Analysis Report on Twitter RDB Application Performance

Hardware Configuration

Operating System: Mac (Macbook M1 Pro 2020)
Number of Cores: 8
RAM: 16 GB

Software Stack

Database: MySQL 8.0.33
Storage Engine: InnoDB
Programming Language: Python 3.9.13
Main libraries Used:

- mysql-connector-python 8.0.33
- pandas 1.5.3

Set-Up Instructions

For set-up instructions, please navigate to README.md in the folder submitted. In this file, you will have access to prerequisites, installation steps, environment configurations, and how to run the application. For any questions, kindly contact me at gollapudi.j@northeastern.edu.

Performance Testing Results

For both performance metrics (i.e. Posting Tweets and Retrieving Home Timeline), I ran 5 iterations, and the results reported are the average results:

1. Tweets Posted Per Second

Iteration 1: 1900.51 [1,000,000/526.17s]
Iteration 2: 1998.94 [1,000,000/500.26s]
Iteration 3: 1907.95 [1,000,000/524.12s]
Iteration 4: 2025.59 [1,000,000/493.68s]
Iteration 5: 1934.76 [1,000,000/516.85s]

Avg Tweets Posted Per Sec = (1900.51 + 1998.94 + 1907.95 + 2025.59 + 1934.76)/5 = **1953.55**

Factors Affecting Performance:

- **System Load:** Assuming that no significant background processes were running, system load was likely not a major factor in performance. However, if there were other resource-intensive applications running, they could have impacted the results.

- **Database State:** The database, if "warm" (having some data already loaded in memory due to previous operations), might have facilitated slightly faster insert operations. A "cold" start (freshly started database) might have slightly slower performance initially due to lack of caching.

- **Network Latency:** As the database was hosted locally (MacBook M1 Pro setup), network latency was likely negligible in this context.

2. Home Timelines Retrieved Per Second

Iteration 1: 35.57 [1,000/28.10s]
Iteration 2: 32.56 [1,000/30.71s]
Iteration 3: 32.08 [1,000/31.16s]
Iteration 4: 28.02 [1,000/35.68s]
Iteration 5: 28.28 [1,000/35.35s]

Avg Home Timelines Retrieved Per Sec = (35.57 + 32.56 + 32.08 + 28.02 + 28.28)/5 = **31.30**

Factors Affecting Performance:

- **SQL Query Complexity:** The SQL query used to retrieve the home timelines involved joining the TWEET and FOLLOWS tables and ordering the results, which can be resource-intensive operations. The complexity of these operations, especially without optimal indexing, could significantly impact retrieval times.

- **Database State:** Similar to posting tweets, the performance could vary depending on whether the database is "warm" (with data cached from previous operations) or "cold" (freshly started).

- **System Resources:** The performance is also likely influenced by the hardware resources of the system, such as CPU speed, available RAM, and disk type. The MacBook M1 Pro,

while efficient, may have constraints compared to a dedicated high-performance server setup.

Analysis

Comparison to Expected Performance:

The observed performance is considerably lower than Twitter's standards (6-10 thousand tweets per second, 200-300 thousand home timeline refreshes per second). This result is expected given the hardware and software limitations compared to a large-scale, distributed system like Twitter's.

Potential Improvements:

- **Optimizing SQL Queries:** Fine-tuning the SQL queries for timeline retrieval, possibly with more efficient JOIN operations or different query structures using prepared statements, could yield better performance.

- **Efficient Data Structures:** In the application layer, exploring alternative data handling methods, possibly avoiding data frames for large datasets, might improve efficiency.

- **Scaling Hardware Resources:** While the MacBook M1 Pro is efficient, upgrading to a system with higher-spec CPUs and more RAM could positively impact performance.

- **Database Tuning:** Adjusting MySQL configurations for performance, such as buffer pool size or query cache, might help in handling larger loads more efficiently.

Conclusion

- The tests provide valuable insights into the scalability and performance of a MySQL-based implementation for a Twitter-like application.

- The performance gap compared to real-world Twitter standards highlights the need for significant enhancements in hardware, software, and possibly database design to meet such demanding requirements.

- These findings underscore the importance of exploring alternative technologies like Redis in the next phase, potentially offering the needed throughput and low latency for high-scale applications.