JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 32 (2024) 246–250
Received: 23 June 2024
Accepted: 10 July 2024

Comparative analysis of the Express.js and ElysiaJS frameworks in the context of web application development

Analiza porównawcza szkieletów programistycznych Express.js i ElysiaJS w kontekście tworzenia aplikacji internetowych

Damian Henryk Kostrzewa*, Marek Miłosz

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
JavaScript-based programming frameworks allow more and more web applications to be built, aiming to become the fastest platform for a particular application. The aim of this paper is, among other things, to conduct study to obtain performance results of two server-based programming libraries in the application of an online transaction database structure. The database structure, the TPC-C benchmark queries and the Bombardier HTTP(S) query measurement tool were used to perform the tests. The research thesis "The Express.js framework has better performance than ElysiaJS in terms of TPS measurement tests in TCP-C structure" was set. The results obtained did confirm the thesis.

Keywords: Bombardier; ElysiaJS; Express.js; TPC-C

Streszczenie
Szkielety programistyczne oparte o język JavaScript umożliwiają budowanie coraz to wiecej aplikacji internetowych dążąc tym samym do uzyskania miana najszybszej platformy w konkretnym zastosowaniu. Celem artykułu jest między innymi przeprowadzenie badań, które pozwolą na uzyskanie wyników wydajności dwóch serwerowych bibliotek programistycznych w zastosowaniu struktury bazy danych transakcji online. Do wykonania testów wykorzystano strukturę bazy danych, zapytania benchmarku TPC-C oraz narzędzie pomiarowe do zapytań HTTP(S) Bombardier. Postawiono tezę badawczą „Szkielet programistyczny Express.js jest bardziej wydajny niż ElysiaJS pod względem testów pomiaru TPS w strukturze TCP-C”. Otrzymane wyniki potwierdziły postawioną tezę.

Słowa kluczowe: Bombardier; ElysiaJS; Express.js; TPC-C

*Corresponding author
Email address: damian.kostrzewa@pollub.edu.pl (D. H. Kostrzewa)

1. Introduction

Building web applications has always been a challenge for developers who are looking to achieve the best possible performance or optimization results. However, new solutions and entire execution environments are being created to handle these problems, one such environment is Bun and Node.js, on which the JavaScript programming frameworks ElysiaJS and Express.js are based to build a better web application than others. These frameworks are used for the server side of the application, handling all the necessary logic to communicate with the database and additional functionality that often goes beyond the page, such as online payments or Application Programming Interface (API) servers. One of the most important aspects of any web application is that the data, user information [1] or actions performed on the site are recorded in a database. As the functionality of a web application increases, so does the amount of data stored in the database, which also results in a greater load on the application server side itself [2], which has to handle the communication between the user and the very collection of data it is accessing.

To test the performance of the development framework, automated HTTP(S) query tools can be used or performance tests from trusted organizations such as the Transaction Processing Performance Council [3] can be used. This will allow reliable results to be obtained in a specific day-to-day application of a web application, as well as obtaining information on the latency of queries made between server-side points and the database.

The purpose of this paper is to benchmark two server-based development frameworks in identical but separated environments and to obtain information on which one achieves better performance for a specific application, in this case a transaction service.

2. Purpose and scope of work

The purpose of analysing two development frameworks, ElysiaJS and Express.js, is to conduct a study to answer the thesis “The Express.js framework has better performance than ElysiaJS in terms of TPS measurement tests in TCP-C structure”, TPS measurement tests in the context of web application development are crucial for determining the overall bandwidth of requests between the application and the database. Measurement criteria include support for HTTP requests, memory management, scalability and ease of configuration in development on prepared databases and datasets. The analysis will also consider aspects of the tool ecosystem, documentation, community support and developer feedback. Results of the analysis will provide recommendations on the appropriate type of application for each framework, taking into

account differences in flexibility, modularity and available resources.

3. Literature review

The initial question is whether both programming frameworks are capable of creating an application for a specific type of application. The technical documentation of both Express.js and ElysiaJS [4-5], specifies that they are minimalistic and flexible frameworks based on the Node.js and Bun run-time environment [6-7]. Performance is the most important consideration for developers when choosing a platform. Node.js, known for its event-driven, non-blocking input/output model, has stood the test of time as a solid and efficient choice for handling multiple concurrent calls as presented in [8]. Looking at the differences that are offered by the two environments, Node.js has an ”ecosystem” designed to control events and handle concurrent connections thanks to its non-blocking I/O model. Bun, on the other hand, directly declared at the outset that the main goal is the speed and performance of the environment, while to determine which framework based on which execution environment is better for specific architectures it is necessary to use performance tests prepared for real-world problems.

The database server has a major role in transactional testing for a specific architecture, thanks to the research provided by the article [9] that provides new insights and findings about the performance comparison between SQL and NoSQL databases. Study conducted an investigation on various operations such as read, write, delete, instantiate, and iterate on both types of databases. The results indicate that not all NoSQL databases outperform SQL databases, and the performance varies depending on the operation. communication between the server side and the database resource is needed, this communication is the most resource-intensive in terms of execution operations. When the case involves huge data sets that bank or warehouse applications can store, the necessary optimization must be applied to the server-side link with the database. Seemingly small optimizations can translate into a surprising end result, because everything must be multiplied by the amount of data that can be executed in a given second, this unit is called transactions per second. According to the article [10], it is possible to obtain information to optimize ”Create, Read, Update, Delete” (CRUD) queries and indicate when to use parameterized queries and when it is better to bet on performance and resources.

Although everything comes to performance measure that is crucial to determine which of backend server frameworks (ElysiaJS and Express.js) performs better, thanks to paper [11] which compared two types of TPC benchmarks and conclude comparison of various I/O characteristics of the two traces, including request types, sizes, spatial and temporal locality it is possible to choose benchmark that will meet expectations in terms of throughput, latency and terms of realistic workload simulation and accurate performance evaluation. The literature review helped to select appropriate research methods and practices for the execution of the tests and the final presentation of the results.

4. Research methods

Two suitable test methods were undertaken to perform the tests in order to obtain meaningful results. All tests were carried out according to the table of hardware and software specifications in Table 1.

Table 1: Environment specifications of hardware and software

Parameter	Specification
Operating system	Windows 10 64-bit
Processor	Intel Core i7-11850H @ 2.50GHz
RAM	SO-DIMM DDR4 64GB 3200MHz
SSD	PM9A1 NVMe Samsung 512GB
Internet	Wi-Fi 6 AX201 160MH

In addition, both development frameworks, i.e. ElysiaJS and Express.js, used the following versions of the installed packages as listed in Table 2.

Table 2: Specification of framework versions and packages

Specification	ElysiaJS	Express.js
Node version	21.11.1	21.11.1
Express.js version	-	4.19.2
NPM version	-	10.2.4
Bun version	1.1.8	-
ElysiaJS version	1.0	-
MySQL driver version	3.9.7	3.9.7

The latest stable versions of the frameworks were used, and all dependencies were updated to their most recent versions.

4.1. Database and queries

The database implementation will use the MySQL relational database version 8.2.12, with the XAMPP package version 3.3.0 installed, which will simply and securely reflect the schema presented by the TPC-C test assumptions. The database used will be unambiguous for both development environments, thus avoiding time measurement errors. Each programming framework will operate on a newly created database structure as specified in Figure 1 filled with uniform and pre-prepared data correct due to TPC-C documentation.

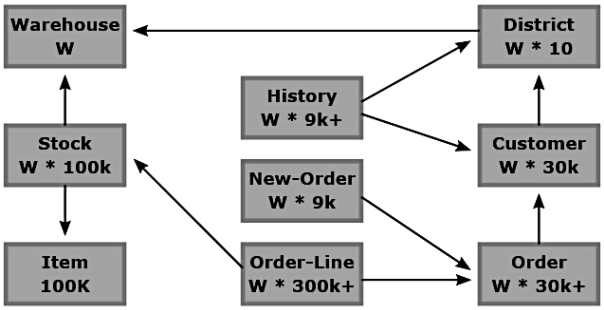

Figure 1. TPC-C Database schema.

Presented in Figure 1 schema assumes 9 entities, each warehouse (W) under minimum assumptions has 10 predefined districts (table District), and each district has

3,000 customers (table Customer) assigned to it, for a total of 30,000 customers. Each customer has the ability to order from a catalog of 100,000 products (table Item). There is also an order entity, which can be divided into 3 different entities and contains at least 30,000 orders per warehouse (W), all due to the need for a kind of buffer for this entity, the order has a predecessor as new-order, and a successor order-line, through which it is possible to queue orders and possibly modify the inventory of the warehouse, each warehouse has to have at least 300,000 records. Added to the whole is the history entity, which is intended only for inserting records. The tables in this database have unique assumptions, the item entity is read-only, warehouse, district, customer and stock are read and write, and new-order is write, read and insert because it is used as the previously mentioned buffer. Order and order-line receive introductions and can be read occasionally.

4.2. Queries

Each transaction is a group of queries required to complete action in online web transaction in warehouse, some of which require almost every possible data manipulation method on database. Implementation may vary depending on optimalization in code, thus transaction queries will be described as following:

- Payment transaction: First, a database transaction is started. Next, the store's annual balance total is updated, and then the store's address data and name are retrieved. The next step is to update the annual total of the district balances and retrieve the address data and the district name. Depending on the value of byname, if byname is true, the customers with the given name are counted and the customer data based on the name is retrieved; if byname is false, the customer data based on ID is retrieved. The customer's balance is updated. If the customer has a special credit type (contains 'BC'), additional customer data is retrieved, new customer data is formatted, and then the balance sheet and customer data are updated. If the customer does not have a special credit type, only the customer balance sheet is updated. Finally, a new record is inserted in the transaction history table.
- New order transaction: Initially, customer and store data are retrieved, such as customer discount, name, credit type and store tax. Next, the district data is retrieved, including the next order ID and district tax. The next step is to update the district's next order ID. New records are then inserted into the orders and new orders table. For each order line, the data is processed: the supplier's warehouse, item ID and quantity are retrieved, and then the item data such as price, name and additional information is retrieved. After this, the warehouse data, such as the quantity in stock and additional distribution information, are retrieved, and then the quantity in stock is updated. Based on the item and warehouse data, the type of brand (original or generic) is determined. The order line amount is calculated, taking into account taxes and discounts, and then a record is inserted into the order line table. Each of these operations is performed in a loop for all order lines until all the data has been processed and stored in the database.
- Order status transaction: First, the namecnt variable is initialized. If byname is true, the number of customers with the given name, district and store identifier is counted and then the data of these customers, such as balance, first name, middle name and identifier, is retrieved. The customer from the middle of the list is selected for further processing. If byname is false, the customer's data is retrieved based on its ID, district ID and warehouse. The data of the last order and the order line data are then retrieved and displayed.
- Delivery transaction: The DIST_PER_WARE constant is set to 10, and then the storage ID is displayed. Then, for each d_id from 1 to DIST_PER_WARE, the following operations are performed: Starting by querying the new order IDs for the given district id and warehouse id, if there are no results, the iteration moves to the next district id. The first order id in the new order table from the query results is retrieved, then the corresponding new order is removed based on the order id in the new order table, district id and warehouse id. The order customer id for the relevant new order id, district id and warehouse id are retrieved, then the order carrier id for the order is updated based on the order id, district id and warehouse id. The delivery date for the order line is then updated based on the order id in the new order, district id and warehouse id table. The total amounts for the order line are calculated, and the customer balance is updated with the order line total. Finally, information on the operations performed for the district and new order is displayed. Each cycle of the loop processes the data for the following districts in the warehouse.
- Stock level transaction: First, the value of d_next_o_id is taken from the district table as order id for the specified warehouse id and district id. Next, the number of different items whose stock is below the specified threshold is counted. The query includes the order line and stock table, where the conditions are warehouse id, district id, order id, and item id and quantity in stock. The result is stored in the stock count variable.

4.3. Bombardier method

The first method involves checking the requests per second, latency and throughput rate and is performed using Bombardier's HTTP(S) benchmark tool [12]. The test simulates real-world scenarios to provide a comprehensive assessment of each framework's capabilities. Method was prepared in such way to simulate different types of web application workloads with different concurrent connections and amount of requests. Each framework will be run 10 times to ensure statistical

significance. The following parameters were used for the Bombardier test:

- Concurrent connections: 25, 50, 100
- Amount of requests: 10000, 100000, 1000000

Each group of tests will run only on payment transaction query due to complexity of queries used in that transaction. Therefore, worth mentioning is fact that during tests object “createPool” will be used with property connectionLimit set to 100. The reason for that is concurrent connections boundary won’t be exceeded during tests.

4.4. TPC-C method

The method of using the TPC-C benchmark is to perform N number of queries per transactional endpoint in 60 seconds, the measure thus obtained, hereafter referred to as tpmC, will allow a reliable and relative result to be obtained in a real-world example of query application and database structure usage.

Each of the tests will be repeated 10 times to obtain reliable results for calculating the statistics of the average number of queries per 60 seconds for the ElysiaJS and Express.js development framework. By comparing the results of the two programming frameworks against the individual operations performed on the database in a bar chart, it will be possible to identify clear performance differences. Request endpoints are followed with similar code in both frameworks as shown in Listing 1.

Listing 1. Code example of tpmC transaction query in ElysiaJS

```
.get("/test/delivery/:time",
  async ({ params: { time } }) => {
    let totalElapsedTime = 0;
    let totalQueries = 0;

    // seconds
    time *= 1000;
    while (totalElapsedTime < time) {
      try {
        const startTime = new Date();

        const connection = await pool.getConnection().then(async (connection) =>
          const transaction = await delivery(connection, 1, 1, 1).then(()=>{
            return new Date() - startTime
          })
          return transaction;
        })
        totalElapsedTime += connection;
        totalQueries++
      } catch (err) {
        console.error("Error in Delivery transaction:", err);
      }
    }
    console.log(totalQueries)
    return totalQueries
  },
  {
    params: t.Object({
      time: t.Numeric(),
    }),
  }
)
```

5. Study results

5.1. Bombardier results

The tests, which were conducted according to a specific methodology, produced reproducible and reliable results, as can be seen in the averaged Table 2, each group of parameters, i.e. 25 concurrent connections per 10000 requests, 50 concurrent connections per 100000 requests etc., was performed 10 times for each development framework.

As a result, the average latency results shown in Figure 2 were achieved. Equally important information is the fact of the examined average standard deviation, which for the Express.js framework behaves over the three parameter groups in a range between 6 and 12 ms, where the compared ElysiaJS development framework from 11.5 to 58 ms. The comparison suggests significantly better optimization of the Express.js language, when handling multiple queries simultaneously.

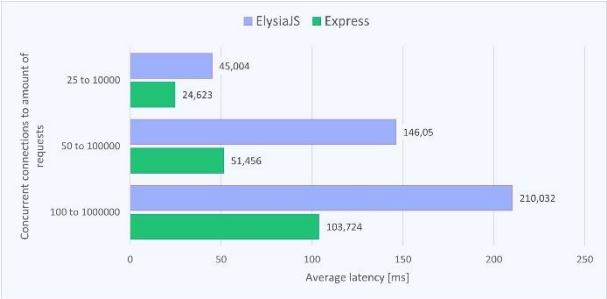

Figure 2. Summary of average latency results for each variant of test.

Another of the parameters analyzed is the average number of queries per second. When comparing the results in Figure 3, one can see an almost twofold difference in performance over the trials.

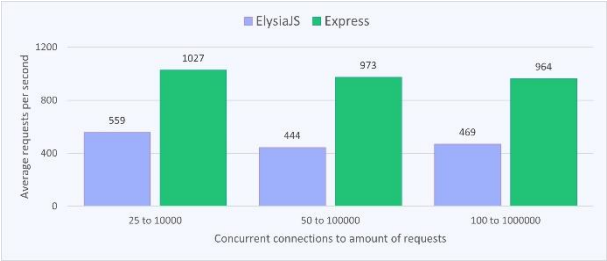

Figure 3. Summary of average request results for each variant of test.

5.2. TPC-C results

In the following part of the study, a single execution of each transaction query was performed, repeated 10 times. The difference in proceeding with the TPC-C method is due to the execution of a given transaction for 60 seconds until the N result, which is the number of transactions performed, that in other words tpmC. According to the collated data shown in Figure 4, it is possible to indicate which transactional queries were executed the slowest and which the fastest. The transactional process with the highest number of executions in a full minute is the stock level operation, with a score in the range of 236 to 255 thousand, to check the stock level. On the other hand, the process with the fewest executions is "order status" with a score in the range of 200 to 300 transactions made, serving to filter the data to check the status of an order and returning the corresponding data.

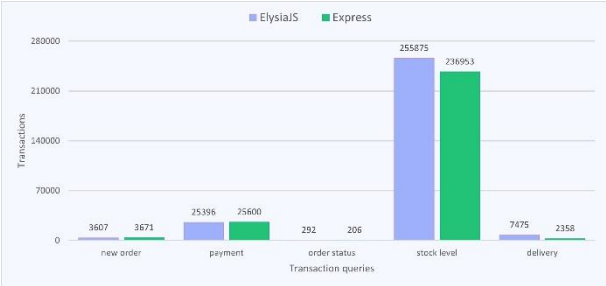

Figure 4. Average tpmC per transaction query between ElysiaJS and Express.js.

The study used asynchronous unit query execution until the transaction was complete, meaning that each query had only one connection to the database at a time.

6. Conclusions

The research was aimed at identifying which web framework is suitable for a specific and demanding application such as an online transaction service. The Bun runtime environment and the framework based on it, ElysiaJS, was intended and according to existing tests, to be a faster combination than NodeJS and Express.js. However, it appears that in practical use, the Express.js development framework dominates the still relatively recent environments and their development frameworks or does not differ in performance in any significant way from them. It is also worth taking into account the seniority of the competing technologies, as the Bun environment in its official version 1.0 was released on September 8, 2023, where NodeJS has been on the market since 27 May 2009. Years of optimization and fine-tuning of the Express.js framework structure led to the effects observed in the Bombardier tool test.

An aspect that raises the topic of discussion is the result of study 5.2 in which the comparison of tpmC averages obtained is close to each other despite the results of study 5.1, which suggests significant performance and optimization differences between the two developer frameworks. However, despite the deceptively similar performance specificities of the two methods, the key difference is in the behaviour and implementation of the backend server-side functions. Queries performed with the Bombardier load tool used a parameter to simulate parallel connections in a given time unit, in addition to which the createPool object's connectionLimit property was defined to correspond to the maximum number of concurrent connections; in order to avoid unsuccessful repetitions, this limit was set to 100. In contrast, the TPC-C method operated each time on a single connection in a given loop revolution.

Based on the testing performed and analysis of the compared results, it can be concluded that Express.js is optimized for parallel handling of multiple queries and prepared for applications to projects using extensive data structures. Thus, ElysiaJS shows statistically better performance when handling single queries, suggesting performance potential. In response to the research thesis posed, "The Express.js framework has better performance than ElysiaJS in terms of TPS measurement tests in TCP-C structure", the research showed that for handling multiple and parallel queries Express.js outperforms ElysiaJS by almost double, while for single queries the differences in TPS are similar or marginally greater for ElysiaJS.

Bibliography

[1] G. D. Samaraweera, J. M. Chang, Security and Privacy Implications on Database Systems in Big Data Era: A Survey, IEEE Transactions on Knowledge and Data Engineering 33 (2019) 239–258, https://doi.org/10.1109/TKDE.2019.2929794.

[2] A. Lith, J. Mattsson, Investigating storage solutions for large data - a comparison of well performing and scalable data storage solutions for real time extraction and batch insertion of data, Chalmers University of Technology, 2010.

[3] TPC-C Benchmark Standard Specification Revision 5.11, Transaction Processing Performance Council (2010), https://www.tpc.org/tpcc, [08.06.2024].

[4] Documentation of the programming framework, ElysiaJS, https://elysiajs.com/, [16.11.2023].

[5] Documentation of the programming framework, Express.js, https://expressjs.com/, [16.11.2023].

[6] Documentation of the run-time environment, Bun, https://bun.sh/docs, [01.03.2024].

[7] Documentation of the run-time environment, Node.js, https://nodejs.org/docs/latest-v21.x/api/index.html, [01.03.2024].

[8] J. R. Wilson, Node.js the Right Way:Practical Server Side Javascript that Scales, The Pragmatic Bookshelf, Raleigh, 2013.

[9] A. D. Díaz Erazo, M. Raúl Morales Morales, V. K. Pineda Chávez, S. Leonardo Morales Cardoso, Comparative Analysis of performance for SQL and NoSQL Databases, In 2022 17th Iberian Conference on Information Systems and Technologies (CISTI) (2022) 1-14, https://doi.org/10.23919/CISTI54924.2022.9820292.

[10] T. Seser, V. Pleština, F. Marjanica, Performance analysis of SQL Prepared Statements in CRUD operations, In 2022 7th International Conference on Smart and Sustainable Technologies (SpliTech), (2022) 1-5, https://doi.org/10.23919/SpliTech55088.2022.9854303.

[11] S. Chen, A. Ailamaki, M. Athanassoulis, P. B. Gibbons, R. Johnson, I. Pandis, R. Stoica, TPC-E vs. TPC-C: Characterizing the new TPC-E benchmark via an I/O comparison study, ACM Sigmod Record 39 (2011) 5-10, https://doi.org/10.1145/1942776.1942778.

[12] Documentation of http web tool, Bombardier, https://pkg.go.dev/github.com/codesenberg/bombardier, [08.06.2024].