistency of the operation so the data returned is more or less consistent. This property makes the database more CP or AP depending on the CAP theorem, in accordance to the needs of an application. Greater the consistency, higher the latency and vice-versa. For read operations, the number of replicas that needs to reply to a read before giving back data to an application is mentioned by the consistency level. For write operations, the number of replicas that needs to reply to a write operation before calling it successful is mentioned by the consistency level. Usually, the client app mentions a level that is lower than a 'replication factor' given by the keyspace. Usually, the consistency level of QUORUM is maintained for read and write.[13]
  The following consistency levels are supported: ONE,

TWO, THREE, QUORUM, ALL, LOCAL_QUORUM, EACH_QUORUM, LOCAL_ONE, and ANY [6]

- Linearizable consistency: It is an immediate isolation level for lightweight transactions(handles concurrent operations using the Paxos protocol). Locking and transactional dependency do not apply to Cassandra. Race conditions, duplicates, etc can produce inconsistency in replicas. Applying linearizable consistency to a unique identifier like email, even though it is not necessary for all parts of a user account. The Paxos protocol(using a quorum based algorithm) can be used to implement the serial operations. There is no need for a 2-phase commit in lightweight transactions. [9] [7]
- Consistency Calculation: If the condition R + W > N is true, strong consistency is assured.
  If the condition R + W =< N is true, eventual consistency is assured.
  R- read consistency level; W - write consistency level; N - number of replicas. [7]

Security Support

The default package for Cassandra disables all of the security features. This is only to increase the availability among the members of a cluster in the beginning. However, security is considered very seriously in Cassandra. The right configuration of the following 3 properties in the Cassandra.yaml file will help prevent the important attacks that could occur on the DB clusters:

TLS/SSL encryption for communication: This encryption is for two types, node to node and/or client to node encryption. The default cipher suites can also be replaced or overridden. This is not recommended but may be required in cases where the JVM cannot be updated. In Cassandra 4 and beyond, the hot reloading of SSL also works. Which is to say that the subsystem periodically polls for the certificates (trust/key store) and updated certificates can be used for subsequent connections within the polling interval (Default- 10 mins).

Client authentication: The authentication can be enabled in the authenticator settings. This would allow configurations such as applying a simple user name/password-based authentication and setup roles for the user or a group of users if required. It's important to note that the following 2 points be considered before the initialization of the authenticator setting:

a) The client-side config should be initialized. b) There are no clients connected to the cluster undergoing configuration change.

Only the 1st node takes more steps and perhaps more time to configure than the subsequent node. The others merely follow and replicate the 1st node to complete the changes across the cluster. Authorization: Enabling is done on the authorizer setting. This would allow configurations such as applying a complete permission management system and stores the information in the Cassandra tables. CassandraAuthorizer module is responsible for permission management. It's important to note that the following 2 points be considered before the initialization of the authorizer setting:

- Client authentication is already set up.
- The node under update should not take any client-side request as initialization will entail rejection of all requests unless it has the permission.

Similar to client authentication, the 1st node takes more steps and more time to configure than the subsequent nodes. The authorization and the authentication process is an overhead and can deter the quality of service of the cluster. To avoid a significant impact on the cluster it is advised to use the caching feature. This helps in offloading the processing of permissions, role data, and client credentials. Configuration of the caches can be accomplished in the cassandra.yaml file. JMX interface also provides a non-persistent way to modify the caching configurations. [12]

5 NOSQL DATABASE APPLICATION

We start with the idea of the Hotel app that we briefly touched in Section 1. This applications allows guests to search for a hotel in a city, query room availability at a hotel, query amenities provided in each rooms, etc. Our implementation deals with one or more of the these features.

Our application includes hotels, guests that stay in that hotel, collection of rooms, record of reservations, points of interest near a particular hotel etc.

Queries

A basic design of a Hotel application should ideally be modelled after the below queries :

- Finding hotels in a given area
- Booking rooms for guests
- Find available rooms in a hotel

Implementation of the application

We have created our hotel management application on a distributed system using Cassandra. The two systems that we have used is the virtual machine with Ubuntu and our host machine (windows). To make our application distributed, we first changed the cassandra.yaml which is available in the apache cassandra folder in the windows system and the ubuntu system. The seed list, rp/c$_a$$_d$$_d$$_r$$_e$$_s$$_s$$_a$$_n$$_d$$_t$$_h$$_e$$_l$$_i$$_s$$_t$$_e$$_n$$_a$$_d$$_d$$_r$$_e$$_s$$_s$$_w$$_e$$_r$$_e$$_c$$_h$$_a$$_n$$_g$$_e$

Schema

A draft of the schema is given in Figure and we have added a notepad file of our schema4

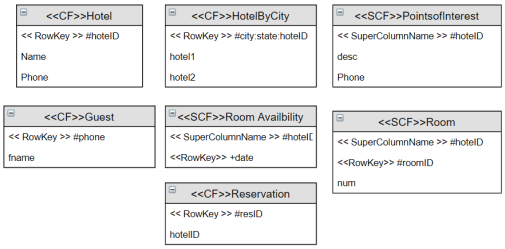

Figure 4: Cassandra data model for Hotel Application

6 FINAL REMARKS

Cassandra	RDBMS
Cassandra is highly famous for its distributed and decentralized system (can handle failures without losing data).	RDBMS does not support distribution of data on multiple machines and cannot handle failure and loss of data.
Cassandra can handle unstructured data and the rows in the table/column families do not necessarily need to have the same number of columns as shown in Fig 2.	RDBMS can only handle data, which is structured, for example it cannot have rows in the relations with different lengths.
Cassandra can store data in the format row key X column name X column key. It can have column families and super column families.	RDBMS support only relations/table and not column families/super column families and can store data is row X column format.
Cassandra uses CQL (Cassandra Query Language) and has syntaxes similar to SQL such as CREATE column family Availability (HotelId INT Primary Key, rooms_available List<text>)	RDBMS uses SQL (Standardized Query Language) which uses syntaxes such as CREATE table Availability(HotelID varchar(255), room_available int, Primary key(HotelID))
Can store lists, sets, hashmap inside one column.	RDBMS can replicate hashmaps only by storing every value of the key-value pair in different columns.
Cassandra cannot create foreign keys or place any constraints in the tables. It does not support operations such as join, aggregation, etc. It can combine data using collections.	RDBMS supports foreign keys and can pplace constraints in the table. Operations such as aggregation, join, etc are supported by RDBMS.
Schema in Cassandra is created after the queries are created. This is because, we only create those relations or column families which are required for the queries.	First, we create schemas in RDBMS and then decide the queries. This makes querying difficult and slow.
Cassandra creates indexes as secondary indexes which are stored in the unsorted format and hence, it is not useful for range-based queries.	RDBMs has indexes in clustered format so it performs well for range-based queries.
Updates are faster in Cassandra. While failure if the master fails, the updates are immediately shifted to the next node with the data.	Updates are slow in RDBMS. If the master fails, then update is paused until the master is up and running.
Cassandra does not follow the ACID Properties. It only follows the AID properties.	RDBMS follows the ACID Properties.

Figure 5: Cassandra vs RDBMS

Cassandra has features like being distributed and decentralized, being elastically scalable, having high availability and being fault tolerant, having tunable consistency, being column oriented, schema free and having high performance.

Unlike RDBMS, Cassandra has "keyspace" as the outer level container for data. Cassandra stores its data in a collection of nodes (cluster) formed in a ring structure. We can define the keyspace by specifying a strategy and defining the number of replicas we want for our data. Inside a keyspace, we have "Column Families" which are analogous to RDBMS tables. In RDBMS the table row / records have a fixed width(same number of columns). We modeled the queries according to the table schema we defined at the start. We do not have the same requirements in Cassandra. Cassandra follows a schema-free model. This allows us to focus more on the queries that we are trying to implement and the provide the data that satisfies the queries,later. Cassandra uses non-primary keys as indexes to increase the performance for a search query.

Cassandra is not good in supporting range based queries using indexes as the secondary-indexes are stored in an unsorted format which requires to access different nodes a multiple times. Whereas, RDBMS can perform well with clustered indexes. Even though the syntax format of RDBMS is not very different from that of Cassandra, Cassandra fails to perform operations such as joins, group by, aggregation, etc.

Cassandra uses CQL while RDBMS is mainly based on SQL. It does not have support for foreign key, joins and group by clauses whereas those are some of the basic things in RDBMS. However, this makes Cassandra more efficient in reading and writing the data. Updates are also efficient compared to RDBMS as no read operations are performed while updating the data in Cassandra.

Cassandra does not satisfy the acid property, instead only follows AID. Partition and availability(AP) is given preference over consistency. It follows the CAP theorem, which says it is not possible to have all 3 properties and have accepting latency. The consistency is configurable using the levels in consistency(tunable consistency). There is no concept of rollback in Cassandra.

7 APPENDIX

Write up for this paper was divided as mentioned below:

- Abstract - Sanyukta, Sahil, Ankita
- Overview - Sanyukta, Sahil, Ankita
- Data Storage and Indexing - Sanyukta, Sahil, Ankita
- Query Processing and Optimization - Ankita, Sanyukta, Sahil
- Transaction Management and Security Support - Ankita, Sanyukta, Sahil
- Database Application - Sanyukta, Sahil, Ankita
- Final Remarks - Sanyukta, Sahil, Ankita

REFERENCES

[1] [n.d.]. CAP theorem. https://wiki.apache.org/cassandra/ArchitectureOverview . Accessed: July 04, 2019.
[2] [n.d.]. Cassandra data model datastax. https://www.datastax.com/dev/blog/basic-rules-of-cassandra-data-modeling . Accessed: July 04, 2019.
[3] [n.d.]. Cassandra Guide. https://learning.oreilly.com/library/view/cassandra-the-definitive/9781491933657/ch05.html . Accessed: July 04, 2019.
[4] [n.d.]. Cassandra Indexing - teddyma. https://teddyma.gitbooks.io/learncassandra/content/model/indexing.html
[5] [n.d.]. Cassandra Keyspace strategies. https://www.vskills.in/certification/tutorial/big-data/apache-cassandra/replica-and-their-placement/ . Accessed: July 04, 2019.
[6] [n.d.]. consistency levels. http://cassandra.apache.org/doc/latest/architecture/dynamo.html . Accessed: July 04, 2019.

[7] [n.d.]. Data Consistency. https://docs.datastax.com/en/archived/cassandra/3.0/cassandra/dml/dmlAboutDataConsistency.html . Accessed: July 04, 2019.
[8] [n.d.]. eventual consistency. https://en.wikipedia.org/wiki/Eventual_consistency . Accessed: July 04, 2019.
[9] [n.d.]. Lightweight Transactions in Cassandra. https://www.datastax.com/dev/blog/lightweight-transactions-in-cassandra-2-0 . Accessed: July 04, 2019.
[10] [n.d.]. Query Optimzation. https://www.scnsoft.com/blog/cassandra-performance . Accessed: July 04, 2019.
[11] [n.d.]. Query Processing. https://www.mahdix.com/blog/2016/06/01/how-apache-cassandra-executes-cql-queries/ . Accessed: July 04, 2019.
[12] [n.d.]. Security. http://cassandra.apache.org/doc/latest/operating/security.html . Accessed: July 04, 2019.
[13] [n.d.]. Transaction properties. https://docs.datastax.com/en/archived/cassandra/2.0/cassandra/dml/dml_about_transactions_c.html . Accessed: July 04, 2019.
[14] 2019. Cassandra Data Model - javatpoint. https://www.javatpoint.com/cassandra-data-model . Accessed: July 04, 2019.
[15] 2019. Cassandra Features - javatpoint. https://www.javatpoint.com/cassandra-features . Accessed: July 04, 2019.
[16] 2019. Cassandra Keyspace Javatpoint - javatpoint. https://www.javatpoint.com/cassandra-create-keyspace . Accessed: July 04, 2019.
[17] Eben Hewitt. 2010. *Cassandra: The Definitive Guide.* O'Reilly Media.