NATIONAL UNIVERSITY OF SINGAPORE

CS4224: DISTRIBUTED DATABASES

APACHE CASSANDRA

DATA MODELING & BENCHMARKING THE PERFORMANCE

SUMITTED BY: TEAM 8

ANUSHA CHOORY BALAJI, A0137777M

PALLAVI SINGH, A0137779J

SAKSHI SAXENA, A0137815B

DESIGNING THE DATA MODEL AND BENCHMARKING THE PERFORMANCE

1. INTRODUCTION

Distributed Databases:

Unlike a RDBMS (Related Database Management System), the Distributed Database Management System (DDBMS) stores and manages data by partitioning the database and thereby stores these partitions across different sites in a computer network. A distributed database is a collection of multiple logically interrelated databases distributed over a computer network [1]. The concept of “Distributed Databases” has been wholly embraced by the technical community in a relatively short period of time because of its abundant flexibility and performance capabilities. Because the entire database entity is distributed in nature, several users can query the database without being perturbed about probable interference from other users or processes. The distributed database management system periodically ensures that scattered data partitions across different sites remain consistent. The distributed databases world is evolving continuously with several flavors of DDBMS flooding the market that support NoSQL, NewSQL and XML database engines [2]. These distributed databases are equipped with robust fault tolerant mechanisms; facilitate high availability and scalability in the system through data replication and transparency of data. Some notable examples of distributed databases are Aerospike, Cassandra, Couchbase, Druid, FoundationDB and OrientDB.

Cassandra:

Cassandra is one such example of a distributed database management system which falls under the open source DDBMS realm. Cassandra was developed at Facebook and it has been sufficiently equipped to handle massive amounts of data across several nodes, while ensuring that the service is available at all times thereby making single point of failure in the database very rare. This supports NoSQL architecture and is considered to be ideal to organize and manage stupendous amount of data across several nodes in the network. Cassandra supports a peer-to-peer technology; hence there is no notion of a master/slave in the network nor is the concept of leader election relevant in this case. On a conceptual level, Cassandra can be represented as a giant hash ring where all the nodes are considered as equal. Data is replicated on “n” servers in the ring and the actual location of the data on the ring is determined by the partition key used for that table. As a result, any given node in the ring is capable of serving the user request. “Replication” in Cassandra ensures that the data is still made available to the user even when a node fails. In Cassandra, data is replicated asynchronously and automatically. The user sets this ‘replication factor’ when he configures the keyspace. The data is always replicated at every server participating in the ‘ring’.

CQL: Cassandra Query Language

The Cassandra Query Language also called CQL is what the users utilize to communicate with the Cassandra database. SQL (Structured Query Language) and CQL (Cassandra Query Language) are very similar to each other. The overall idea of the creation and usage of tables remains the same with some exceptions like support for joins, sub-queries, group by, foreign keys etc.

We make use of the shell cqlsh (CQL shell) to interact with Cassandra. The users are given the flexibility to create keyspaces and tables in the database; they can also execute a plethora of commands like SELECT, INSERT, UPDATE, DESC etc. To ensure a smooth communication between the client and the database, CQL supports various programming language drivers including a one for Java.

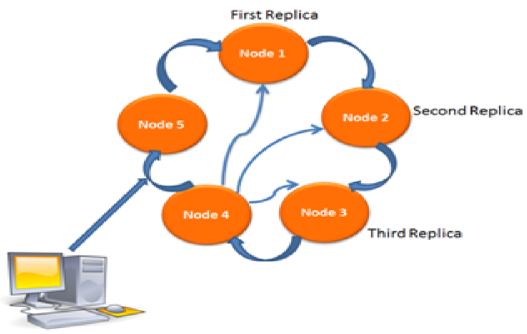


Figure 1: Depicting Replication Factor (RF=3) i.e., 3/5 = 60% of the Nodes in the Cluster have relevant data [3].

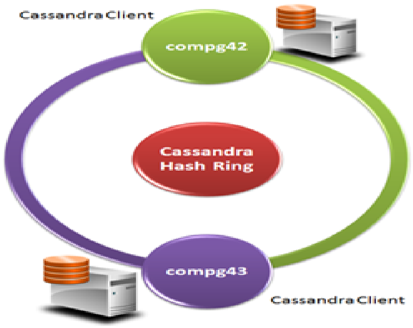


Figure 2: NUS, School of Computing Nodes.

2. DATA MODELS

We developed two data models as a part of this project. We were able to successfully implement **Data Model 1** in Cassandra and were also able to benchmark its performance suitably. The other data model, i.e., **Data Model 2** is our proposal to come up with a database schema which would guarantee a higher throughput.

The data models are provided below:

Data Model 1:

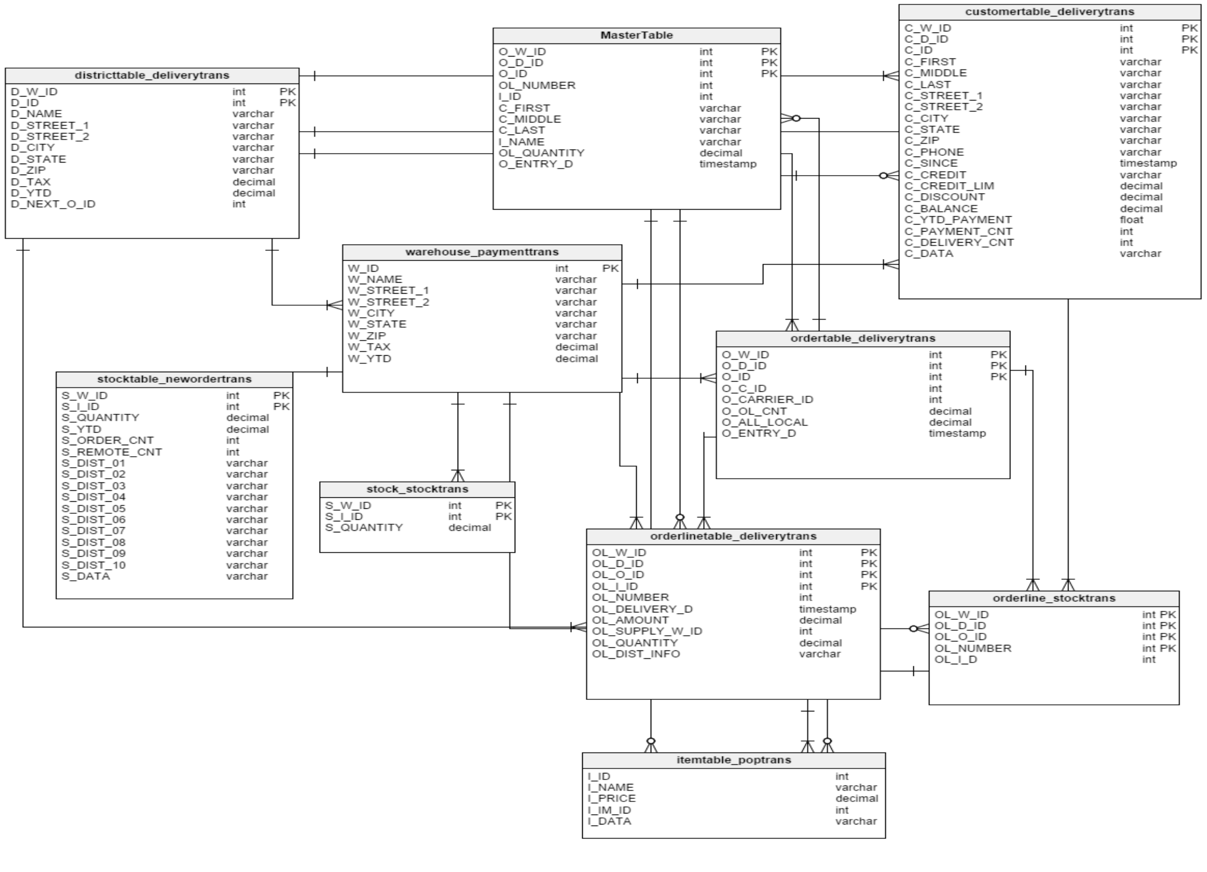


Figure 3: Designing the Data Model - 1

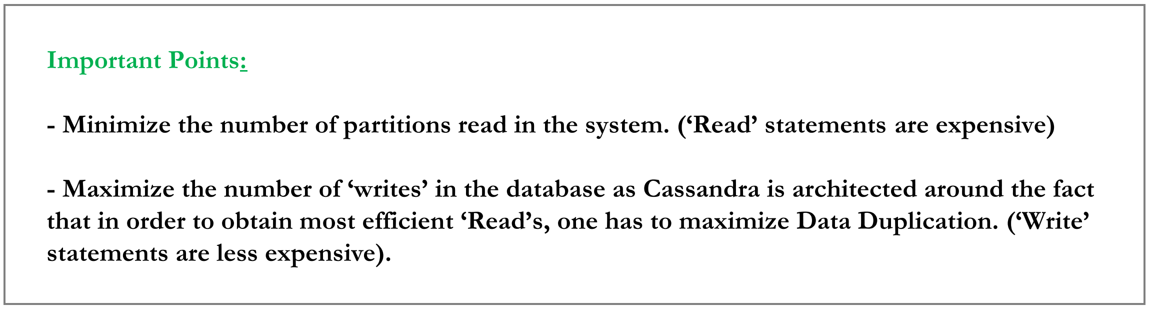
3. JUSTIFICATION

The data model presented above is what we have come up with, to design our distributed database system. We make use of a total of 10 database tables which comprises of all the relevant data which use to query the database. We identify the data access patterns and the type of queries which are executed on the database. Only after such an analysis is performed, we have to make an informed decision concerning the database model architecture. Since the underlying data modeling approach in Cassandra follows a query-driven approach, we ensure that the data is fetched from the tables in an optimal manner without affecting its performance. The tables in the database have been selected in such a way that they currently hold relevant chunks of data and are hence, able to respond to the inbound queries efficiently.

The tables which have been created to cater to the queries/transactions given in the assignment are as follows:

- MASTERTABLE
- DISTRICTTABLE_DELIVERYTRANS
- CUSTOMERTABLE_DELIVERYTRANS
- WAREHOUSE_PAYMENTTRANS
- STOCKTABLE_NEWORDERTRANS
- STOCK_STOCKTRANS
- ORDERTABLE_DELIVERYTRANS
- ORDERLINETABLE_DELIVERYTRANS
- ORDERLINE_STOCKTRANS
- ITEMTABLE_POPTRANS

Each of these tables is being used by one or more given transactions either in entirety or in conjunction with the other tables. By doing so, we restrict the number of tables which have to be queried to obtain a result for the user. Moreover, sufficient care has been also taken to make the updates of the data in its corresponding tables accurately. This underscores the fact that data is being made consistent in the overall database system.



Important Points:

- Minimize the number of partitions read in the system. ('Read' statements are expensive)

- Maximize the number of 'writes' in the database as Cassandra is architected around the fact that in order to obtain most efficient 'Read's, one has to maximize Data Duplication. ('Write' statements are less expensive).

Figure 4: Cassandra – Salient points

In the sections that follow, we will discuss how each of the tables comes into play while querying Cassandra.

4. DEEP DRIVE INTO TRANSACTIONS

1. New Order Transaction:

The ‘New-Order Transaction’ is used to process a new order from a given customer. It takes as input W_ID, D_ID, C_ID, NUM_ITEMS, ITEM_NUMBER[i], SUPPLIER_WAREHOUSE[i] and QUANTITY[i]. It makes use of the below mentioned tables and selects/updates/inserts appropriate data into the other corresponding tables as per the processing logic mentioned. This transaction ultimately creates a ‘new’ Order and a ‘new’ Order-Line Tuple in the database system, thus signifying the creation of an entirely new customer order.

Tables Involved:

DISTRICTTABLE_DELIVERYTRANS,WAREHOUSE_PAYMENTTRANS,
CUSTOMERTABLE_DELIVERYTRANS,ORDERTABLE_DELIVERYTRANS,
STOCKTABLE_NEWORDERTRANS, ORDERLINETABLE_DELIVERYTRANS.

2. Payment Transaction:

The Payment transaction outlines the processing steps which are involved when a payment (for a certain service) is made by a customer. It takes as input C_W_ID, C_D_ID, C_ID and the payment amount i.e., PAYMENT. It updates the corresponding table entries of the warehouse, district and customer based on the predefined logic requirements and also prints out their details on the screen.

Tables Involved:

DISTRICTTABLE_DELIVERYTRANS,WAREHOUSE_PAYMENTTRANS,
CUSTOMERTABLE_DELIVERYTRANS.

3. Delivery Transaction:

The Delivery transaction takes into consideration the processing of data which is related to the oldest order which is yet to be delivered to the customer for the ten districts in a given warehouse. It takes as input W_ID and CARRIER_ID and subsequently processes the minimum order ID by suitably updating the order, order-line and customer tables in the database.

Tables Involved:

CUSTOMERTABLE_DELIVERYTRANS,ORDERTABLE_DELIVERYTRANS,
ORDERLINETABLE_DELIVERYTRANS.

5. Order-Status Transaction

The Order-Status transaction is used to when the status of the last order of the customer is to be found out. It takes as input the customer identifier: C_W_ID, C_D_ID and C_ID. After processing, the customer’s details, order ID details from the customer’s last order and individual item details from the customer’s last order are displayed as output.

Tables Involved:

CUSTOMERTABLE_DELIVERYTRANS,ORDERTABLE_DELIVERYTRANS,
ORDERLINETABLE_DELIVERYTRANS.

6. Stock-Level Transaction

The Stock-Level transaction is used to monitor the last "L" items which were sold at a specified warehouse-district. It reports the number of items that have a stock level quantity below a specified threshold. It takes in as input W_ID, D_ID, Stock Threshold T and Number of last sold items, L and after subsequent processing steps it outputs the total number of items where its stock falls lower than the threshold value "T".

Tables Involved:

DISTRICTTABLE_DELIVERYTRANS,STOCK_STOCKTRANS,
ORDERLINETABLE_DELIVERYTRANS.

7. Popular Item Transaction

The popular item transaction is used to find out the most popular item/items in each of the specified last "L" orders at a given district. It takes in W_ID, D_ID and L as input. The transaction then computes the most popular item based on certain specifications which are mentioned. It then prints out the district, last order information and its corresponding customer details as output.

Tables Involved:

DISTRICTTABLE_DELIVERYTRANS,WAREHOUSE_PAYMENTTRANS,
CUSTOMERTABLE_DELIVERYTRANS,ORDERTABLE_DELIVERYTRANS,
ORDERLINETABLE_DELIVERYTRANS, ITEMTABLE_POPTRANS.

The code and processing logic for all the given transactions is found in wholeSaleDataTransactions.java file.

8. PROPOSED DATA MODEL

Data Model 2:

The data model presented above is another proposal constructed on the basis of the data/ queries given. We had followed the key points mentioned in Figure [4] to derive this architecture.

According to this approach, each of the 6 transactions query only a single database table each (which has to be preprocessed sufficiently so that it retains all the essential table properties). By doing so, they minimize their 'reads' and thus, save up on the overall 'SELECT'/Read costs in the database. Duplication of data is allowed in this kind of an approach as same data would be required by different applications and hence this data could be made to reside in different tables. Sufficient care must be taken to perform updates in such a case. Modeling the Cassandra database by employing such an approach is supposed to guarantee much higher performance.

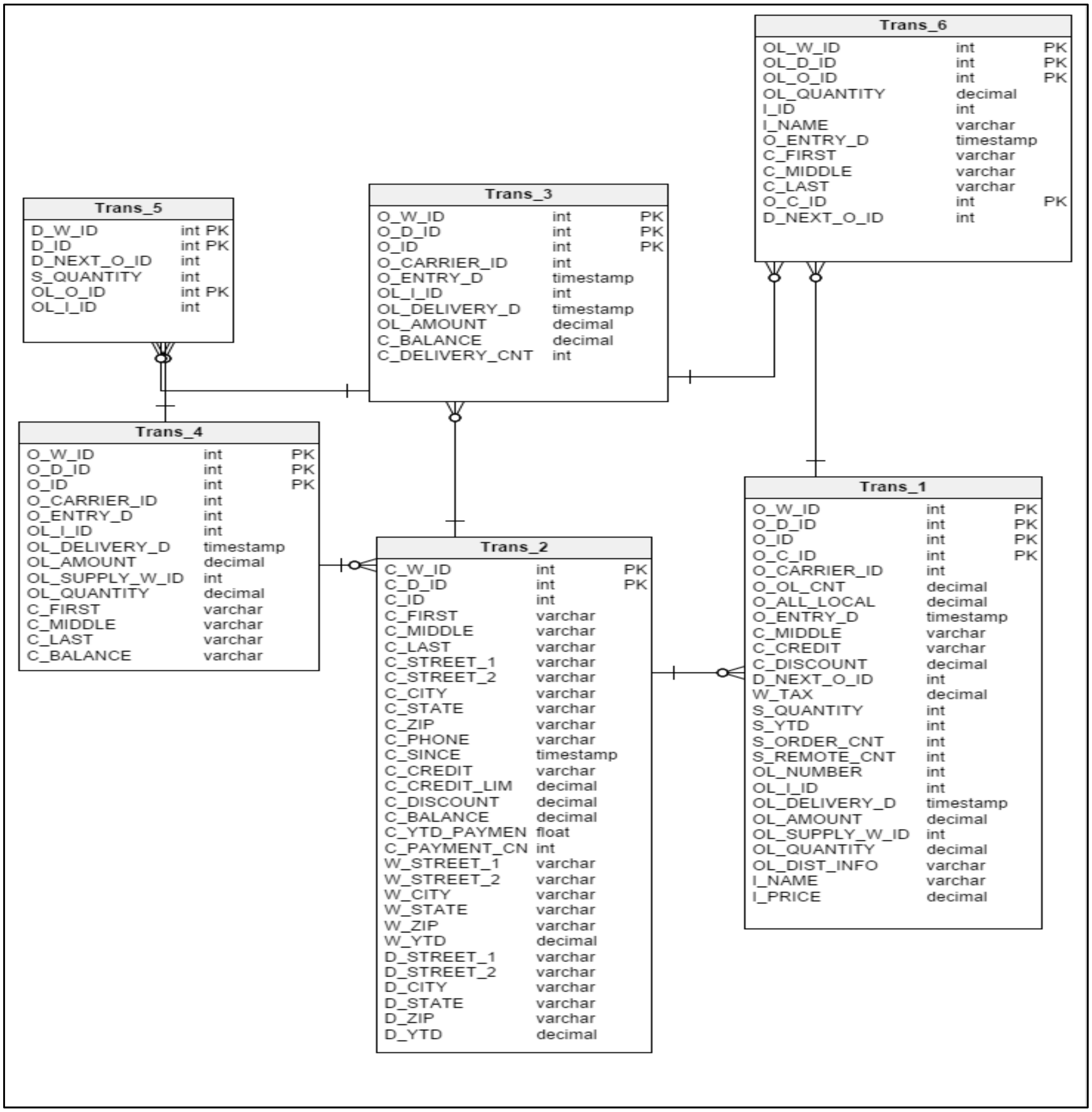


Figure 5: Designing the Data Model - 2

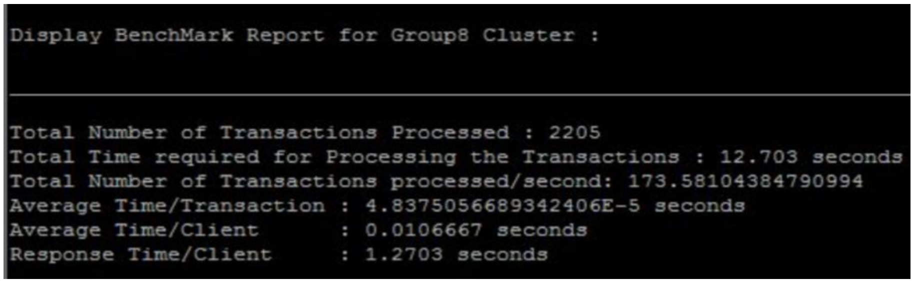
9. RESULTS: PERFORMANCE BENCHMARKING

Experiment 1: Single Node Configuration

Application Instance: (X, Nodes, Z) for X ∈ {D8, D40}, Z = 10, 20, 40, 100, and Nodes = {1, 2}

For (D8, 1, 10): Meaning D8 database, with one node configuration and 10 instances of the program being run. After data modelling and enriching the tables with content, we set these preconditions and run Cassandra for benchmarking its performance for varying loads.

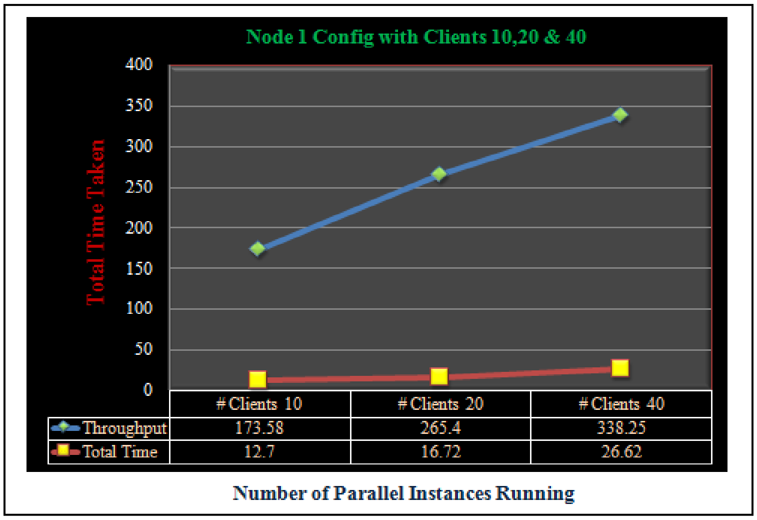
Obtained Output:



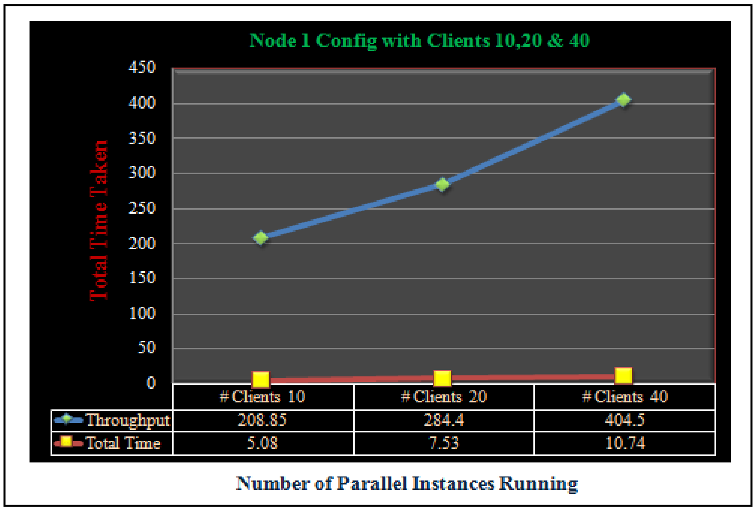
```
Display BenchMark Report for Group8 Cluster :

Total Number of Transactions Processed : 2205
Total Time required for Processing the Transactions : 12.703 seconds
Total Number of Transactions processed/second: 173.58104384790994
Average Time/Transaction : 4.8375056689342406E-5 seconds
Average Time/Client      : 0.0106667 seconds
Response Time/Client     : 1.2703 seconds
```

For (D8, 1, Z):

The outputs for the remaining application sets are computed in a similar way. The graphs depicting the total time taken to process the transactions and its corresponding throughputs are obtained for each load and are plotted for further analysis.



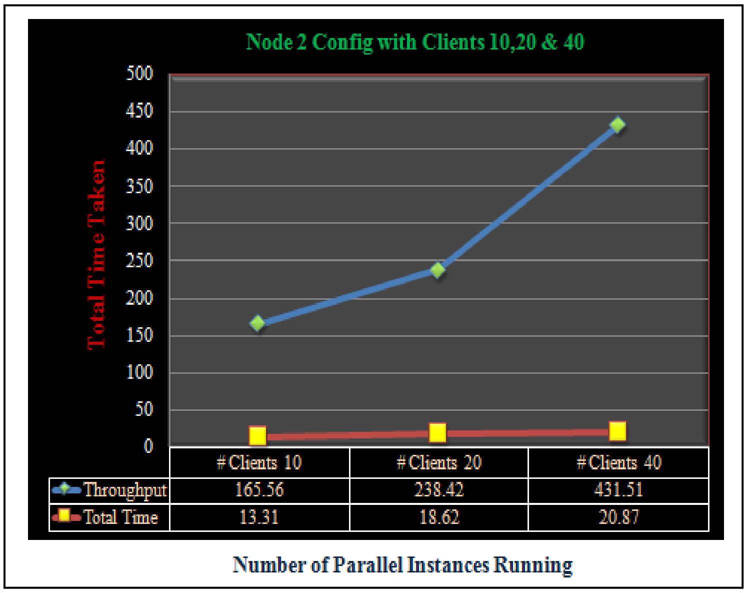
For (D40, 1, Z):



Experiment 2: 2 Nodes Configuration

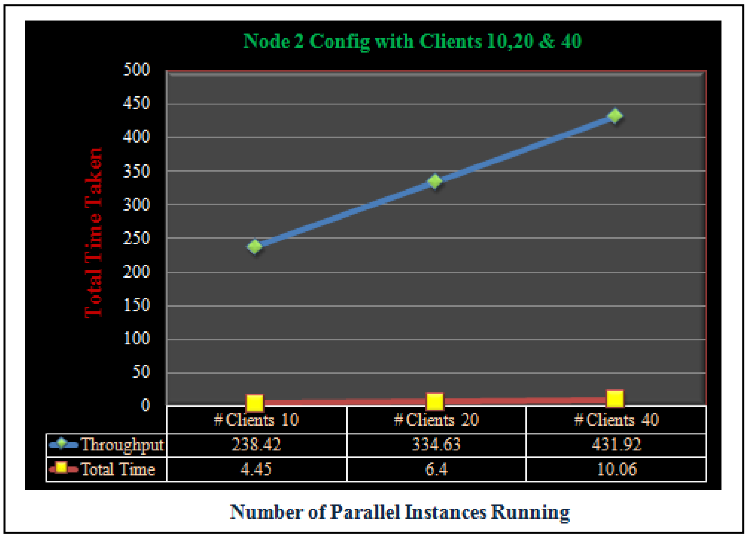
Application Instance: (X, 2, Z) for X ∈ {D8, D40}, Z = 10, 20, 40.

For (D8, 2, Z):



	# Clients 10	# Clients 20	# Clients 40
Throughput	165.56	238.42	431.51
Total Time	13.31	18.62	20.87

For (D40, 2, Z):



	# Clients 10	# Clients 20	# Clients 40
Throughput	238.42	334.63	431.92
Total Time	4.45	6.4	10.06

10. LESSONS LEARNT

Some of the pain points which we had encountered during the project are listed below:

- We experienced several Cassandra timeout issues while running queries on Cassandra.
- Sometimes when running Cassandra on a slow network, we observed that it got disconnected often.

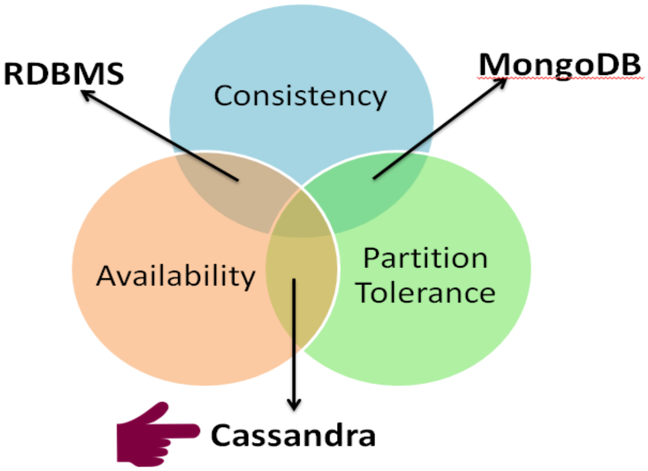
- There was a Cassandra version/ Java conflict for the software version we had downloaded. Because Cassandra 2.2.2 did not support the latest version of Java, we had to resort to using a previous version of JDK to establish the connection.
- The yaml file configuration in Cassandra turned out to be different in various release versions of Cassandra which made configuration a challenging task.
- We had planned to represent the performance metrics of our experiment by establishing a connection to a web based Cassandra output tracker. However, we found it difficult to configure the web based application to support our compgXX nodes.
- If there was a standard benchmarking tool or application which was given to us, it would have been easier to gauge the performance metrics of our model. In addition, all the teams' outputs and benchmarking results would have been consistent.

11. DISCUSSION

Based on the experiments conducted on Cassandra and after a thorough analysis of the traditional and distributed database management systems, we observe that decisions pertaining to the type of database management system (traditional/distributed) to be chosen and for example, what flavor of a distributed database model to pick must be selected on the basis of what the application demands. Even then, neither of the approaches can be declared as ideal and the solutions would only be 'near-perfect', because in the real world there are always trade-offs which cannot be left out of the equation. To give an example, if your database schema has tables with financial and highly sensitive data, it is advisable to use a SQL database which supports ACID properties as finance-related transactional data might require rollback capabilities which Cassandra does not support. During the evaluation of database systems, one should take into the consideration the CAP theorem – wherein the user of the database is allowed to pick only two 'desirable properties' out of the following:

CAP Theorem - C: Consistency A**:** Availability **P:** Partition Tolerance

Cassandra has been fine-tuned so that it remains "Available" at all times and is "Partition Tolerant" but it is only "eventually" Consistent.



Cassandra supports horizontal scalability, which means that if one keeps adding nodes to the existing ring, the performance is bound to increase manifold. This is what we have observed in our experiment with Node count = 1 and Node count = 2. Thus, we see that when the numbers of participating nodes are 2, the throughput increases linearly.

REFERENCES

[1] Distributed Databases http://www.inf.unibz.it/dis/teaching/DDB/ln/ddb01.pdf
[2] Distributed Database https://en.wikipedia.org/wiki/Distributed_database
[3] Datastax Academy: Cassandra https://academy.datastax.com/courses/ds201-cassandra-core-concepts
[4] Cassandra –The Definitive Guide, Eben Hewitt
[5] Cassandra - A Decentralized Structured Storage System, Avinash Lakshman, Prashant Malik