Performance Analysis of Relational and Non-Relational Database Management Systems for Searching Attributes in a Web Application

Lucas Dohmann, Jason Oltzen, Mert Torun, Luca Uckermann

Abstract—**This report presents the results of a research project focused on investigating the effectiveness and performance of different methods for searching algorithms within a database. The goal of the project is to compare the performance of relational and non-relational database management systems. The research question addressed in this study is whether there are measurable or noticeable differences between various search methods and which method performs best. The project uses the Python-based web framework Django and the Google Cloud Platform to implement a coupon web application. Performance data on different search methods and database management systems will be collected and analyzed to determine the most effective approach. The results of this study will provide insight into optimizing database performance in web applications and identifying the most efficient search methods.**

Index Terms—**Cloud, Large-Scale system, Database models, Performance**

Contents

I. Introduction

TODAY, many businesses use coupons to attract customers and increase sales. With the rapid growth of online shopping and e-commerce in recent years, coupon web applications have become increasingly popular and important to reach a wider audience. Due to the large number of coupons available, it is becoming increasingly difficult for consumers to keep track of them. Therefore, it is important to provide users with an effective search method.

This project investigates the effectiveness of different search methods using the attributes of coupons stored in a database. The performance of relational and non-relational database management systems is compared in terms of response times. The coupon web application will be implemented using Python and a cloud platform. Data will be collected on the response times of different search methods and database management systems. The results will then be analyzed to determine which method performs best. The goal of this project is to identify the most effective search methods and how to optimize database performance in web applications.

The following sections of this report provide a review of the relevant literature, the system architecture of the web application prototype, the results, and the final conclusion.

II. Research Question

A. Motivation

When working with databases, one is likely to come across the debate of which is the most effective type of database management system in terms of storing and managing data. There are two main types: relational database management systems and non-relational database management systems. The choice of database management system can significantly affect the performance of an application. Therefore, the focus of this project is to investigate the effectiveness and performance of different database query methods.

B. Formulation

Are there measurable and noticeable differences between different methods of searching the coupon attributes within the database management system, and which method performs best? In terms of response times, is a relational or non-relational database management system the best option?

C. Approach

To answer the research question, the Python-based web framework Django and the Google Cloud Platform are used. The application allows users to upload a coupon, write comments under coupons, create hashtags, vote up and down a coupon, and see the score of the related coupon (see figure 1). A main aspect of the application is searching for a specific coupon based on a hashtag. Both relational and non-relational database management systems are implemented to store the coupon data and to compare their performance in terms of response times for each search method.

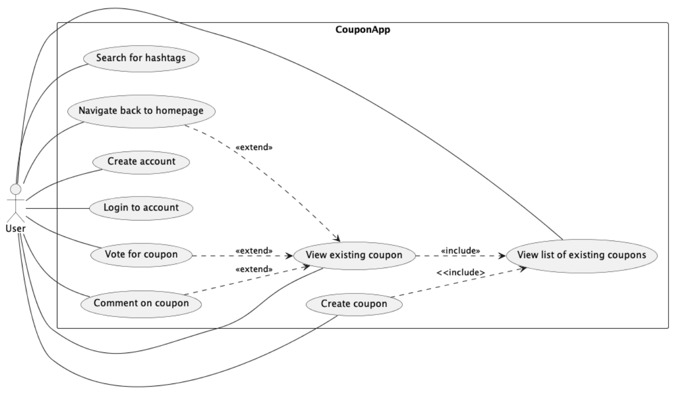

Fig. 1. UML use case diagram - Coupon application

Performance testing tools allow to simulate many users accessing the coupon web application and measure response times for different search queries. After data collection, the mean response times and standard deviations for each search method and database management system are compared to determine if there are significant differences in performance.
The general approach is to develop a prototype coupon web application, test its performance using performance testing tools, and analyze the data collected from these tests to identify the most effective search methods and database management systems. The goal is to provide valuable insights for companies looking to optimize their web applications and improve the user experience for their customers.

III. Preliminary or Related Work

This section introduces related work. The coupon application is inspired by the Mydealz web application [2]. This project implements a lightweight version of Mydealz. Mydealz is an online platform where users can post and vote on deals, discounts, and coupons from different industries.

In the article ”Enhanced query processing over semantic cache for cloud based relational databases” [5], published in April 2020, the authors proposed the approach to improve the hit rate of the cache system by using an implicit semantic matching algorithm, which includes four different scenarios, each designed to optimize the cache hit rate based on the query predicate and attribute matching. Furthermore, they evaluated different experiments, which showed that it outperformed existing techniques in terms of time complexity and hit rate. The experiments were conducted on a prototype system that included a database management system and were cached on the same device.

The paper ”Integrated Environment Based on Anytime Solution Search Algorithms and A Non-Relational Database for Real-Time Intelligent Systems” [4] describes an integrated environment for real-time intelligent systems that uses anytime solution search algorithms and a non-relational database. The use of anytime algorithms and NoSQL databases improves the efficiency of working with graph-based database management systems and speeds up the process of finding a solution.

The article ”BANKS: Browsing and Keyword Searching in Relational Databases” [3] discusses Browsing And Keyword Searching (BANKS), which allows easy publishing of relational and XML data on the web. Relational database management systems require knowledge of the schema and query language for a query, which is used to retrieve specific information from a database when needed, making it difficult for casual users to perform searches. An alternative would be an HTML form that allows users to use predefined queries, but creating forms for each task is tedious and confusing. The best solution would be to create web search engines that provide a simpler alternative through unstructured querying and browsing using keywords and hyperlinks. Keyword search can provide a user-friendly mechanism for casual users to access database information without requiring schema knowledge.

IV. Prototype System Architecture

A. Key Design Decisions

B. System Context

The system is described based on the web-based system architecture from the lecture (see figure 2) reduced to the key components used in the coupon application.

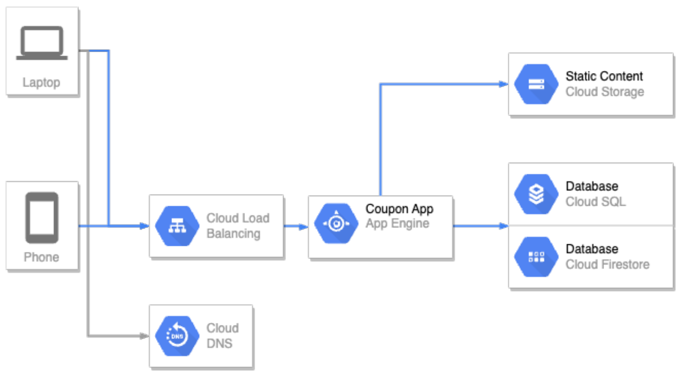

Fig. 2. Web-based system architecture

C. Component View

The Component View diagram, shown in figure 3, provides an overview of the different components that make up the coupon application. The diagram shows that the application is

Decision	Category	Explanation and Rationale
Django as backend framework	Technology	Coupon application is built using the Django web framework as backend. Django provides a robust set of tools and features for web development, including ORM and authentication, which will speed up development and ensure maintainability.
Google Cloud SQL	Technology	The coupon application uses Google Cloud SQL as its relational database. Google Cloud SQL provides the scalability, managed backups and high availability that are critical for a production-ready system.
Google Firestore	Technology	In addition to Google Cloud SQL, the coupon application also uses Google Firestore as a NoSQL/non-relational database. Firestore is a flexible data model and its real-time updates make it ideal for storing dynamic and unstructured data efficiently.
RESTful API architecture	Architectural Style	The coupon application follows a RESTful API architecture to enable interoperability and flexibility. RESTful APIs provide standard HTTP methods and can be easily consumed by different clients (for example, companies that want to automatically upload their coupons to the coupon application), ensuring compatibility and ease of integration.

TABLE I
KEY DESIGN DECISIONS

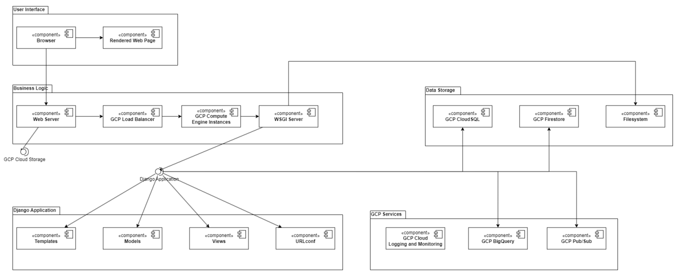

Fig. 3. Component view

divided into three main components: the client-side, the server-side and the database. The client-side component includes the user interface and the web browser, while the server-side component includes the web server and the application server. The database component includes the database management system and the database itself. The diagram also shows the communication channels between the different components, indicating the flow of data and requests between them. Overall, the Component View diagram provides a high-level view of the coupon application architecture and the various components that make it up.

Component(s)	Explanation
Laptop Phone	The devices of the users can alternate. Laptops as well as phones can access the coupon application through the web browser.
Cloud Load Balancing	The load balancer acts as a traffic distributor, evenly distributing incoming client requests across multiple application server instances. It helps ensure high availability, scalability and efficient resource utilization.
Cloud DNS	The Domain Name System service publishes the domain names to the global DNS.
App Engine	The application engine uses the Django web framework, to handle routing, data retrieval and response generation.
Cloud Storage	The cloud storage stores the static files of the application, such as JavaScript and CSS files, so the user is able to access them.
Database	The databases are a relational database and a non-relational database. The relational database is an instance of GCP's Cloud SQL using PostgreSQL. The non-relational database is Google Cloud Firestore.
Data flow between browser client and application server	The browser client communicates with the application server using the HTTP(S) protocol. It sends requests to retrieve web pages, submit forms, or interact with the application's APIs. The application server responds with the appropriate HTML, CSS, or JavaScript files to render in the browser.
Data flow from App Engine to Databases	The engine connects to the databases to retrieve and store data. This connection is established using an appropriate database driver or library, such as Django's database connector for relational databases like PostgreSQL.
Search indexing	For search, the popular GCP storage service Google Cloud Firestore is used.

TABLE II
WEB-BASED SYSTEM ARCHITECTURE

D. Data Schema

The figure 4 represents the data schema of the coupon application and has four tables: ”user”, ”coupon”, ”comment” and ”hashtag”.

1) Users table:
   - ”id” of type integer, which serves as the primary key to uniquely identify each user.
   - ”username” of type varchar, stores the username of the user.
   - ”first_name” of type varchar, stores the first name of the user.
   - ”last_name” of type varchar, stores the surname or last name of the user.
   - ”email” of type varchar, stores the email address of the user.
   - ”password” of type varchar, stores the password associated with the user’s account.
   - the remaining attributes are implemented by Django and are not relevant for this project.
2) Coupon table:
   - ”name” of type varchar, stores the name or title of the coupon.
   - ”expiring_date” of type timestamp, represents the expiration date of the coupon.
   - ”discount_amt” of type integer, stores the amount of the total discount in Euro offered by the coupon.
   - ”score” of type integer, represents the score or rating associated with the coupon.
   - ”code” of type varchar, stores the coupon code which is needed to redeem it
   - ”comments_amt” of type integer, stores the number of comments the coupon has
3) Comment table:
   - ”text” of type varchar, stores the actual content or text of the comment.
   - ”created_date” of type timestamp, represents the creation date of the comment
4) Hashtag table:
   - ”name” of type varchar, stores the name or text of the hashtag.

   The diagram also contains relationships between these tables indicated by the reference lines:
   - The ”user” table is referenced by the ”user_id” column in both the ”comment” and ”coupon” tables.
   - The ”coupon” table is referenced by the ”coupon_id” column in both the ”comment” and ”hashtag” tables.
   - The ”coupon” table is referenced by the ”coupon_id” column in the ”hashtag” table.

E. Runtime View

The Runtime View, shown in figure 5, is a UML activity diagram that provides an overview of the main use cases of the coupon application. The diagram shows the different activities that occur during the use of the application, including creating a new coupon, uploading a coupon and creating a new comment. The arrows between the different activities indicate the flow of control between them, showing the sequence of steps that occur during each use case. Overall, the Runtime View provides a high-level view of the functionality of the coupon application and the different activities that occur during its use.

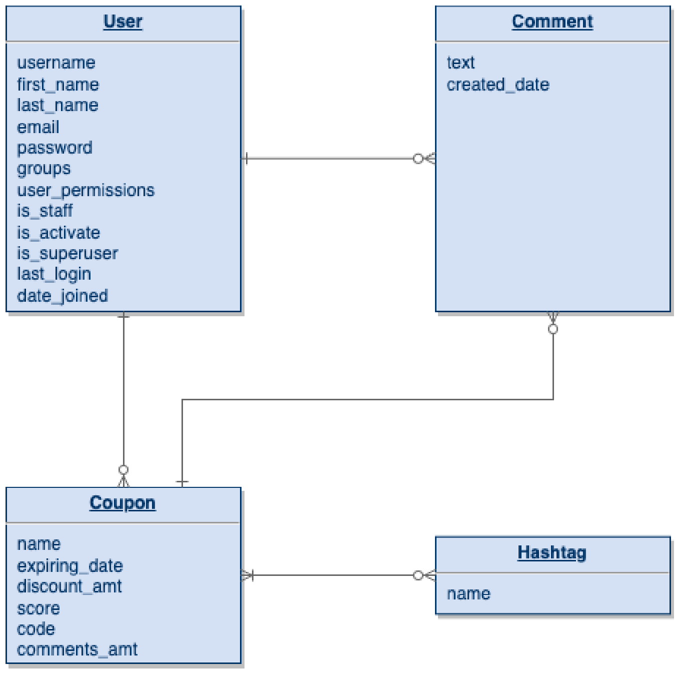

Fig. 4. Data Schema View

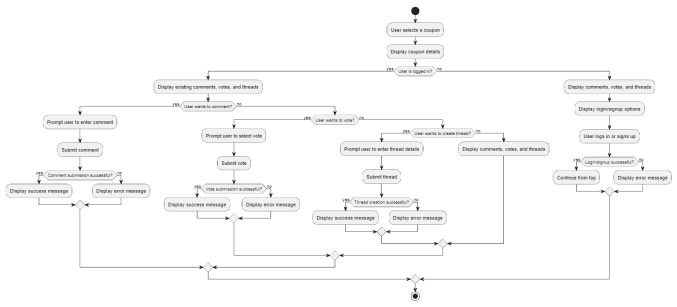

Fig. 5. Runtime View

F. Crosscutting Concepts

Crosscutting concept	Category	Explanation and Rationale
Caching	Performance	Caching can be used to store frequently accessed coupon data in memory, reducing the number of database queries and improving application performance. Django provides built-in caching support through its cache framework and it can be easily configured to work with CloudSQL and Cloud Firestore.
Scaling	Scalability	Scaling allows for the application to handle increasing amounts of traffic and data. Both CloudSQL and Cloud Firestore provide scaling options, such as read replicas and sharding, which can be used to distribute the load across multiple database instances.
Authentication	Security	Authentication can be used to ensure that only authorized users are able to access and modify coupon data. Django provides built-in authentication support through its authentication framework, which can be extended to work with external authentication providers such as Google OAuth.
Search functionality	User experience	Search functionality allows users to quickly find relevant coupons based on specific criteria. Cloud Firestore offers advanced search capabilities, such as full-text search and complex queries, which can be used to implement a robust search functionality.
Load testing and simulation	Operational Concepts	To simulate high levels of traffic and user activity on the coupon application, which allows to identify and resolve performance bottlenecks before they become critical, tools such as the Google Cloud Load Testing can be used to generate realistic load tests .
Database monitoring	Operational Concepts	Database monitoring involves tracking and analyzing the performance and health of a database to ensure optimal operation and to identify and resolve issues such as slow queries, high resource usage, or errors. For CloudSQL on Google Cloud Platform, tools such as Cloud Monitoring and Cloud Logging provide insight into database performance, availability and utilization.

V. Measurements and Findings

A. Experimental Setup

To test the different database architectures of the coupon application, the tool ”Locust.io” [1] is used. The following different scenarios are tested to measure the performance of the web application.

1) Simple Read: Fetch a single document or row based on a unique identifier.
2) Simple Write: Insert a single document or row with predefined data.
3) Complex Query: Extract data based on multiple conditions, involving JOINs (for PostgreSQL) or aggregations (for MongoDB).
4) Bulk Insert: Insert 1,000 rows or documents with varying data patterns.

The scenarios are tested on the one hand with PostgreSQL and on the other hand with MongoDB. The databases are compared in terms of execution times for simple read and write operations (fetch or insert single coupon), complex queries (searching coupons by hashtag), and bulk insert tasks (insert multiple coupons).

B. Measured results

	MongoDB	PostgreSQL
Simple Read Operation (one coupon) (ms):	0.72	0.54
Simple Write Operation (one coupon) (ms):	1.24	2.31
Complex Query (searching multiple coupons by hashtag) (ms)	40.36	37.22
Bulk Insert (inserting 1.000 coupons) (ms)	137.95	83.89

TABLE III
Results for both databases

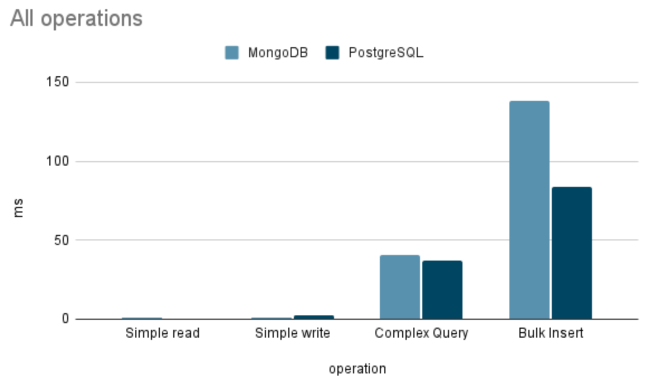

Fig. 6. Results all operations

Table III and figures 6 & 7 show the measured results. Both applications are much slower when performing more extensive tasks such as complex queries and bulk inserts than when performing simple tasks such as read and write operations. The bulk insert requires nearly 200 times more time, so the magnified figure 7 is needed to correctly display the simple read and write operations.

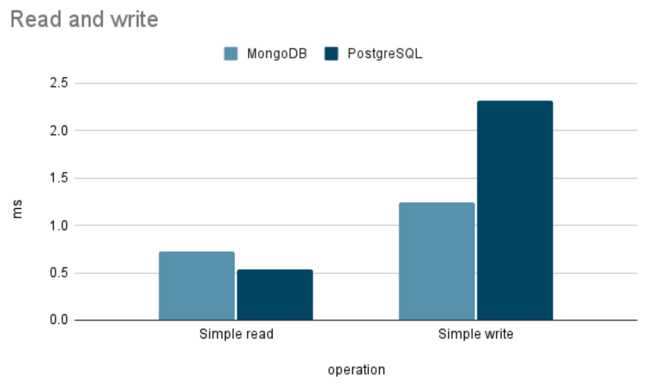

Fig. 7. Results read and write operations

VI. Conclusion

This project looked at two popular databases: MongoDB and PostgreSQL, to see which is better for the coupon web application.

The tests show that both have their strengths. PostgreSQL is faster at reading data, working with complex queries and adding a lot of data at once. On the other hand, MongoDB is faster when adding single pieces of data.

While performance metrics provide insight and help evaluate these databases, the choice between MongoDB and PostgreSQL should be made on a project-by-project basis, taking into account other issues such as the suitability of the data model, scalability requirements, and familiarity of the development team. As the technology evolves, so will the capabilities of these systems. The decision should be driven by the specific needs and context of the application at hand.

List of Tables

List of Figures

References

[1] Locust.io. https://locust.io. Accessed: 2023-08-01.
[2] Mydealz. hhttps://www.mydealz.de/. Accessed: 2023-04-26.
[3] Soumen Chakrabarti Arvind Hulgeri Charuta Nakhe Parag S. Sudarshanxe B. Aditya, Gaurav Bhalotia. Banks: Browsing and keyword searching in relational databases. https://www.sciencedirect.com/science/article/pii/B9781558608696501141. Accessed: 2023-04-26.
[4] Paniavina Eremeeva, Poliushkina. Integrated environment based on anytime solution search algorithms and a non-relational database for real-time intelligent systems. https://ceur-ws.org/Vol-2648/paper26.pdf. Accessed: 2023-04-26.
[5] Atta Rahman Rachid Zagrouba Fahd Alhaidari Tariq Ali Farzana Zahid Munir Ahmad, Muhammad Abdul Qadir. Enhanced query processing over semantic cache for cloud based relational databases. https://link.springer.com/article/10.1007/s12652-020-01943-x. Accessed: 2023-04-26.