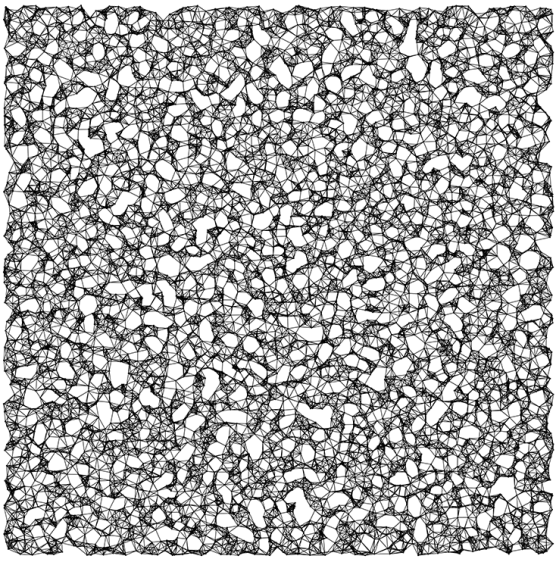
205.2 - Beyond relational DB
Prof. Delgado, Prof. Schumann
Semestre d'automne 2024-2025

Informatique et systèmes de communication ISC
Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften



KNN graph – Inspired by Marcus Volg

Rapport de projet pour la filière ISC

Avec une mise en page avec `Typst`

Nathan Antonietti

13 avril 2025

Table des matières

1 - Introduction

This report presents the results of a comparative analysis of three database systems: **Postgres**, **MongoDB**, and **Redis**. The analysis focuses on three key operations:

- **Insert Time**: Measuring the time taken to insert data into the database.
- **Query Time**: Measuring the time taken to execute a query.
- **Lookup Time**: Measuring the time taken to retrieve a specific record.

Additionally, the report evaluates the **CPU usage** and **memory consumption** during these operations to assess the resource efficiency of each database.

The source code can be found at https://github.com/NathanAnto/db_comparison

2 - Methodology

The tests were conducted using Docker containers for each database system. The following steps were performed:

1. **Data Preparation**: A mock dataset was generated using `data.py` and stored in `data.csv`.
2. **Environment Setup**: Docker containers were initialized using `docker-compose`.
3. **Performance Testing**: Python scripts (`insert-time.py`, `query-time.py`, `lookup-time.py`) were executed to measure operation times and resource usage.
4. **Resource Monitoring**: CPU and memory usage were captured using `docker stats`.

3 - Analysis

The following graphs were plotted with 5 iterations of each operation.

3.1 - Performance Metrics

The performance metrics for **Insert Time**, **Query Time**, and **Lookup Time** are summarized in the graphs.

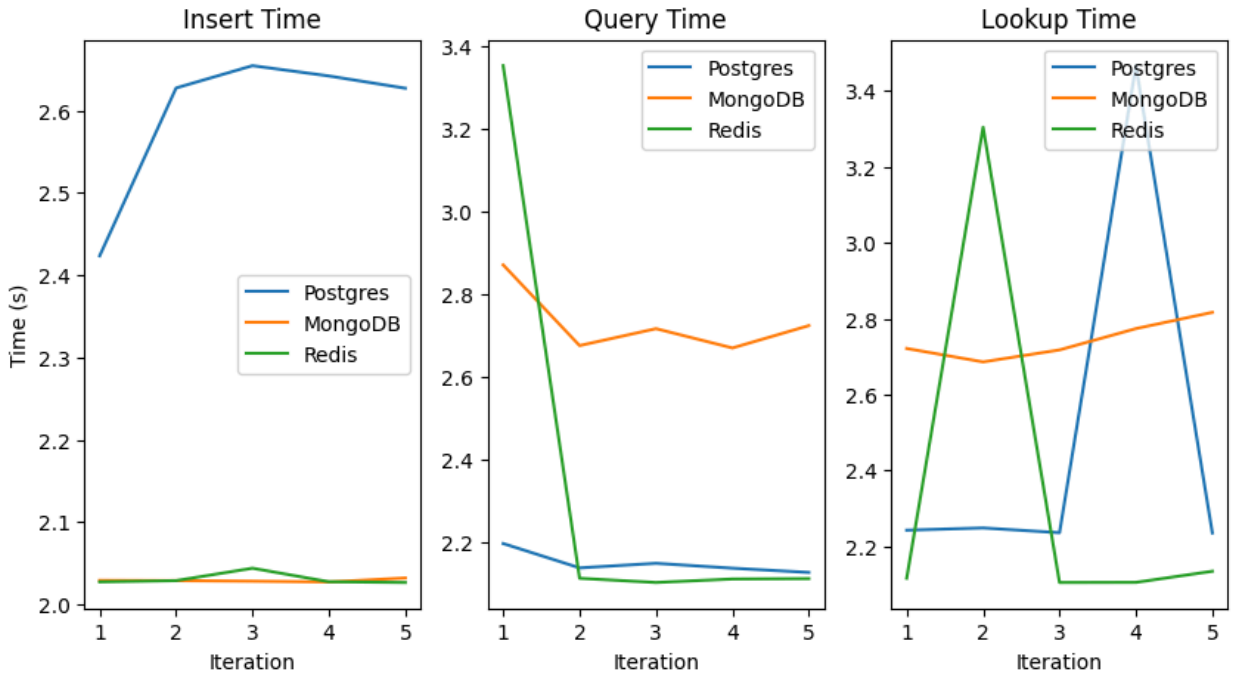


Figure 1 - CPU usage per database

3.1.1 - Insert Time

- **Postgres**: Insert times increase slightly over iterations, peaking at around 2.6 seconds. This could indicate some overhead in managing relational constraints or indexes.
- **MongoDB**: Maintains consistent insert times across iterations, averaging around 2 seconds. This highlights its efficiency in handling bulk inserts.
- **Redis**: Also shows consistent insert times, similar to MongoDB, averaging around 2 seconds. This is expected due to its in-memory operations.

3.1.2 - Query Time

- **Postgres**: Query times remain stable across iterations, averaging around 2.2 seconds. This demonstrates its reliability for complex queries.
- **MongoDB**: Query times are higher than Postgres, averaging around 2.8 seconds. This could be due to the overhead of managing document-based queries.
- **Redis**: Has the fastest query times, averaging around 2.1 seconds. This is expected, as Redis is designed for fast key-value lookups.

3.1.3 - Lookup Time

- **Postgres**: Lookup times show variability, with some iterations peaking at over 3.5 seconds. This suggests occasional overhead in retrieving specific records.
- **MongoDB**: Lookup times are consistent, averaging around 2.7 seconds. This indicates MongoDB's efficiency in retrieving documents by key.
- **Redis**: Has the fastest and most consistent lookup times, averaging around 2.1 seconds. This highlights its suitability for scenarios requiring frequent lookups.

Insight: Redis consistently outperforms Postgres and MongoDB in query and lookup times, making it ideal for real-time applications. Postgres excels in stability, while MongoDB offers a balance between performance and flexibility.

3.2 - CPU Usage

The average CPU usage for each database is summarized in the graph. Key observations include:
- **MongoDB**: Exhibits the highest CPU usage, averaging around 0.5%. This is likely due to its document-based architecture and the overhead of managing flexible schemas.
- **Postgres**: Has the lowest CPU usage, with minimal variation between initial and final CPU usage. This makes it highly CPU-efficient.
- **Redis**: Shows moderate CPU usage, higher than Postgres but significantly lower than MongoDB. Its in-memory operations contribute to its lightweight CPU usage.

Insight: MongoDB's higher CPU usage may be a trade-off for its flexibility and scalability, while Postgres and Redis are more CPU-efficient.

The light blue Initial CPU(%) consists of the average initial cpu usage before each operation, while the dark blue CPU Difference (%) indicates the added usage after the operation.

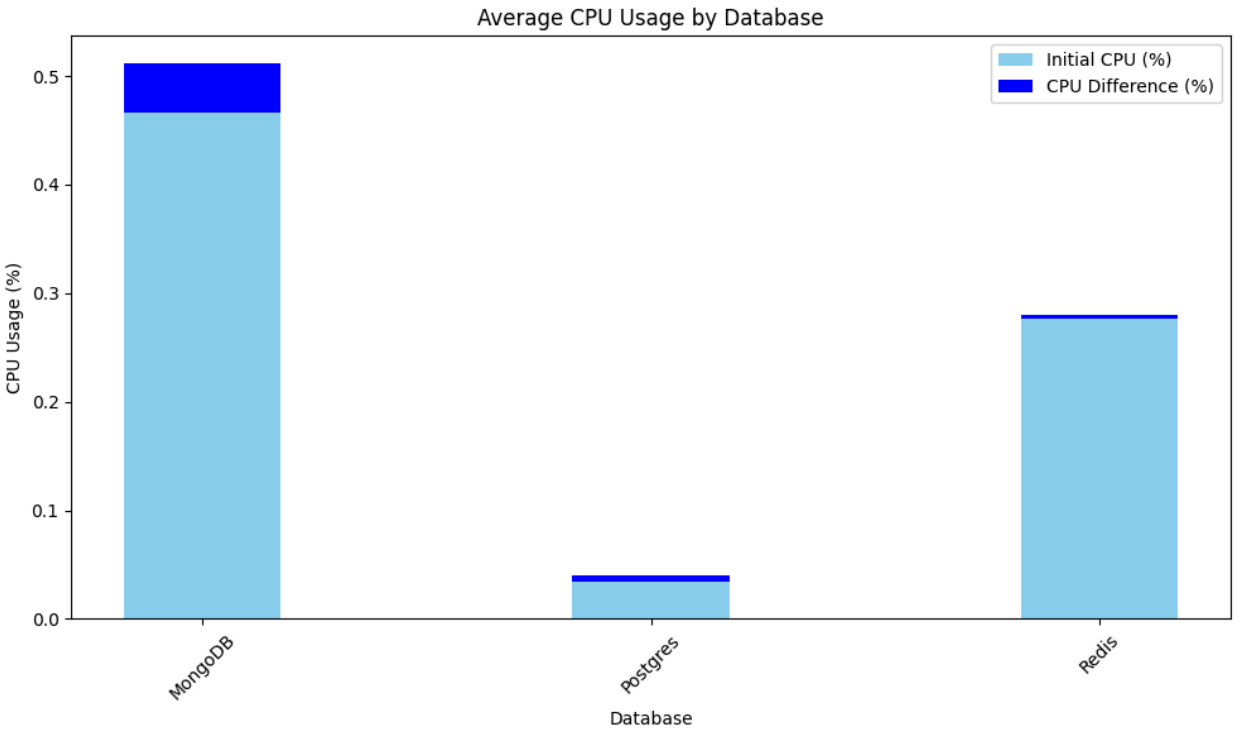


Figure 2 - CPU usage per database

3.3 - Memory Usage

The average memory usage for each database is summarized in the graph. Key observations include:

- **MongoDB**: Consumes the most memory, averaging over 300 MiB. This is consistent with its design, as MongoDB often caches data in memory to improve query performance.
- **Postgres**: Uses moderate memory, averaging around 50 MiB. This indicates a balance between memory usage and performance.
- **Redis**: Has the lowest memory usage, averaging around 15 MiB. Its in-memory architecture is optimized for minimal memory overhead.

Insight: MongoDB's high memory usage makes it suitable for systems with abundant memory, while Redis and Postgres are more memory-efficient.

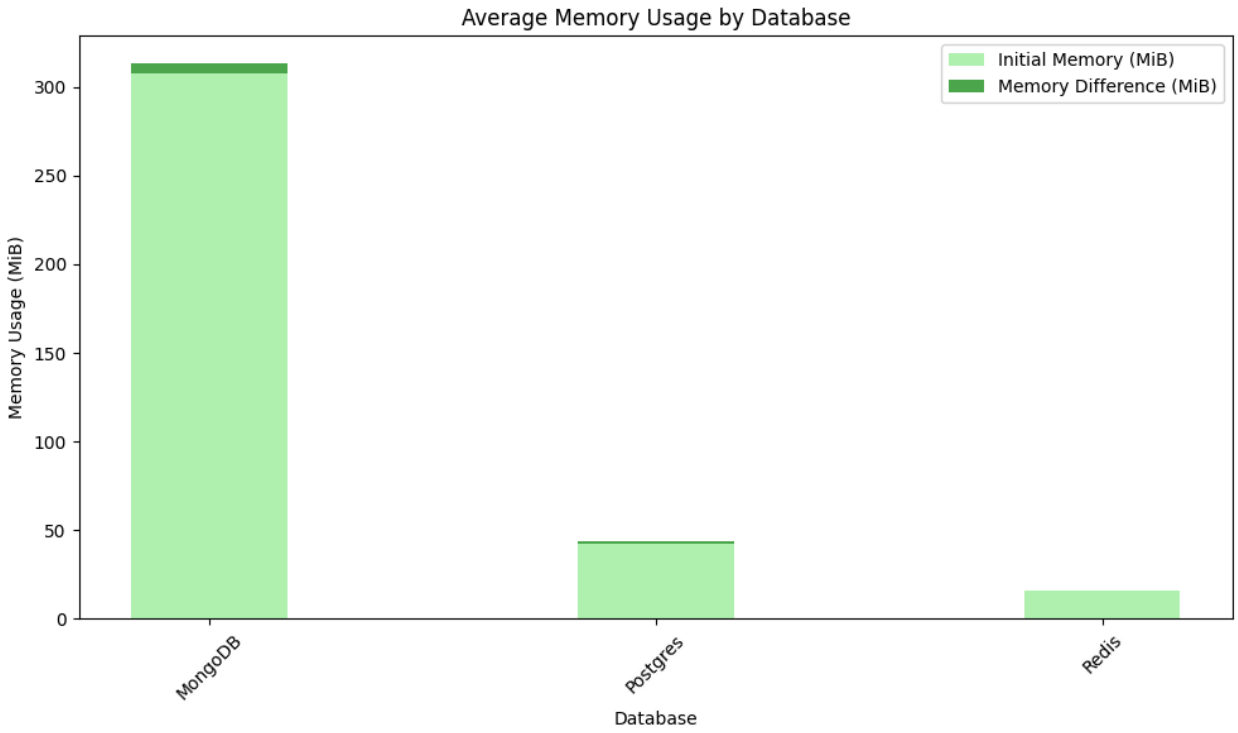


Figure 3 - Memory usage per database

3.4 - In Summary

1. **Performance**:
   - Redis is the fastest for queries and lookups, making it ideal for caching and real-time applications.
   - MongoDB offers consistent performance across operations, suitable for flexible schema designs.
   - Postgres provides stable performance but shows variability in lookup times, which may be influenced by indexing or relational constraints.
2. **Resource Usage**:
   - MongoDB is the most resource-intensive, with high CPU and memory usage.
   - Postgres is resource-efficient, with minimal CPU and moderate memory usage.
   - Redis is the most lightweight, with low CPU and memory usage.
3. **Use Cases**:
   - **Postgres**: Best for complex queries and relational data.
   - **MongoDB**: Suitable for applications requiring flexible schemas and high memory availability.
   - **Redis**: Optimal for caching, real-time lookups, and scenarios with limited resources.

The results indicate the following:

- **CPU Usage**: MongoDB has the highest CPU usage, while Postgres is the most CPU-efficient. Redis strikes a balance between the two.
- **Memory Usage**: MongoDB consumes the most memory, followed by Postgres. Redis is the most memory-efficient.
- **Performance**:
  - Redis consistently outperforms Postgres and MongoDB in query and lookup times.
  - MongoDB offers consistent performance across operations but at the cost of higher resource usage.
  - Postgres provides stable performance for inserts and queries but shows variability in lookup times.

These findings highlight the trade-offs between performance and resource usage for each database system.

4 - Conclusion

This analysis highlights the strengths and weaknesses of each database system:

- **Postgres**: Best suited for complex queries and relational data.
- **MongoDB**: Ideal for flexible schema designs and high memory availability.
- **Redis**: Optimal for fast lookups and caching.

Future work could include testing additional operations, such as updates and deletes, and evaluating performance under concurrent workloads.