JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 36 (2025) 296–302
Received: 1 April 2025
Accepted: 18 May 2025

Comperative analasys of JavaScript runtime environments

Analiza porównawcza środowisk uruchomieniowych JavaScript

Konrad Kalman*, Marek Miłosz

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

Abstract

This article presents a comparative analysis of two JavaScript runtime environments: Node.js, the industry leader since 2009, and Bun, a newer and promising alternative introduced in 2022. The study evaluates their performance in two distinct experimental scenarios: handling HTTP requests and executing a computationally intensive algorithm for calculating Fibonacci numbers. These scenarios allow for the assessment of both I/O-bound and CPU-bound workloads. Benchmarking tools were used to measure the most important metrics, including total request handling time, average latency, and peak memory usage. By analyzing architectural differences and runtime optimizations, the article highlights the advantages and trade-offs of both environments. The findings offer developers valuable insights for selecting an appropriate platform for high-performance and scalable server-side applications, and contribute to the ongoing discussion on the evolution of JavaScript runtimes.

Keywords: JavaScript runtime environments; Node.js; Bun; HTTP handling; memory usage analysis

Streszczenie

Artykuł przedstawia porównawczą analizę dwóch środowisk uruchomieniowych JavaScript: Node.js, lidera branży od 2009 roku, oraz Bun, nowszej i obiecującej alternatywy wprowadzonej w 2022 roku. W badaniu oceniono wydajność obu środowisk w dwóch odrębnych scenariuszach eksperymentalnych: podczas obsługi żądań HTTP oraz podczas wykonywania wymagającego obliczeniowo algorytmu obliczającego liczby ciągu Fibonacciego. Pozwoliło to na analizę zarówno zadań opartych na operacjach wejścia/wyjścia, jak i obciążeń procesora. W testach wykorzystano narzędzia benchmarkujące do pomiaru najważniejszych metryk jak, pomiar całkowitego czasu obsługi żądań HTTP, średniego opóźnienia, a także szczytowego zużycia pamięci. Na podstawie różnic architektonicznych i zastosowanych optymalizacji wydajności, artykuł ukazuje zalety i kompromisy obu technologii. Wyniki te mogą pomóc programistom w wyborze odpowiedniego środowiska do tworzenia skalowalnych i wydajnych aplikacji serwerowych oraz wzbogacają dyskusję na temat rozwoju środowisk uruchomieniowych JavaScript.

Słowa kluczowe: środowiska uruchomieniowe JavaScript; Node.js; Bun; obsługa HTTP; analiza użycia pamięci

*Corresponding author

Email address: **konrad.kalman@pollub.edu.pl** (K. Kalman)

1. Introduction

JavaScript runtime environments are essential for building modern web applications, enabling the execution of JavaScript code outside of the browser. Node.js, introduced in 2009 by Ryan Dahl, has become the most popular runtime environment, widely used for building application servers, development tools, and microservices. Powered by the high-performance V8 engine [1] developed by Google and also used in the Chrome browser, Node.js revolutionized server-side development with its asynchronous, event-driven approach to I/O handling, quickly gaining industry-wide adoption [2].

However, in recent years, new initiatives have emerged aiming to improve performance and enhance the capabilities of modern applications. One of these is Bun, a newer competitor to Node.js introduced in 2022. Bun was created by Jarred Sumner and is built on JavaScriptCore [3], the JavaScript engine developed by Apple and used in the Safari browser. Bun aims to accelerate JavaScript execution and improve efficiency through a tightly integrated toolchain, faster startup times, and more optimized resource management [4].

Beyond Node.js and Bun, other JavaScript runtime environments have also gained attention in the development community. Deno, created by the original author of Node.js, is one such example, offering a secure and modern runtime with TypeScript support out-of-the-box. Another example is Hermes, developed by Meta (formerly Facebook), which is optimized for running JavaScript in mobile environments, particularly in React Native applications.

In terms of implementation, Node.js is primarily written in C++, JavaScript, and C, while Bun is implemented in Zig, a low-level systems programming language known for its performance and safety features. These underlying technologies significantly influence how each runtime handles system-level operations, concurrency, and memory management.

This article conducts a comparative analysis of Node.js and Bun, focusing on technical aspects such as HTTP handling and memory usage, which are critical for developing efficient server-side applications. The analysis includes benchmarking tests to evaluate how both runtimes perform in real-world scenarios, such as processing HTTP requests and executing CPU-bound algorithms.

As Bun emerges as a compelling alternative to Node.js with the potential for significant performance improvements, this analysis seeks to explore its capabilities in depth. The findings will provide valuable insights for developers and system architects, helping them choose the most suitable runtime for their specific needs.

2. Literature review

The performance of JavaScript runtime environments plays a crucial role in the development of modern web applications. Among the most widely used JavaScript runtimes are Node.js and Bun [5]. While Node.js has dominated the server-side JavaScript landscape for over a decade, Bun, a relatively new entrant, promises performance improvements in various areas. This literature review examines recent studies and sources that provide insights into the performance of these runtimes, particularly in HTTP handling and memory usage.

Node.js has been the go-to JavaScript runtime for server-side development since its release in 2009. Built on Google's V8 JavaScript engine [1], Node.js is known for its asynchronous, event-driven architecture, which allows it to handle a large number of concurrent connections efficiently. It excels in handling I/O-bound operations, particularly in scenarios requiring non-blocking behaviour such as HTTP request processing. However, a recent performance benchmark [6] comparing Node.js to Bun, Go, C#, and Python revealed some of its limitations. In tests involving functionally identical REST APIs, Node.js ranked fourth in terms of request throughput, significantly behind both Go and Bun, which tied for first place. While Node.js remains popular due to its maturity and vast ecosystem, these results suggest that newer or alternative runtimes may offer substantial performance or memory usage benefits depending on the application's requirements.

Performance benchmarks comparing Node.js to other runtimes reveal that while it remains efficient in many use cases, its memory consumption can be relatively high, especially when handling large-scale applications or concurrent HTTP requests. For instance, Node.js can exhibit uneven memory usage, which may lead to performance degradation under high load conditions [7, 8]. This has prompted some developers to explore alternatives that promise better resource management without compromising speed.

Bun, introduced as a modern JavaScript runtime, has quickly gained attention for its high performance and innovative features. Built on the JavaScriptCore engine, Bun has been designed to optimize JavaScript execution and improve resource management compared to Node.js. One of the most important claims of Bun is its ability to handle HTTP requests more efficiently. Performance benchmarks have shown that Bun can process more requests per second than Node.js, making it a strong contender for applications requiring fast HTTP handling [8].

A significant advantage of Bun is its memory usage. Several studies have pointed out that Bun consumes less memory than Node.js when handling HTTP requests, which is crucial for applications with tight memory constraints. In a benchmark comparing Bun, Node.js, and other runtimes, Bun demonstrated that it could handle the same number of HTTP requests while using significantly fewer resources [9]. These optimizations make Bun an attractive choice for developers focused on memory efficiency and faster response times, particularly in high-performance scenarios.

In the context of performance testing of JavaScript runtime environments, several studies compare popular frameworks and libraries, such as AngularJS and ReactJS. Kowalczyk and Plechawska-Wójcik [10] conducted a detailed analysis of the performance of both technologies in the context of web applications. Their research revealed that AngularJS offered better memory efficiency compared to ReactJS, particularly in larger applications where memory management plays a crucial role. These findings serve as a reference for our tests, which focus on comparing the performance of runtime environments like Node.js and Bun, specifically in HTTP handling.

Comparative benchmarking between Node.js and Bun reveals interesting findings regarding their performance in different areas. For basic HTTP request handling, Bun consistently outperforms Node.js in both speed and memory consumption. According to a performance test conducted by the Bun development team, Bun showed a notable reduction in cold start times and improved module loading speeds compared to Node.js, contributing to faster overall execution [8]. This makes Bun a promising choice for developers who require faster execution and a leaner runtime.

In study on JavaScript framework performance, Grudniak and Dzieńkowski [11] carried out a detailed comparison between Express and Hapi, concentrating on performance and memory usage. Their analysis showed that Express tends to perform better than Hapi, especially in high-traffic situations with many concur-rent connections, where low latency and high throughput are essential. This research offers useful insights for our own investigation, which compares the performance of runtime environments like Node.js and Bun, with a particular focus on HTTP server perfor-mance and memory efficiency.

The comparative analysis of Node.js and Bun highlights significant differences in their performance, particularly in areas such as HTTP handling and memory usage. Node.js, with its well-established presence and large ecosystem, continues to be a reliable choice for many developers. However, as shown by recent benchmarks, Bun offers substantial performance improvements, especially in terms of HTTP request handling and resource management. Its lower memory consumption and faster execution times make it an appealing alternative for high-performance applications.

While Bun holds great potential, developers must weigh its benefits against its relatively new status and the lack of a mature ecosystem. Further research and long-term performance testing will be crucial to determining whether Bun can truly compete with Node.js in production environments at scale. For now, both runtimes offer valuable tools for different application needs, and the choice between them will depend on the specific requirements of the project, such as performance, memory constraints, and ecosystem compatibility.

3. Experiment settings

3.1. Environment specifications

The performance tests for this study were conducted in a controlled environment using a Windows PC. The testing environment was carefully selected to ensure consistency across all benchmarks. Unlike some previous benchmarks - such as those by F. Ahmod [9], which were conducted on MacBook hardware, this study uses a desktop-class Windows PC to ensure consistent performance under sustained loads. The difference in operating systems and hardware may account for some of the variations in results compared to prior studies. All tests were carried out according to the table of hardware and software specifications in Table 1.

Table 1: The specifications of the testing machine

Component	Specification
Processor	Intel i5-10400 @2.9Ghz
RAM	16GB DDR4 3600MHz CL17
Operating System	Windows 11 Pro 64-bit 24H2
Storage	WD 1TB M.2 PCIe NVMe Blue SN570

For accurate and up-to-date comparisons, the latest stable versions of both Node.js [12] and Bun [4] were used. The table below lists the versions of the JavaScript runtimes and associated tools used in this study. In addition to the runtimes, several software tools were necessary to conduct the performance testing. These tools were used to measure various performance metrics such as HTTP handling throughput, memory usage, and CPU-intensive tasks.

To measure HTTP throughput, Bombardier [13] was utilized. This high-performance HTTP benchmarking tool, written in Rust, is designed to generate high loads and test HTTP servers' capabilities under stress. It provides valuable insights into total completion time and server responsiveness under heavy traffic.

For accurate measurement of execution times, Hyperfine [14] was used. This command-line benchmarking tool is renowned for its high precision in evaluating the execution time of commands. It aids in comparing different runtimes by offering detailed results regarding time variations and consistency.

Process Explorer [15] was used to monitor system performance during the tests. This system monitoring tool provides an in-depth view of processes, memory usage, and CPU consumption. It was crucial in ensuring that resource usage was accurately tracked throughout the testing process.

Table 2: The specifications of runtimes and tools

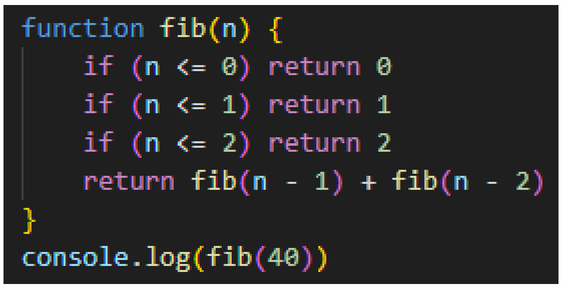
Environment/Tool	Version
Node.js	v22.13.0 LTS
Bun	v1.1.43
Bombardier	v1.2.6
Hyperfine	v1.19.0
Process Explorer	v17.06

3.2. Execution time test for recursive Fibonacci calculation

To evaluate the performance of Node.js and Bun in CPU-bound scenarios, a comparative test was designed using a deliberately inefficient recursive algorithm for calculating Fibonacci numbers. The structure of this test was inspired D. Herron [16], which discusses the use of a naive recursive implementation to emphasize the impact of CPU load on JavaScript runtimes.

The test employed a synchronous, single-threaded function that recursively computes Fibonacci numbers without any form of optimization such as memoization. The function structure strictly follows a naive recursive pattern, where each number is calculated by summing the results of two prior recursive calls. This approach leads to exponential growth in the number of recursive calls as the input increases, thus creating a highly CPU-intensive task.

Listing 1: Fibonacci calculation setup



```
function fib(n) {
    if (n <= 0) return 0
    if (n <= 1) return 1
    if (n <= 2) return 2
    return fib(n - 1) + fib(n - 2)
}
console.log(fib(40))
```

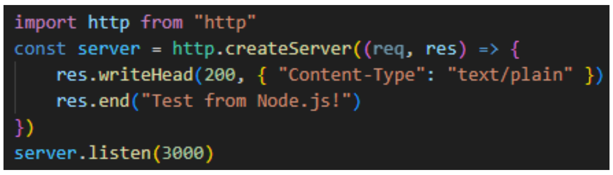
The same synchronous function was executed under both Node.js and Bun environments for input values ranging from 25 to 40. These values were selected to simulate increasing computational complexity and CPU stress. The benchmarking tool Hyperfine was used to perform 50 executions of the Fibonacci program on both Node.js and Bun, with 20 warmup runs to stabilize the runtimes before measurement began. Hyperfine ensures accurate measurement of execution time. This setup effectively highlights differences in execution speed and efficiency between the two runtimes in tasks that are purely CPU-bound, offering insights that are not influenced by asynchronous I/O operations.

3.3. HTTP server request handling and concurrency test

The second test aimed to evaluate the ability of Node.js and Bun to handle high-concurrency HTTP requests, a critical requirement for real-world web applications. As web servers often face fluctuating levels of concurrent traffic, this experiment measured how well each runtime manages high volumes of simultaneous connections under stress.

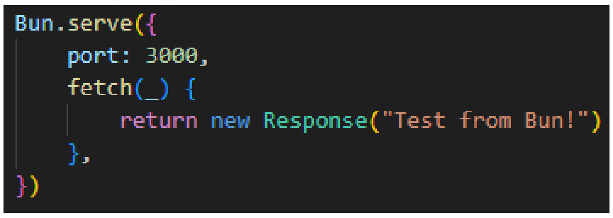
Two minimal HTTP servers were implemented for this purpose. The Node.js version used the built-in http module (Listing 2), while the Bun implementation leveraged its native Bun.serve function (Listing 3). Both servers listened on port 3000 and returned a short plain-text response.

Listing 2: Node.js HTTP Server



```
import http from "http"
const server = http.createServer((req, res) => {
    res.writeHead(200, { "Content-Type": "text/plain" })
    res.end("Test from Node.js!")
})
server.listen(3000)
```

Listing 3: Bun HTTP Server



```
Bun.serve({
    port: 3000,
    fetch(_) {
        return new Response("Test from Bun!")
    },
})
```

Using the Bombardier benchmarking tool, we simulated load with 10, 100, and 500 concurrent connections. Bombardier was chosen because it can simulate real-world HTTP request loads, allowing for a detailed comparison of how each server performs under pressure. For each level of concurrency, 1 million HTTP requests were sent to the servers, ensuring a substantial load to stress-test the runtimes.

Throughout the tests, several most important performance metrics were measured. The total completion time indicated how long it took each server to handle the full batch of requests at a given concurrency level, reflecting the runtime's ability to manage high volumes of traffic. This metric is particularly useful for evaluating throughput under realistic, large-scale conditions. This is a crucial metric, as it directly relates to the throughput of the server and how well the runtime can manage high-traffic scenarios.

Additionally, average latency was measured, providing a deeper analysis of the delays encountered when processing requests. This reflects the mean delay per request and helps understand the server's responsiveness from the client's perspective. Lower latency values indicate faster and more efficient request handling. By tracking latency across different concurrency levels, we were able to evaluate how quickly requests were processed, particularly under high load. High latency often indicates bottlenecks in the server's ability to handle traffic, and this test was designed to expose any such weaknesses.

3.4. Peak memory usage measurement using Process Explorer

This test aimed to measure the peak memory usage of both Node.js and Bun during the HTTP server request handling tests. Memory usage is a critical factor for web servers, particularly in high-concurrency environments where inefficient memory management can lead to poor performance, increased latency, or even system crashes. By tracking the memory consumption of both runtimes during the HTTP load tests, we sought to determine which runtime handles memory more efficiently while serving large numbers of requests.

The memory usage of the HTTP server applications was tracked using Process Explorer. This tool was chosen for its ability to monitor real-time system metrics, allowing us to track the exact memory consumption of the Node.js and Bun processes during the execution of the HTTP server test.

4. Study results

The performance of Node.js and Bun was evaluated through a series of tests, focusing on both CPU-bound and I/O-bound tasks. In particular, the Recursive Fibonacci Calculation and HTTP server handling tests provided valuable insights into the relative strengths and weaknesses of these two JavaScript runtime environments.

4.1. Recursive Fibonacci calculation execution time

One of the core tests involved calculating the 40th Fibonacci number recursively, a CPU-bound task known for its exponential growth in computational complexity. This kind of task puts significant strain on the runtime's engine, making it an effective benchmark for assessing raw CPU performance. F. Ahmod [9] previously conducted a similar test and found that Bun outperformed Node.js for the 40th Fibonacci number, providing faster execution times. Our results confirm this finding - Bun was indeed faster than Node.js for calculating the 40th Fibonacci number, with Bun completing the task in 561.5 ms, compared to Node.js at 749.8 ms.

Where our research expands on Ahmod's work is in examining the performance of both runtimes across a broader range of Fibonacci values, particularly for smaller inputs. While Ahmod focused on the upper limit of recursive complexity, we investigated values ranging from 25 to 40 to get a more comprehensive understanding of how each runtime behaves under varying levels of CPU demand.

Interestingly, our tests revealed that Node.js consistently outperformed Bun for Fibonacci numbers between 25 and 33, sometimes by a significant margin. This performance trend was not highlighted in Ahmod's study but offers valuable insight for developers whose applications rely on smaller-scale recursive computations. For these less intense CPU loads, Node.js's V8 engine appears to be more efficient, likely due to its mature optimization techniques and well-tuned garbage collector for lighter workloads.

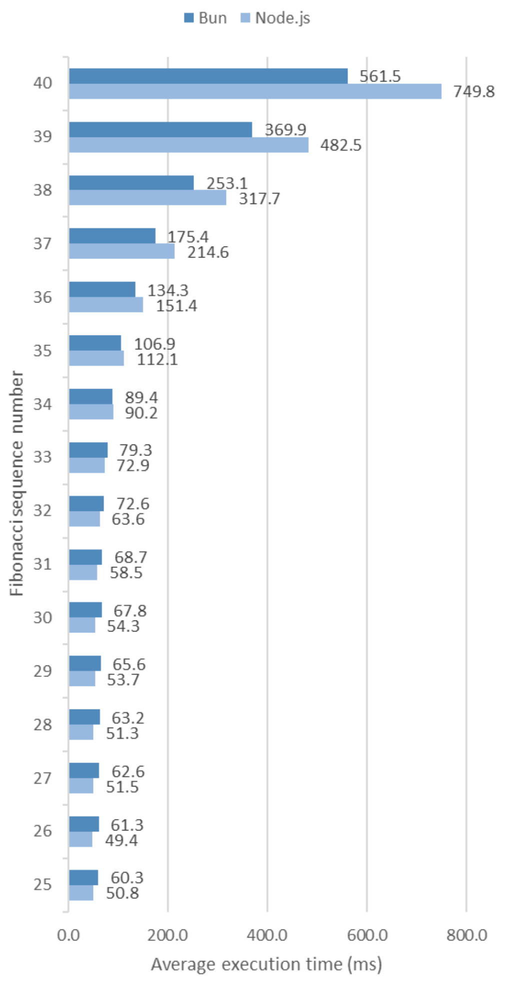


Figure 1: Summary of average execution time results for each runtime.

However, as the input size increased and the recursion depth grew, the performance advantage shifted in favor of Bun. For Fibonacci numbers above 33, Bun started to outpace Node.js, ultimately proving to be significantly faster for the 40th number. This shift suggests that Bun's JavaScriptCore engine, which powers its runtime, may be better optimized for handling deeper recursion and heavier computational tasks. JavaScriptCore's architecture could be leveraging more efficient stack management or optimized inlining strategies that allow it to scale better under CPU-bound conditions.

In conclusion, while Node.js remains a strong performer for tasks involving moderate computational loads, Bun shows clear advantages when scaling into more complex, CPU-intensive scenarios. This distinction is important for developers who need to choose a runtime environment based not only on ecosystem or familiarity but also on the specific nature of the workloads their applications will handle.

4.2. HTTP server performance - completion time and latency

The second set of tests focused on measuring how Node.js and Bun handle high volumes of concurrent HTTP requests, a common and critical scenario in web server workloads. Minimal HTTP servers were deployed in each runtime and subjected to load using Bombardier with 10, 100, and 500 concurrent connections. Each test consisted of processing 1 million HTTP requests.

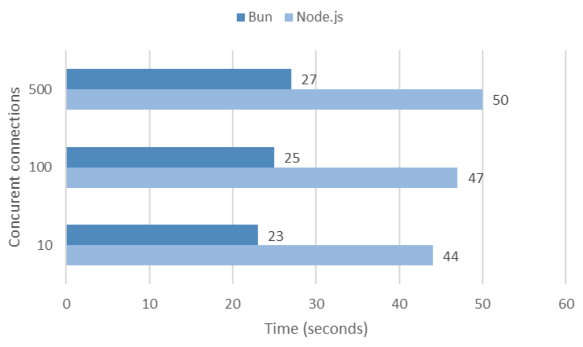


Figure 2: Summary of completion time results for each runtime.

Bun consistently outperformed Node.js in total completion time across all concurrency levels. For instance, Bun completed the test with 10 concurrent connections in 23 seconds, while Node.js took 44 seconds. At 500 concurrent connections, Bun completed in 27 seconds, compared to Node.js at 50 seconds. These results highlight Bun's ability to scale under heavy traffic, finishing the same task in nearly half the time.

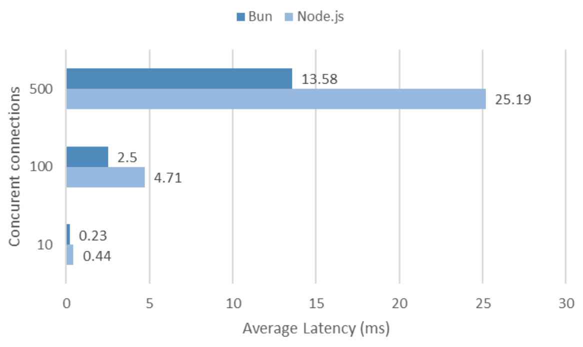


Figure 3: Summary of average latency results for each runtime.

In addition to total time, average latency was tracked to assess responsiveness. Lower latency indicates quicker responses to incoming requests — the most important factor in user experience and real-time performance. Bun achieved an average latency of 0.23 ms at 10 connections and 13.58 ms at 500, while Node.js reported 0.44 ms and 25.19 ms, respectively. These differences demonstrate Bun's capacity to maintain low response times even under heavy concurrency. Overall, Bun's architecture seems better suited to handling high I/O loads efficiently, achieving faster throughput and lower delay.

The time to complete the HTTP server tests provided further evidence of Bun's superior performance under load. In all cases - whether handling 10, 100, or 500 concurrent connections - Bun completed the tests in almost half the time compared to Node.js.

4.3. Memory usage comparison

While Bun demonstrated faster performance in both CPU- and I/O-bound scenarios, this came with a tradeoff in memory usage. Using Process Explorer, we monitored the peak memory consumption during the HTTP server benchmarks at each concurrency level.

Across the board, Bun consumed more memory than Node.js. This result contrasts with findings by F. Ahmod [9], who observed lower or comparable memory usage by Bun. We attribute the discrepancy to differences in testing environments — particularly the operating system and monitoring tools.

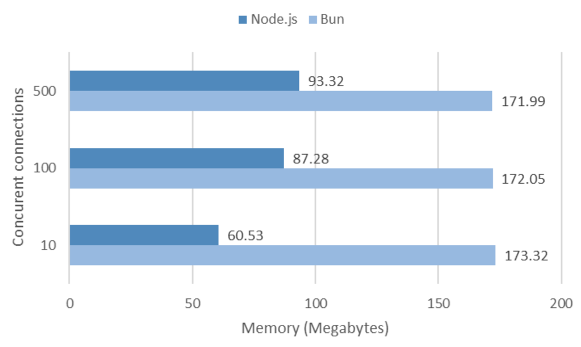


Figure 4: Summary of peak memory usage results for each runtime.

Despite Bun's higher memory footprint, there was no observable performance penalty. In fact, Bun maintained superior speed and lower latency throughout the tests. This suggests that Bun may intentionally trade memory efficiency for higher execution throughput and responsiveness. For environments with sufficient memory, this is a reasonable and often beneficial design decision.

However, developers targeting memory-constrained systems should consider this tradeoff carefully. While Bun delivers better performance metrics, Node.js may be more suitable in environments where memory efficiency is a top priority.

5. Conclusions

The goal of this research was to compare the performance of two JavaScript runtime environments - Node.js and Bun - across different types of tasks, with a particular focus on execution time and memory usage. The study aimed to identify which runtime is more efficient for specific tasks, such as recursive computations and handling large numbers of HTTP requests with varying levels of concurrency.

In the recursive Fibonacci calculation test, our results confirmed the findings of F. Ahmod [9], who noted that Bun outperformed Node.js in calculating the 40th Fibonacci number, completing the task faster. However, this research also uncovered new insights not previously mentioned. For less complex calculations (Fibonacci numbers below 33), Node.js proved to be faster than Bun. This suggests that Node.js may be better suited for simpler, less computationally demanding tasks, while Bun demonstrates its advantages when handling more complex, CPU-intensive operations.

When it came to handling large volumes of HTTP requests, Bun once again emerged as the superior runtime. Bun consistently outperformed Node.js in terms of time to complete and latency, completing tasks nearly twice as fast across all concurrency levels. Node.js, on the other hand, demonstrated slower request handling with higher latency, especially as the number of concurrent connections increased. This indicates that Bun is particularly well-optimized for high-concurrency, I/O-bound operations, making it a strong candidate for applications requiring high throughput and low response times.

While Bun outperformed Node.js in execution speed and request handling, it did so at the cost of higher memory usage. Bun consistently consumed more memory than Node.js during the HTTP server tests, which might be a result of its more aggressive approach to performance optimization. In environments with constrained memory resources, this could be a trade-off that developers need to carefully consider. Despite its higher memory usage, Bun's superior performance in high-concurrency scenarios demonstrates that it effectively leverages available resources to maximize throughput and minimize latency.

Overall, the research demonstrates that Bun is a high-performance alternative to Node.js, particularly for more complex and resource-intensive applications. Its ability to handle large numbers of concurrent HTTP requests more efficiently, combined with its faster execution times for computationally intensive tasks, makes it a strong contender for developers looking to optimize the performance of their applications. However, the higher memory usage of Bun means that it might not be the best choice in memory-constrained environments, where Node.js's more efficient memory management could offer advantages.

Based on the results of this study, it can be concluded that Node.js is more efficient for simpler, less computationally demanding tasks. Bun excels in more complex, high-concurrency environments, offering faster response times and better performance for I/O-bound tasks.

In response to the question of which runtime is better suited for demanding server-side applications, this research shows that Bun outperforms Node.js in terms of both speed and latency, but developers must weigh these advantages against Bun's higher memory usage. As Bun continues to mature, further optimizations may address its memory consumption, making it an even stronger alternative for high-performance web applications.

References

[1] V8 Engine documentation, https://v8.dev/, [02.01.2025].

[2] R. Dahl, 10 Things I Regret About Node.js, In JSConf EU (2018) Berlin.

[3] JavaScriptCore Engine documentation, https://developer.apple.com/documentation/javascriptcore, [02.01.2025].

[4] Bun documentation, https://bun.sh/docs, [02.01.2025].

[5] I. Kniazev, A. Fitiskin, Choosing The Right Javascript Runtime: An In-Depth Comparison Of Node.Js And Bun, Norwegian Journal of Development of the International Science 108 (2023) 72–84, https://doi.org/10.5281/zenodo.7945166.

[6] S. Wortham, Performance Benchmarking: Bun vs. C# vs. Go vs. Node.js vs. Python, https://www.wwt.com/blog/performance-benchmarking-bun-vs-c-vs-go-vs-nodejs-vs-python, [02.01.2025].

[7] K. Carmo, F. Ferreira, E. Figueiredo, Performance Evaluation of Back-End Frameworks: A Comparative Study, Proceedings of the 20th Brazilian Symposium on Information Systems 43 (2024) 1–9, https://doi.org/10.1145/3658271.3658314.

[8] I. Hernandez, Bun vs. Node: Harder, Better, Faster, Stronger?, https://www.dreamhost.com/blog/bun-vs-node/, [08.01.2025].

[9] F. Ahmod, JavaScript Runtime Performance Analysis: Node and Bun, Master's thesis, Faculty of Information Technology and Communication Sciences, 2023.

[10] K. Kowalczyk, M. Plechawska-Wójcik, AngularJS and ReactJS libraries - performance analysis, Journal of Computer Sciences Institute 2 (2016) 114–119, https://doi.org/10.35784/jcsi.126.

[11] M. Grudniak, M. Dzieńkowski, REST API performance comparison of web applications based on JavaScript programming frameworks, Journal of Computer Sciences Institute 19 (2021) 121–125, https://doi.org/10.35784/jcsi.2620.

[12] Node.js documentation, https://nodejs.org/docs/latest-v22.x/api/, [11.01.2025].

[13] Bombardier documentation, https://github.com/codesenberg/bombardier, [02.01.2025].

[14] Hyperfine documentation, https://github.com/sharkdp/hyperfine, [02.01.2025].

[15] Process Explorer documentation, https://learn.microsoft.com/en-us/sysinternals/downloads/process-explorer, [02.01.2025].

[16] D. Herron, Node. js Web Development: Server-side web development made easy with Node 14 using practical examples, Packt Publishing Ltd, 2020.