JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

Scalable Database Optimization Techniques in Spring Boot Microservices

Ankit
M.Tech Student
Department of Computer Science and Engineering
BRCM College of Engineering and Technology, Bahal, India

Abstract: A microservices architecture has greatly changed the manner in which modern applications are being designed and deployed due to its rapid adoption. Spring Boot is now one of the most used architectures to develop microservices because of its simplicity, modularity and integration abilities. Nevertheless, with more micro-services, the issue of database scalability emerges as a matter of concern. The poor performance of the system can be caused by bottlenecks in the form of inefficient queries, poor connection pooling, poor indexing, and overloading on centralized databases.

Objectives: The main goal of this study is to research and analyze optimization methods of the database that help to increase the scalability of the Spring Boot microservices. The work will also seek to identify some of the common performance bottlenecks and suggest solutions that can be implemented in the real world system.

Methods: This research approach is aimed at finding, applying, and assessing scalable database optimization strategies in Spring Boot microservices. It has adapted a structured approach and this would entail system design, optimization strategies and experimental evaluation.

Findings: The results of the study confirm that the implementation of database optimization methods in Spring Boot microservices can be considered as the method which enhances the scalability and the performance greatly. Connection pooling using HikariCP decreased the overhead of creating new connections and the average response time decreased by about 40%.

Novelty: The originality of the study consists in the fact that numerous techniques in database optimization within the framework of Spring Boot microservices were thoroughly and practically integrated and tested in the cloud-native context. The other important contribution is the testing assessment of performance of the experiment using containerized micro services with simulated real-world work loads under conditions of low and high workload, and offers measurable accuracy of response time, throughput and stability of the system in optimized conditions compared to the pre-optimization conditions.

Keywords: Spring Boot, Microservices, Database Optimization, Scalability, Query Optimization, Caching, Connection Pooling, Sharding, Replication, Cloud-Native Applications

1. Introduction

Over the last few years, the Spring Boot microservices architecture has emerged as one of the leading strategies to develop scalable, flexible, and maintainable enterprise applications. Micro services break large monolithic systems into small independent services that may be developed, deployed and scaled independently. Yet, although this architecture increases agility and modularity, it also presents considerable problems with managing and optimizing databases, which in many cases in micro services become the main bottleneck to achieve real scalability.Traditional monolithic systems normally use a single centralized database, but in microservices, several independent services may demand high concurrency, real-time data access, and distributed data management. The more users and transactions, the less responsive the query, the higher the connection overhead, the higher the latency and resource contention can be as a result of poor database operation optimization. These performance problems have direct impact on the reliability and user experience of applications based on microservices.

The study fills such gaps by presenting and analyzing Scalable Database Optimization Techniques in Spring Boot Microservices. Through the integration of various optimization measures and experimental performance analysis within the framework of containerized and distributed conditions, this research paper offers useful information to the developers and researchers who want to enhance the scalability and resiliency of microservices applications.

2. Methodology

1. To address the ultimate design, the initial phase of the methodology was to design a cloud native microservices architecture application on spring boot. Services like User, Order and Inventory were implemented as independent with their respective databases and communication interface. This modular design guaranteed loose coupling, and simplified the application of optimization techniques at the service level.
2. Under the second step, connection pooling was applied with HikariCP to effectively handle connections with the database. This assisted in lowering the overhead of establishing and closing new connections repeatedly and enhanced the overall response time of microservice under parallel workloads.
3. The third phase dealt with caching system in which Redis was incorporated to save data accessed on a regular basis. Latency was minimised because it allowed the server to respond to repeated requests using the cache rather than accessing the database and minimised the amount of workload on the primary database.
4. The fourth step brought about the design of queries and indexing in an optimized way. SQL queries were re-organised to remove repetition and proper indexing of queries was performed to optimize search and retrieval process. This move saw to it that databases were able to manage huge datasets with much ease.
5. Hybrid database solutions were implemented in step five. Structured transactional data was stored using postgreSQL and unstructured as well as highvolume data was selected using MongoDB. This combination used the advantages of both relational and NoSQL databases to become scaled and flexible.
6. Sharding spread out the data to a number of database instances, and replication enhanced the availability and fault tolerance of data. The combination of these methods increased the horizontal scalability of the system.
7. Performance testing with Apache JMeter was the last process. Workloads of different numbers of simultaneous users were applied in simulation to test throughput, latency, and utilization of resources.

3. Results and Discussion

1. Connection pooling with the application of HikariCP led to the tangible decrease in response time. The average latency fell by almost 35 percent under simulated concurrent workloads versus the baseline system where there was no pooling. This illustrates that connection management is essential when the heavy traffic is managed in Spring Boot micro services.
2. Database load and query execution time were greatly decreased with the integration of the Redis caching. Queries that were regularly formulated and then previously consumed a few milliseconds were presented in micro seconds using the cache. This did not only enhance the performance of the application but also provided a high level of scalability when the traffic is at peak levels.
3. Maximized SQL queries and indexing techniques displayed a steady enhancement in data recovery operations. AS an example, search operations that used to take a full table scan could be finished in a fraction of the time once they were aptly indexed. This supported the significance of query optimization of database-intensive microservices.
4. The mixed database model, which accommodated both PostgreSQL and MongoDB was a good solution that offered the flexibility to deal with various kinds of data. Whereas PostgreSQL guaranteed the integrity of transactions of structured data, MongoDB handled large amount of unstructured information efficiently. Such a multi-database structure minimized the number of bottlenecks and made microservices scalable.
5. The use of sharding and replication techniques further enhanced scalability and fault tolerance. Sharding spread the workload among the multiple nodes and therefore the load per server on an instance was lowered and replication ensured that data was still available in case one of the instances failed. Through this strategy, the system realized an improved reliability and higher throughput.
6. The optimizations were validated using performance testing with Apache JMeter. The optimized system continued to have low latency and high throughput as the simulated users concurrently grew in numbers well beyond the simulated number of users. The utilization of resources was balanced among the services, which confirmed that the techniques applied were effective in the scaling of the system.
7. A batch was also done to combine several single query inserts/updates, as opposed to executing multiple single query inserts/updates, which resulted in a close to 40-percent decrease in query execution overhead. This greatly enhanced efficiency in this system when the load was high in terms of transactions. Leakage of connections in HikariCP avoided the decline in performance with time. Some connections were open without need, draining

database resources, because of no proper leak detection. Facilitating leak detection meant that idle connections were released in time, which stabilized system performance over long running operations.

4. Conclusion

The study conducted on Scalable Database Optimization Techniques in Spring Boot Microservices proves the fact that one of the most important elements that can affect the scalability of distributed system is the database performance. Although microservices architecture provides modularity and flexibility, it is inherently reliant on the ability to manage high throughput and low latency on the basis of efficient database interactions. The paper has discussed some optimization methods including indexing, refining querying, caching, batches, asynchronous processing, replication, and partitioning. All these approaches played their individual roles in making response times shorter, stabilizing system load, and maintaining a steady provision of services even during periods of heavy user access.As the outcomes made it quite clear, database optimization is not a one-time activity but an ongoing process that has to be actively monitored and profiled and tuned. As an example, caching popular data minimized unwanted hits to the database, whereas query optimisation minimised the time taken to execute the query. Equally, such methods as connection pooling and batching did not only reduce processing overhead but also increased the overall responsiveness of microservices. With replication and sharding, the system could allocate workloads in an effective way, thereby avoiding the occurrence of bottlenecks in times of heavy demand. These results support the concept of scaling a microservices-based application not only with the addition of more servers or services but also with the faster, smarter and more efficient access to the data. Moreover, the study highlights observability and automation as the key role of modern microservice ecosystems. In the absence of suitable monitoring tools, it is difficult to identify query bottlenecks, lock contentions, or slow response times. The database queries are optimized and automated scaling mechanisms are built to keep the system dynamically scaled to change of work-loads. The integration plays a crucial role in organizations wishing to create fault-tolerant and high-performance apps that can be used by thousands of concurrent users.

5. References

1.Newman, S. Microservice Design: Fine-Grained Systems. OʹReilly Media, 2021.

2.Richardson, C. Microservices Patterns: With Examples in Java. Manning Publications, 2018.

3.Fowler, M., & Lewis, J. ʺMicroservices: a definition of this new architectural term,” Martin Fowler, 2014.

4.Thönes, J. ʺMicroservices,” IEEE Software, vol. 32, no. 1, pp. 116–116, 2015.

5.Bansal, S., & Kumar, A. ʺPerformance optimization techniques for database systems,” International Journal of Computer Applications, vol. 180, no. 26, pp. 15–21, 2018.

6.Kumar, P., & Singh, R. ʺDatabase optimization in distributed applications,” Journal of Computer Science and Applications, vol. 9, no. 2, pp. 45–53, 2020.

7.Chodorow, K. MongoDB: The Definitive Guide. OʹReilly Media, 2022.

8.Stonebraker, M., & Çetintemel, U. One size fits all: An idea whose time has come and gone, 21 st International Conference on Data Engineering (ICDE), IEEE, 2005.

9.Gaurav, A., and Sharma, V. ʺPerformance improvement of Spring boot microservices with caching mechanisms, International Journal of Innovative

10.Technology and Exploring Engineering (IJITEE), vol. 9, no. 4, 210215, 2020.

11.Microsoft. ʺDatabase Sharding Patterns,” Azure Architecture Center, 2023.

12.Oracle. ʺBest Practices for Database Performance Tuning,” Oracle White Paper, 2022.

13.MySQL Documentation, ʺOptimization Overview,” Oracle Corporation, 2023.

14.PostgreSQL Documentation, ʺPerformance Optimization Guide,” PostgreSQL Global Development Group, 2023.

15.Kubernetes Documentation. ʺHorizontal Pod Autoscaler and Scaling Strategies,” CNCF, 2023.

table 1:Performance Comparison of Optimization Techniques in Spring Boot Microservices

Technique Applied	Average Response Time (ms)	Throughput (req/sec)	CPU Utilization (%)	Memory Usage (MB)
Baseline (No	420	180	72	850
Database	250	310	65	870
Connection	190	350	60	880
Caching	120	480	58	900
Database	160	410	62	920
Combined (Indexing + Caching +	95	520	55	910