Leo Maršić

BENCHMARK REPORT

Cache for dimension tables: read response time with (join on server from cache)

vs

without (app simulates joins by doing additional queries) cache

Architecture

The benchmark focused on evaluating the performance of a Redis cache implementation within a system architecture. Redis is an in-memory data structure store known for its high-speed data caching and retrieval capabilities. It significantly enhances system performance by storing frequently accessed data in memory, reducing the need to query the primary database repeatedly.

While Redis offers speed and efficiency, drawbacks may include limited storage capacity due to its reliance on memory, potentially increasing costs for larger datasets. Applications like real-time analytics, session caching in web applications, and high-throughput systems benefit from Redis caching by improving response times and reducing database load.

Environment:

Redis Container: A Docker container was instantiated to host the Redis instance, leveraging its in-memory caching capabilities. This containerized Redis served as the caching layer for optimizing data retrieval and storage during the benchmark.

Main Application Container: Docker container encapsulated the main application, integrating the benchmarking logic and facilitating interactions with Redis for caching and SQLite for data operations. This containerized main application provided a controlled environment for executing the benchmarking procedures.

SQLite Container: SQLite, functioning as the database component, was encapsulated within its Docker container. This container hosted the SQLite instance, allowing the main application container to interact with it for data retrieval and manipulation during the benchmarking process.

Local GUI: Additionally, local graphical user interface (GUI), provided an interface for initiating operations. GUI allowed for user interaction and visualization of the benchmarking results.

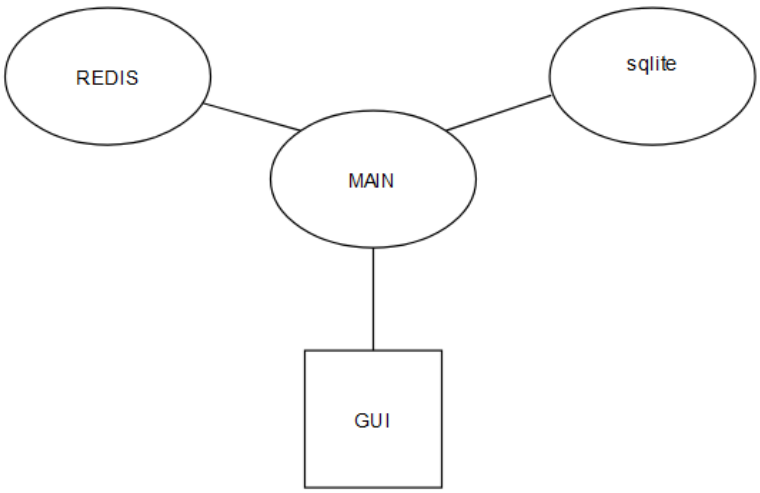

Figure 1: System Architecture

Simulating Load and Data with Faker for Benchmarking:

The benchmarking process involved the generation of synthetic data using Faker to simulate a representative workload for evaluating caching mechanisms. This approach was chosen for its simplicity and ease in creating a dataset that mirrors real-world scenarios.

Measurement of Response Times or Throughput:

The assessment of response times was conducted by integrating time measurement functions within the main application code. By importing the 'time' module, the benchmarking process incorporated functionality to capture timestamps at the commencement and completion of specific processes. This approach was adopted to precisely measure the elapsed time between these timestamps, determining the duration required for the execution of data retrieval, processing, and caching.

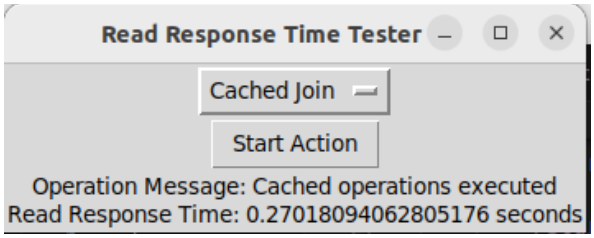

Figure 2: Cached operation

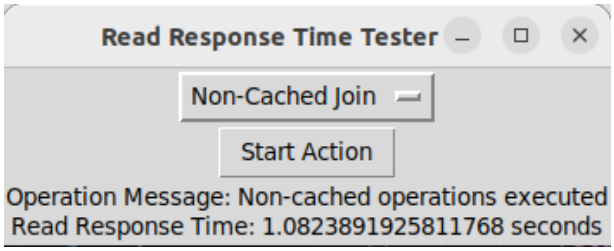

Figure 3: Non-cached operation

Summary

The benchmark was realized using Docker containers to encapsulate components of the system: Redis for caching, the main application, SQLite as the database, and a local graphical user interface (GUI).
Redis container used in-memory caching, optimizing data retrieval, while the main application interacted with Redis for caching and SQLite for data operations. The GUI container provided a user interface for initiating actions. Docker's containerization facilitated portability, scalability, and ease of setup, ensuring consistent deployment across various environments.
This modular approach allowed for efficient assessment and comparison of performance metrics while ensuring isolation and reproducibility in the benchmarking environment.