Final Report on Search Application Project

Group Number: 13

Team Members: Meiqiao Shi, Sarthak Singh, Phan Nguyen Huong Le, Alvin Alex

Abstract

The Twitter Search Application Project undertaken by Group 13 focuses on developing an advanced, scalable search tool capable of handling large-scale Twitter data sets with high efficiency and responsiveness. This project combines the strengths of both relational (MySQL) and non-relational (MongoDB) databases to manage and analyze Twitter data, highlighting a dual-database approach to effectively handle structured and dynamic data types. Enhanced by strategic caching and indexing techniques, the system supports real-time data processing and complex querying capabilities. This report outlines the project's methodology, from data acquisition during critical events to the implementation of the search application using Flask, showcasing our innovative solutions for real-time social media data analysis.

Introduction

In the realm of social media analytics, the rapid growth of data presents unique challenges, particularly during significant global events. Our project addresses these challenges by developing a robust Twitter Search Application that efficiently manages vast datasets and delivers rapid analytical insights. The system integrates MySQL for structured data like user profiles and tweet metadata, ensuring robust data management and complex querying capabilities. MongoDB is employed for its flexibility in managing unstructured data such as tweet content and metadata, enhancing data retrieval efficiency. Together, these technologies underpin a high-performance search tool that leverages an advanced caching system and a user-friendly interface built with Flask, enabling diverse search functionalities and real-time interaction with data. This introduction outlines our project's approach and objectives, emphasizing our strategic use of technology to optimize real-time social media data analysis.

Dataset and Processing Overview

The dataset, crucial to the development of the Twitter Search Application Project, is a curated collection of Twitter activities during a critical period of the COVID-19 pandemic. It consists of two primary files, corona-out-2 and corona-out-3, each representing data snapshots from distinct dates in April 2020, a time when global attention was heavily focused on the pandemic.

corona-out-2 (Data from April 12, 2020):

- This file contains 18,518 lines of data, encompassing 7,356 original tweets and 11,162 retweets.
- Duplicate tweet IDs (such as 1249406404013764608, and 1249407126847524864) are present, suggesting instances of repeated data collection or retweeting patterns.

corona-out-3 (Data from April 25, 2020):

- This larger file consists of 101,916 lines, including 40,804 original tweets and 61,112 retweets, alongside 32,788 URLs, reflecting a continuation of active information sharing and discussion on the platform.
- Similar to corona-out-2, it also contains duplicates of certain tweet IDs.

Database Structure Overview

Employing a hybrid database approach addresses the need for efficient management of both structured and dynamic data types inherent in the Twitter dataset. Structured data, such as user profiles and key tweet metadata (including elements like tweet ID, creation time, and user ID), are managed using a relational database (**MySQL**). This approach is optimal for these data types due to their fixed schema and the relational database's capability to handle complex queries and joins effectively.

Conversely, the more varied and dynamic nature of the tweet data, specifically the tweet content and associated entities like hashtags and URLs, are handled by **MongoDB**. This non-relational database is adept at managing unstructured data, offering flexibility for the evolving structure of tweets and ensuring efficient retrieval for the search application.

Tweet Data Structure and Storage:

Each tweet's structure includes text content, metadata, interaction data, and unstructured data like hashtags and URLs.

Data Processing for Storage

- **Methodology:** Implemented a Python script for real-time tweet parsing and processing.
- **Data Transformation**: Extracting and transforming data to fit defined schemas before database loading.
- **One at a time:** loaded data one at a time into the databases to replicate the real-life Twitter streaming. Used the collections.insert_one() to insert the tweets one by one for the MongoDB database. For the SQL database, using a self-defined function to process each tweet one at a time

Handling Retweets:

Retweets are managed by storing the original tweet ID, allowing analysis of retweeting behaviors. Retweet counts are taken into account for the original tweet (as retweeting a retweet only increases the retweet count for the original tweet).

Data Models and Schemas

Users Table Schema (Relational Database- MySQL)

Field Name	Data Type	Nullable	Key	Description
user_id	bigint	No	PRI	Unique identifier for the user
name	varchar(255)	Yes		Name of the user
screen_name	varchar(255)	Yes		The Twitter handle of the user
location	varchar(255)	Yes		Location of the user
followers_count	int	Yes		Number of followers

friends_count	int	Yes		Number of friends
statuses_count	int	Yes		Number of tweets posted
created_at	DateTime	Yes		Account creation date

Tweets Table Schema (Relational Database- MySQL)

Field Name	Data Type	Nullable	Key	Description
tweet_id	bigint	No	PRI	Unique identifier for the tweet
user_id	bigint	Yes	MUL	Identifier of the user who tweeted
created_at	DateTime	Yes		Date and time of tweet post.
is_quote_status	tinyint(1)	Yes		Indicates if the tweet is a quote
quote_count	int	Yes		Number of times the tweet was quoted
reply_count	int	Yes		Number of replies to the tweet
retweet_count	int	Yes		Original Tweet Retweet Count.
favorite_count	int	Yes		Number of likes on the tweet
is_retweet	tinyint(1)	Yes		Indicates if the tweet is a retweet
original_tweet_id	bigint	Yes		ID of the original tweet (if retweeted)

Tweets Document Schema (Non-relational: MongoDB)

Field Name	Data Type	Description
_id	ObjectId	A unique identifier generated by MongoDB
tweet_id	long	Unique identifier for the tweet

text	String	Content of the tweet
hashtags	Array of Strings	List of hashtags in the tweet
user	Object	-
├─ **user_id**	long	Identifier of the user who tweeted
├─ **name**	String	Name of the user
└─ **screen_name**	String	The Twitter handle of the user
retweet_count	int	Original Tweet Retweet Count
favorite_count	int	Number of likes on the tweet
created_at	DateTime	Date and time of the tweet.

Indexing Strategy

- Relational Database: Default indexing on primary keys
- Non-Relational Database (MongoDB): Automatic _id indexing; full-text indexing for tweet texts; the increasing index for screen_name; the increasing index for hashtags. In order to ensure faster queries when searching by username, hashtags, and tweet context, we added a text index on the “text” field. We added an increasing index on the “screen_name” field and the “hashtags” field.

Cache Design

The indexing strategy is complemented by the LRUCache system, which is designed with an optimal capacity of 100 items to balance memory efficiency and data accessibility. The cache's Time-to-Live (TTL) of 3600 seconds ensures data remains current, enhancing search result relevance and accuracy. The caching mechanism optimizes response times for popular queries and frequently accessed data by employing an eviction policy that adheres to the Least Recently Used (LRU) principle. This allows the cache to dynamically adjust to user demands, providing fast and reliable access to pertinent information. Persistence is a key feature of the caching

strategy; it involves periodic serialization to disk, allowing the system to maintain continuity through restarts without data loss. The cache is routinely monitored to ensure the 'freshness' of its contents, with outdated items promptly removed to maintain efficiency and performance.

```
INFO:root:Access took 0.00 ms
INFO:root:Access took 0.00 ms
INFO:root:Cache access for key 1249403767108668930 (hit) took 0.018 ms
INFO:root:Cache access for key 1249403768023678982 (hit) took 0.010 ms
INFO:root:Cache access for key 1249403769193779202 (hit) took 0.006 ms
INFO:root:Database fetch for key 1249403767108668930 (miss) took 0.010457 seconds
INFO:root:Database fetch for key 1249403768023678982 (miss) took 0.010610 seconds
INFO:root:Database fetch for key 1249403769193779202 (miss) took 0.010419 seconds
INFO:root:Database fetch for key 1249403767108668930 (miss) took 0.010143 seconds
INFO:root:Database fetch for key 1249403768023678982 (miss) took 0.011164 seconds
INFO:root:Database fetch for key 1249403769193779202 (miss) took 0.010536 seconds
INFO:root:Cache access for key 1249403767108668930 (hit) took 0.016 ms
INFO:root:Cache access for key 1249403768023678982 (hit) took 0.007 ms
INFO:root:Cache access for key 1249403769193779202 (hit) took 0.005 ms
```

Figure 1: Cache Performance while querying.

Search Application Design

The search functionality includes searching by username, hashtag, and tweet content, with the ability to filter by time range on the backend. Search results are prioritized by relevance—by favorite count for tweets and follower count for user profiles. Drill-down features enable displaying additional tweets or retweets from selected authors when their username is clicked. Additionally, backend functions were developed to showcase the top 10 tweets and users by favorite and follower counts, respectively. Search queries are routed to appropriate data stores based on their prefixes: '@' for usernames, '#' for hashtags, and general text for content searches.

Division of Work

- **Sarthak Singh:** Focused on the relational datastore and SearchTimingComparisons, ensuring efficient handling and querying of structured data.
- **Alvin Alex:** Managed the non-relational datastore using MongoDB and developed indexing strategies to enhance search performance.
- **Phan Nguyen Huong Le:** Designed the caching system using LRU with TTL and persistence, optimizing data retrieval and maintaining data freshness.
- **Meiqiao Shi:** Developed the GUI interface and integrated the search application, making it user-friendly and accessible for diverse user needs.

Search Application Interface and Functionality Showcase

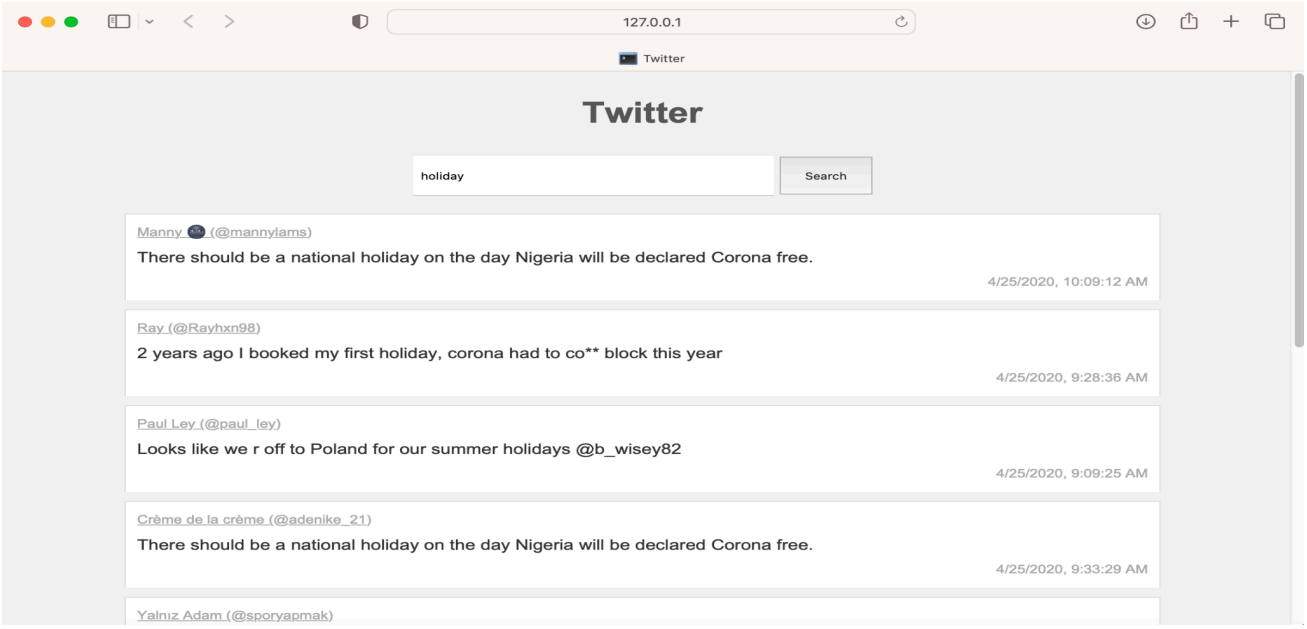

Figure 2: GUI snapshot of Search By User

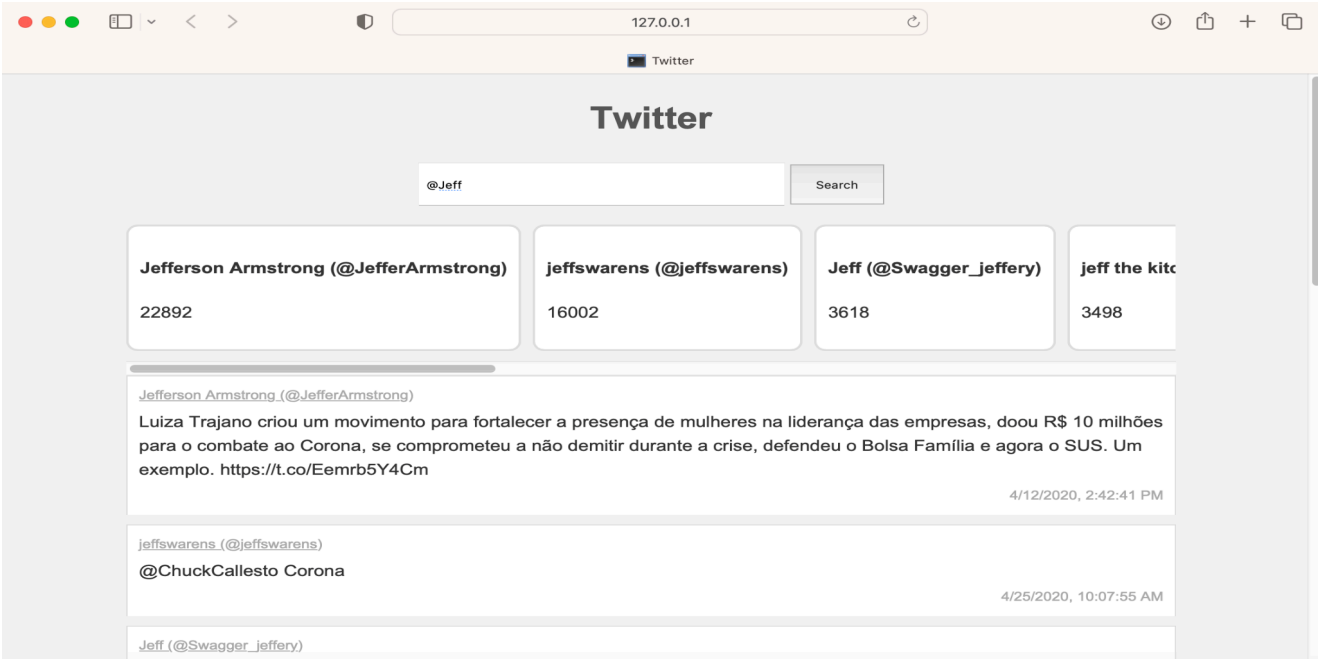

Figure 3: GUI snapshot of Search by text.

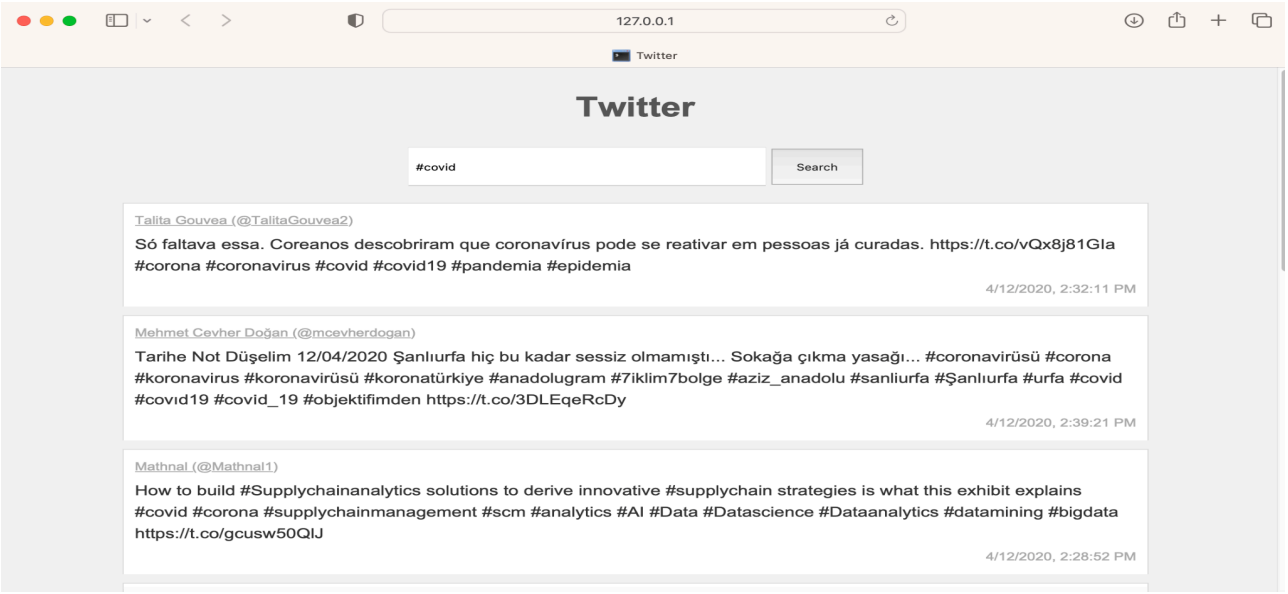

Figure 4: GUI snapshot of Search by hashtag.

GitHub Repository and Usernames:

The project can be tracked on GitHub at Main Github Repo. Our team members' GitHub usernames are Meiqiao Shi: meiqiaoshi, Sarthak Singh: manasarthak, Phan Nguyen Huong Le: valle1306, and Alvin Alex: aalvinn03.

Conclusion

The completion of the Twitter Search Application Project marks a significant milestone in the effective management and querying of large-scale Twitter datasets. The integrated system, which combines relational and non-relational databases, supports complex searches and delivers insights with high responsiveness. This project underscores the effectiveness of merging structured data management with flexible data handling to cater to the dynamic nature of social media data. Strategic caching and advanced indexing significantly enhance system performance, enabling real-time data processing and rapid information access. This application serves as a valuable tool for researchers and analysts to delve into Twitter data and lays the groundwork for future enhancements in social media analytics. The collective expertise of the team has been crucial in surmounting technical challenges and achieving project objectives, demonstrating the importance of teamwork and specialized knowledge in developing sophisticated data solutions.