Performance Comparison for accessing Product Catalog in Relational vs. MongoDB database systems

Kartik Bansal
Heinz College of Information Systems and Public Policy
Carnegie Mellon University
Pittsburgh, USA
kartikbansal@cmu.edu

***Abstract* - In this paper, we take the scenario of fetching product information for an online retailer. This scenario deals with the highest load in terms of data pull in comparison to all other aspects of the website. We have designed and conducted an experiment to compare the performance of Relational Databases with MongoDB NoSQL databases for storing and fetching product information. The MongoDB database pulled data in almost 1/10th time in comparison to the Oracle SQL database. This was expected as documents in MongoDB allow data to be stored at a single place rather than spread across multiple tables and hence data has to be read just from a single place.**

I. Introduction

The E-Commerce sector has seen a boom in the last decade. This is apparent from the news of Amazon hitting a valuation of $1 Trillion in Q3 of 2018 (1). This has been possible because of the gradual shift of customers from brick and motor shops to the online retail platform. Due to the convenience of ordering from home, more variety, and 24hr availability, we see an ever-increasing number of people ordering online. With more than 200 million online shoppers in the US (2) alone, it becomes very important to handle the high volume of requests that too with acceptable speed. E-commerce also has a large variety of data. This variety has been increasing constantly to provide better customer experience. Therefore, along with speed, it also becomes important to store data in a way that updating the data doesn't harm its availability.

II. Data Storage Options

A. Relational Data Models

The relational model means that the logical data structures - the data tables, views, and indexes - are separate from the physical storage structures. This separation means that database administrators can manage physical data storage without affecting access to that data as a logical structure.

To ensure that data is always accurate and accessible, relational databases follow certain integrity rules. Some of the main features of relational databases are Data Consistency, Commitment, and Atomicity, Stored Procedures, Database Locking, and Concurrency. (3) To avoid data redundancy and modification issues, relational databases are usually normalized into multiple tables depending on their purpose. Thus, pulling data requires joining multiple tables together.

B. NoSQL Databases

NoSQL databases store data in a format other than relational tables. NoSQL databases come in a variety of types characterized by their data model. Examples include document, key-value, wide-column, and graph. They typically provide flexible schemas and the ability to easily scale with large amounts of data and high user loads. NoSQL data models allow related data to be nested within a single data structure.

Types of NoSQL Databases –

- Document databases store data in documents similar to JSON (JavaScript Object Notation) objects. Each document contains pairs of fields and values. The values can typically be a variety of types including things like strings, numbers, booleans, arrays, or objects, and their structures typically align with objects developers are working within code
- Key-value databases are a simpler type of database where each item contains keys and values. A value can typically only be retrieved by referencing its value, so learning how to query for a specific key-value pair is typically simple
- Wide-column stores store data in tables, rows, and dynamic columns. Wide-column stores provide a lot of flexibility over relational databases because each row is not required to have the same columns. Many consider wide-column stores to be two-dimensional key-value databases
- Graph databases store data in nodes and edges. Nodes typically store information about people, places, and things while edges store information about the relationships between the nodes (4)

III. Experiment Setup

Product Catalog data is one of the most frequently accessed data for an e-commerce site. This is expected because only around 3% of customers go on to buy a product from its product page. Thus, other pages like account information, cart, payment page are accessed comparatively less frequently. This makes it important to store product data in a way that maximizes the availability and fetch speed of the data.

Since the inception of e-commerce, they have used relational databases to store the product catalog data. A relational database is still a common choice when it comes to building the data layer for any e-commerce platform. These databases give utmost importance to consistency. This worked well until the database started to see a high number of data requests. The systems failed to meet the demand, especially during peak times. Due to this, companies started to increase their tolerance towards data inconsistencies to improve the read and write performance under highly concurrent conditions. (5)

In Product Catalog systems, each product has 1:Many relationship with the reviews i.e. each product comprises of multiple reviews. This means that the data has a hierarchical structure to it. This would make MongoDB ideal to store such data as it supports the embedding of documents within a document. Also, product catalogs must have the capacity to store many different types of objects with different sets of attributes. These kinds of data collections are quite compatible with MongoDB's data model (6). This makes MongoDB an ideal choice for building a Product Catalog system in comparison to other NoSQL databases.

We have chosen Oracle as the relational database in this comparison because it is the most widely used database in the world. It is also very feature-rich especially because it has been in the market since the 1970s and continuous improvements have been made on it.

Therefore, the objective of this research paper is to compare the Oracle database with the MongoDB NoSQL database.

IV. Data Used

The data used to perform the experiment is of a Brazilian e-commerce website that has been sourced from Kaggle (7). Thus, the experiment will try to mimic the product catalog system of a real-world e-commerce platform. The dataset contains products, orders, and reviews data. The dataset is explained in much more detail below –

A. *Product Listings Data*

This dataset includes product characteristics. It comprises of 32340 products. This table is the primary table of a product catalog system and therefore needs to be accessed most frequently. A sample row is shown in Table 1.

TABLE I. Product Listing Example

Columns	Sample Values
Product ID	1e9e8ef04dbcff4541ed26657ea517e5
Product Name	Perfume ABC
Product Category	Perfumes
Product Description	ABC
Product Photos Qty	2
Product Weight Gms	225
Product Length	16
Product Height	10
Product Width	14

B. *Orders Data*

The dataset contains order information for 99442 orders made on the website. This table contains ProductID as a foreign key. This table will be useful for mapping the reviews and ratings given for each order to its product. A sample row is shown in Table 2.

TABLE II. Order Example

Columns	Sample Values
Order ID	e481f51cbdc54678b7cc49136f2d6af7
Customer ID	9ef432eb6251297304e76186b10a928d
Order Status	Delivered
Order Purchase Date	mm/dd/yyyy
Order Approved Date	mm/dd/yyyy
Order Delivered Carrier Date	mm/dd/yyyy
Order Delivered Customer Date	mm/dd/yyyy
Order Estimated Delivery Date	mm/dd/yyyy

C. *Reviews Data*

The dataset contains 10000 ratings and reviews for a given order product combination. This table contains OrderID as a foreign key. This table can be mapped to a specific product using the order table. As a product page displays ratings and reviews information along with the product details, this table is also required to be accessed very frequently. A sample row is shown in Table 2.

TABLE III. Review Example

Columns	Sample Values
Product ID	1e9e8ef04dbcff4541ed26657ea517e5
Order ID	73fc7af87114b39712e6da79b0a377eb
Review Score	4
Review Comment Title	ABC
Review Comment	ABC
Review Date	mm/dd/yyyy
Review Upvotes	16

V. Database Design

The approach for designing the database varies depending on the database being tested. This is done to ensure that we meet all the constraints imposed by the database while making sure that we leverage their strengths. The approach for both the database is described in detail below –

A. *Relational Databases*

We have 3 separate tables for Product Info, Orders, and Reviews. The ProductID is the primary key for the Product Table which is the foreign key in the Orders Table. The Order table is used to map the Ratings and Reviews with their

respective product. This is how e-commerce platforms typically store product information. This is done to reduce data redundancy and manipulation issues. This architecture is shown in Figure 1.

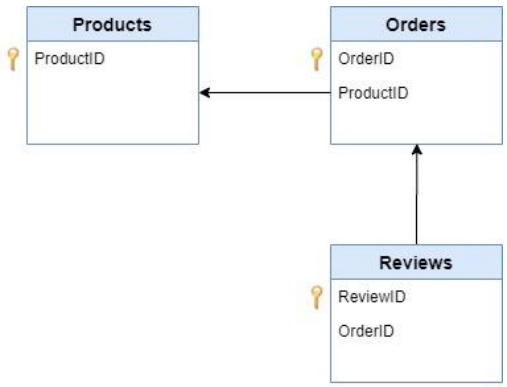

Fig. 1. Database design for Oracle Relational Database for the Experiment

B. *MongoDB*

We design the database in a way that leverages the tree hierarchical storage of MongoDB i.e. storage of document within a document.

This was done by embedding Orders and Reviews data in the Product data. Thus, a single document of Product will comprise of all the information required to populate the product page. This is illustrated in Figure 3.

Fig. 2. Database design for MongoDB Database for the Experiment

VI. Scenarios and Queries

The systems will be tested for two scenarios. The first scenario is of pulling information for hundreds of products. This is to simulate the scenario of populating home page and product search pages with product information. The second scenario is of pulling single product information multiple times. This will simulate the scenario of people accessing a single product page.

1) Hundreds of products: This will again comprise of two types of queries –

a) Homepage: The home page is mostly populated as a combination of products belonging to a category. Thus, the query for this will filter for top products in the category based on rating. The queries for Oracle and MongoDB databases are shown in figures 3 & 4 respectively.

```
SELECT
    Productid,
    Categoryname,
    Productname,
    Productdescription,
    Productphotosqty,
    Productweightgms,
    Productlengthcms,
    Productheightcms,
    Productwidthcms,
    Reviewid,
    Reviewscore,
    Reviewcommenttitle,
    Reviewcommentmessage,
    Reviewcreationdate
FROM
    Productinfo
    JOIN Orderitemsinfo USING (ProductID)
    JOIN Reviewinfo USING (OrderID)
WHERE
    Categoryname = 'perfumaria'
ORDER BY Reviewscore DESC;
```

Fig. 3. SQL Query to pull products belonging to a category

```
db.ProductInfoAll.find(
{"CategoryName": "perfumaria"},
{_id:0,
"ProductID":1,
"Categoryname":1,
"ProductName":1,
"Productdescription":1,
"Productphotosqty":1,
"Productweightgms":1,
"Productlengthcms":1,
"Productheightcms":1,
"Productwidthcms":1,
"Reviewid":1,
"Reviewscore":1,
"Reviewcommenttitle":1,
"Reviewcommentmessage":1,
"Reviewcreationdate":1}
).sort({"Reviews.Score":-1})
```

Fig. 4. MongoDB Query to pull products belonging to a category

b) Search page: The search page is populated based on searching the products based on its name. The queries for MySQL and MongoDB are shown in figures 5 & 6 respectively.

```
SELECT
    Productid,
    Categoryname,
    Productname,
    Productdescription,
    Productphotosqty,
    Productweightgms,
    Productlengthcms,
    Productheightcms,
    Productwidthcms,
    Reviewid,
    Reviewscore,
    Reviewcommenttitle,
    Reviewcommentmessage,
    Reviewcreationdate
FROM
    Productinfo
    JOIN Orderitemsinfo USING (ProductID)
    JOIN Reviewinfo USING (OrderID)
WHERE
    Productname Like 'moveis%'
ORDER BY Reviewscore DESC;
```

Fig. 5. SQL Query to pull products based on search keywords

```
db.ProductInfoAll.find({"CategoryName": /^moveis/},
{_id:0,
"ProductID":1,
"Categoryname":1,
"ProductName":1,
"Productdescription":1,
"Productphotosqty":1,
"Productweightgms":1,
"Productlengthcms":1,
"Productheightcms":1,
"Productwidthcms":1,
"Reviewid":1,
"Reviewscore":1,
"Reviewcommenttitle":1,
"Reviewcommentmessage":1,
"Reviewcreationdate":1}
).sort({"Reviews.Score":-1})

```

Fig. 6. MongoDB Query to pull products based on search keywords

2) Single product: In this scenario, the product will be pulled using its product id alone. This is to replicate the scenario when a user clicks on a product listing to open the product page. Hence, the product id will be known while pulling the data. This scenario will be tested in the following two ways –

a) Pulling multiple products at once – We will pull data for 5 randomly selected products. As a product page also shows similar or related products, this will simulate the data request for populating a product page more accurately

b) Pulling one product multiple times – Here, we will run the previous query 10 times sequentially. This is to simulate the scenario of the server receiving multiple product requests from different users.

As the queries for part a) and b) are the same, only queries for part a) are shown for MySQL and MongoDB databases in figures 7 & 8 respectively.

```
SELECT
    Productid,
    Categoryname,
    Productname,
    Productdescription,
    Productphotosqty,
    Productweightgms,
    Productlengthcms,
    Productheightcms,
    Productwidthcms,
    Reviewid,
    Reviewscore,
    Reviewcommenttitle,
    Reviewcommentmessage,
    Reviewcreationdate
FROM
    Productinfo
    JOIN Orderitemsinfo USING (ProductID)
    JOIN Reviewinfo USING (OrderID)
WHERE
    Productid = '1e9e8ef04dbcff4541ed26657ea517e5'
    OR Productid = '46805be02f14add01c364b5ca18959ec'
    OR Productid = 'cfbd417c9c7f510b60243124a6489883'
    OR Productid = 'c5f5b4b17e4cd989c39986cd0dddc018'
    OR Productid = '58bf57007fcbbcc3c327d9de8cf5dbf8';


```

Fig. 7. SQL Query to pull products using the Product ID

```
db.ProductInfoAll.find({ $or:[
{"ProductID": "1e9e8ef04dbcff4541ed26657ea517e5"},
{"ProductID": "46805be02f14add01c364b5ca18959ec"},
{"ProductID": "cfbd417c9c7f510b60243124a6489883"},
{"ProductID": "c5f5b4b17e4cd989c39986cd0dddc018"},
{"ProductID": "58bf57007fcbbcc3c327d9de8cf5dbf8"},
]},
{_id:0,
"ProductID":1,
"Categoryname":1,
"ProductName":1,
"Productdescription":1,
"Productphotosqty":1,
"Productweightgms":1,
"Productlengthcms":1,
"Productheightcms":1,
"Productwidthcms":1,
"Reviewid":1,
"Reviewscore":1,
"Reviewcommenttitle":1,
"Reviewcommentmessage":1,
"Reviewcreationdate":1}
)

```

Fig. 8. MongoDB Query to pull products using the Product ID

For the experiment, we tested the performance of the two databases by measuring the time taken for the execution of the query. To make sure that the comparison of performance is fair, similar conditions were maintained for the Oracle SQL and MongoDB databases. These are mentioned in detail below –

1) Oracle Database: In the case of the Oracle database we have set the following testing procedure -

- Time is measured by using dbms_utility.get_time attribute which is an inbuilt attribute in SQL commonly used to measure the time of a query
- Display of results on the console has been set to off by using command – SET TERMOUT OFF
- To make the query execution time comparable to MongoDB, all the results are saved in Spool instead of showing on the interface
- ProductID column is the primary key for Product Listings data

2) MongoDB: In the case of the Oracle database we have set the following testing procedure -

- Time is measured by using built-in method db.collection.explain("executionStats") (8)
- The display of results on the console is by default off when using the explain method
- An index has been created on ProductID column

VII. Results

Results for all the different scenarios are as follows –

1) Hundreds of products: This comprised of the following two scenarios –

a) Homepage: MongoDB pulled the results in almost 1/8th of the time on average in comparison to SQL. The query run times in milli-seconds [ms] for SQL and MongoDB are shown in Table IV.

TABLE IV. HOMEPAGE SCENARIO

Metric	*Database*	
	Relational Databases	***MongoDB***
Average Latency [ms]	232.0	28.8
Min latency [ms]	220.0	24.0
Max latency [ms]	270.0	31.0

b) Search Page: MongoDB pulled the results in almost $1/12^{th}$ of the time on average in comparison to SQL. The query run times in milli-seconds [ms] for SQL and MongoDB are shown in Table V.

TABLE V. SEARCH SCENARIO

Metric	*Database*	
	Relational Databases	***MongoDB***
Average Latency [ms]	556.0	42.8
Min latency [ms]	540.0	41.0
Max latency [ms]	570.0	45.0

2) Single Product: MongoDB pulled the results in almost $1/40^{th}$ and $1/30^{th}$ of the time on average in comparison to SQL in both the scenarios. The query run times in milli-seconds [ms] for SQL and MongoDB are shown in Table VI.

TABLE VI. SINGLE PRODUCT SCENARIO

Metric	*Database*			
	Relational Databases		***MongoDB***	
	Once	***Ten Times***	***Once***	***Ten Times*** [a]
Average Latency [ms]	24.0	214.0	0.4	6.4
Min latency [ms]	10.0	200.0	0.0	1.2
Max latency [ms]	30.0	220.0	1.0	14.2

[a.] Time calculated by subtracting the time it takes to connect to Mongo server

The query run times might vary greatly depending on the system specifications and program version of Oracle and MongoDB. Also, the actual systems are much more complicated that the experiment conducted here. These test does not account for processing of the data as well other overheads such as data maintenance and security.

VIII. ANALYSIS OF THE RESULTS

The MongoDB database pulls information in almost $1/10^{th}$ of the time on an average when pulling data for hundreds of products in comparison to the Oracle relational database. For a single product, this difference increases to close to $1/30^{th}$. This difference is very likely to increase when SQL join many more tables provide the product information. E.g. joining product stock information to show product availability on the page.

These results were expected as MongoDB did not require multiple joins to fetch the required attributes. With all the data for a product stored in a single document, rather than spread across multiple relational tables, the database only needs to read and write to a single place. Also, having the data for an object in one place makes it easier for developers to understand and optimize query performance. But storing the data this way might not be optimal for fetching data in other scenarios. Hence, we might have to create databases in MongoDB tailor made for each scenario. This will increase the storage requirements of the system.

Another important aspect of documents that was not tested here is their flexibility. We can add new attributes when we need to, without having to alter a centralized database schema. Changing schema causes downtime or significant performance overhead in a relational database like MySQL. (9) This is one of the main advantages of No SQL databases over a traditional database especially in e-commerce as product storage requirements change quite frequently.

IX. CONCLUSION

All the tests show that MongoDB database provides much faster read times in comparison to the Oracle SQL database. The magnitude of difference might vary in an actual system, but the direction should remain same. This speed comes with a hit on consistency. It is important for the database architect to consider the pros and cons of both the systems.

The speed requirements for all the e-commerce platforms will keep on increasing, as more and more users make a transition to online shopping. Therefore, the retailers should make a timely assessment of their current system to see if it meets the data requirements or not. If not, they should consider transitioning to a suitable NoSQL database as one of the solutions.

REFERENCES

[1] "Amazon Hits $1 Trillion Valuation" Laura Stevens and Amrith Ramkumar [Online] Available: https://www.wsj.com/articles/amazon-hits-1-trillion-valuation-1536075734 [Accessed: 11 April 2020]

[2] "Number of digital shoppers in the United States from 2016 to 2021" [Online] https://www.statista.com/statistics/183755/number-of-us-internet-shoppers-since-2009/ [Accessed: 11 April 2020]

[3] "What a Relational Database Is" [Online] Available: https://www.oracle.com/database/what-is-a-relational-database/

[4] "NoSQL Databases Explained" [Online] Available: https://www.mongodb.com/nosql-explained [Accessed: 12 April 2020]

[5] "All Things Distributed" [Online] Available: https://www.allthingsdistributed.com/2008/12/eventually_consistent.html [Accessed: 12 April 2020]

[6] "Use Cases of MongoDB: Product Catalog" [Online] Available: https://docs.mongodb.com/drivers/use-cases/product-catalog [Accessed: 16 April 2020]

[7] "Brazilian E-Commerce Public Dataset by Olist" [Online] Available: https://www.kaggle.com/olistbr/brazilian-ecommerce#olist_products_dataset.csv [Accessed: 25 April 2020]

[8] "Analyze Query Performance" [Online] Available: https://docs.mongodb.com/manual/tutorial/analyze-query-plan/ [Accessed: 26 April 2020]

[9] "MongoDB vs. MySQL" [Online] Available: https://www.mongodb.com/compare/mongodb-mysql [Accessed: 26 Apr 2020]