Hewlett Packard Enterprise

HPE CTY

Network Flow Database

A Report on MongoDB

Documented by:

Raghavendra J R

Performance of MongoDB

Introduction :

MongoDB is a leading NoSQL database, renowned for its flexibility and scalability. As a document-oriented database, MongoDB stores data in flexible, JSON-like documents, making it ideal for handling semi-structured data and accommodating dynamic schemas. This open-source database is developed by MongoDB Inc. and has gained widespread adoption for its ease of use and powerful features.

Key Features:

1. **Schema Flexibility**: MongoDB uses a flexible schema model that allows documents to vary in structure, providing adaptability to changing data requirements without the need for expensive schema migrations.
2. **Scalability and Performance**: MongoDB supports horizontal scaling through sharding, distributing data across multiple servers to ensure high availability and large-scale performance.
3. **Rich Query Language**: The database offers a powerful query language that includes support for ad hoc queries, indexing, and real-time aggregation, enabling complex data retrieval and analytics.
4. **Indexing**: MongoDB supports various indexing strategies, including single-field, compound, multikey, geospatial, and text indexing, which significantly improve query performance.
5. **Replication and High Availability**: Through replica sets, MongoDB provides automated failover and data redundancy, ensuring data availability and fault tolerance.
6. **Aggregation Framework**: The aggregation framework in MongoDB allows for efficient data processing and transformation using a pipeline approach, suitable for creating sophisticated analytics and reporting.
7. **ACID Transactions**: MongoDB ensures ACID (Atomicity, Consistency, Isolation, Durability) compliance for multi-document transactions, providing reliability for critical applications.
8. **Community and Ecosystem**: MongoDB has a vibrant community and a comprehensive ecosystem, including tools for data visualization, ETL (Extract, Transform, Load), and cloud-based database services.

Use cases :

1. **Web Applications**: MongoDB's flexible schema and JSON data format make it a perfect fit for modern web applications, allowing rapid development and iteration.
2. **Big Data and Analytics**: Its ability to handle large volumes of data and perform real-time analytics makes MongoDB suitable for big data applications.
3. **Content Management Systems**: The flexibility and scalability of MongoDB are beneficial for content management systems that require frequent updates and complex queries.
4. **IoT Applications**: MongoDB can efficiently manage and analyse the vast amount of data generated by IoT devices, thanks to its scalable and flexible nature.

MongoDB for designing Network Flow Database

Structure of Network Flow Database:

A Network Flow database is designed to efficiently capture, store, and analyse packet information passing through the switches in the network. This allows users to efficiently store, retrieve, and modify information about different packets sent in a network.

The Network flow database contains the following attributes:

1) **src_ip** : This field is used to store the source IP address of the packet.

2) **dest_ip** : This field is used to store the destination IP address of the packet.

3) **src_port** : This field stores the source port number of the application.

4) **dest_port** : This field stores the destination port number.

5) **ip_type** : (IPv4 or IPv6) This stores the protocol used in the network layer.

To avoid redundancy, all five attributes should be unique in the database. Thus, any packet containing all five-attribute information is stored in the network flow database and used as per the client's requirements.

Considering this structure of the database, MongoDB is a good option to design this database for the following reasons:

1. **Schema flexibility:** MongoDB is a NoSQL database, meaning it does not require a fixed schema like SQL databases. This allows for more flexible and dynamic data models, making it easier to evolve your application over time without having to modify the entire schema.
2. **Scalability**: MongoDB is designed to scale out horizontally, allowing you to distribute your data across multiple servers easily. This makes it suitable for handling large volumes of data and high throughput applications.
3. **High performance**: MongoDB uses internal memory mapping for storage and supports indexing, which results in faster read and write operations, especially for large datasets.
4. **JSON-like document model**: MongoDB stores data in JSON-like documents, which makes it easy to work with data in a format that's similar to how objects are represented in many programming languages. This can simplify development and reduce the need for complex ORM (Object-Relational Mapping) layers.
5. **Querying flexibility**: MongoDB supports rich query expressions, including queries based on document structure, nested fields, and even geospatial queries. It also supports aggregation pipelines, which allow for complex data processing within the database.
6. **Horizontal scalability**: MongoDB's architecture allows for easy distribution of data across multiple nodes, enabling horizontal scalability. This means you can add more servers to handle increased load without significant changes to your application.

7. **Replication and fault tolerance**: MongoDB supports replica sets, which provide automatic failover and data redundancy. In the event of a node failure, MongoDB can automatically promote a secondary node to primary, ensuring high availability and data durability.
8. **Community and ecosystem**: MongoDB has a large and active community of developers, which means there are plenty of resources, tutorials, and third-party libraries available to help you get started and solve problems.

Disadvantages of MongoDB :

- **Schema Design Complexity**: While MongoDB's flexible schema can be an advantage, it can also lead to complexities in schema design, especially for applications with complex relationships between data entities.
- **Memory Usage**: MongoDB tends to consume more memory compared to traditional SQL databases, particularly for certain operations like aggregations and indexing.
- **Transaction Support**: MongoDB's support for transactions has improved over the years, but it may still not be as robust as traditional SQL databases, especially in multi-document transactions.
- **Data Consistency**: MongoDB's default configuration favors availability and partition tolerance over consistency (following the CAP theorem). While this is suitable for many use cases, it may not be ideal for applications that require strict consistency guarantees.
- **Lack of Joins**: MongoDB's query language does not support joins in the same way as SQL databases. While it offers some alternatives like the $lookup aggregation stage, complex join operations can be less efficient compared to SQL databases.
- **Indexing Challenges**: While MongoDB supports indexing, improper indexing strategies or over-indexing can lead to performance issues. Maintaining indexes can also impact write performance.
- **Maturity and Ecosystem**: While MongoDB has been around for a while and has a large user base, it may still be perceived as less mature compared to some traditional SQL databases. Additionally, the ecosystem around MongoDB, including tools and libraries, may not be as extensive as that of some other databases.
- **Data Durability**: By default, MongoDB acknowledges write operations after writing to memory, which means there's a risk of data loss in the event of a failure before the data is persisted to disk.
- **Resource Intensive**: MongoDB can be resource-intensive, especially in terms of CPU and disk usage. This may require careful capacity planning and resource allocation.
- **Not Open Source** : MongoDB is not open source and enterprises need to get a commercial license in order to utilize the source code.

Key features required for Network Flow Database

Relational /Non-Relational?	In memory/Disk based (RAM v/s SSD)	Publisher-Subscriber notification support for row/column updates (Y/N)	Partial key search support (Y/N)	Opensource License used	Client Library language support	Multi-threaded publisher/subscriber support
Relational	Both	Yes	Yes	SPL(versions released after Oct 16, 2018) GNU AGPL v3.0 (for all versions released prior to October 16, 2018)	C, Python, C++, Java, Rust, Swift, Node.js, Ruby, Kotlin, C#, Scala, Typescript	-

MongoDB is a non-relational database and it is primarily disk based. But it uses RAM for caching the frequently accessed data. MongoDB supports partial key search. MongoDB community edition is free for all but for an enterprise edition, they need to have license for commercialisation. MongoDB supports most of the programming languages widely used in industries and provides cloud services as well. Also MongoDB provides a feature called change streams to support publisher-subscriber notification support.

Additional feature which helped in improving performance

Bulk-Write operations :

- In MongoDB, bulk write operations allow for the efficient execution of multiple write operations, such as inserts, updates, and deletes, as a single batch.
- These operations can significantly improve performance by reducing the number of round-trips between the application and the database server. Bulk write operations are particularly useful in scenarios where large volumes of data need to be processed or manipulated in a single transaction-like operation.
- By batching multiple write operations together, MongoDB can optimize resource utilization and minimize overhead, resulting in faster and more efficient data processing.
- Additionally, bulk write operations ensure atomicity, meaning that either all operations in the batch succeed, or none of them are applied, maintaining data consistency and integrity. This feature is invaluable in high-throughput applications where maintaining data consistency is critical.

Performance testing of MongoDB using Python Modules

We have written code to test the performance of MongoDB by performing bulk insertions, deletions, and updates on a large dataset. Here's a brief explanation:

Generation of Dataset for the operation :

For the dataset to be used in performance measurement, this Python script generates random unique IP address and port number pairs, simulating real network packet exchanges. It includes functions to generate random IPv4 and IPv6 addresses, random ports, and to randomly select the IP type. The script initializes parameters, tracks unique pairs with a dictionary, and uses a counter to ensure a large number of unique request-response pairs. Each pair and its reverse (representing the response packet) are written to a CSV file, ensuring unique and realistic network packet data.

Code Overview (batch operations):

1. **Setup and libraries**

- The script imports necessary libraries such as pymongo for MongoDB operations, datetime for measuring time, matplotlib.pyplot for plotting graphs, and csv for reading CSV files.
- Data Reading Function: The `read_data_from_file` function reads data from a shuffled CSV file and processes it into a list of tuples..
-

2. **Main Functionality :**

- Database Connection: Establishes a connection to the MongoDB database using the MongoDB client.
- Insert, Delete and Update operations using MongoDB API.

3. **Batch Operations Menu:**

- A menu-driven interface allows the user to select operations
- Delete Tuples: Reads data, copies it to a temporary table, and performs deletions in batches using bulkWrite() method
- Insert Tuples: Reads data and inserts it into the main table (`flow_table`) in batches using the insert_many() command.
- Update Tuples: Reads data and performs updates in batches using bulkWrite method

4. **Time Measurement and display** :

- Print Table: A function (`print_table`) is defined to print the timing results in a tabular format.
- Plot Graph: Another function (`print_graph`) is provided to plot the timing results using matplotlib.

5. **Error Handling** :

- Exception Handling: Catches and prints any errors that occur during database operations.

Usage flow :

1. **Run Script**: The user runs the script, which connects to the MongoDB database.
2. **Display Menu**: The script displays a menu for the user to select an operation. 3. Perform Operation: Based on user input, the script performs the selected operation (delete, insert, update) on the database in batches.
3. **Time Calculation**: The script measures the time taken for each operation and calculates the average time per record for each file.
4. **Display Results**: The script prints the average time taken for each file.

This structure ensures efficient handling of large datasets, provides clear performance metrics, and maintains the integrity of the database operations through proper error handling and resource management.

Github:

https://github.com/Raghavendra-J-R/Pymongo/blob/main/pymongo-5/crud_batch_operations.py

Code Overview (normal operations) :

Here each tuple is inserted, deleted and updated individually

1. **Setup and libraries**

- The script imports necessary libraries such as pymongo for MongoDB operations, datetime for measuring time, matplotlib.pyplot for plotting graphs, and csv for reading CSV files(though `matplotlib` is not used in the code).
- Data Reading Function: The `read_data_from_file` function reads data from a CSV file and processes it into a list of tuples

2. **Main Functionality :**

- Database Connection: Establishes a connection to the MongoDB database using 'MongoClient'
- Insert, Delete and Update operations using MongoDB API
- Insertion of 100, 1000, 10000, 100000 and 1000000 into the database followed by delete and update operations.

3. **Menu-driven Interface:**

- User Menu: A menu is displayed to allow users to choose between three operations: deleting, or inserting tuples or updating tuples, or exiting the script.

4. **Operations :**

- Insertion operation is performed by inserting the csv file data (`data_100_tuples.csv`, `data_1000_tuples.csv`, etc.) into the database serially.
- Delete operation is performed by reading the files (`data_100_tuples.csv`, `data_1000_tuples.csv`, etc.) and deleting serially.
- Update operation is performed by reading the files(`data_100_tuples.csv`, `data_1000_tuples.csv`, etc.) and updating the src_port to 0 in the respective document.

5. **Time Operations and display :**
    - The code records the time before and after each operation to measure the duration.
    - It calculates and prints the average time taken for deletions, insertions, and updates for each file.

6. **Error Handling :**
    - The code includes a try-except block to catch and print any exceptions that occur during database operations.

Usage flow :

1. **Run Script**: The user runs the script, which connects to the MongoDB database.
2. **Display Menu**: The script displays a menu for the user to select an operation. 3. Perform Operation: Based on user input, the script performs the selected operation (delete, insert, update) on the database.
3. **Time Calculation**: The script measures the time taken for each operation and calculates the average time per record for each file.
4. **Display Results**: The script prints the average time taken for each file.

This structure provides a way to measure the performance of individual deletions, insertions, and updates on a MongoDB database. It records the time taken for each operation and calculates average times, helping users understand the performance characteristics of their database operations.

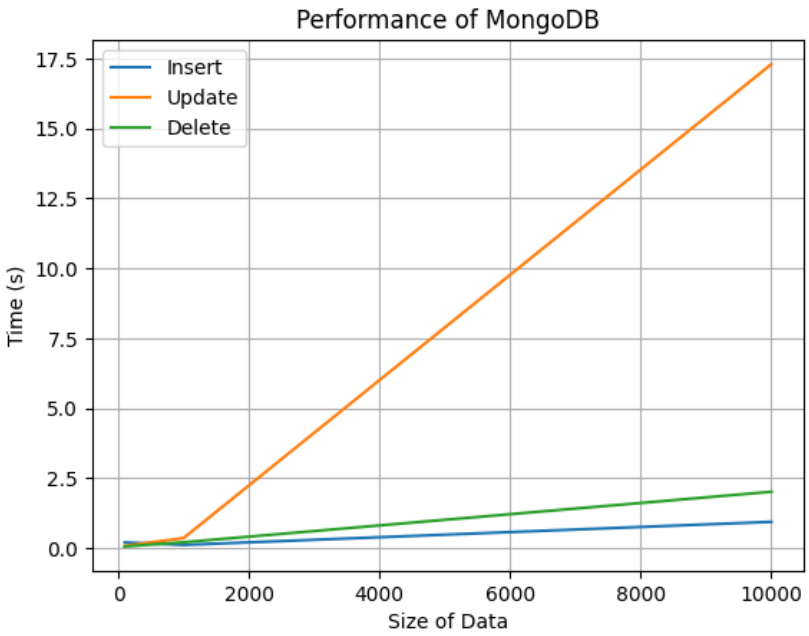
Github :

https://github.com/Raghavendra-J-R/Pymongo/blob/main/pymongo-6/crud_normal_operarions.py

Results

1. Performance of Normal Operations:

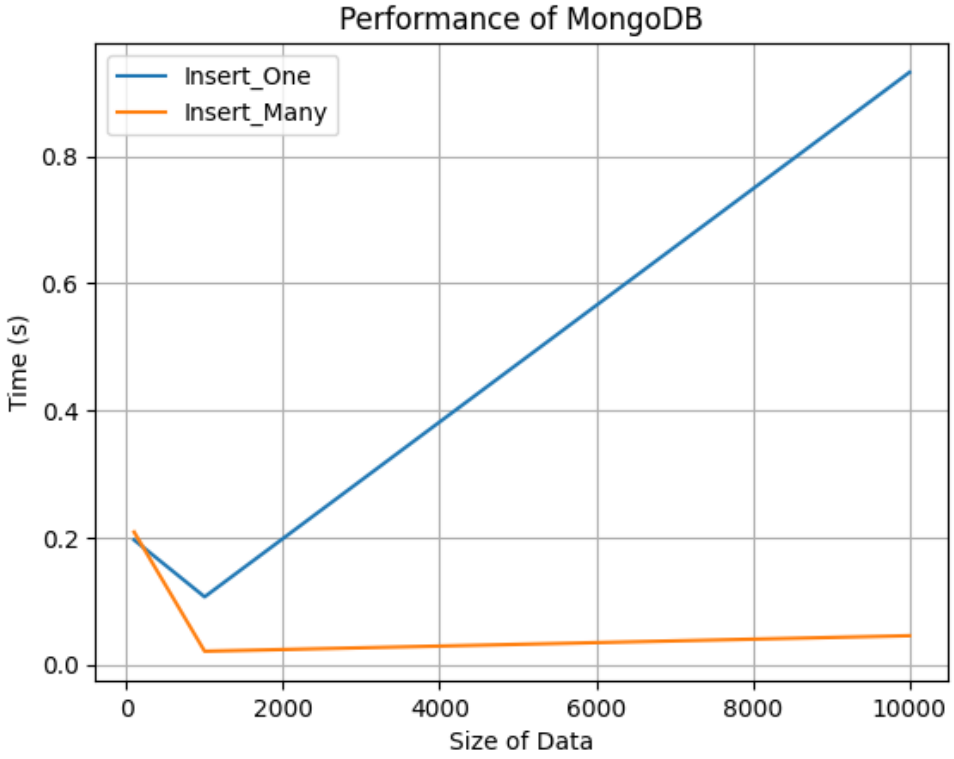
MongoDB CRUD Operations Latency – Normal Operations (in sec)			
Tuple size	**100**	**1000**	**10000**
Insert	**0.1961**	**0.1065**	**0.9326**
Update	**0.07341**	**0.35**	**17.3005**
Delete	**0.0532**	**0.20157**	**2.009**



From the above data it can be concluded that MongoDB efficiently performs the insert operation but takes lot of time while dealing with updation and deletion of large volumes of data.

2. Difference in the performance of InsertMany and InsertOne during normal operation :

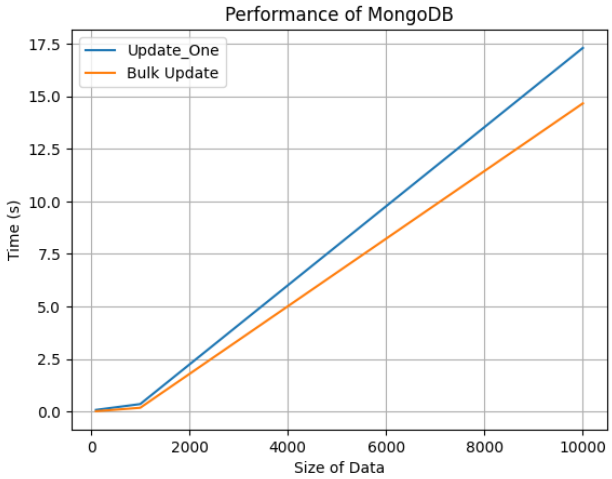
Comparison of Insertion in Normal operation (in sec)			
Tuple size	**100**	**1000**	**1,0000**
Insert_many	**0.2085**	**0.020829**	**0.045271**
Insert_one	**0.1961**	**0.1065**	**0.9326**



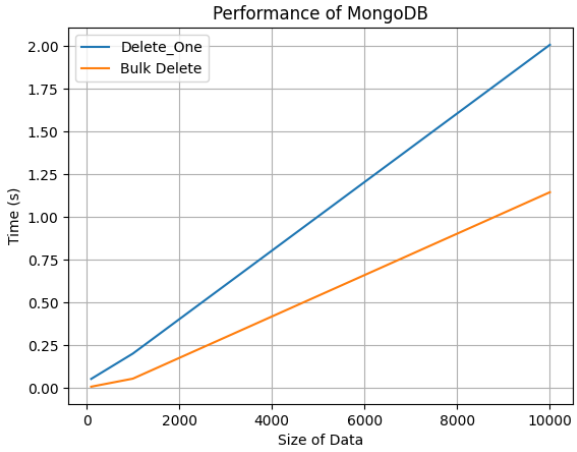
From the above results it can said that InsertMany performs well for inserting data in a bulk fashion rather than sequential inserts.

3. Comparison between normal and bulkWrite methods :

Comparison of Updation in Normal operation (in sec)			
Tuples Size	**100**	**1000**	**10000**
Update-One	**0.07341**	**0.35**	**17.3005**
Bulk-Update	**0.009894**	**0.174184**	**14.661395**

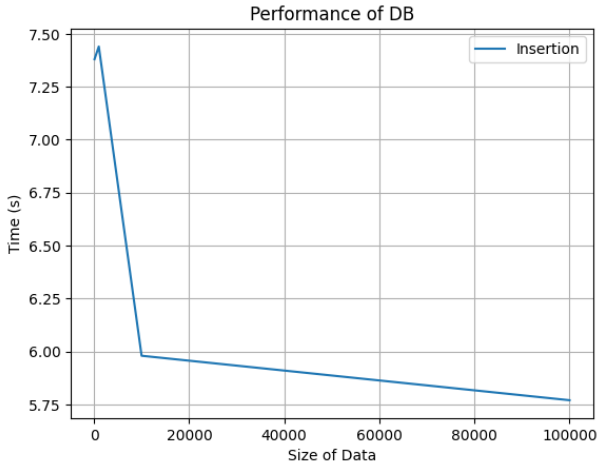


Comparison of Deletion in Normal operation (in sec)			
Tuples Size	**100**	**1000**	**10000**
Delete One	**0.0532**	**0.20157**	**2.009**
Bulk-Delete	**0.007648**	**0.054735**	**1.146129**



4. Data size of 10 lakh and batch operations were executed in batches of 100, 1000, 10000 and 100000

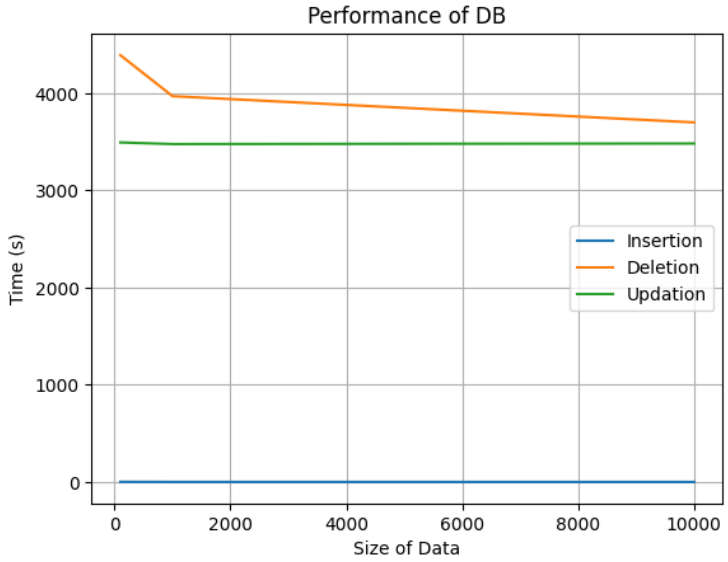
MongoDB CRUD Operations Latency – Batch Operations (in sec)				
Batch size	**100**	**1000**	**10000**	**1000000**
Insert	**7.38**	**7.44**	**5.98**	**5.58**



In the above experiment update and delete have been not used because of large volumes of the dataset.

5. Data size of 1 lakh and batch operations were executed in batches of 100, 1000, 10000

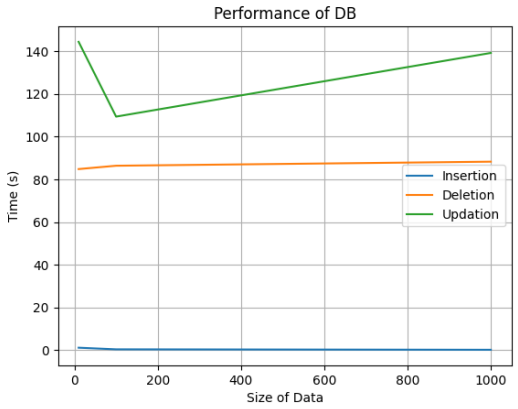
MongoDB CRUD Operation Latency – Batch Operations (in sec)			
Batch Size	**100**	**1000**	**10000**
Insert	**2.12**	**0.9424**	**0.68**
Delete	**3494**	**3477**	**3483**
Update	**4392**	**3970**	**3700**



From the above plot, it can be said that it takes more time to delete and update in batches rather than insertion.

6. Data size of 1 lakh and batch operations were executed in batches of 10, 100, 1000 and 100000

MongoDB CRUD Operation Latency – Batch Operations (in sec)			
Batch Size	**10**	**100**	**1000**
Insert	**1.1451**	**0.385**	**0.9863**
Delete	**80.852**	**78.963**	**96.321**
Update	**144.37**	**109.3626**	**139.1828**



Here also Insertion time remains almost constant but updates and deletes take long time compared to insertion.

Performance Comparison with YCSB tool

Introduction to YCSB :

The Yahoo! Cloud Serving Benchmark (YCSB) is an open-source database benchmarking suite and a critical analytical component of cloud-based database management system (DBMS) evaluation. It allows users to comparatively measure how various modern SQL and NoSQL DBMS perform simple database operations on generated datasets.

These are the types of workloads in YCSB

Workload A : Update heavy workload: 50/50% Mix of Reads/Writes

Workload B : Read mostly workload: 95/5% Mix of Reads/Writes

Workload C : Read-only: 100% reads

Workload D : Read the latest workload: More traffic on recent inserts

Workload E : Short ranges: Short range-based queries

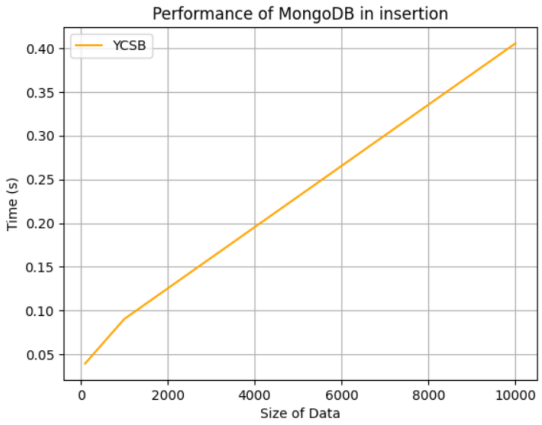
Workload F : Read-modify-write: Read, modify and update existing records

In order to compare with the script values, certain defaults have been set :

- Synchronous MongoDB driver.
- Upsert option set to 'false'.
- Update performed in ycsb by making 'readmodifywrite = 1'.
- For all tuple size, operation count = tuple size.

1. Insertion of Tuples of size 100, 1000, 10000

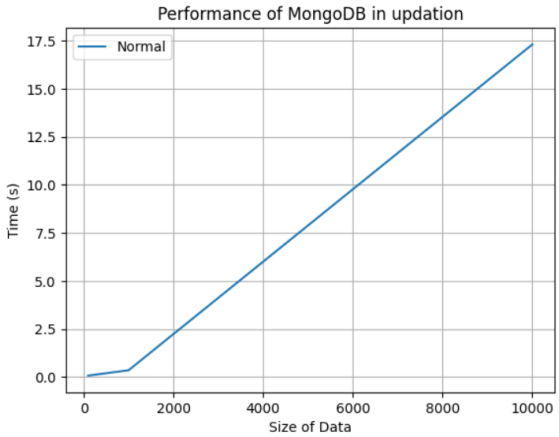
Comparison of script values and YCSB data (in sec)		
Count	**Script value**	**YCSB**
100	**0.1961**	**0.0395**
1000	**0.1065**	**0.09405**
10000	**0.9326**	**0.4055**

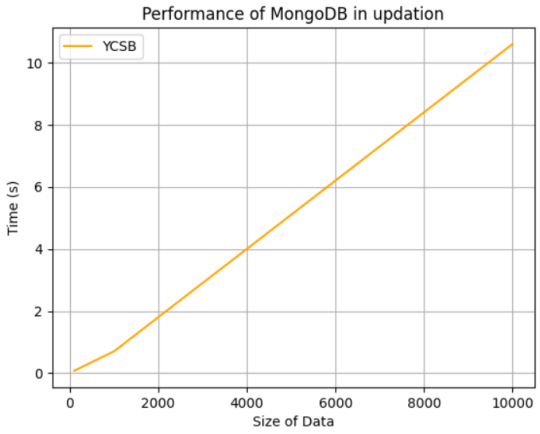


From the plot it can be said that Insertion can be compared well with the benchmarking data.

2. Updating tuples of size 100,1000,10000

Comparison of script values and YCSB data (in sec)		
Count	**Script Values**	**YCSB**
100	**0.07341**	**0.080658**
1000	**0.350**	**0.7063**
10000	**17.3005**	**10.596**





From the plot it can be said that MongoDB can be compared well with the benchmarking data

Defaults used in the experiment/benchmarking :

- 16GB cache size with a RAM capacity of 32GB
- All operations performed in a Linux environment
- Synchronous drivers for performing the operations (Pymongo client)
- No indexes on the data used

Additional Explorations on MongoDB :

- Upon thoroughly inspecting the official documentation of MongoDB, I found out that MongoDB by default uses 50 % of (RAM-1)GB of space for the cache.
- The storage engine used in MongoDB is called WiredTiger and I was successful in changing the cache configuration of the file. But this did not help in upgrading the performance of the database as the document on a whole consumed nearby 100MB of storage and cache size would really fit in for this dataset.
- WiredTiger engine usually uses internal compression algorithms like snappy , zlib and zstd and these compression algorithms have an effect on reducing the storage.
- A document in MongoDB limits its size to 16MB
- Apart from these factors, MongoDB also provides cloud storage structure called – MongoDB Atlas where the users can use the MongoDB server on a specific region.
- But for the Network Flow Database the latency should be considered while connecting to mongod instances.
- The concept of indexes did not fit to the use case as it would have been increasing the latency on heavy writes as well as leading to increased storage size

CPU and Memory utilization during the operations :

The CPU and memory utilisation has been recorded with the help of top command by getting the process ID of MongoDB server running on the system

1. For batch operations :

MongoDB Batch Operation and CPU usage using top		
Operations	**CPU %**	**Memory %**
Insert	**27.9**	**1.7**
Update	**100**	**1.6**
Delete	**100**	**4.2**

2. For Normal Operations :

MongoDB Normal Operation Memory and CPU usage using top		
Operations	**CPU%**	**Memory%**
Insert	**24**	**4**
Update	**100**	**4.3**
Delete	**100**	**4.3**

It can be inferred from the above results that the memory consumption is least for the operations performed but CPU utilization is very high

Conclusion

As a keen explorer of MongoDB, I would like to say that MongoDB has a very good insertion performance and datasets can be easily ingested into the database. With the bulk operations for inserting the data, the database performs well compared to serial insertion of the data. But in the aspect of insertion and deletion, the operations have been performed by reading the csv files and then looking the data in the database. But this would be costlier if the csv file has files in a highly random order as a result of which the database has to search for all five tuples and sequentially delete it. But compared to update, delete performs well. Since the storage capacity of a huge data volume like 10 lakh data size nearly consumed 60-100MB of size including the indexes, I would say that MongoDB acts as a good database for storing the network flow. Meanwhile further experiments need to be done in more robust and ideal environment where there is less network latency and more powerful processors.

References

- https://www.mongodb.com/docs/
- https://www.mongodb.com/docs/manual/core/wiredtiger/
- https://github.com/brianfrankcooper/YCSB