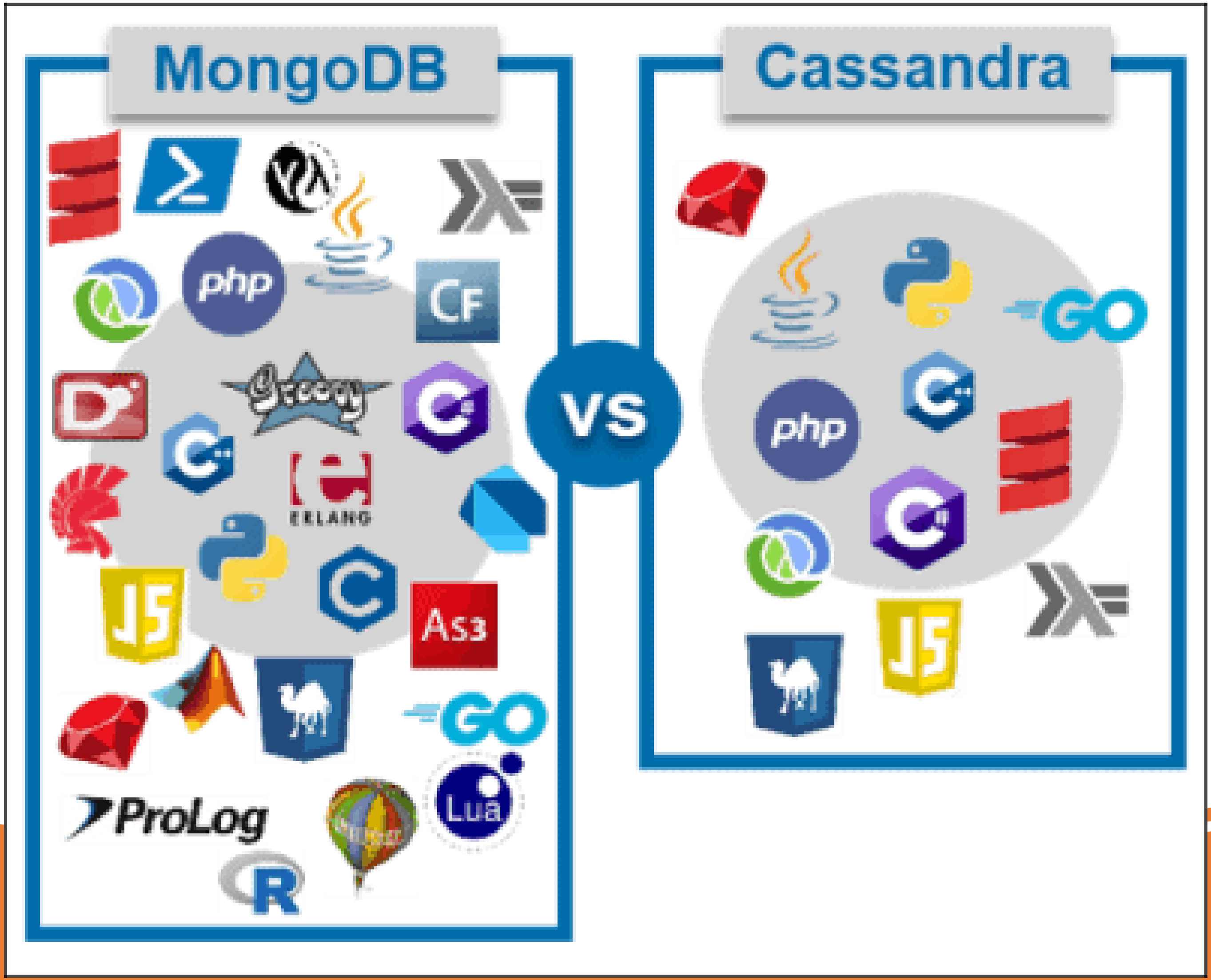
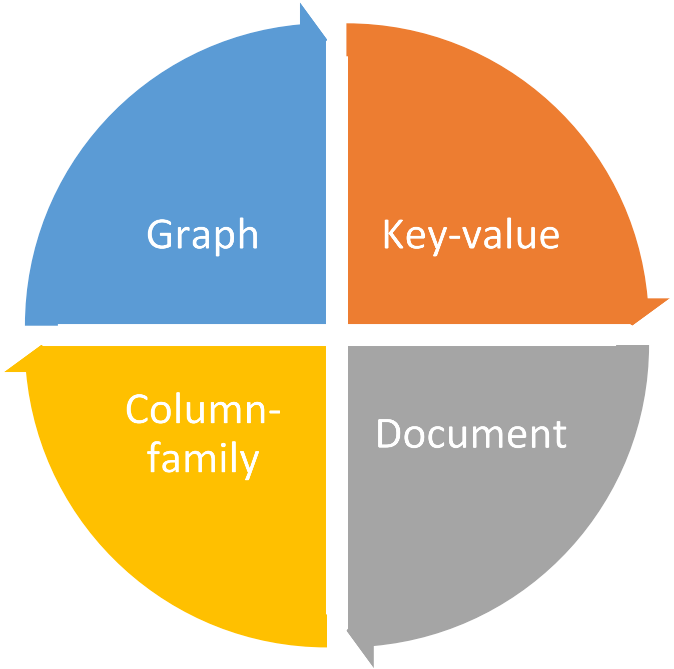
Comparative Performance Analysis between Cassandra and MongoDB in tracking efficiency of NBA Players



Introduction: What is the problem?

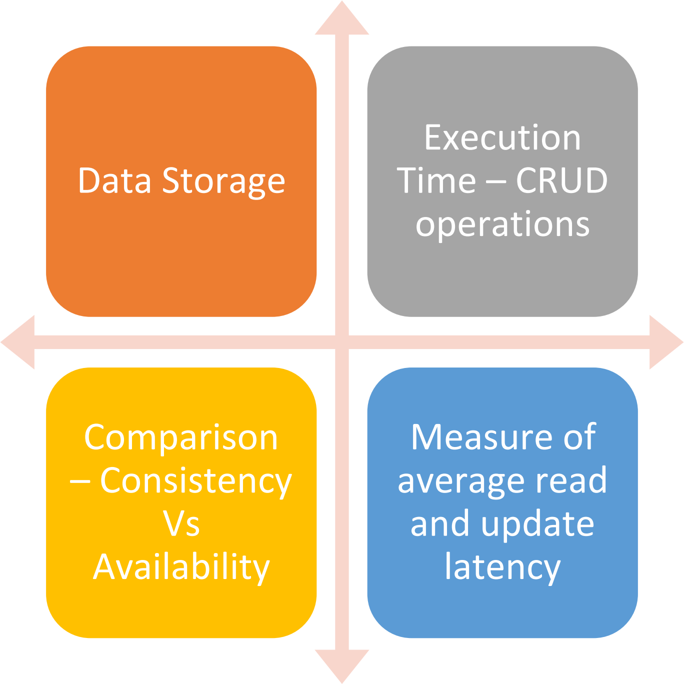
- Finding a right database for a specific workload or an application can be very challenging due to the availability of a huge number of NoSQL databases.
- **Hence, evaluating the performance of NOSQL in-order to select a right database as per the requirement of each application is the need of the hour.**
- Each application communicates with the database using the CRUD operations
- The amount of data, the type of the data to be handled, the run-time for CRUD operations can affect the performance of the NOSQL.
- MongoDB and Cassandra is selected for evaluating their performance in terms of data storage, query handling and run-time for each operation.



Core Categories of NoSQL

Introduction: Scope of Investigation

- ❑The scope of this project is to do an in-depth analysis on the performance of Cassandra and MongoDB as these two databases are widely used in many industrial applications.
- ❑ The dataset consists of performance parameters of NBA players and other game measures.
- ❑The comparison will be done based on the following parameters:



Data Storage

Execution Time – CRUD operations

Comparison – Consistency Vs Availability

Measure of average read and update latency

Theoretical bases & literature review: Our Solution to solve this problem

- The research papers studied showed performance analysis between different databases for synthetic data that was generated using YCSB benchmark.
- Hence, a real-life scenario dataset was selected to measure the efficiency of NBA players to make a performance comparison between MongoDB and Cassandra.
- This will give a better and meaningful insights about the effectiveness of both databases.

Theoretical bases & literature review: Where our solution different from others?

Data Storage

- Analyze which data storage model makes querying easier

CAP

- Understand which database is better option when focusing on consistency over availability or vice versa.

Real-life Scenario dataset

- Simulating a real-scenario dataset

Theoretical bases & literature review: Why our solution is better?

Comparing the time taken to import data in both the databases

Analysis on CAP theorem

Comparing the execution time of READ, UPDATE, DELETE operation on both the databases

Exploring the average read and update latency of both the databases along with throughput rate.

Hypothesis/Goals

To test and verify if MongoDB performs better in read operations and Cassandra performs better in write operations.

- From the research papers studied, it was observed that MongoDB performs better for read and delete operations whereas Cassandra performs better for write operations. We would like to verify if the same results are obtained for our analysis.

To test and verify if Cassandra has low update and read latency as compared to MongoDB, when the load increases.

- As per the research paper, Cassandra has low average read/update latency and high throughput rate when the load increases. This is due to the fact that they have peer-peer replication. On the other hand, MongoDB latency increases exponentially when the load is increased.

Methodology: How to generate/collect input data

Downloading Data

Dataset downloaded from Kaggle website

Upload the data in Pandas for data wrangling, if required

Check the shape and size of the dataset for better understanding

Data Loading in MongoDB

Installing the MongoDB Module and MongoDB compass

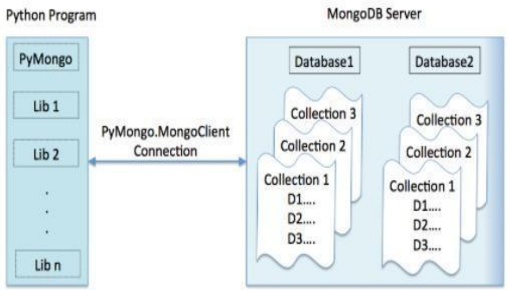
Importing required modules in notebook and connecting to the mongo client i.e. local host and port

Converting the uploaded CSV files to Json format with the help of Python-Pandas

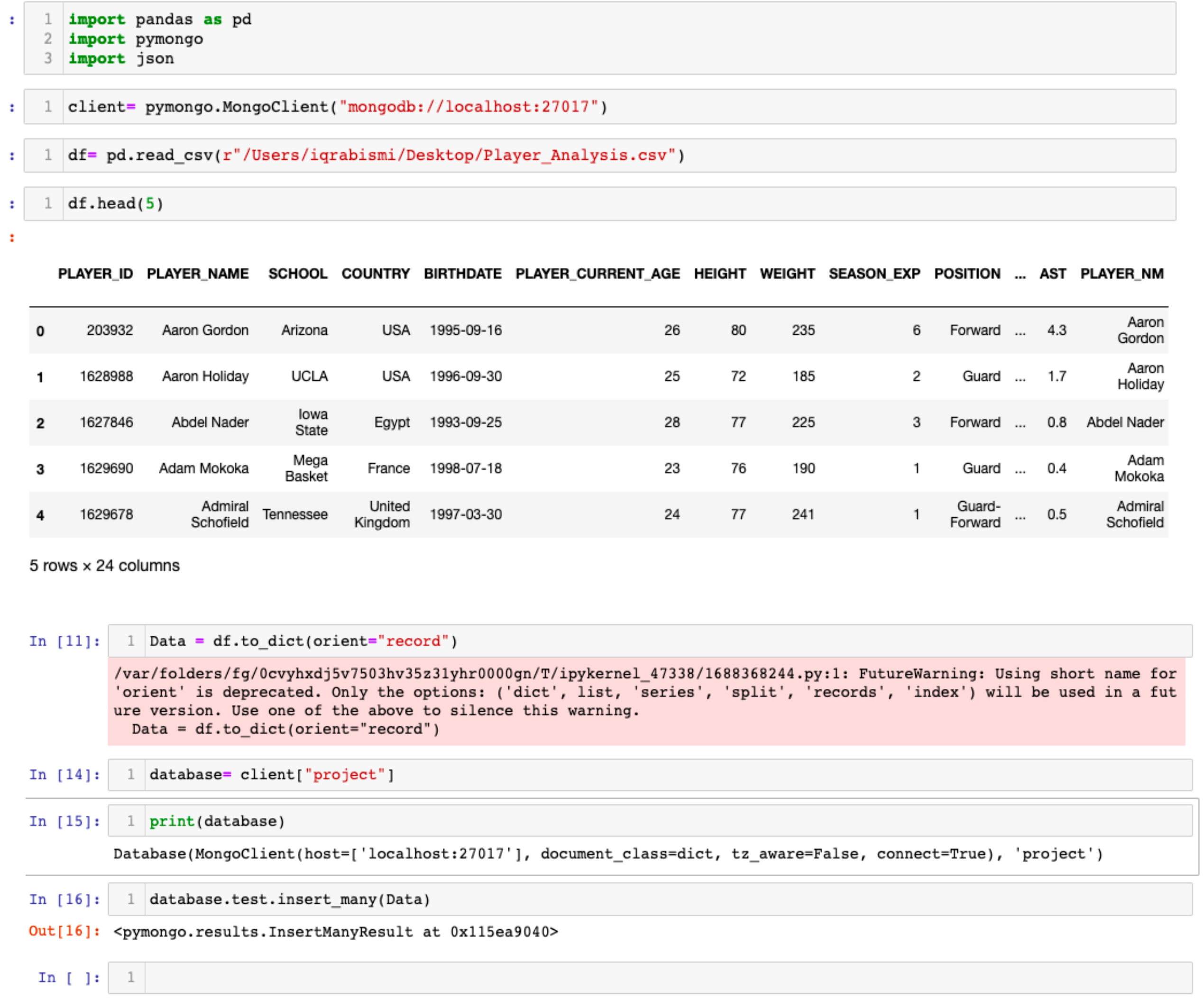
Creating database and collection in the Mongo cluster

Inserting data in the database and collection in MongoDB compass by using command **insert_many()**

Imported data can be viewed using MongoDB Compass and execution time in data loading shall be noted



Data Loading in MongoDB



```
import pandas as pd
import pymongo
import json
```

```
client= pymongo.MongoClient("mongodb://localhost:27017")
```

```
df= pd.read_csv(r"/Users/iqrabismi/Desktop/Player_Analysis.csv")
```

```
df.head(5)
```

	PLAYER_ID	PLAYER_NAME	SCHOOL	COUNTRY	BIRTHDATE	PLAYER_CURRENT_AGE	HEIGHT	WEIGHT	SEASON_EXP	POSITION	...	AST	PLAYER_NM
0	203932	Aaron Gordon	Arizona	USA	1995-09-16	26	80	235	6	Forward	...	4.3	Aaron Gordon
1	1628988	Aaron Holiday	UCLA	USA	1996-09-30	25	72	185	2	Guard	...	1.7	Aaron Holiday
2	1627846	Abdel Nader	Iowa State	Egypt	1993-09-25	28	77	225	3	Forward	...	0.8	Abdel Nader
3	1629690	Adam Mokoka	Mega Basket	France	1998-07-18	23	76	190	1	Guard	...	0.4	Adam Mokoka
4	1629678	Admiral Schofield	Tennessee	United Kingdom	1997-03-30	24	77	241	1	Guard-Forward	...	0.5	Admiral Schofield

5 rows × 24 columns

```
In [11]:  Data = df.to_dict(orient="record")
```

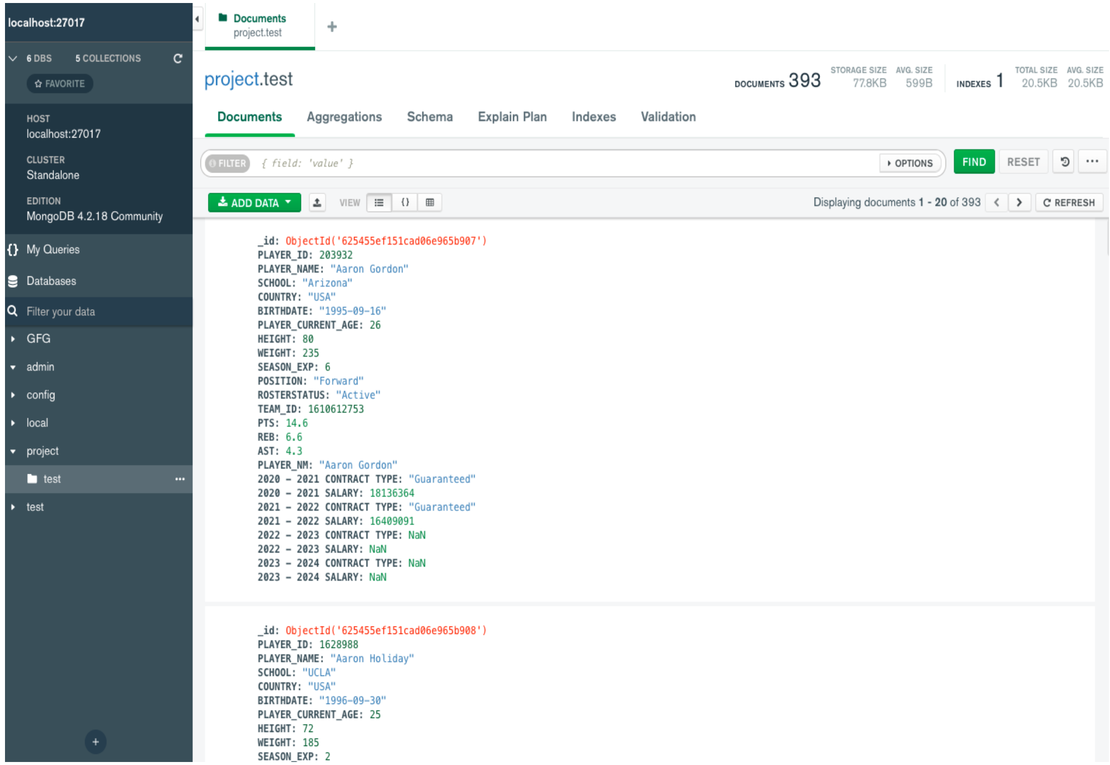
```
/var/folders/fg/0cvyhxdj5v7503hv35z31yhr0000gn/T/ipykernel_47338/1688368244.py:1: FutureWarning: Using short name for
'orient' is deprecated. Only the options: ('dict', list, 'series', 'split', 'records', 'index') will be used in a fut
ure version. Use one of the above to silence this warning.
  Data = df.to_dict(orient="record")
```

```
In [14]:  database= client["project"]
```

```
In [15]:  print(database)

Database(MongoClient(host=['localhost:27017'], document_class=dict, tz_aware=False, connect=True), 'project')
```

```
In [16]:  database.test.insert_many(Data)
Out[16]:  <pymongo.results.InsertManyResult at 0x115ea9040>
```

```
In [ ]:
```



```
_id: ObjectId('625455ef151cad06e965b907')
PLAYER_ID: 203932
PLAYER_NAME: "Aaron Gordon"
SCHOOL: "Arizona"
COUNTRY: "USA"
BIRTHDATE: "1995-09-16"
PLAYER_CURRENT_AGE: 26
HEIGHT: 80
WEIGHT: 235
SEASON_EXP: 6
POSITION: "Forward"
ROSTERSTATUS: "Active"
TEAM_ID: 1610612753
PTS: 14.6
REB: 6.6
AST: 4.3
PLAYER_NM: "Aaron Gordon"
2020 – 2021 CONTRACT TYPE: "Guaranteed"
2020 – 2021 SALARY: 18136364
2021 – 2022 CONTRACT TYPE: "Guaranteed"
2021 – 2022 SALARY: 16409091
2022 – 2023 CONTRACT TYPE: NaN
2022 – 2023 SALARY: NaN
2023 – 2024 CONTRACT TYPE: NaN
2023 – 2024 SALARY: NaN
```

```
_id: ObjectId('625455ef151cad06e965b908')
PLAYER_ID: 1628988
PLAYER_NAME: "Aaron Holiday"
SCHOOL: "UCLA"
COUNTRY: "USA"
BIRTHDATE: "1996-09-30"
PLAYER_CURRENT_AGE: 25
HEIGHT: 72
WEIGHT: 185
SEASON_EXP: 2
```

Methodology: How to generate/collect input data

Data Loading in Cassandra

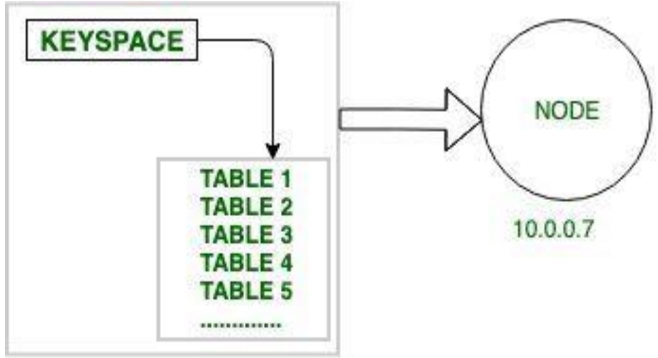
Installation of Cassandra 4.0.3 on docker

Creating Keyspace and column families

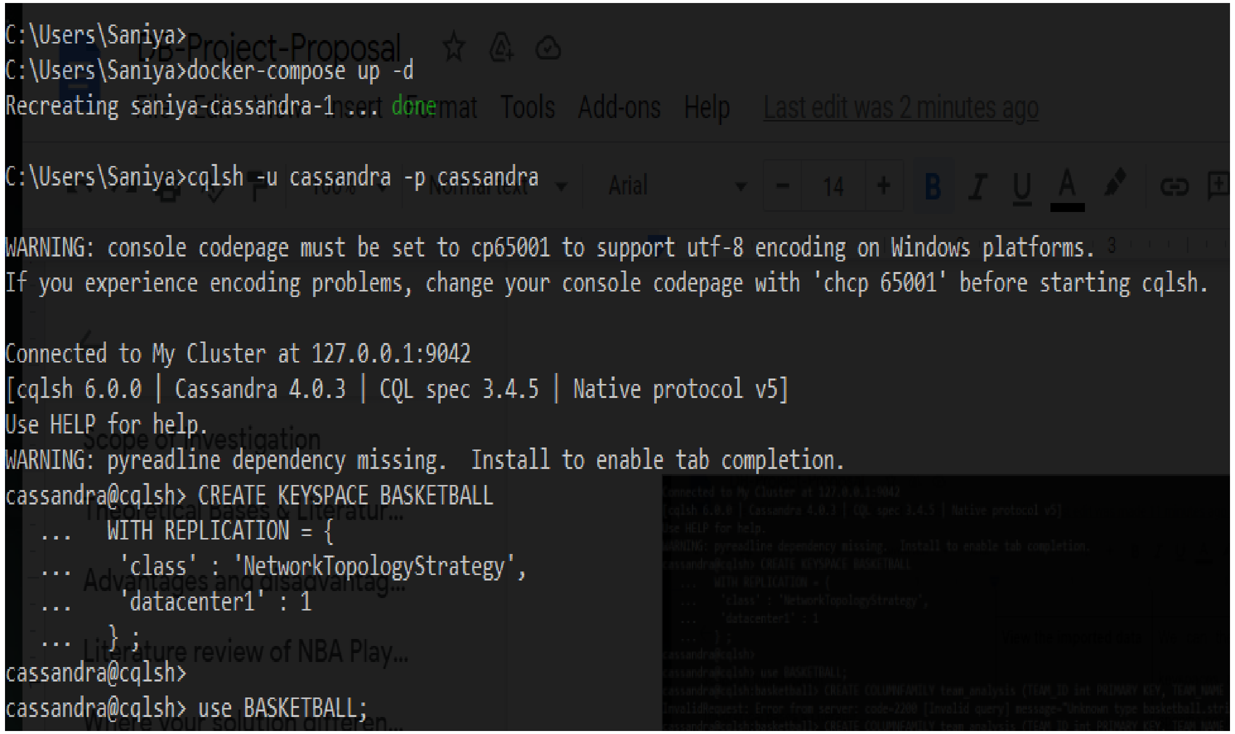
Importing the csv file to the column family using the **COPY** command

Data can be seen from the keyspace and specific column family by using **select *** command

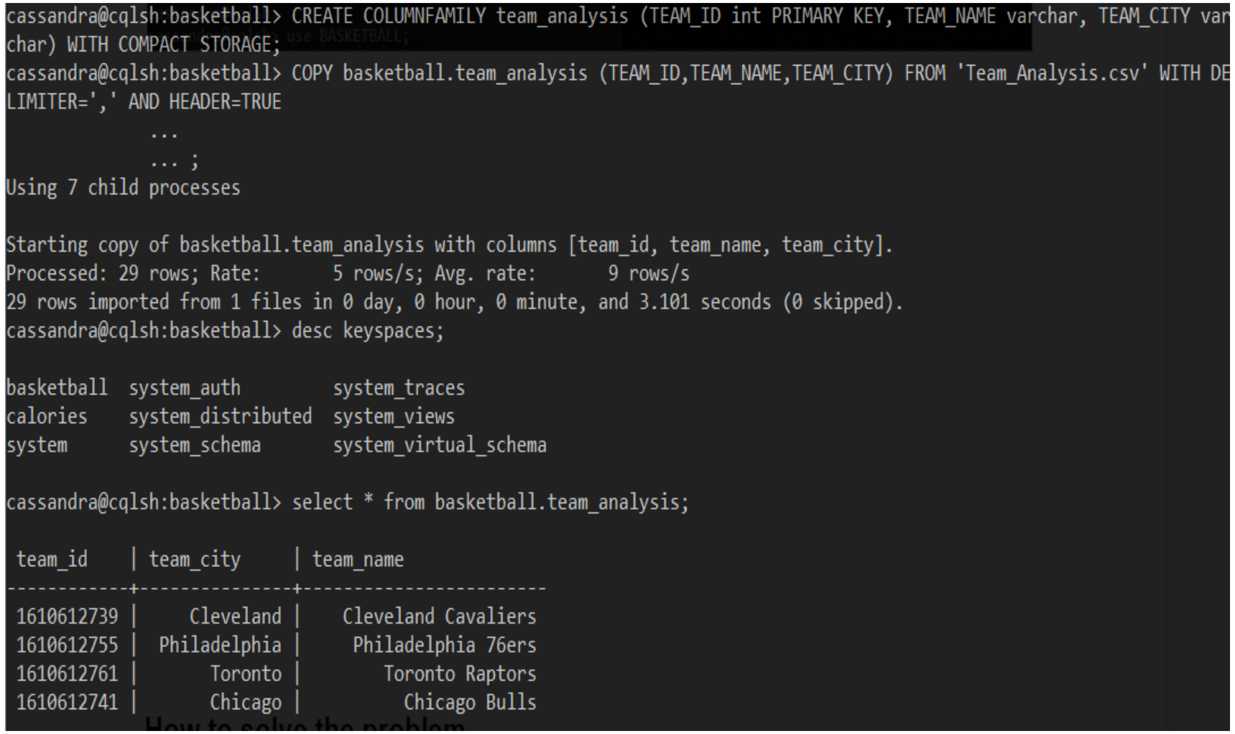
Time taken to load data can be seen in the command prompt



Data Loading in Cassandra

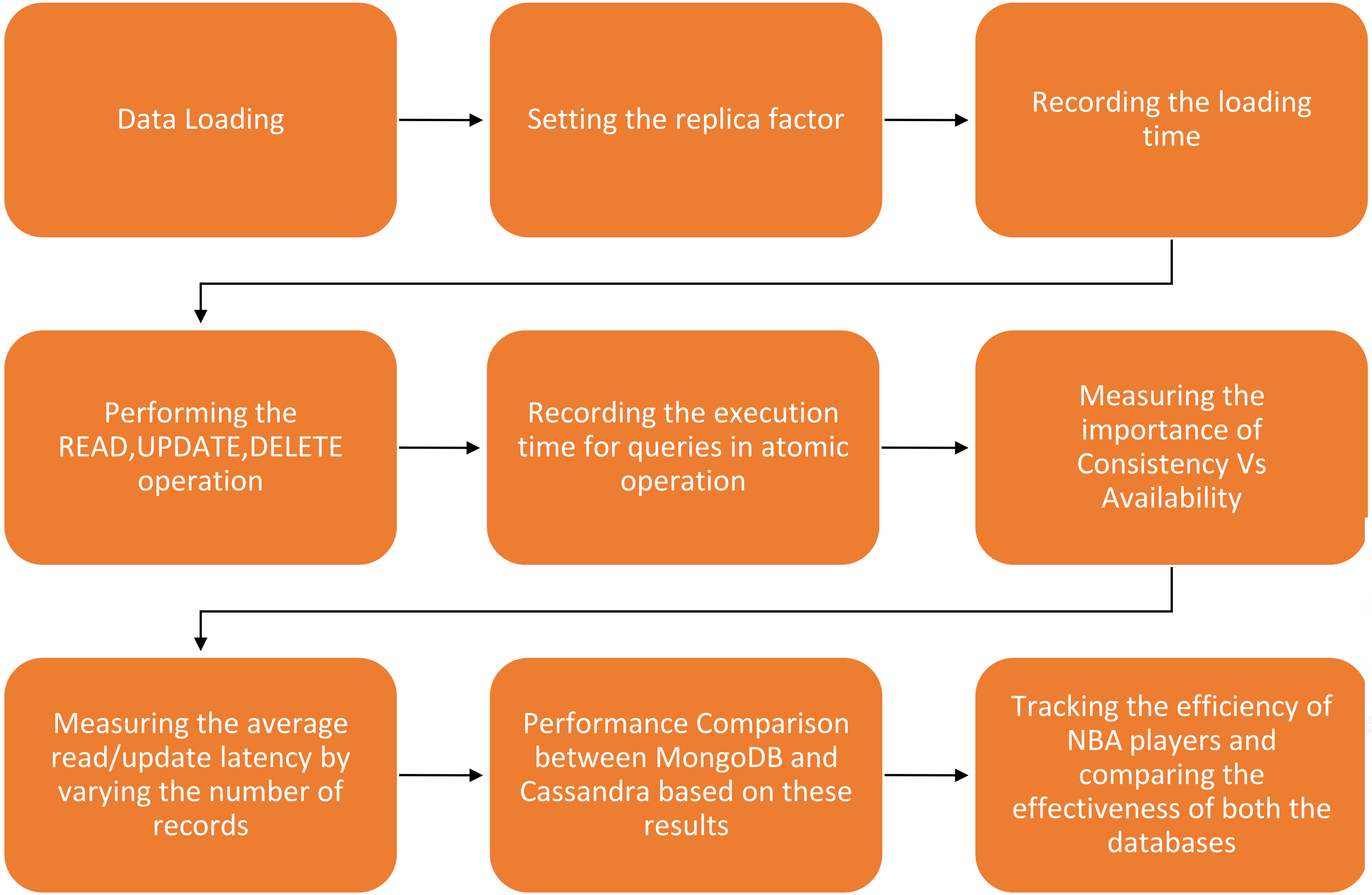


```
C:\Users\Saniya>
C:\Users\Saniya>docker-compose up -d
Recreating saniya-cassandra-1 ... done

C:\Users\Saniya>cqlsh -u cassandra -p cassandra

WARNING: console codepage must be set to cp65001 to support utf-8 encoding on Windows platforms.
If you experience encoding problems, change your console codepage with 'chcp 65001' before starting cqlsh.

Connected to My Cluster at 127.0.0.1:9042
[cqlsh 6.0.0 | Cassandra 4.0.3 | CQL spec 3.4.5 | Native protocol v5]
Use HELP for help.
WARNING: pyreadline dependency missing.  Install to enable tab completion.
cassandra@cqlsh> CREATE KEYSPACE BASKETBALL
   ...   WITH REPLICATION = {
   ...    'class' : 'NetworkTopologyStrategy',
   ...    'datacenter1' : 1
   ...   } ;
cassandra@cqlsh>
cassandra@cqlsh> use BASKETBALL;
```



```
cassandra@cqlsh:basketball> CREATE COLUMNFAMILY team_analysis (TEAM_ID int PRIMARY KEY, TEAM_NAME varchar, TEAM_CITY var
char) WITH COMPACT STORAGE;
cassandra@cqlsh:basketball> COPY basketball.team_analysis (TEAM_ID,TEAM_NAME,TEAM_CITY) FROM 'Team_Analysis.csv' WITH DE
LIMITER=',' AND HEADER=TRUE
              ...
              ... ;
Using 7 child processes

Starting copy of basketball.team_analysis with columns [team_id, team_name, team_city].
Processed: 29 rows; Rate:       5 rows/s; Avg. rate:       9 rows/s
29 rows imported from 1 files in 0 day, 0 hour, 0 minute, and 3.101 seconds (0 skipped).
cassandra@cqlsh:basketball> desc keyspaces;

basketball  system_auth         system_traces
calories    system_distributed  system_views
system      system_schema       system_virtual_schema

cassandra@cqlsh:basketball> select * from basketball.team_analysis;

 team_id    | team_city     | team_name
------------+---------------+-----------------------
 1610612739 |     Cleveland |    Cleveland Cavaliers
 1610612755 |  Philadelphia |     Philadelphia 76ers
 1610612761 |       Toronto |        Toronto Raptors
 1610612741 |       Chicago |          Chicago Bulls
```

Methodology: Algorithm Design



Data Loading

Setting the replica factor

Recording the loading time

Performing the READ,UPDATE,DELETE operation

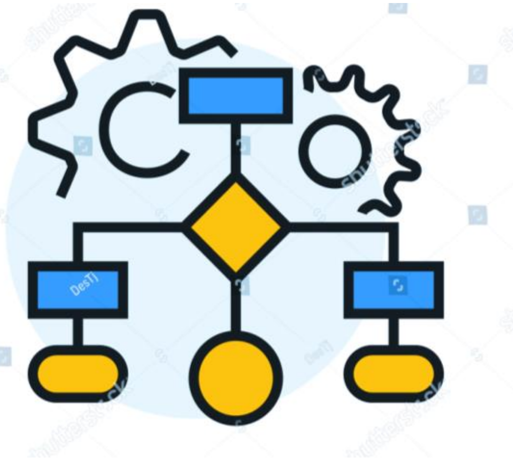
Recording the execution time for queries in atomic operation

Measuring the importance of Consistency Vs Availability

Measuring the average read/update latency by varying the number of records

Performance Comparison between MongoDB and Cassandra based on these results

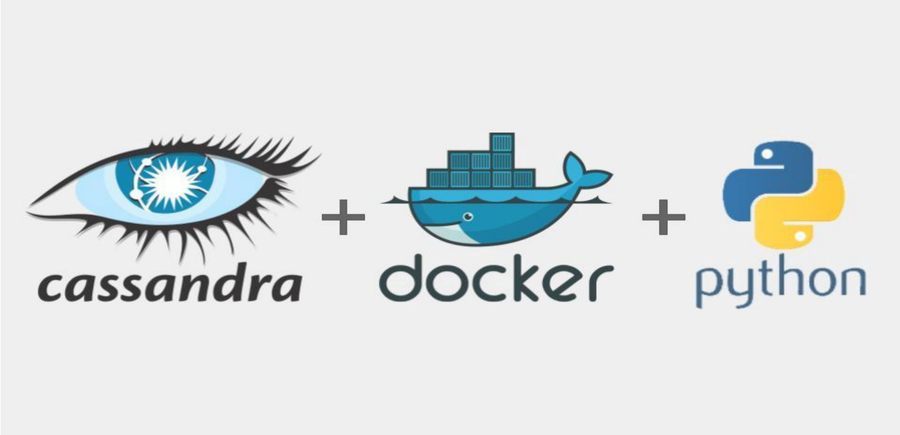
Tracking the efficiency of NBA players and comparing the effectiveness of both the databases



Methodology: Language Used

Language	Purpose
CQL	For querying in Cassandra
Python - Pandas	For ETL and data wrangling
PyMongo	For querying and inserting data in MongoDB

PyMongo is a Python distribution containing tools for working with MongoDB, and is the recommended way to work with MongoDB from Python

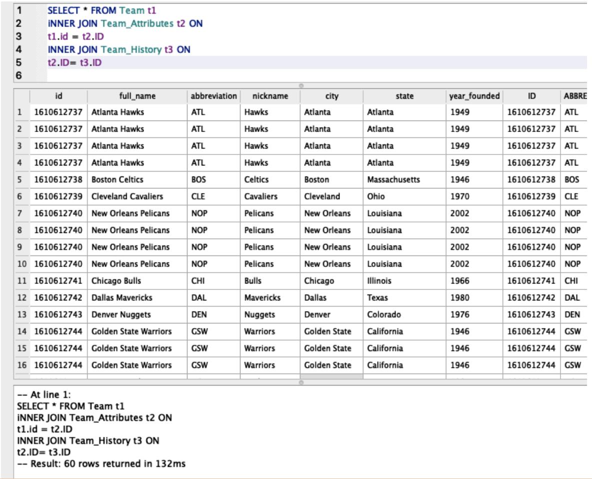


Methodology: Tools Used

Tools	Purpose
Jupyter notebook	For converting csv file to json format, data visualization and data cleaning
MongoDB Compass	For managing collection and document. Also, for querying, aggregating, and analyzing MongoDB data in a visual environment
MongoDB Atlas	To access the data virtually in the cloud, we will establish a connection to the cloud using MongoDB Atlas
Docker	Docker Image is one of the methods of installing Cassandra
Cassandra	For data storing and data analysis

Output Generation

SQLite DB browser

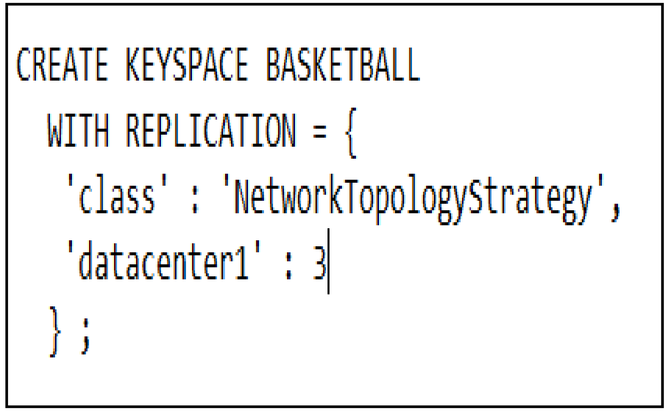


```
SELECT * FROM Team t1
iNNER JOIN Team_Attributes t2 ON
t1.id = t2.ID
INNER JOIN Team_History t3 ON
t2.ID= t3.ID
```

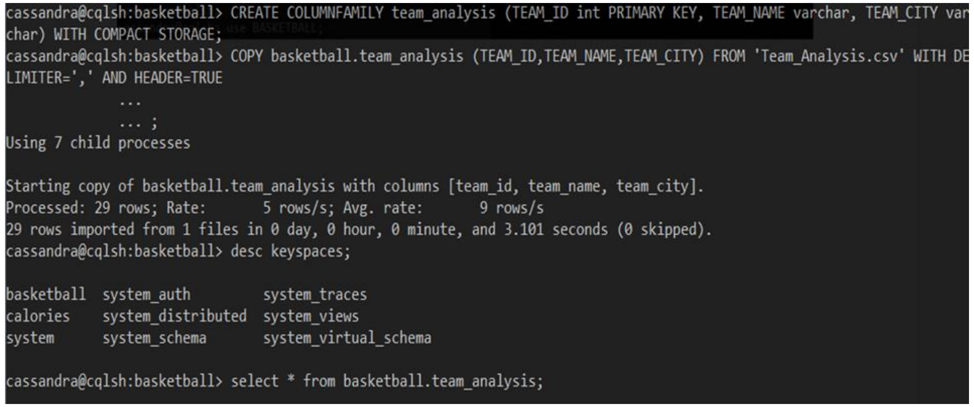
	id	full_name	abbreviation	nickname	city	state	year_founded	ID	ABBRE
1	1610612737	Atlanta Hawks	ATL	Hawks	Atlanta	Atlanta	1949	1610612737	ATL
2	1610612737	Atlanta Hawks	ATL	Hawks	Atlanta	Atlanta	1949	1610612737	ATL
3	1610612737	Atlanta Hawks	ATL	Hawks	Atlanta	Atlanta	1949	1610612737	ATL
4	1610612737	Atlanta Hawks	ATL	Hawks	Atlanta	Atlanta	1949	1610612737	ATL
5	1610612738	Boston Celtics	BOS	Celtics	Boston	Massachusetts	1946	1610612738	BOS
6	1610612739	Cleveland Cavaliers	CLE	Cavaliers	Cleveland	Ohio	1970	1610612739	CLE
7	1610612740	New Orleans Pelicans	NOP	Pelicans	New Orleans	Louisiana	2002	1610612740	NOP
8	1610612740	New Orleans Pelicans	NOP	Pelicans	New Orleans	Louisiana	2002	1610612740	NOP
9	1610612740	New Orleans Pelicans	NOP	Pelicans	New Orleans	Louisiana	2002	1610612740	NOP
10	1610612740	New Orleans Pelicans	NOP	Pelicans	New Orleans	Louisiana	2002	1610612740	NOP
11	1610612741	Chicago Bulls	CHI	Bulls	Chicago	Illinois	1966	1610612741	CHI
12	1610612742	Dallas Mavericks	DAL	Mavericks	Dallas	Texas	1980	1610612742	DAL
13	1610612743	Denver Nuggets	DEN	Nuggets	Denver	Colorado	1976	1610612743	DEN
14	1610612744	Golden State Warriors	GSW	Warriors	Golden State	California	1946	1610612744	GSW
15	1610612744	Golden State Warriors	GSW	Warriors	Golden State	California	1946	1610612744	GSW
16	1610612744	Golden State Warriors	GSW	Warriors	Golden State	California	1946	1610612744	GSW

```
-- At line 1:
SELECT * FROM Team t1
iNNER JOIN Team_Attributes t2 ON
t1.id = t2.ID
INNER JOIN Team_History t3 ON
t2.ID= t3.ID
-- Result: 60 rows returned in 132ms
```

Cassandra

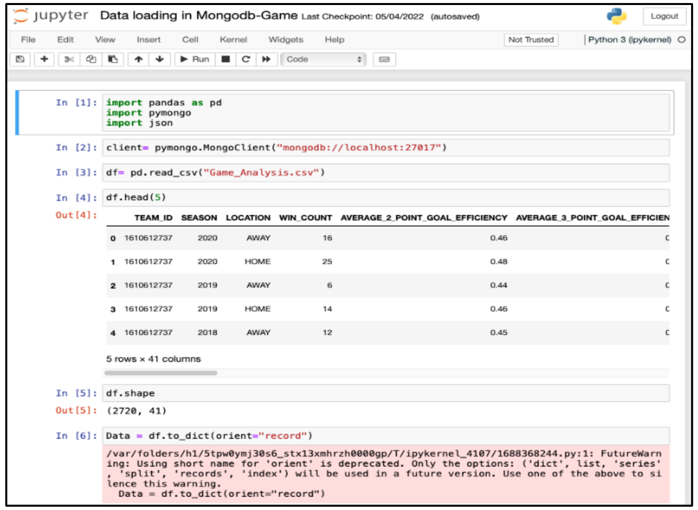


```
CREATE KEYSPACE BASKETBALL
  WITH REPLICATION = {
   'class' : 'NetworkTopologyStrategy',
   'datacenter1' : 3
  } ;
```

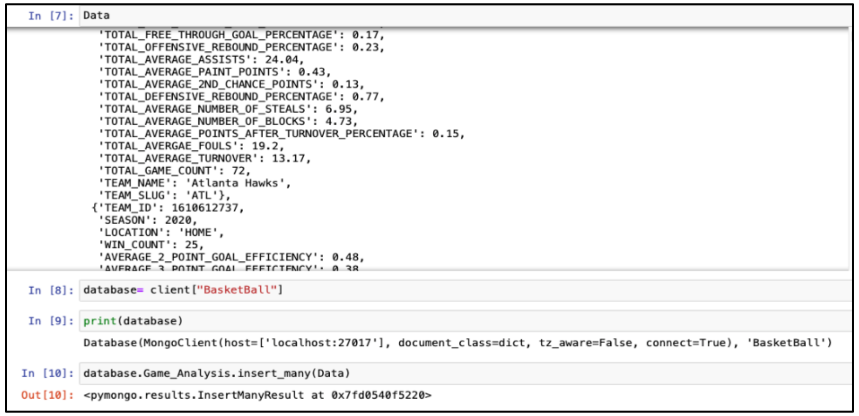


```
cassandra@cqlsh:basketball> CREATE COLUMNFAMILY team_analysis (TEAM_ID int PRIMARY KEY, TEAM_NAME varchar, TEAM_CITY var
char) WITH COMPACT STORAGE;
cassandra@cqlsh:basketball> COPY basketball.team_analysis (TEAM_ID,TEAM_NAME,TEAM_CITY) FROM 'Team_Analysis.csv' WITH DE
LIMITER=',' AND HEADER=TRUE
              ...
              ... ;
Using 7 child processes

Starting copy of basketball.team_analysis with columns [team_id, team_name, team_city].
Processed: 29 rows; Rate:       5 rows/s; Avg. rate:       9 rows/s
29 rows imported from 1 files in 0 day, 0 hour, 0 minute, and 3.101 seconds (0 skipped).
cassandra@cqlsh:basketball> desc keyspaces;

basketball  system_auth         system_traces
calories    system_distributed  system_views
system      system_schema       system_virtual_schema

cassandra@cqlsh:basketball> select * from basketball.team_analysis;
```

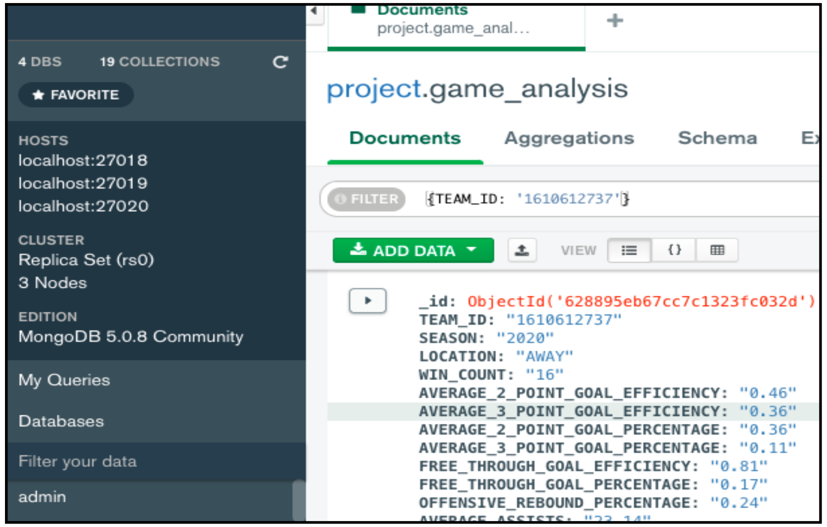
MongoDB



```
In [1]: import pandas as pd
        import pymongo
        import json

In [2]: client= pymongo.MongoClient("mongodb://localhost:27017")

In [3]: df= pd.read_csv("Game_Analysis.csv")

In [4]: df.head(5)
Out[4]:
      TEAM_ID  SEASON  LOCATION  WIN_COUNT  AVERAGE_2_POINT_GOAL_EFFICIENCY  AVERAGE_3_POINT_GOAL_EFFICIEN
0  1610612737    2020      AWAY         16                             0.46                               C
1  1610612737    2020      HOME         25                             0.48                               C
2  1610612737    2019      AWAY          6                             0.44                               C
3  1610612737    2019      HOME         14                             0.46                               C
4  1610612737    2018      AWAY         12                             0.45                               C

5 rows × 41 columns

In [5]: df.shape
Out[5]: (2720, 41)

In [6]: Data = df.to_dict(orient="record")
        /var/folders/h1/5tpw0ymj30s6_stx13xmhrzh0000gp/T/ipykernel_4107/1688368244.py:1: FutureWarn
        ing: Using short name for 'orient' is deprecated. Only the options: ('dict', list, 'series'
        , 'split', 'records', 'index') will be used in a future version. Use one of the above to si
        lence this warning.
          Data = df.to_dict(orient="record")
```



```
In [7]: Data
          'TOTAL_FREE_THROUGH_GOAL_PERCENTAGE': 0.17,
          'TOTAL_OFFENSIVE_REBOUND_PERCENTAGE': 0.23,
          'TOTAL_AVERAGE_ASSISTS': 24.04,
          'TOTAL_AVERAGE_PAINT_POINTS': 0.43,
          'TOTAL_AVERAGE_2ND_CHANCE_POINTS': 0.13,
          'TOTAL_DEFENSIVE_REBOUND_PERCENTAGE': 0.77,
          'TOTAL_AVERAGE_NUMBER_OF_STEALS': 6.95,
          'TOTAL_AVERAGE_NUMBER_OF_BLOCKS': 4.73,
          'TOTAL_AVERAGE_POINTS_AFTER_TURNOVER_PERCENTAGE': 0.15,
          'TOTAL_AVERGAE_FOULS': 19.2,
          'TOTAL_AVERAGE_TURNOVER': 13.17,
          'TOTAL_GAME_COUNT': 72,
          'TEAM_NAME': 'Atlanta Hawks',
          'TEAM_SLUG': 'ATL'},
         {'TEAM_ID': 1610612737,
          'SEASON': 2020,
          'LOCATION': 'HOME',
          'WIN_COUNT': 25,
          'AVERAGE_2_POINT_GOAL_EFFICIENCY': 0.48,
          'AVERAGE_3_POINT_GOAL_EFFICIENCY': 0.38

In [8]: database= client["BasketBall"]

In [9]: print(database)
        Database(MongoClient(host=['localhost:27017'], document_class=dict, tz_aware=False, connect=True), 'BasketBall')

In [10]: database.Game_Analysis.insert_many(Data)
Out[10]: <pymongo.results.InsertManyResult at 0x7fd0540f5220>
```



{TEAM_ID: '1610612737'}

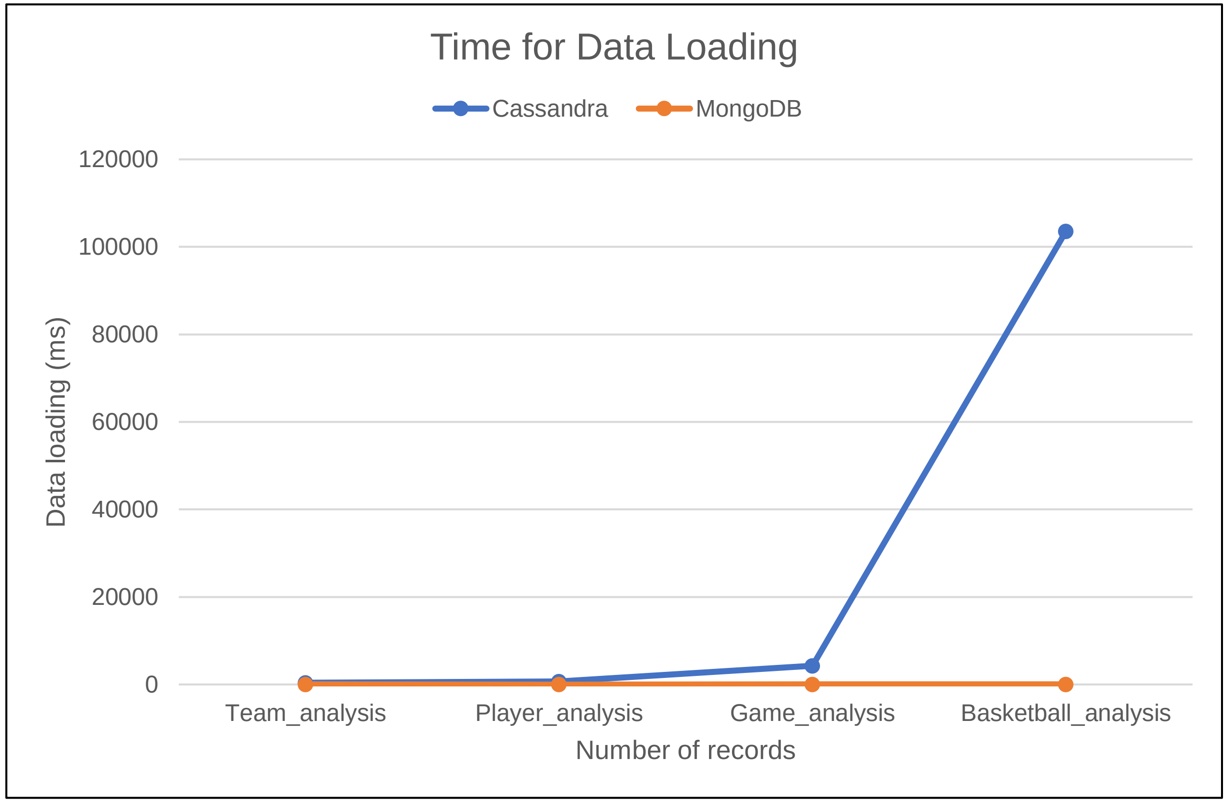
```
_id: ObjectId('628895eb67cc7c1323fc032d')
TEAM_ID: "1610612737"
SEASON: "2020"
LOCATION: "AWAY"
WIN_COUNT: "16"
AVERAGE_2_POINT_GOAL_EFFICIENCY: "0.46"
AVERAGE_3_POINT_GOAL_EFFICIENCY: "0.36"
AVERAGE_2_POINT_GOAL_PERCENTAGE: "0.36"
AVERAGE_3_POINT_GOAL_PERCENTAGE: "0.11"
FREE_THROUGH_GOAL_EFFICIENCY: "0.81"
FREE_THROUGH_GOAL_PERCENTAGE: "0.17"
OFFENSIVE_REBOUND_PERCENTAGE: "0.24"
AVERAGE_ASSISTS: "23.14"
```

Output Analysis & Discussion

Comparison of data loading time

- The execution time required for loading data for both the databases

Data Loading	Number of Records	Cassandra (ms)	MongoDB (ms)
Team_analysis	29	398	1
Player_analysis	394	659	0
Game_analysis	2720	4275	4
Basketball_analysis	34567	103549	26

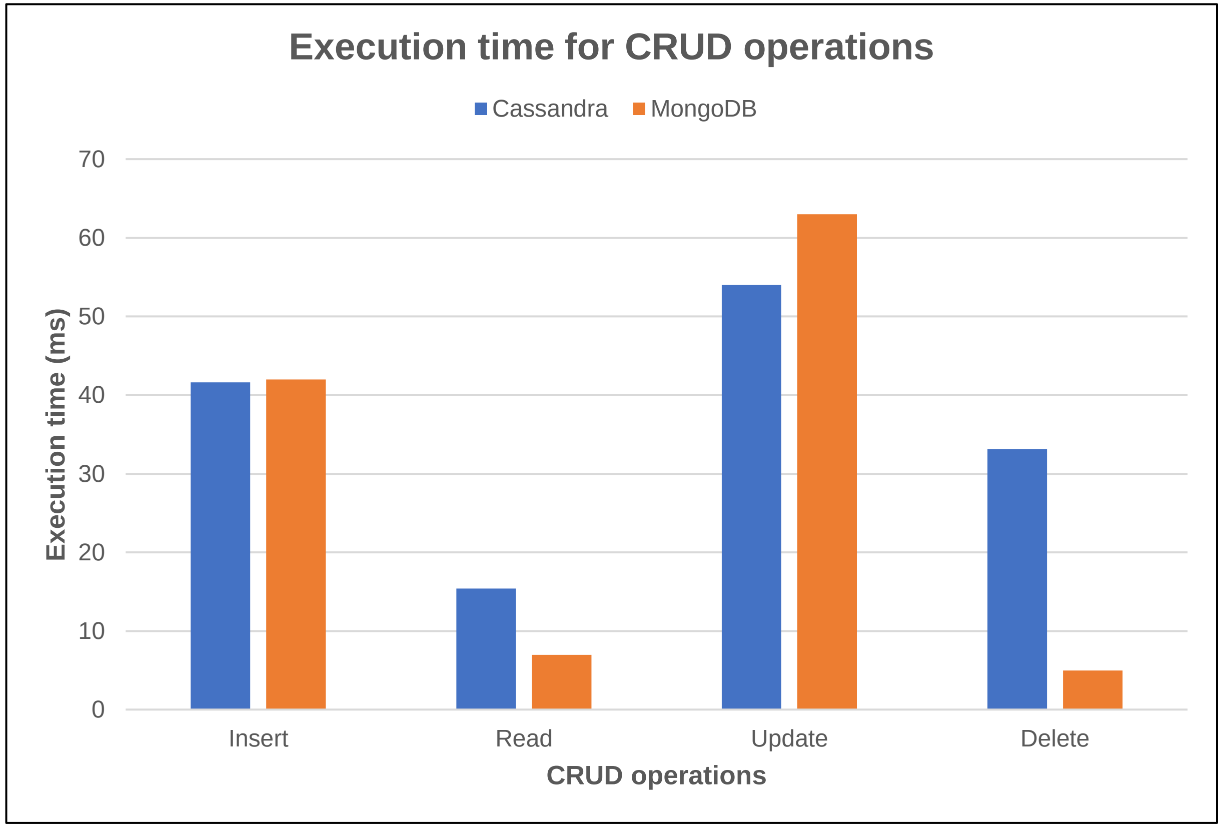


Output Analysis & Discussion

Comparison of execution time for various CRUD operations

- The time taken for various CRUD operations for both the databases

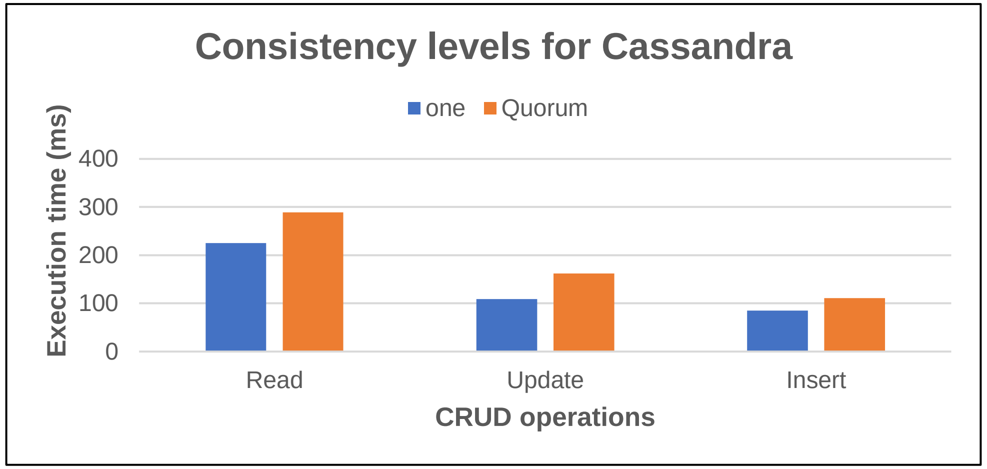
CRUD Operations	Cassandra (ms)	MongoDB (ms)
Insert	41.616	42
Read	15.403	7
Update	54.020	63
Delete	33.122	5

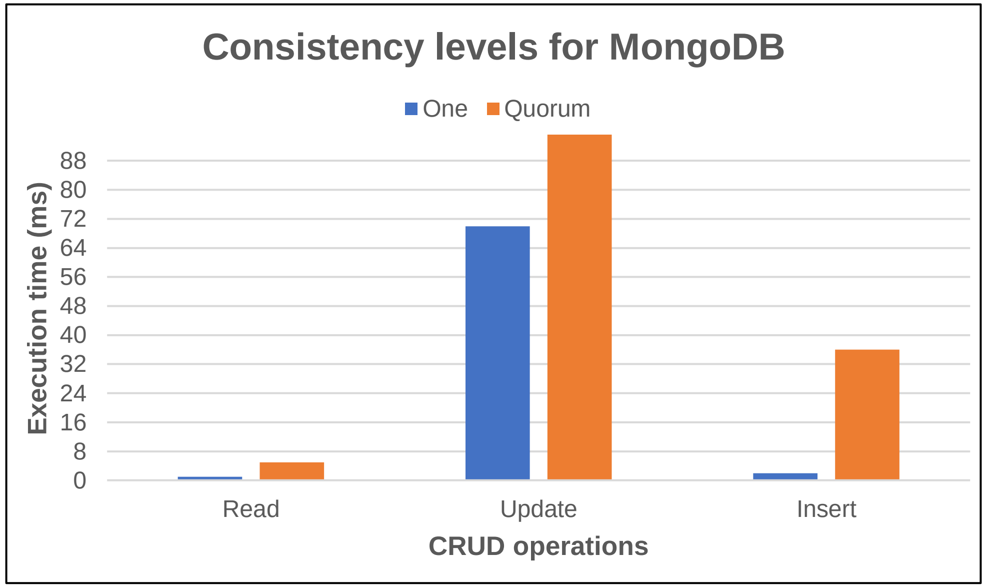


Output Analysis & Discussion

Comparison of execution time for different consistency levels

CRUD	Consistency levels	Cassandra (ms)	MongoDB (ms)
Read	One	225.147	1
	Quorum	228.763	5
Update	One	108.951	70
	Quorum	162.396	556
Insert	One	85.303	2
	Quorum	111.234	36

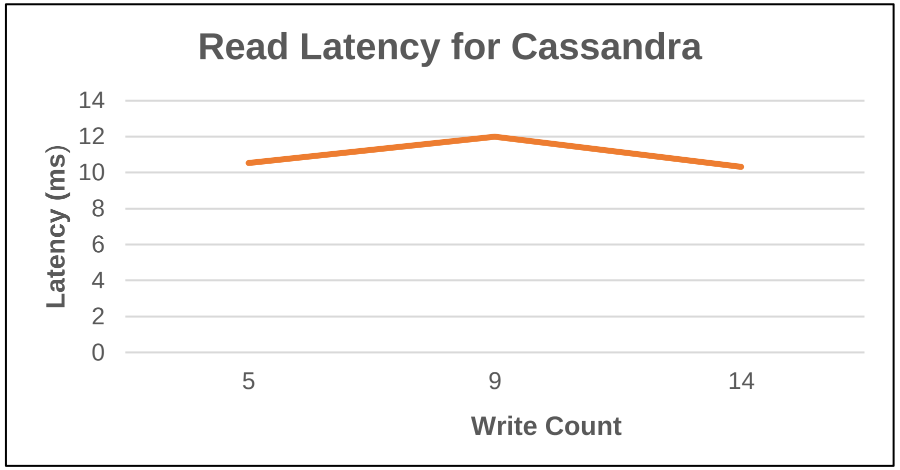


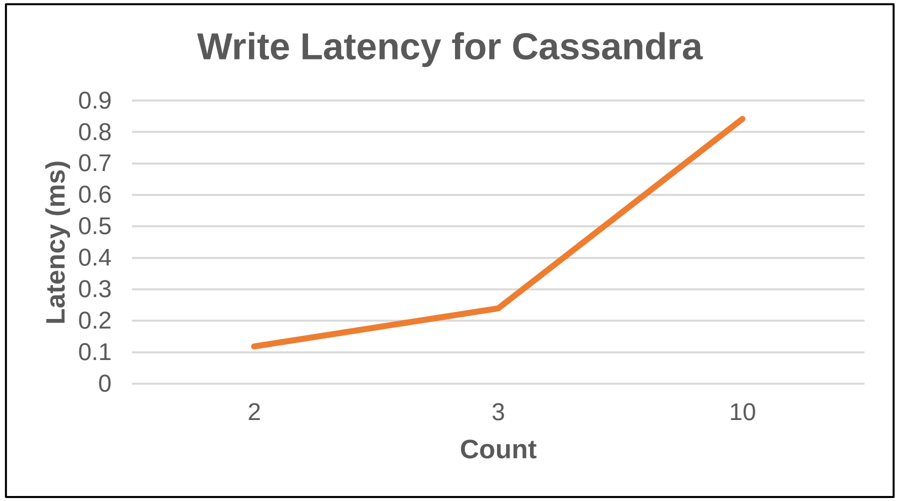


Output Analysis

Cassandra: Comparison of Read and Write latency

Read Operation		Write Operation	
Count	Time Taken (ms)	Count	Time Taken (ms)
9	11.994	2	.0118
5	10.532	3	0.239
14	10.311	10	0.8415

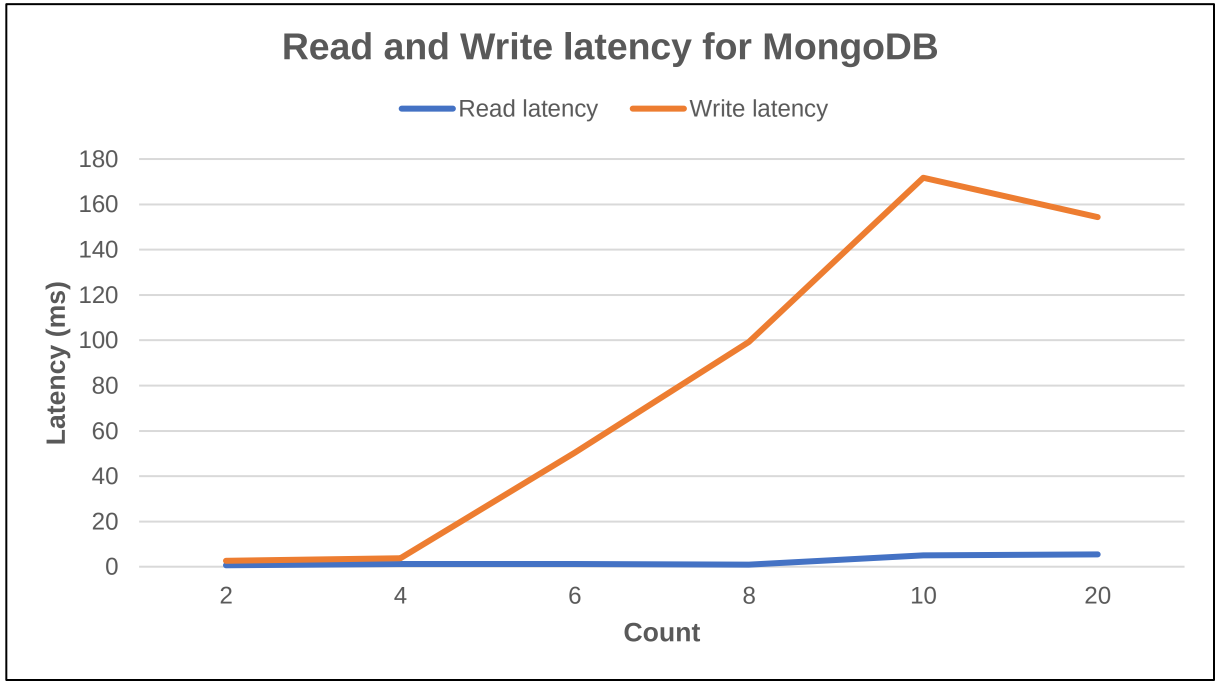




Output Analysis

MongoDB: Comparison of Read and Write latency

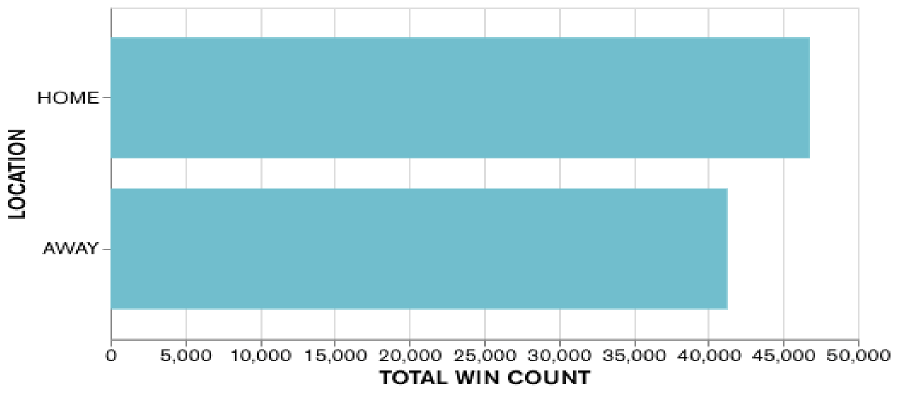
Read Operation		Write Operation	
Count	Time Taken (ms)	Count	Time Taken (ms)
2	0.64	2	2.048
4	1.28	4	2.56
6	1.28	6	49.152
8	1.024	10	98.304
10	5.12	12	166.644
20	5.434	15	148.936



Compare output against hypothesis

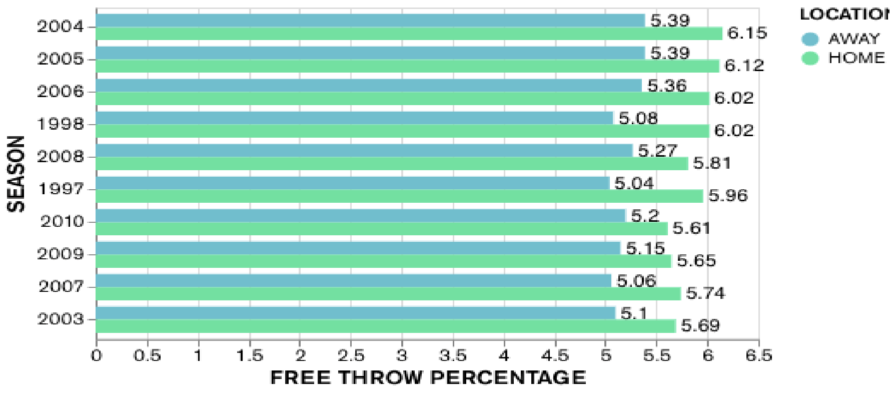
Hypothesis 1 : “To test and verify if MongoDB performs better in CRUD operations and Cassandra performs better in write operations”

Performance evaluation for NBA Players



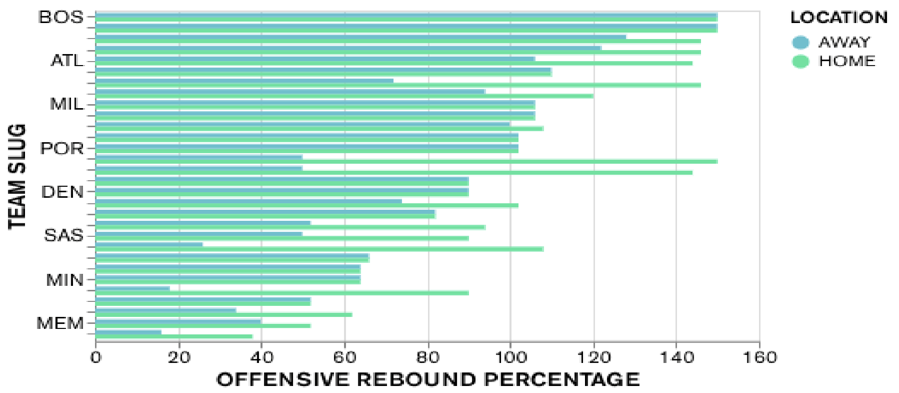
Win Count of the team Home team Vs Away team

Winning counts of the team when they have played as a Home team and as an Away …



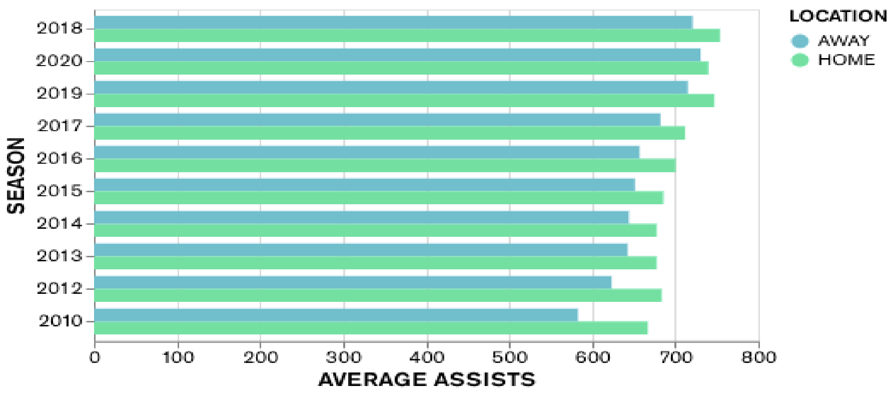
Free Throw % at Home Vs. Away locations across years

Increase in free-throw goal percentage of Home teams compared to the Away teams



Offensive Rebound Percentage Home Vs Away Team

Offensive rebound is important for a team to score points by 2nd chance, thus increa…



Average Assists Home Vs Away Team

Average number of assists per game is relatively higher for most of the teams when t…

Summary & Conclusions

❖ **CRUD Operation**

- MongoDB loads the data faster in comparison to Cassandra
- Except for UPDATE operation, MongoDB is more efficient than Cassandra for READ and DELETE operations.

❖ **Latency**

- Cassandra performed much better in terms of latency (lower latency) as the workload increased
- Write latency for MongoDB increased significantly with increase in workload

❖ **Consistency**

- Cassandra and MongoDB: At consistency level one, the time required for each CRUD operation to be executed is low as compared to the time required for Quorum consistency

❖ **Data Visualization**

- Evaluated the performance of the NBA players using the MONGO charts in Atlas
- Data analysis using mongo charts is more efficient as data is updated at regular intervals

Recommendations

- More number of read and write operations can be performed to better understand the latency patterns as well as the CRUD operations of Cassandra and MongoDB
- Analysis can be performed on larger datasets which shall help to get better understanding about performance of both the databases
- Comparison between Cassandra and MongoDB can be done on real time data such as for on-field game analytics to get better insights of their performance metrics in real time
- On-field game analytics can make use of Mongo charts for data analysis as the data is updated at regular interval
- Horizontal scalability can also be measured by increasing the number of nodes

Thank you