Comparison of NoSQL Databases
MongoDB and Cassandra

Raghav Mehta
Student ID : x17155151
School of Computing (MSc in Data Analytics)
National College of Ireland
Dublin, Ireland
x17155151@student.ncirl.ie

Abstract

Data is being generated at a tremendous pace in today's world. We classify such large datasets as Big Data. There is an increasing growth of unstructured data, which set a challenge in storage, management and querying on big data. Traditional relational databases do not come handy when such large datasets need to be stored as a single logical view of related data is broken into stripes and stored across multiple tables. This is referred as the impedance mismatch problem. Modern day applications create exceptionally large volumes of ever increasing data and ever changing data types. Hence it is not feasible to store such data in the conventional relational databases as they are not optimised for horizontal scaling. Therefore there is a requirement for a better database solution. This paves the way for NoSQL databases and in this research we compare the architecture and performance of two such NoSQL databases ie. MongoDB and Cassandra.

Introduction

As majority of the data being generated is semi-structured and unstructured, we need a database solution which implements a schema-less model. NoSQL databases tackles this problem along with proven horizontal scaling capabilities, a must for rapid growth. NoSQL databases are a requirement for applications generating Big Data or large amounts of real time data. Data is not stored in tabular format with relations and joins between the tables. While querying is slower as compared to relational databases, NoSQL is advantageous as denormalized data is stored. Hence transferring it to another server or horizontally scaling out is convenient through auto-sharding. Adding new dimensions to existing data is also quick as NoSQL uses a schema-less model, as opposed to relational models where a new schema needs to be made and the existing data and new dimensions need to be transferred to it.

NoSQL databases can be classified into 5 categories on the basis of the different architecture in each data model:

1. Column Store : eg. Google BigTable, Cassandra, HBase, Vertica
2. Document Store : eg. MongoDB, Couchbase
3. Key-value Store : eg. Amazon DynamoDB, Oracle NoSQL database, Zookeeper
4. Graph Store : eg. Apache Giraph, Neo4j, Virtuoso
5. Multi-model Store : eg. Apache Ignite, ArangoDB, InfinityDB

This paper is divided in 6 sections, the first section discussing the key characteristics of two popular NoSQL databases ie. MongoDB and Cassandra. In the second section we highlight the architecture of these databases and then we evaluate both the databases on scalability, availability and reliability in section three.

Section four is dedicated to the test plan for benchmarking these databases and the parameters used. In the final two sections we evaluate the test results and draw our final conclusion on the two databases.

I. Key Characteristics

MongoDB

- It is written in C/C++ and JavaScript and is an open-source document store database.
- Records, in this case documents, can be nested within another and be easily remodified by adding or removing fields without restructuring the entire document.
- Data is stored in BSON (Binary JSON) format.
- MongoDB has its own expressive querying language which is dynamic and can execute complex queries.
- It uses an auto generated *'_id'* field to index the documents within a collection and also allows the use of secondary indexes, hence it is efficient in accessing the stored data.
- MongoDB is not dependent on a pre-defined schema and hence is classified as a schema-less database.
- Supports horizontal scaling out through auto-sharding on commodity hardware. Hence it is scalable while being cost efficient.
- MongoDB enforces a 'single-master' model where the master node controls the slave nodes. In case of failure of master node, a slave node is automatically elected as the master node. However there can be write downtime during node failure.
- As per Brewer's theorem, MongoDB is consistent and partition tolerant.
- MongoDB uses a unique replication method consisting of the master node, several slave nodes and an arbiter node.
- MongoDB supports batch processing and aggregation operations.
- MongoDB provides lower latency for read requests.
- The metadata is stored with the value data in MongoDB and hence provides ease in transferring on servers.

Cassandra

- It is written in Java and requires Python support.
- It is an open-source column store database where each set of column family is identified by a row key.
- Cassandra is probably more traditionally structured in the form of rows and tables.
- It uses column family structure which needs to be defined at the time of creation. Hence it is not schema independent.
- Querying is done through CQL which is fairly similar to SQL.
- In Cassandra, querying can be done using only the primary index, hence does not support secondary indexes.
- It has no central node and data can be read/write from any node in a cluster.
- It is optimized for write operations and provides 100% uptime.
- Cassandra enforces a 'multi-master' model. Therefore there is no single point of failure, as the node is replaced by another node in an event of failure. Peer to peer architecture is implemented instead of 'master-slave' model.
- As per Brewer's theorem, Cassandra is available and partition tolerant.
- Cassandra provides lower latency for write requests.
- Replication is simple in Cassandra and only needs to be indicated the number of nodes the data needs to be replicated to.
- It supports horizontal scalability and can be integrated seamlessly to increase performance by addition of more servers in a cluster.
- It is eventually consistent.

	MongoDB	Cassandra
Rich Data Model	✓	✕
Storage Method	BSON Files	Column Family
Secondary Indexes	✓	✕
Availibility	10-40 seconds delay	100% uptime
Write Scalability	✕	✓
Replication	Master-Slave	Multi-Master
CAP Theorem	CP	AP
Language	JSON Fragments	CQL

Table 1 : Key Characteristics

II. Database Architecture

MongoDB

In MongoDB we create a collection of documents, where each document is defined by a unique ID . For the same collection, documents can have different heterogeneous and diverse fields leading to a dynamic schema. Data is stored in Binary JSON format. These documents are stored contiguously.

MongoDB uses a master-slave replication technique. A master node authorizes the slave nodes and in times of failure of a slave node, the master node redistributes the workload on other slave nodes. If the master node fails, then the arbiter node chooses a new master node from the existing secondary nodes. This takes about 10-40 seconds. In case of failure of the arbiter node, it leads to a single point of failure.

The primary server handles the write operations and the secondary servers can only be used for read operations. At times of master node failure, no writes can be performed.

MongoDB performs horizontal scaling through auto sharding. Sharding, which occurs at the collection level, leads to data being distributed amongst several machines in a cluster.

As seen in Fig. 1, the application requests the CRUD operations through the Query router as the Application is not allowed to directly access the shards. The query router decides which replica set ie. shard needs to be forwarded the workload.

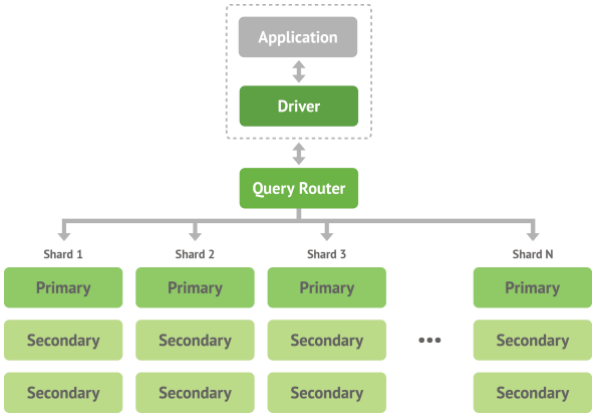


Fig. 1: Sharding in MongoDB

MongoDB enforces consistency and partition tolerance. However it does so by comprising availability.

Cassandra

Being a column store database, Cassandra functions by storing data in column family. Each column family consists of key-value pair. The column key defines the data attributes and a set of columns define a tuple. Each column family is identified using a unique row key.

Cassandra is decentralized and distributed. Due to its peer-to-peer ‘multi-master’ architecture, there is no single point of failure. At any point, a failed node is replaced without any downtime in operations. Since all nodes are same, scaling Cassandra is relatively easier and requires barely any configuration.

In Cassandra, the client application can connect and request read or write functionality using CQL. The node receiving such a request, then coordinates by acting as an intermediate between the client application and actual node where data is being requested from based on the configuration. When a

write activity is executed, a Commit log is created. After this it is written back to the memory structure known as MemTable.

Cassandra also enforces high availability as any failed node in the cluster can be replaced without any downtime. The write operation designated to the failed node is automatically redirected to another node.

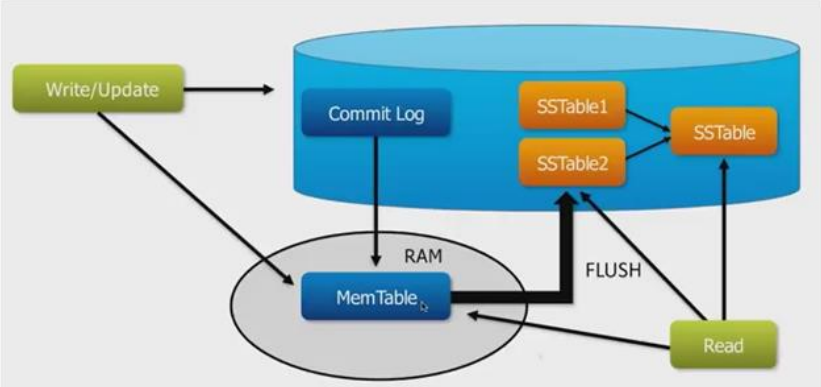


Fig. 2: Operation Cycle in Cassandra

Cassandra provides flexibility in consistency and hence referred to as 'tuneable consistency'. It lets the client application decide the consistency of the database. To achieve high availability, it sacrifices on consistency.

III. Scalability, Availability and Reliability

Scalability

- Vertical scaling has limitations, hence we prefer horizontal scaling. MongoDB uses sharding to achieve horizontal scaling. Using cost effective commodity hardware, additional servers are added to the existing infrastructure. Then data in MongoDB is distributed across the machines to ensure high throughput. Sharding takes place at collection level in MongoDB, distributing the shards across the cluster. A shard key, that exists in every document in a collection is used for this process.
- Scalability is easily achieved in Cassandra. This is because it implements a peer-to-peer architecture. Simply new nodes need to be added to the cluster. Once they are added, they automatically assume responsibility for an equal amount of workload from other nodes on that cluster. There is no downtime in adding or removing a node from a cluster. Cassandra exhibits both – vertical scaling through addition of new nodes and horizontal scaling through addition of new data centers.

Availability

- There can be a single point of failure when it comes to MongoDB. This is due to the master-slave architecture, where a master node authorizes the secondary nodes. If a secondary node fails, then the master node redistributes the workload on the other nodes. On failure of the master node, the arbiter node elects a slave node to become the master node. This process takes from 10 to 40 seconds and during this period the replica set is down. Hence there can be no write operations.
- However in the case of Cassandra, failure of a node is simply redirected to another node. Therefore it does not affect the write operations and the write operations have 100% uptime. Data can also be replicated to many data centers to improve performance. When a new node is

added to the cluster, then the workload is equally distributed with it. Hence Cassandra is highly available.

Reliability

- MongoDB has strong availability, consistency and has sharding features. Hence it is very reliable in read operations. MongoDB writes are not highly reliable. The database often encounters failure in update/write operations which need to be manually queried, checked and confirmed.
- Cassandra is known for it's node replication and eventual consistency. The ease in replication and addition of new nodes to clusters is what makes Cassandra highly reliable. It has a very robust replication architecture to ensure reliability and fault tolerance.

IV. Performance Test Plan

For our testing, we are choosing the two databases discussed in this paper ie. MongoDB and Cassandra. We shall test each of the two databases using Yahoo! Cloud Serving Benchmark framework. The YCSB tool consists of a ycsb-client which generates the workload and the defined workloads which are the scenarios to be executed by the client.

To analyse the performance of the test, we shall choose 2 pre-defined **workloads A & B** and run the tests for 5 different operation counts. To ensure accurate and consistent results, **each of the test will be performed 3 times** and the average of the output parameters will be taken to evaluate, interpret and benchmark the databases.

The workloads refer to a range of scenarios which are a combination of read, write and update operations. The workloads as chosen in our testing are as follows :

- Workload A : Update heavy workload – 50/50 reads and writes
- Workload B : Read mostly workload – 95/5 reads and writes

We will carry out our testing on an instance created on the Open Stack cloud server of NCI, with a disk size of 40GB and RAM of 4GB. For the testing, we are using **YCSB v12**, along with **Java Runtime Environment 8** and **Python 3.5**. Additionally we have installed **Apache Cassandra v3.11.2**, running with replication factor 3 and **MongoDB v3.2.10**.

The number of operations chosen for testing are **100000, 250000, 500000, 750000 and 1000000**.
We are also using the default distribution in selecting the records to be operated on ie Uniform.

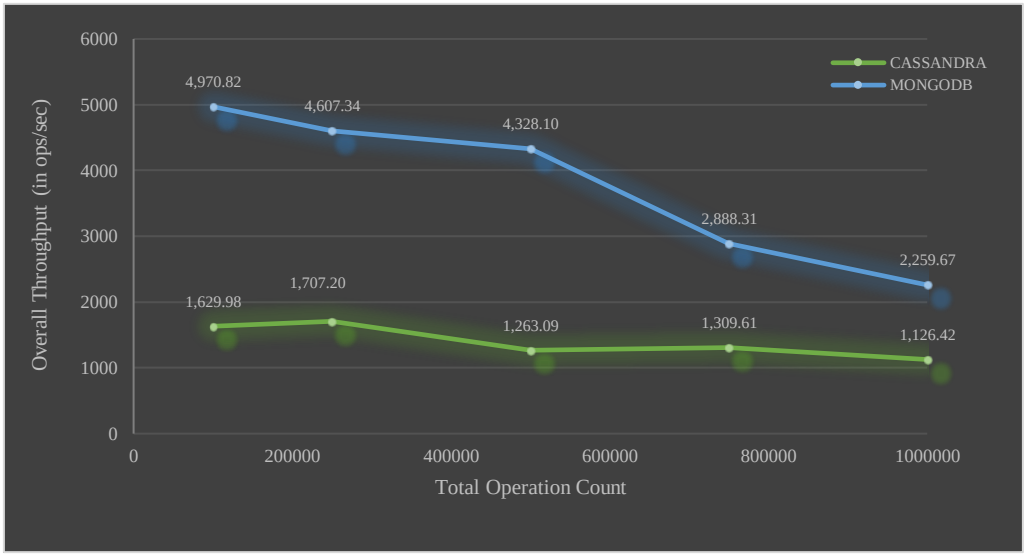
V. Evaluation and Results

All tests have been successfully run, ensuring no insert, read or update failure for both the databases and workloads across any operation count.

A. Workload A

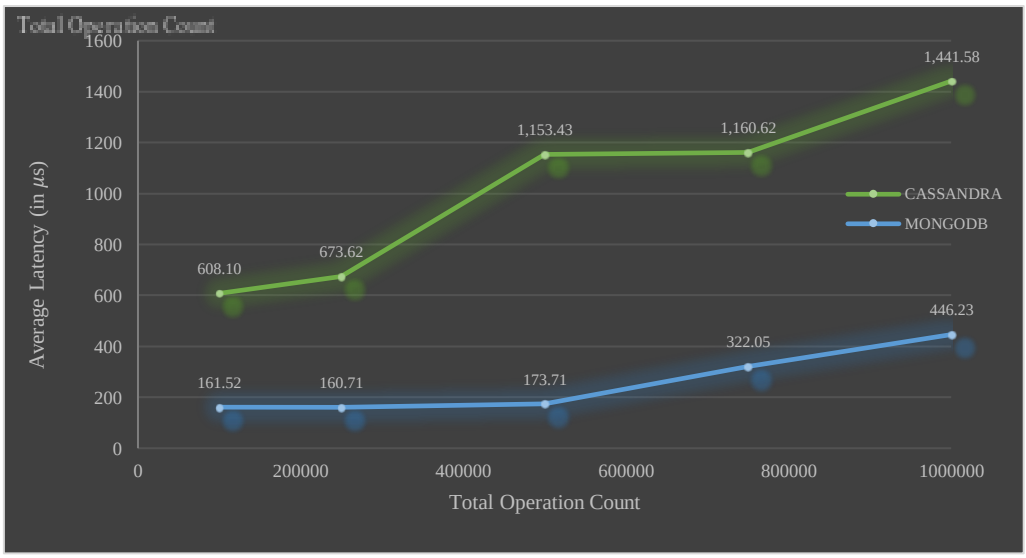
{Read Operations and Update operations account for 50% of total operations each}

a. Overall Throughput (operations per second) vs Total Operations



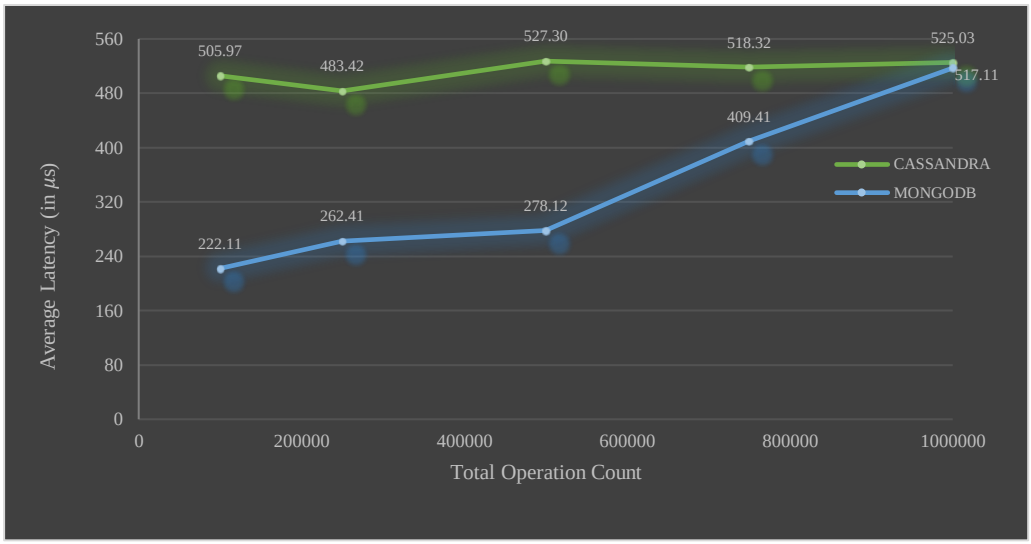
Both databases have a steep decline in the overall throughput as the number of operations increase. MongoDB had a throughput 3x of that of Cassandra when tested for 100,000 operation counts. However this decreases to only 2x as the operation counts were raised to 1,000,000. Hence there is a higher decline in overall throughput in the case of MongoDB, yet it provides better throughput.

b. Read Average Latency (in μs) vs Total Operations



There is a steady increase in the Read Average latency for both the databases. Cassandra shows higher read latency. MongoDB has only 26.5% of read latency when compared to Cassandra at 100,000 total operations (which is approximately 50,000 read operations) The latency grows steadily for MongoDB and reaches 31% to that of Cassandra.

c. Update Average Latency (in μs) vs Total Operations



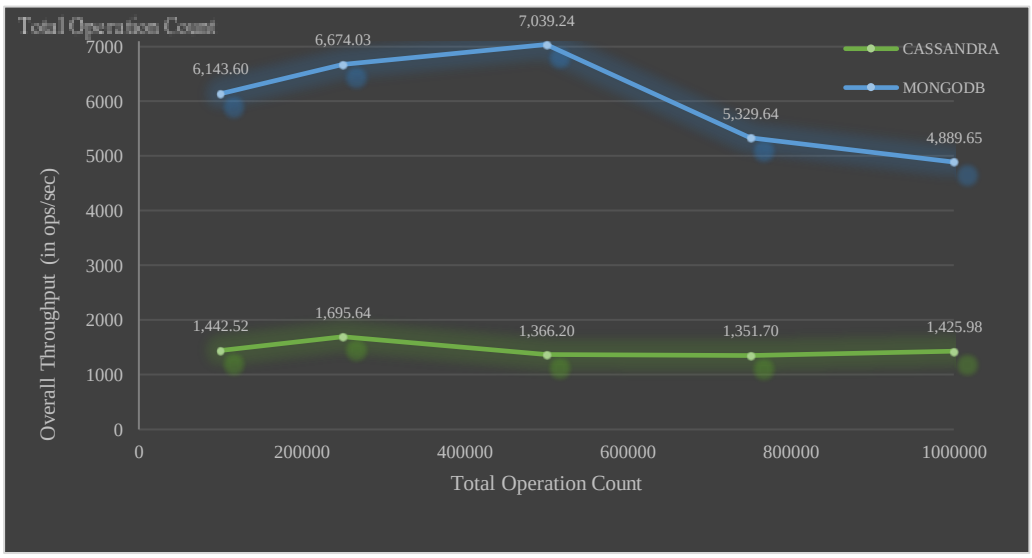
The update latency has a slightly different curve as compared to the overall throughput and read latency graphs. As the operation counts increase from 100,000 to 1,000,000 the latency rises sharply for MongoDB while it is relatively stable for Cassandra. The difference in latency moves from 2.28x to almost equal. However it is noticeable that MongoDB still has low latency in update operations.

> Results for Workload A are as expected, because MongoDB is a read-optimized database as compared to Cassandra which is a typical write-optimized database. This difference should be visible more clearly in Workload B results, as 95% read and only 5% update operations run in each test iteration.

B. Workload B

{Read Operations and Update operations as 95% and 5% of total operations respectively}

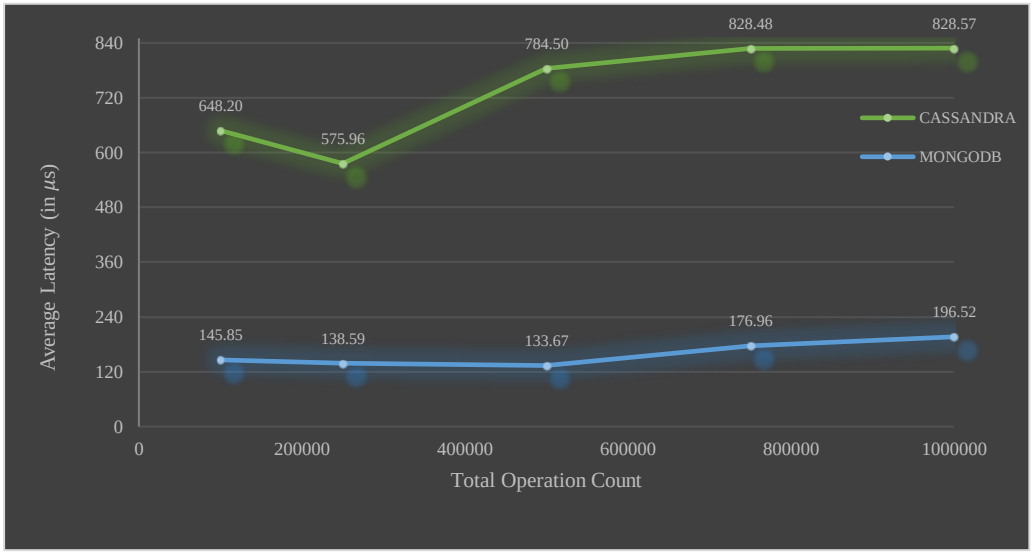
a. Overall Throughput (operations per second) vs Total Operations



The overall operations per second in workload B is much higher for MongoDB as compared to Cassandra. As the operation count increases, Cassandra maintains a

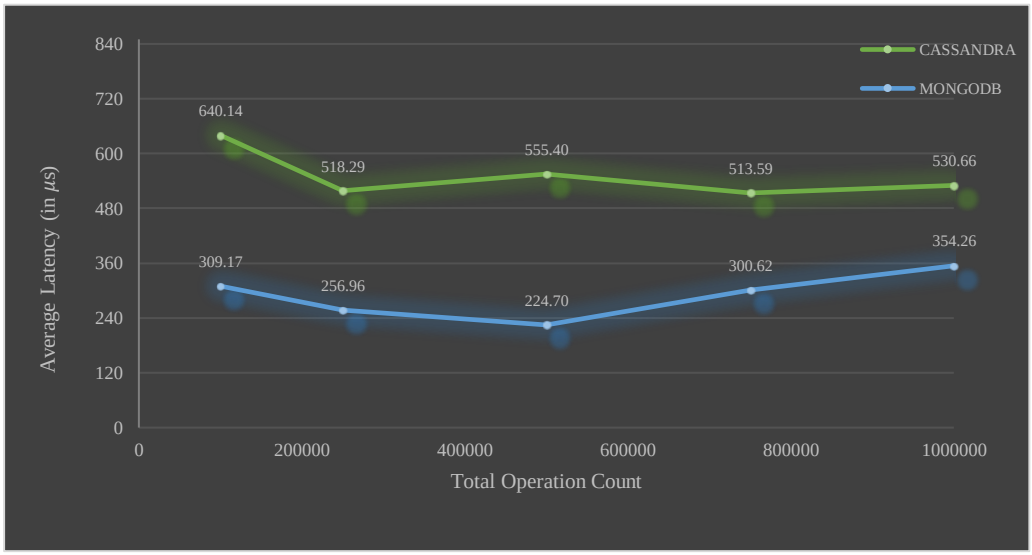
steady throughput however it falls in the case of MongoDB. It falls sharply when 750,000 and 1,000,000 operations are run. Yet MongoDB maintains a much higher throughput.

b. Read Average Latency (in μs) vs Total Operations



As expected with a 95% read operation mix, the latency in read operations are considerably lower for MongoDB. There is only a marginal increase in MongoDB's latency which is due to the fact that large number of operations are running.

c. Update Average Latency (in μs) vs Total Operations



Again we notice, that the update latency in workload B is still higher for Cassandra. With increase in operation counts, there is a decrease in the latency, which is increasing in the case of MongoDB. The average latency comes down from 2.07x to 1.49x that of MongoDB.

VI. Conclusion

Based on the tests and evaluating the results we can conclude that MongoDB provides superior performance than Cassandra in our testing scenarios. The overall throughput is much higher for MongoDB, sometimes attaining more than 5x of Cassandra.

MongoDB also exhibits much lower latency in both the workloads across all operation counts.

Hence we can conclude from our test results and analysis, that MongoDB is a better performing NoSQL database.

However we must not restrict our test research conduct to over here, a different environment and different operation parameters ie. read-write-modify, may lead to Cassandra outperforming MongoDB.

VII. References

[1] Cooper, B., Silberstein, A., Tam, E., Ramakrishnan, R. and Sears, R. (2018). Benchmarking cloud serving systems with YCSB.
[2] Carpenter, J. and Hewitt, E. (2017). Cassandra. Beijing: O'Reilly.
[3] Bazar, C., Sebastian, C. (2014). "The transition from RDBMS to NoSQL. A comparative analysis of three popular non-relational solutions: Cassandra MongoDB and Couch base", Database systems Journal, vol. 5, no. 2.
[4] Datt, N. (2016). Comparative Study of CouchDB and MongoDB – NoSQL Document Oriented Databases. International Journal of Computer Applications, 136(3), pp.24-26.
[5] GOTO2012 Conference – Introduction to NoSQL by Martin Fowler
[6] https://www.mongodb.com/nosql-explained
[7] https://www.mongodb.com/mongodb-architecture
[8] https://scalegrid.io/blog/cassandra-vs-mongodb