An Overview of Document Database, MongoDB, DynamoDB, compared with MySQL

Illinois Institute of Technology

CSP 554 Big Data Technology

Chao Yang cyang72@hawk.iit.edu

Contents

Abstract

Document databases are widely used in the current technology world, company used it for many reasons, to accommodate with different types of data, to store and retrieve big data, to achieve faster data operations for applications. In this project, in order to achieve an overview of document databases we will do experiments on document databases MongoDB and DynamoDB, to compare with traditional relational database MySQL. Benchmark code will be implemented to measure the data operations among databases including insert, query, update and delete. Experiment result is intuitionally but also confused, the project explains problems in the experiment and what we could to solve the deficit.

Keyword: document database, MongoDB, DynamoDB MySQL, Benchmark

Introduction

A document database (also known as a document-oriented database or a document store) is a database that stores information in documents. What is a document? According to the official website of MongoDB, a document is a record in a document database. A document typically stores information about one object and any of its related metadata. Document databases are considered to be non-relational databases aka NoSQL databases. NoSQL databases have more types, including document databases, key-value stores, column-oriented databases and graph databases.

In this project, we will choose three typical document databases MongoDB, DynamoDB which is supported by Amazon Web Service and one relational database MySQL (8.0.30 for windows) for comparison purpose. The project will test these databases in these fields: performance in data insertion, data query, data update and data deletion. During the coding, we find another perspective of comparison as well, for example, easy use of the database, SQL syntax friendly. In the method, the project tried the best to control all elements which could affect the result, however, network latency, cloud stability and other hidden reasons are out of control and could affect the final performance, so experiment result is given for reference purpose. In the discussion, we discuss the deficit in the project. Through the project, we can achieve a better understanding on why we need document databases

Method

The experiment will be divided into two parts, each part will test and research on the properties of the document databases. We will use python 3.9 to write codes for creating connections and performing operations.

Database benchmark

A few datasets in different sizes will be prepared for these databases, number of rows range from 10, 100, 1000, 10,000, 160,000, each of them only contains name and address as string, and are saved as csv format. In the benchmark, the experiment is designed at CRUD operations including creating database/tables/collections, writing data, querying data, updating data and deleting data The pseudocode of part I is as following:

1. Read file data
2. Create connection and table
3. Writing data into database
4. Query a row of data
5. Update a row of data
6. Delete a row of data

The following python libraries are used in the experiments:

Boto3: DynamoDB API

Pymongo: python MongoDB API

MySQl and MySQL connector: python MySQL APi

Results

Database benchmark:
A few challenges were occurred when fetch the result during the experiment:

- NoSQL databases in the experiment are all hosted remotely which means data transformation is completed by a series of network interactions, however, MySQL has its local host which can be installed and run locally.
- DynamoDB has bad performance when loads large size data even batch writing is enabled, in the experiment bad performance appeared when load the file with 160,000 rows, time has exceeded for more than 6 hours but results are still unknown.

Write					
	10	100	1000	10,000	160,000
DynamoDB	0.34048	0.65674	17.0248	696.31155	
MongoDB	0.52981	0.54626	0.73366	2.00388	29.61833
MySQL	0.00501	0.01451	0.08806	0.70242	11.49354

In the writing operations, writing time increased as the file size increased, but obviously MySQL always achieve the best performance among three databases, this can be explained: 1. MySQL is hosted locally, there's no network latency. 2. Size of data is still small; type of data is simple.
For DynamoDB, batching writing is enabled, multiple records are stored in one batch, and then the batch is sent to DynamoDB remotely, which makes whole writing takes longer, but it doesn't mean DynamoDB can't process large files, we could simply upload large file to Amazon S3 bucket and make DynamoDB ingest data directly from S3, it can accelerate data writing, though it will break the fairly measurement for another two databases.
MongoDB achieves better performance than DynamoDB too, overall MongoDB support various data types and have a large size of document up to 16MB, while DynamoDB only allows item sizes of up to 400KB.
Except for writing performance, here's code snippet of writing operations in the database for estimating easy use performance:
DynamoDB:

```
with table.batch_writer() as writer:
    for row in reader:
        mdict = {'name': row[0], 'address': row[1]}
        writer.put_item(Item=mdict)
```

The dictionary contains two key-value pairs, and write into a batch called writer, writer will send data to DynamoDB

MongoDB:

```
mycol.insert_many(mydata)
```

Same as DynamoDB, however mydata here is an array with dictionaries of name and address. Using insert_many api, it can write multiple records in one execution. Code

is simple and neat.

MySQL:

```
insert_query = 'insert into customers (name, address) values
(%s, %s)'
for row in reader:
    array.append((row[0], row[1]))
# insert time
mycursor.executemany(insert_query, array)
```

As relational database, the SQL is required to interact with MySQL, we need an insert query which contains destination table, schema, and values.

Overall, MongoDB has simple and straightforward characteristic than DynamoDB and MySQL.

Query					
	10	100	1000	10,000	160,000
DynamoDB	0.06825	0.08054	0.06574	0.07541	
MongoDB	0.0	0.0.0	0.00097	0.0	0.0
MySQL	0.13118	0.12080	0.11516	0.12391	0.26816

In the table, query time for MongoDB is very interesting, it can take almost zero second to fetch one row of data from multiple records, here is code:

```
mydoc = mycol.find({'address':'YANG'})
```

the code is trying to find a pair of data with key is ‘address’, value is ‘YANG’. We assume query time 0 is caused by query syntax is too simple, so we make another complicated syntax:

```
mydoc = mycol.find({"address": { "$regex": "^S" } })
```

This code finds all address with letter ‘S’ in address, but query time is still 0

Easy use performance

DynamoDB achieve better performance than MySQL, that’s why DynamoDB is used more for high frequency data transformation under circumstances like cache, web servers.

```
table.query(KeyConditionExpression=Key('name').eq('CHAO'))
```

DynamDB has simple sytax in query too

```
mycursor.execute("delete from customers where name = 'CHAO'")
```

MySQL has update SQL in its update function.

Update					
	10	100	1000	10,000	160,000
DynamoDB	0.07086	0.07199	0.08096	0.06966	
MongoDB	0.04865	0.70906	0.61467	0.46761	0.30217
MySQL	0.12507	0.18523	0.14039	0.11267	0.33310

In the update table, three databases all have a different feature, when size of file is increased, but update time is not always increased. The step of update includes find data and then replace data with updated version. Take query table into consideration, update takes longer time than query, we could simplify the update operation as a

combination of find and update operations, direct update operation will not take most of time, but find could vary. DynamoDB achieves better performance among three databases, especially when file size increased.

Easy use performance:

DynamoDB:

```
table.update_item(Key={'name': 'CHAO', 'address': 'YANG'},
                  UpdateExpression='set tax = :tax',
                  ExpressionAttributeValues={
                      ':tax':'1',
                  },
                  ReturnValues='UPDATED_NEW')
```

MongoDB:

```
mycol.update_one({'address': 'YANG'}, { "$set": { "address": "Canyon 123" } })
```

MySQL:

```
mycursor.execute(
    "UPDATE customers SET address = 'YANG' WHERE address = 'ZHE' "
)
```

MySQL and MongoDB show its simplicity in update operations, MongoDB has more coding intuitionally and attractive, it's similar as a dictionary operation in python.

Delete					
	10	100	1000	10,000	160,000
DynamoDB	0.07967	0.07620	0.06236	0.07232	
MongoDB	0.04409	0.04737	0.04202	0.41328	0.26987
MySQL	0.16360	0.20802	0.15266	0.10395	0.32579

DynamoDB is very stable in delete operations, no matter the size of file. MongoDB has the best performance among three databases and its result is stable. Two NoSQL databases beat MySQL.

Easy use performance:

DynamoDB

```
table.delete_item(Key={'name': 'ZHE', 'address': 'YANG'})
```

MongoDB:

```
mycol.delete_one({'name': 'CHAO'})
```

MySQL

```
mycursor.execute("delete from customers where name = 'CHAO'")
```

In the delete operation, all syntax are easy to implement.

Discussion

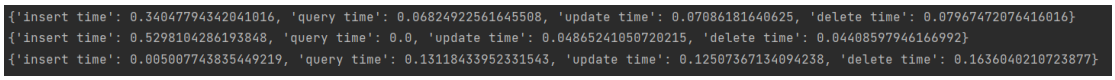
- Deficit: DynamoDB has no result in the test when data size reaches to 160,000, it can be explained by network reason, or it's not the right use condition. DynamoDB is excellent for scalability and caching. What's more, data in the experiment is still too small to big data level, so it doesn't present the great power for NoSQL in handling big data. What's more, scenarios in the experiment are focused on common use, data type is simple, no image, no JSON files, and no time requirements
- Overall, MySQL shows its performance in data loading stage, it beats other two NoSQL databases. Although its advantage derived from local host, but its boundedness is also about local host, current business industries are facing cloud service, based on business need of high-speed data throughput, distributed customers. For personal users and small business, MySQL can meet their requirements. NoSQL shows it ability in the query, update and delete stages, which means they achieve better performance in the data interactions, they can meet complex tasks and keep consistence. Working in the cloud means it has excellent scalability and fault-tolerance, cloud serve will monitor the database and enable backup if needed.

Tables and Figures

Running results

File size 10

	Insert	Query	Update	Delete
DynamoDb	0.34048	0.06825	0.07086	0.07967
MongoDb	0.52981	0.0	0.04865	0.04409
MySQL	0.00501	0.13118	0.12507	0.16360

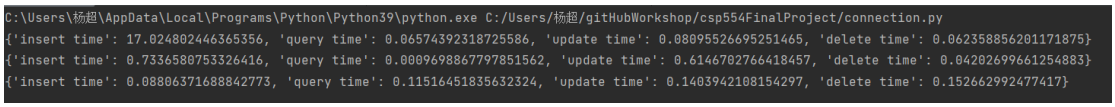


File size 100

	Insert	Query	Update	Delete
DynamoDb	0.65674	0.08054	0.07199	0.07620
MongoDb	0.54626	0.0	0.70906	0.04737
MySQL	0.01451	0.12080	0.18523	0.20802

File size 1000

	Insert	Query	Update	Delete
DynamoDb	17.0248	0.06574	0.08096	0.06236
MongoDb	0.73366	0.00097	0.61467	0.04202
MySQL	0.08806	0.11516	0.14039	0.15266



File size 10,000

	Insert	Query	Update	Delete
DynamoDb	696.31155	0.07541	0.06966	0.07232
MongoDb	2.00388	0.0	0.46761	0.41328
MySQL	0.70242	0.12391	0.11267	0.10395

File size 160,000

Appendix

1. “Document Database - Nosql.” MongoDB, https://www.mongodb.com/document-databases.
2. NoSQL vs SQL databases (no date) MongoDB. Available at: https://www.mongodb.com/nosql-explained/nosql-vs-sql (Accessed: October 16, 2022).
3. Dearmer, Abe. “Firebase vs. Mysql: A Database Comparision.” *Integrate.io*, Integrate.io, 18 Nov. 2020, https://www.integrate.io/blog/firebase-vs-mysql/#:~:text=Firebase%20and%20MySQL%20Differences,-There%20are%20various&text=Firebase%20uses%20NoSQL%3B%20MySQL%20uses,data%3B%20MySQL%20has%20predefined%20schemas.
4. Getting Started with Dynamodb and the AWS Sdks - Amazon Dynamodb. https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GettingStarted.html.
5. Github code repository: cyangIIT/csp554FinalProject (github.com)