SPRINT: Scaling Performance of R-DBMS with IMDG and Text Analytics

Amartya Chaudhuri
amartyac@usc.edu
University of Southern California

Aditya Dutta
adityadu@usc.edu
University of Southern California

Abstract

As volumes of data grow and the demand for real-time data processing rises, traditional relational database management systems (R-DBMS) like MySQL increasingly face performance limitations, especially in scalability, query speed, and full-text search capabilities. To address these challenges, our project proposes a novel integration of MySQL with Redis In-Memory Data Grid (IMDG) and Elasticsearch, aimed at enhancing the performance and scalability of R-DBMS for complex and data-intensive applications. Redis IMDG allows the system to incorporate high-speed transactions and efficient data caching, while Elasticsearch boosts performance in full-text search and analytics performance. This hybrid approach holds the promise of providing significant gains in throughput and query execution times,therefore making it a very strong solution in modern enterprise data management needs. Preliminary experimentations have shown performance increases by an order of magnitude over stand-alone MySQL setups, thus confirming the efficiency of our integrated system for the challenge of data management today and tomorrow.

1 Introduction

With the rise of modern, data-driven applications, there is a requirement like never before: a database capable of managing large-scale analytics and high-speed data operations. MySQL is a great staple among the world of Relational Database Management Systems (R-DBMS) and is increasingly being challenged as it scales to meet the needs of high-volume, complex queries, and real-time data access. Traditional architectures of R-DBMS, particularly MySQL, are affected negatively by single-threaded query executions and dependence on disk-based storage, leading to significant performance bottlenecks and inefficiencies in handling simultaneous operations and large datasets. Additionally, MySQL's built-in full-text search capability using B-Tree indexes is becoming less and less suited to the complexities of modern data volumes.

This project strives to propose an efficient and transformative solution that would integrate MySQL with an in-memory data grid, Redis, and a distributed full-text search and analytics engine called Elasticsearch for solving these most pervasive problems. Such integration is designed to leverage Redis' power in fast data access and effective caching with the great full-text search and real-time analytics capabilities of Elasticsearch. The synergy of these technologies working in tandem reach for MySQL performance beyond current limits, enriching transaction speed and data retrieval process. Engineered to surpass limitations of the MySQL architecture, this hybrid system dramatically improves throughput, reduces query execution time, and increases scalability to provide a resilient framework intended for the dynamic needs of the enterprise data management of today.

Preliminary experimental evaluations of the integrated system reveal that there is great potential, with improvements in key performance metrics such as average query execution times and speedup ratios observed with respect to a standalone MySQL configuration. These advances are very important not only for the bettering of user experience but also for businesses to leverage their data assets into real-time analytics in a more effective way. Our approach is bound to provide a substantial uplift in R-DBMS capabilities, set a standard beyond current performance, and be fit for higher levels of scalability that are demanded by currently escalating data loads. This study will show that this kind of integrated solution is not only a solution but will aggressively counter the limitations of the traditional R-DBMS and open up new possibilities within data management technology.

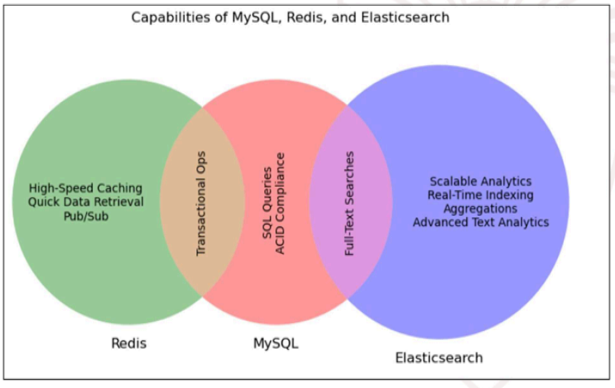


Fig 1 : Functionalities of Components

2 Proposed Solution

At the core, our proposed solution integrates two advanced technologies with MySQL: Redis IMDG and Elasticsearch. The detailed skeleton of our approach is as follows:

Redis IMDG Integration: With Redis, an in-memory data grid, we leverage its quick data access nature to help in high-speed transaction handling and for caching operations that are inefficient in disk-based storage systems like MySQL. It acts as an extremely responsive data layer that quickens the read operations, hence speeding the operations of data retrieval and transaction handling to a great extent.

Elasticsearch Integration: Elasticsearch is used for performing powerful full-text search and real-time analytics; it performs well under situations requiring fast searching over large datasets and more efficiently with complex query requirements than the inbuilt full-text search features in MySQL. The system integrates Elasticsearch by obtaining the implementation of nuanced search functionalities required for any modern application.

3 System Architecture

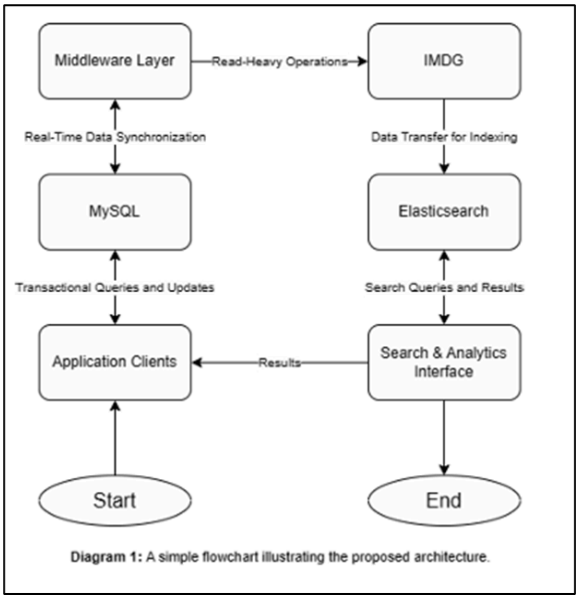


Fig 2 : Proposed System Architecture

The system architecture is a multi-component setup that outlines the flow of operations in a distributed e-commerce website. Here's a breakdown of the final software stack as we have imagined:

i. Middleware Layer:
This layer serves as the central hub for directing operations. It receives real-time data synchronization inputs from external or internal sources, to stay current with the latest information. It handles read-heavy operations by interfacing with the In-Memory Data Grid (IMDG), which is optimized for high-performance read operations due to its in-memory nature. This suggests that the middleware layer is responsible for balancing load between persistent storage and fast, read-optimized storage.

ii. In-Memory Data Grid (IMDG):
The IMDG is used for its quick data retrieval capabilities, such that the system architecture is ready for scenarios where speed is critical, such as real-time analytics or high-speed transactions. Data from the IMDG is then transferred for indexing in Elasticsearch, as the system not only requires quick access to data but also needs efficient search and analytics capabilities.

iii. MySQL:
MySQL, a relational database management system, is used here for transactional queries and updates, as the system requires ACID-compliant transactions, which MySQL provides. The flow of data from MySQL goes back to the middleware layer. MySQL is used for operational data storage, and the middleware orchestrates the transactional operations.

iv. Elasticsearch:
This search and analytics engine receives data from the middleware layer for indexing, which would enable complex search operations and analytics. In theory, it would send search queries and results to the proposed "Search & Analytics Interface" based on user inputs, indicating that Elasticsearch is the engine powering the search and analytics functionality of the system.

v. Application Clients:
Clients interact with the system, performing transactional queries and updates which interact with the MySQL database through the middleware layer. The clients also receive results from the Search & Analytics Interface, where they could see other clients' choices and reviews to make decisions.

vi. Search & Analytics Interface:
This is a user interface (service endpoint) where the results of the search and analytics performed by Elasticsearch are consumed. This is the final output of the system available to the clients.

vii. Overall Flow:
The architecture is linear, starting from the real-time data input to the middleware and flowing through the system's components in a sequence that reflects the processing of data from raw input to

transactional updates, and ending with the delivery of search results and analytics.

The architecture emphasizes the separation of concerns, with different systems responsible for different aspects of data processing, such as transaction handling, real-time data caching, and search and analytics, with middleware orchestrating the overall process flow.

4 Pseudocode for Middleware Synchronization

```
Function Main:
    Initialize Database Connections to
MySQL, Redis, Elasticsearch
    Start Listening to Changes in
MySQL

Function Initialize Database
Connections:
    Connect to MySQL
    Connect to Redis
    Connect to Elasticsearch

Function Start Listening to Changes in
MySQL:
    While true:
        Check for new or updated
records in MySQL
        For each changed record:

UpdateRecordInRedis(record)

UpdateRecordInElasticsearch(record)
        Sleep for a configured
interval (to reduce load on the
database)

Function UpdateRecordInRedis(record):
    Generate a key for Redis based on
record's unique identifier
    Store relevant record data in
Redis using SET command

Function
UpdateRecordInElasticsearch(record):
    Prepare a document from the record
data
    Index or update the document in
Elasticsearch based on record's unique
identifier

Function Check for new or updated
records in MySQL:
    Execute a SQL query to fetch new
or updated records since the last
check
    Return the list of records
```

5 Methodology

In our project, we used Docker containers to spawn isolation environments for MySQL, Redis, and Elasticsearch with the settings to run on dedicated ports: 3306 for MySQL, 6379 for Redis, and 9200 for Elasticsearch. Such isolation resulted in a significant decrease in network latency but also simplified the management of configurations on a local server, enabling accurate performance testing and benchmarking.

For this purpose, the configuration of the database was executed through command-line interfaces to enable the running of SQL scripts for schema creation that would be similar to the structure of the common e-commerce datasets. This structure was very important in order to create realistic test scenarios. In addition to that, a middleware layer developed in Jupyter Notebook also enabled interactive testing and debugging. This middleware was crucial for integrating the databases, with a key function, **sync_data()**, orchestrating data flow between MySQL and both Redis and Elasticsearch, ensuring effective synchronization across these platforms.

We populated the MySQL tables using Python Faker library to generate synthetic data, simulating real-world e-commerce transactions to reflect normal operational conditions. This approach allowed us to evaluate the system's performance across a spectrum of data volumes and query complexities.

Performance metrics were meticulously gathered through a series of test benches that recorded running times and throughput. Running times were measured using the Python formula:

$$elapsed = time.perfcounter() - starttime$$

and throughput was quantified as transactions per second using:

$$throughput = len(data)/elapsed$$

These tests were executed on data sets with 100, 1000, and 5000 records in number, respectively, in order to check for systematic scalability and performance variations under different loads.

We had further conducted relative analysis for measuring the efficiency improvement provided by Elasticsearch for full-text search capabilities. The formula used for calculating speedup ratios is:

$$\text{Speedup} = \frac{\text{Execution Time of MySQL}}{\text{Execution Time of Elasticsearch}}$$

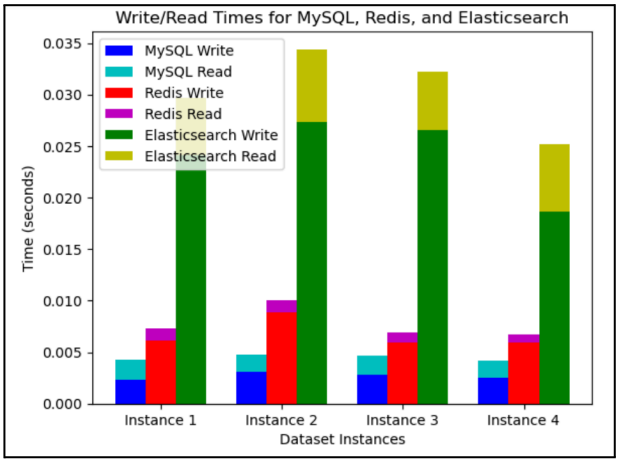
This gives a clear picture of the performance improvements that would be realized by using integration with Elasticsearch, instead of just using the normal SQL queries executed by MySQL.

This extensive methodology allowed the entire system to be fully evaluated, thus emphasizing that it was fully capable of improving query performance and keeping the data in synchronization between distinct storage solutions, thus showing its potential to scale up and serve the needs of modern, data-intensive applications.

6 Results

i. Running Time:

a) On Varied Read-Write Operations:

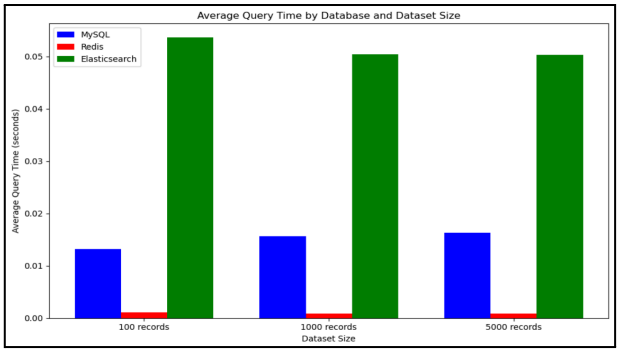


The accompanying analysis of the graph reveals that Redis consistently outperforms MySQL in read operation times across various dataset instances, highlighting its efficiency in retrieval operations. Furthermore, a trend is observed where Redis's write times decrease as the dataset sizes increase, indicating potential optimizations or caching mechanisms that improve write performance with larger datasets. It is expected that Redis's write times will fall below mysql's on huge datasets spanning millions of tuples, provided RAM capacity is sufficient. So our system would be more efficient on average read functions than standalone mysql for its Redis component.

Instance 1 Output as example:

```
MySQL Write Time: 0.0023 seconds
MySQL Read Time: 0.0020 seconds
Redis Write Time: 0.0061 seconds
Redis Read Time: 0.0012 seconds
Elasticsearch Write Time: 0.0241 seconds
Elasticsearch Read Time: 0.0056 seconds
```

b) On INSERT operations:

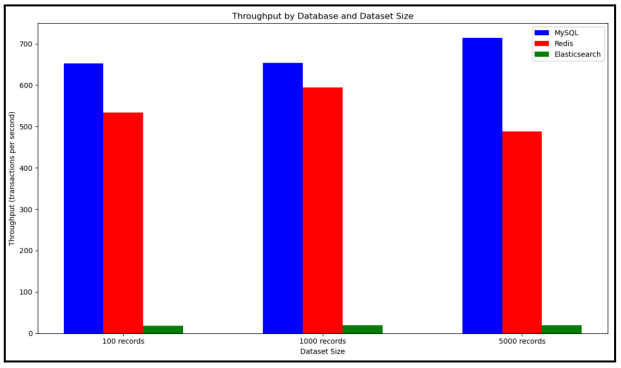


The graph highlights that Redis consistently offers the lowest average query times across all dataset sizes, showcasing its speed and efficiency. In contrast, MySQL and Elasticsearch exhibit higher query times that increase with the size of the dataset, indicating a relative decrease in performance as the data volume grows.

5000 INSERTS Instance Output:

```
MySQL Average Query Time: 0.0163 seconds
Redis Average Query Time: 0.0009 seconds
Elasticsearch Average Query Time: 0.0503 seconds
```

ii. Throughput:

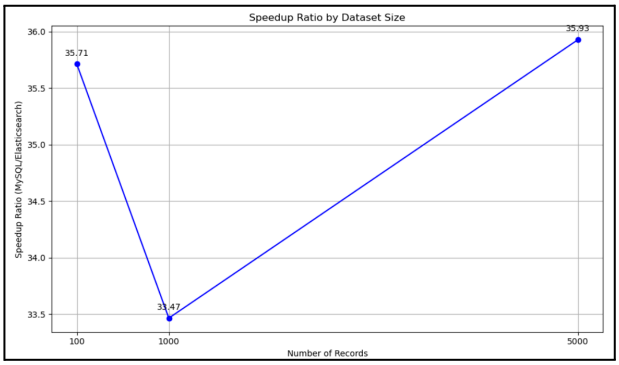


The graph illustrates that MySQL's throughput generally increases with the dataset size, peaking at 5000 records, whereas Redis shows a peak at 1000 records before a slight decline. Elasticsearch maintains a relatively constant but lower throughput across all dataset sizes, suggesting less variability with changing data volumes. This is concerning as Redis by itself gives much higher throughput than MySql on average than our system shows. This might be due to RAM constraints on our physical device, which could be solved by hosting the system on a cloud service.

5000 Records Instance Output:

```
MySQL Throughput: 714.00 transactions per second
Redis Throughput: 487.70 transactions per second
Elasticsearch Throughput: 19.68 transactions per second
```

iii. Speedup Ratio:



The line graph of speedup ratios demonstrates that Elasticsearch significantly outperforms MySQL in full-text search times across all dataset sizes, as indicated by the higher speedup ratio, signifying that lower search times are better for performance. The trend suggests that as the dataset size increases, the performance advantage of Elasticsearch remains fairly consistent.

5000 Records, Full-Text Searches Output:

```
ElasticSearch Full-Text Search Average Time: 0.0014 seconds
MySQL Full-Text Search Average Time: 0.0503 seconds
```

7 Conclusion

While this is still a very nascent idea, a data store combining the powers of MySQL, Redis and ElasticSearch shows potential. Apart from leveraging Redis's incredible on-RAM data retrieval speeds and ElasticSearch's robust full-text search capabilities that outperform MySQL's, we could also use their distinct features together such as a Publish-Subscribe message delivery system, Aggregation, Real Time Indexing and Advanced Text Analytics along with traditional MySQL roles under the same roof, which remains to be explored.

8 Future Work

Advanced Caching Mechanisms: Developing some of the most advanced caching strategies within the IMDG, which could enable an additional reduction in query times, especially for complex analytical queries.

Machine Learning Optimization: Analyze past patterns in queries and implement machine learning algorithms on middleware layer to predict and optimize performance of queries.

Cross-Platform Scalability: The architectural aspect of the system extended to operate seamlessly on different cloud platforms in order to ensure the portability and scalability of the system.

Real-Time Analytics Enhancements: Refining the middleware layer for better real-time analytics capabilities, including stream processing and complex event processing.

Fault Tolerance and Recovery: Developing robust fault-tolerance mechanisms and quick recovery strategies for the distributed system to minimize downtime.