Issue 15, 2024

https://doi.org/10.23939/sisn2024.15.159

УДК 004

DEVELOPING A PERFORMANCE EVALUATION BENCHMARK FOR EVENT SOURCING DATABASES

Roman Malyi[1], Pavlo Serdyuk[2]

Lviv Polytechnic National University,
Software Engineering Department, Lviv, Ukraine
[1] roman.m.malyi@lpnu.ua[1], 0000-0002-2255-1132[1]
[2] pavlo.v.serdiuk@lpnu.ua[2], 0000-0002-2677-3170[2]

In the domain of software architecture, Event Sourcing (ES) has emerged as a significant paradigm, especially for systems requiring high levels of auditability, traceability, and intricate state management. Systems such as financial transaction platforms, inventory management systems, customer relationship management (CRM) software, and any application requiring a detailed audit trail can significantly benefit from this approach. Numerous aspects of ES remain unexplored, as they have yet to be thoroughly investigated by scientific research. The unique demands of such systems, particularly in terms of database performance and functionality, are not adequately addressed by existing database benchmarks. By establishing benchmarks, organizations can compare different databases to determine which best meets their needs for applications. This aids in selecting the most appropriate technology based on empirical data rather than assumptions or marketing claims.This paper introduces a novel benchmarking framework specifically designed for evaluating databases in the context of event sourcing. The framework addresses critical aspects unique to ES, including event append performance, efficient handling of Projections (separate databases for read operations), strong consistency, ordered data insertion, and robust versioning controls. Through rigorous testing and analysis, this framework aims to fill the gap in existing database benchmarking tools, providing a more accurate and relevant assessment for ES systems. We also conducted experiments that not only demonstrated the effectiveness of our approach but also yielded meaningful results, substantiating its practicality and applicability.

Key words: event sourcing; MongoDB; EventStoreDB; PostgreSQL; events; NoSQL; performance comparison.

Introduction

Event-driven systems have emerged as a pivotal paradigm for addressing the challenges of real-time data processing and communication. Event sourcing architecture and its benefits are deeply described in the works of (Alongi, 2022) [1] and (Overeem, 2021) [2].

Data stores utilized for event sourcing must fulfill specific prerequisites to effectively support the event-driven paradigm, as described in the paper (Overeem, 2021) [2] it's mainly two operations: read and append. They also must offer durable storage to safeguard the chronological sequence of events, ensuring accurate reconstruction of system states. Immutability of stored events is essential to maintain data integrity. Furthermore, efficient querying mechanisms are necessary for reconstructing past states.

Scalability is vital to accommodate growing event streams. Finally, support for distributed architectures ensures fault tolerance and high availability, critical for consistent event sourcing implementations.

The criticality of selecting an appropriate database for ES is underscored by the diverse and often challenging requirements these systems present. These include high throughput for event writes, efficient strategies for event reads, rapid reconstruction of states from events, and the ability to scale horizontally in response to fluctuating workloads. Additionally, the inherently append-only nature of event logs in ES poses unique challenges for database management systems, especially in terms of long-term data growth and query optimization.

Formulation of the problem

The benefits of ES, such as improved audit trails, historical state reconstruction, and enhanced system resilience, are well-documented; they also introduce a set of complex challenges in database management. The primary concern is the selection of an optimal database system that aligns with the specific requirements of ES, including handling high-volume event streams, ensuring efficient state reconstruction, and maintaining data integrity over prolonged periods.

Existing benchmarks primarily focus on general database operations or are tailored towards other specific applications, such as OLTP or data warehousing. These benchmarks, while comprehensive in their domains, do not adequately capture the nuances and specialized demands of ES systems.

Analysis of recent research and publications

The paper (Sfaxi, 2021) [3] presents a detailed methodology for benchmarking a cash management platform used by an investment bank. This is achieved using a generic benchmarking solution named BABEL. The paper emphasizes the modular design of BABEL and offers an evaluation methodology along with best practices for its application in real-world systems. A key outcome of this study is the ability to identify appropriate trade-offs between consistency and availability, aligning with the service level agreements specified by clients. Additionally, the paper demonstrates that the integration of BABEL with the platform incurs minimal overhead during runtime. The workloads suggested in the article are similar to the YCSB benchmark.

The work (Aluko, 2019) [4] presents a comprehensive study of four popular Big SQL systems: Apache Hive, Spark SQL, Apache Impala, and PrestoDB. The study aims to analyze the performance characteristics of these systems using three different benchmarks: TPC-H, TPC-DS, and TPCx-BB. The study finds that Textfile formats showed the lowest performance, while compressed formats like ORC provided better performance. The Parquet file format generally offered the highest performance for most queries. The article concludes with valuable insights and lessons learned from the experiments, providing guidance for future research and practical applications in Big SQL system benchmarking.

The survey paper (Fuad, 2020) [5] provides an extensive overview and analysis of various benchmarking efforts in the realm of big data systems. The survey discusses the Yahoo! Cloud Serving Benchmark (YCSB), introduced in 2010, which was designed to assess the performance characteristics of NoSQL databases. The HiBench benchmark suite, presented for Hadoop, includes synthetic micro-benchmarks and real-world applications to assess the Hadoop framework. The StreamBench benchmark, introduced by Lu et al., addresses standard stream processing scenarios and operations. It measures different aspects of systems like multi-recipient performance, fault tolerance, and durability. The survey mentions studies like those by Barnawi et al., which focused on assessing the performance of big graph processing systems such as Giraph and GraphLab. In the emerging domain of big machine/deep learning systems, few benchmarking efforts have been made. The study by Boden et al., for example, implemented distributed machine learning algorithms on Apache Flink and Apache Spark, focusing on scalability with high-dimensional data. Overall, the paper underscores the complexity and diversity in benchmarking big data systems, highlighting the need for comprehensive and versatile benchmarks that can cater to various system types and application scenarios.

(Han, 2018) [6] presents a comprehensive survey of state-of-the-art big data benchmarking efforts. The paper categorizes existing benchmarks into three groups: micro benchmarks, end-to-end benchmarks, and benchmark suites. Micro benchmarks focus on individual system components or specific behaviors. End-to-end benchmarks assess entire systems using typical application scenarios. Benchmark suites combine different micro and/or end-to-end benchmarks to provide comprehensive solutions. Examples include HcBench and MRBS for Hadoop-related systems, and HiBench, CloudSuite, and BigDataBench for various big data systems. A significant part of the paper is dedicated to discussing workload generation techniques for big data benchmarks. Workloads are categorized based on the type of operations used to form them, including I/O operations, algorithms, and elementary operations (like standard SQL operators or similar syntaxes). This categorization helps in understanding how different benchmarks simulate real-world scenarios and the effectiveness of their methodologies.

In the work of (Yang, 2020) [7] explained the importance of applying metrology principles to benchmarking to improve the quality and authority of measurement results in benchmarks. It argues that benchmarks are essential for both users and manufacturers in making informed decisions and optimizing product performance.

The paper lists several benchmarks in various fields:

- BigBench for DBMS and MapReduce systems.
- CloudSuite for scale-out workloads in machine learning and cloud services.
- HiBench by Intel for MapReduce applications.
- CALDA for Hadoop and RDBMS systems.
- YCSB for NoSQL databases.
- AMP Benchmarks for real-time analysis applications.
- LinkBench for social graph databases.
- CloudBM for cloud data management systems.
- Various AI benchmarks like Fathom, DeepBench, BenchNN, DNN-Mark, Tonic Suite, and DAWNBench.

The paper concludes that adopting metrology principles in benchmark design and development enhances the measurement quality, making benchmarks more authoritative and reliable for evaluating IT products.

We discovered an abundance of research papers that delve into the comparative analysis between MongoDB and various other database systems. These papers extensively examine the distinctions and performance characteristics between MongoDB and alternative database solutions. The substantial body of literature reflects a keen interest in evaluating MongoDB's features within the broader context of database technology.

(Deari, 2018) [8] conducts a thorough analysis and comparison between document-based and relational databases, assessing data storage, management principles, and CRUD operation performance using MongoDB and MySQL as representative examples. The findings offer valuable insights into the strengths and limitations of each database model.

Cloud users face challenges when transferring data across different cloud storage services due to differing paradigms across platforms. In the work of (Khan, 2023) [9] examines articles addressing cloud data portability, interoperability, and software architectures of SQL and NoSQL databases. State-of-the-art studies extensively compare Oracle RDBMS and NoSQL Document Database (MongoDB), revealing NoSQL databases as a tailored option for big data analytics and SQL databases as optimal for online transaction processing (OLTP).

(Mukherjee, 2019) [10] explores a range of attributes inherent to NoSQL databases, outlines their distinct advantages over traditional RDBMS systems, and contemplates the future trajectory of NoSQL technology. The article also provides a foundational understanding of NoSQL and its practical applications.

Formulation of the purpose of the article

The primary purpose of this article is to introduce a benchmarking framework designed explicitly for assessing database performance in event sourcing (ES) environments. Recognizing the distinct requirements and challenges posed by ES architectures, this framework aims to fill a gap in the current landscape of database performance evaluation tools.

Secondary purpose is to illustrate how the benchmarking tool can be applied in real-world scenarios, providing insights into its usability and effectiveness.

Presenting main material

Overview of existing benchmarks

Database benchmarks are critical tools for evaluating the performance and efficiency of database systems across various areas of software development. Here are some well-known examples of database benchmarks, each focusing on different aspects and use cases:

1. YCSB (Yahoo! Cloud Serving Benchmark):

Specifically designed for evaluating the performance of NoSQL databases. It provides a framework for creating and executing typical read, update, and scan operations across different cloud database systems. It has 6 default workloads:

- Update-heavy: 50 % read, 50 % update.
- Read-mostly: 95 % read, 5 % update.
- Read-only.
- Read latest (delete old ones, insert new ones and read mostly the new ones).
- Read-modify-write.
- Short range scan.

2. TPC-C Benchmark (Transaction Processing Performance Council)

One of the most popular benchmarks for evaluating the performance of Online Transaction Processing (OLTP) systems. It simulates a complete computing environment where a population of users executes transactions against a database.

Default Workload: Simulates a complete computing environment where a population of users executes transactions against a database. The primary operations involve order-entry, payment, status updates, delivery, and stock level control. The workload is characterized by a mix of read and write operations with a significant emphasis on transaction integrity.

3. SPEC Database Benchmarks (Standard Performance Evaluation Corporation)

These benchmarks are used to evaluate the performance of database systems in different configurations and environments, providing a comprehensive overview of system capabilities.

One of the popular workloads is an e-commerce scenario where users interact with a web application for purchasing items. This includes browsing, adding items to the shopping cart, and processing orders.

Design of a new benchmark

The conventional benchmarks for database performance, while robust in their general application, are not adequately tailored for event store databases, particularly due to their specific operational nuances. A critical aspect of event sourcing is that it often does not rely solely on direct reads from the event store. Instead, it frequently employs the concept of Projections – separate databases specifically designed for read operations. These Projections are tailored to facilitate efficient querying and data representation, distinct from the event store's primary function of recording events.

In revising the specifications for a benchmarking framework tailored to event store databases, it's crucial to incorporate the aspect of strong consistency. This framework should rigorously evaluate how effectively the database maintains orderliness in data insertion, ensuring that all events are stored in a precise, sequential manner. This is essential in event sourcing, as the order of events directly impacts the accuracy of state reconstruction and system behavior.

In Fig. 1, we explained the main concepts of three proposed benchmark scenarios.

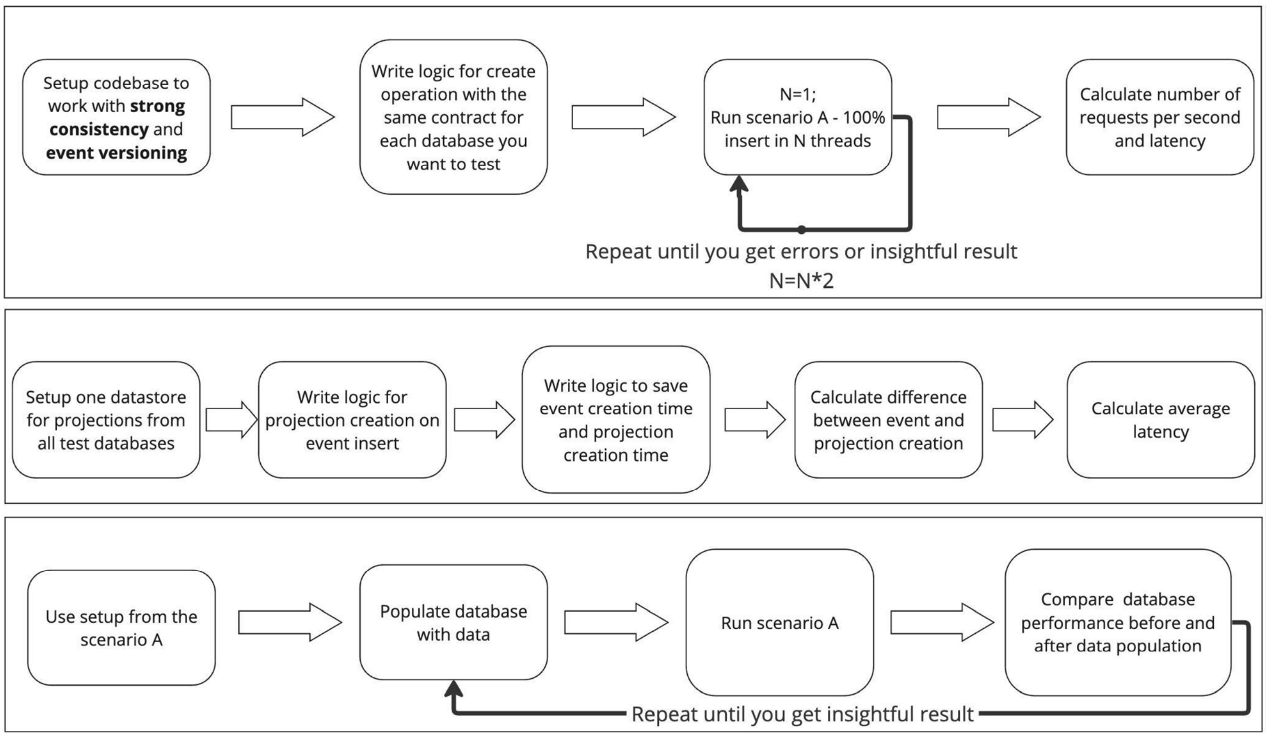
Setup codebase to work with **strong consistency** and **event versioning**
Write logic for create operation with the same contract for each database you want to test
N=1; Run scenario A - 100% insert in N threads
Calculate number of requests per second and latency
Repeat until you get errors or insightful result N=N*2
Setup one datastore for projections from all test databases
Write logic for projection creation on event insert
Write logic to save event creation time and projection creation time
Calculate difference between event and projection creation
Calculate average latency
Use setup from the scenario A
Populate database with data
Run scenario A
Compare database performance before and after data population
Repeat until you get insightful result

Fig. 1. Benchmark overview

Scenario A only focused on the insert operation, as it is one of the most crucial parts of ES systems. It's important to be able to support big amounts of inserts and have the possibility to scale if needed.

Scenario B is intended to test the latency of creating read models in order to verify that delay is suitable for the system.

Scenario C is optional and intended to predict database performance in the long run - when amounts of data will grow over time.

Experiments

Financial data is well-suited for event sourcing due to its inherent chronological nature and audit trail significance. Event sourcing captures every state transition, enabling accurate reconstruction of financial system history. This approach ensures compliance, transparency, and the ability to trace and analyze complex market transactions over time.

In the work (Qu, 2022) [11] authors mention that limitations of traditional stand-alone relational database management systems (RDBMS) become evident in handling scalability challenges for extensive applications like Securities Exchange. From this, we conclude that the incorporation of stock data into performance testing becomes equally crucial for both SQL and NoSQL database solutions. That's why we decided to use real-world financial data for our tests. You can check data structure in Table 1.

Table 1

Example of financial data used for test

Time	Tick_Volume	Real_Volume	High	Low	Open	Close
2023.02.28 23:59:00	2	0	1.058	1.0579	1.0579	1.058
2023.02.28 23:58:00	1	0	1.0579	1.0579	1.0579	1.0579

The structure from Table 1 was converted to the following model in c#:

```
public class AddCurrencyInfoCommand
{
   public DateTime Time { get; set; }
   public int TickVolume { get; set; }
   public int RealVolume { get; set; }
   public decimal High { get; set; }
   public decimal Low { get; set; }
   public decimal Open { get; set; }
   public decimal Close { get; set; }
}
```

Experimental setup: we ran the tests on the laptop with macOS version 14.2.1, Apple M2 Pro processor with 16G of physical memory and solid-state drive.

We decided to use the two most popular databases PostgreSQL and MongoDB also, we used EventStoreDB because it's designed for event sourcing. Even though we ran tests from the laptop all databases were created in the Cloud. We used DigitalOcean cloud service provider.

Databases specifications:

1. MongoDB v6 / 2 vCPU / 8 GB RAM / storage: 30GB SSD / data center region: Frankfurt.
2. PostgreSQL v16 / 2 vCPU / 8 GB RAM / Connection limit: 197/ storage: 30GB SSD / data center region: Frankfurt.
3. EventStoreDb v23.10.1 using Docker based on virtual machine (VM) with 2 vCPU / 8 GB RAM / 40 GB NVMe SSD/ data center region: Frankfurt. EventStoreDb is not supported directly by DigitalOcean, so we used docker to create it on a VM.

You can check the entire structure of our test setup in Fig. 2. For load generation we chose a free and open source load testing framework called NBomber [12]. Code snipped for Scenario A:

```
private static ScenarioProps CreateScenario(string name, HttpClient httpClient,
int testDurationSeconds, int numberOfScenarioInstances){
ScenarioProps scenario = Scenario.Create($"{name}Scenario", async context =>{
var request = Http.CreateRequest("POST", $"http://localhost:5093/api/{name}/currency")
                .WithHeader("Accept", "application/json")
                .WithBody(new StringContent(JsonConvert.SerializeObject(CreateCommand()),
Encoding.UTF8, "application/json"));

var response = await Http.Send(httpClient, request);
         return response;
              })
              .WithInit(async context => { await Task.Delay(TimeSpan.FromSeconds(1)); })
              .WithoutWarmUp()
              .WithLoadSimulations(Simulation.KeepConstant(numberOfScenarioInstances,
TimeSpan.FromSeconds(testDurationSeconds)));
return scenario;}
```

API was created using C# language and ASP.NET Core Runtime 8.0. Only EventStoreDb can support strong consistency and event versioning by default. So, for PostgreSQL we used a .Net library called Marten [13]. For MongoDb we used persistence library – NEventStore [14]. Then we added support for projections based on each event and decided to store it directly in memory because we want to calculate the time from the new event's arrival to the API till the moment when a new projection for it is created and can be stored. Of course, we understand that real examples of read models (projections) can be much more complicated, but for test purposes this is more than enough.

To be able to react on each event insert for PostgreSQL we used the "Event Projections" functionality that already exists in the Marten library. EventStoreDB has "catch-up subscriptions" functionality. Unfortunately, the NEventStore library that we used for MongoDb has no built-in support for projections, so for MongoDb we chose the "change streams" database functionality. You can check out the entire source code in the public repository [15].

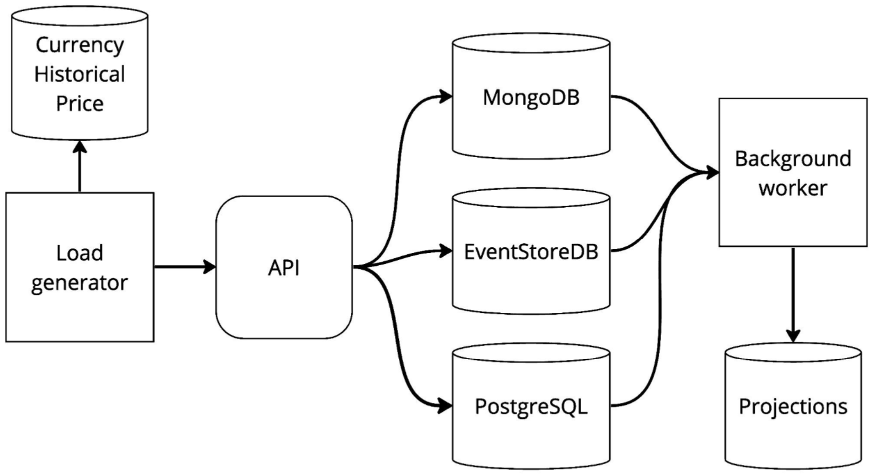

Fig. 2. Our test setup

As you can see from the code snipped above we can change `testDurationSeconds` and `numberOfScenarioInstances` variables. So we made 4 runs for 5 minutes for all 3 databases at the same time. Simulation.KeepConstant – means that our test framework will always keep one active HTTP request and as soon as one response is received a new request will be sent. For every run we have general statistics and a detailed report for each Scenario, see Fig. 3.

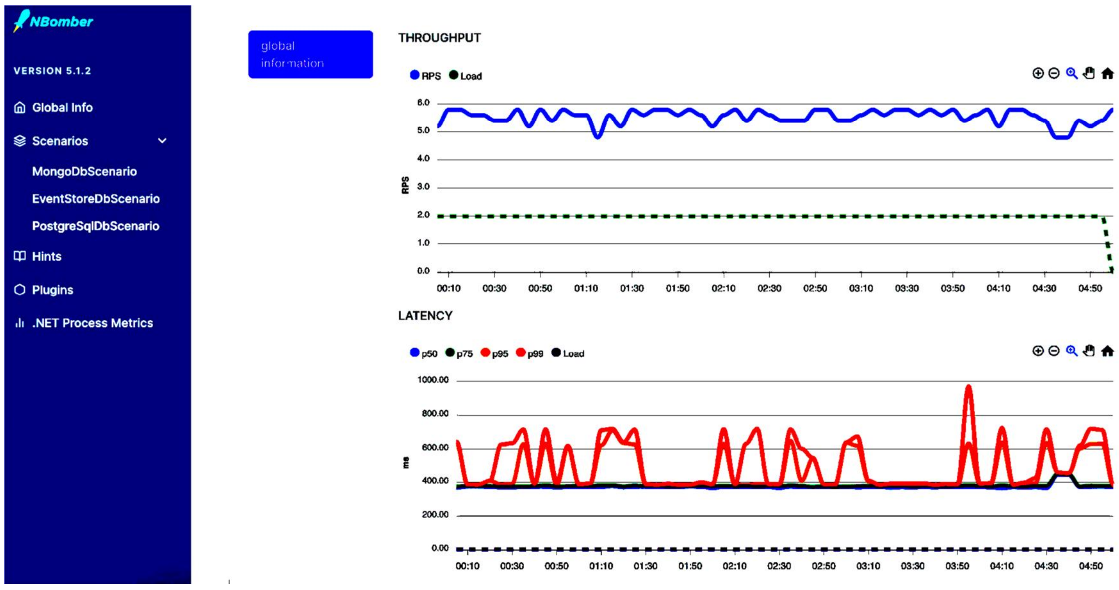

Fig. 3. Report for PostgreSQL Scenario A for 2 threads

In order to make information more clear and understandable we combined information from all 4 runs for all databases based on average request per second (RPS) characteristics, see Fig. 4.

Based on the information from Fig. 4 it's obvious that EventStoreDB has the best write performance and it increases when we add new threads with test workload. MongoDb setup had concurrent issues even with two threads which means that the current implementation with the NEventStore library is not suited for high load. However, it's not strange that a document-oriented database is not performing very well in cases when each event write requires reads and writes in two separate collections. Document databases are not designed to work in such scenarios. Anyway, it's an advantage of MongoDB that it's possible to use for simple event-sourced applications without high loads. PostgreSQL had no errors, but RPS decreased in the test with four threads compared to two, so it also can't handle a lot of concurrent requests and slows down, most likely because of locks on database tables as it's required for strong consistency and event versioning support.

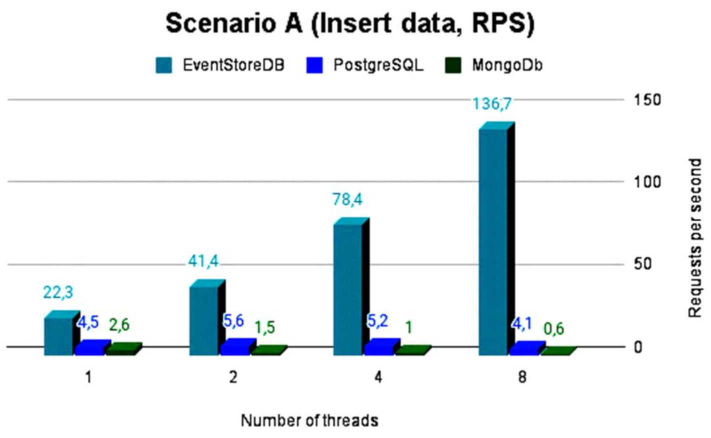

Fig. 4. Average RPS for each test run

In order to test Scenario B, we added two fields to the event: ApiCallTime – the time when the command entered the API, and SaveTime – the time when the event entered the background service in order to create a model for reading. This allowed us to calculate the delay between the request and the creation of the projection, the average values for one of the tests performed are shown in Fig. 5. Results are almost the same as for write performance – EventStoreDB is the winner with a huge advantage.

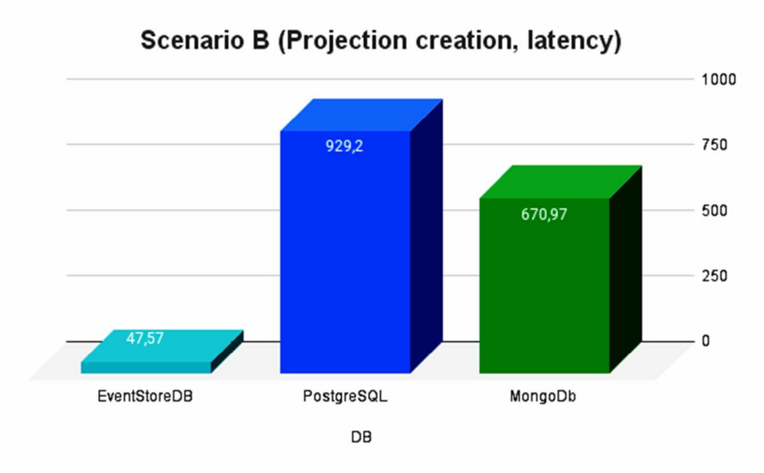

Fig. 5. Average latency for projection creation

We haven't run Scenario C mostly because it will take a huge amount of time to populate MongoDB or PostgreSQL with big amounts of data when we see an RPS of less than 5.

Conclusion

In this work, we conducted a study of current research and the state of existing benchmarks for NoSQL data storages and proposed a brand new benchmark for event-sourcing databases. We used proposed scenarios to test three databases: MongoDB, PostgreSQL, and EventStoreDB.

In summary, the evaluation of performance across selected databases has yielded intriguing insights into the behavior of these data storage solutions under varying workloads. Notably, our findings emphasize the importance of understanding the impact of concurrency and thread management on write operations in these systems.

EventStoreDB exhibited exceptional write performance, consistently demonstrating its ability to handle a significant volume of data. What stands out as particularly noteworthy is the fact that EventStoreDB's write performance improved as the number of threads increased, showcasing its strong concurrency capabilities. This behavior aligns with its design as a dedicated event sourcing solution, optimized for capturing and persisting events efficiently.

Conversely, the other databases in our evaluation displayed contrasting patterns. As we increased the number of threads from 1 to 2 and then to 4 and 8, the number of writes observed in these databases exhibited a decrease relative to previous runs. This behavior suggests potential limitations in their concurrency handling for write operations, as well as the need for careful configuration and tuning to optimize performance in multi-threaded scenarios.

We also compared the performance of selected databases for creating projections based on stored events and found that EventStoreDB is the best choice in terms of speed.

The practical value of this work lies in the ability to test any database and find out whether it is suitable for event sourcing under the required load. You can also use the results of this work to compare several databases in the context of ES and choose the best one for specific use cases.

References

1. Alongi, F., Bersani, M. M., Ghielmetti, N., Mirandola, R., & Tamburri, D. A. (2022). Event-sourced, observable software architectures: An experience report. John Wiley & Sons Ltd., 2127–2151. https://doi.org/10.1002/spe.3116.
2. Overeem, M., Spoor, M., Jansen, S., & Brinkkemper, S. (2021). An Empirical Characterization of Event Sourced Systems and Their Schema Evolution – Lessons from Industry. J. Syst. Softw. https://doi.org/10.1016/j.jss.2021.110970.
3. Sfaxi, L., & Ben Aissa, M. (2021). Designing and implementing a Big Data benchmark in a financial context: application to a cash management use case. Computing, 103, 1983–2005. https://doi.org/10.1007/s00607-021-00933-x.
4. Aluko, V., & Sakr, S. (2019). Big SQL systems: an experimental evaluation. Cluster Computing, 1–31. https://doi.org/10.1007/s10586-019-02914-4.
5. Bajaber, F., Sakr, S., Batarfi, O., Altalhi, A., & Barnawi, A. (2020). Benchmarking big data systems: A survey. Computer Communications, 149, 241–251. https://doi.org/10.1016/j.comcom.2019.10.002
6. Han, R., John, L., & Zhan, J. (2017). Benchmarking Big Data Systems: A Review. IEEE Transactions on Services Computing, PP, 1–1. https://doi.org/10.1109/TSC.2017.2730882
7. Yang, K., Wu, T., Shen, Q., Cui, W., & Zhang, G. (2020). Benchmark Researches from the Perspective of Metrology. https://doi.org/10.1007/978-3-030-49556-5_31
8. Deari, R., Zenuni, X., Ajdari, J., Ismaili, F., & Raufi, B. (2018). Analysis And Comparision of Document-Based Databases with Relational Databases: MongoDB vs MySQL. 2018 International Conference on Information Technologies (InfoTech), 1–4. https://doi.org/10.1109/InfoTech.2018.8510719.
9. Khan, W.; Kumar, T.; Zhang, C.; Raj, K.; Roy, A. M.; Luo, B. (2023). SQL and NoSQL Database Software Architecture Performance Analysis and Assessments – A Systematic Literature Review. Big Data Cogn. Comput. 7, 97. https://doi.org/10.3390/bdcc7020097
10. Mukherjee, S. (2019). The battle between NoSQL Databases and RDBMS. http://dx.doi.org/10.2139/ssrn.3393986
11. Qu, L., Wang, Q., Chen, T., Li, K., Zhang, R., Zhou, X., … Zhou, A. (2022). Are current benchmarks adequate to evaluate distributed transactional databases? BenchCouncil Transactions on Benchmarks, Standards and Evaluations, 2, 100031. https://doi.org/10.1016/j.tbench.2022.100031.
12. (n. d.). .NET load testing framework. NBomber. Retrieved March 15, 2024, from https://nbomber.com/
13. (n. d.). Marten. Martendb. Retrieved March 15, 2024, from https://martendb.io/
14. (2023, August 16). Event sourcing library for .NET. Nuget.org. Retrieved March 15, 2024, from https://www.nuget.org/packages/NEventStore/
15. Malyi, R. (2024, February 16). Test project source code. GitHub. Retrieved March 15, 2024, from https://github.com/RomanMalyi/DatabaseComparison

РОЗРОБКА ТЕСТУ ЕФЕКТИВНОСТІ БАЗ ДАНИХ ДЛЯ EVENT SOURCING

Роман Малий[1], Павло Сердюк[2]

Національний університет "Львівська політехніка",
кафедра програмного забезпечення, Львів, Україна
[1] roman.m.malyi@lpnu.ua[1], 0000-0002-2255-1132[1]
[2] pavlo.v.serdiuk@lpnu.ua[2], 0000-0002-2677-3170[2]

У сфері архітектури програмного забезпечення Event Sourcing (ES) стало важливою парадигмою, особливо для систем, які потребують високого рівня перевірки, відстеження та складного управління станом. Такі системи, як платформи фінансових транзакцій, управління запасами, програмне забезпечення для управління взаємовідносинами з клієнтами (CRM) і будь-які програми, які потребують детального аудиту, можуть отримати значну користь від цього підходу. Численні аспекти ES залишаються невивченими, оскільки їх ще повинні ретельно дослідити науковці. Унікальні вимоги до систем ES, зокрема щодо продуктивності та функціональності баз даних, не відповідають належно наявним тестам баз даних. Встановлюючи контрольні показники, організації можуть порівнювати різні бази даних, щоб визначити, яка найкраще відповідає їхнім потребам у додатках. Це допомагає вибрати найвідповіднішу технологію на підставі емпіричних даних, а не припущень чи маркетингових тверджень. Стаття містить нову структуру порівняльного аналізу, спеціально розроблену для оцінювання баз даних у контексті Event Sourcing. Фреймворк розглядає критичні аспекти, унікальні для ES, зокрема продуктивність додавання подій, ефективне опрацювання проєкцій (окремі моделі для операцій читання), надійну узгодженість, упорядковане вставлення даних і надійні засоби керування версіями. Завдяки ретельному тестуванню та аналізу ця структура має на меті заповнити прогалину в наявних інструментах порівняльного аналізу баз даних, надаючи точну та релевантну оцінку для систем ES. Автори також виконали експерименти, які не тільки продемонстрували ефективність запропонованого підходу, але й дали вагомі результати, обґрунтовуючи його практичність і застосовність.

Ключові слова: Event Sourcing; MongoDB; EventStoreDB; PostgreSQL; події; NoSQL; порівняння продуктивності.