Performance Evaluation Report

Aadarsha Gopala Reddy

M.Sc. Computer Science
Washington University in St. Louis

Contents

1 Introduction

This study evaluates the performance of different web servers and programming languages under varying load conditions. I utilized ApacheBench to benchmark both nginx and Apache servers by sending a specific number of concurrent requests. Performance metrics such as requests per second, time per request, and transfer rate were recorded for different request volumes to assess scalability and efficiency. Additionally, I compared the performance of JavaScript and Python, by implementing a prime number search algorithm in both languages. The algorithm was tested with different input sizes to evaluate the performance of each language. The results of this study can be used to determine the optimal web server and programming language for a given application based on performance requirements. Additionally, Python was employed to develop automation scripts for executing the benchmarks, processing the collected data, and generating visualizations. Python's extensive libraries and ease of integration made it an essential tool for streamlining the testing workflow and analyzing the performance results effectively.

2 Web Servers

Apache vs. Nginx

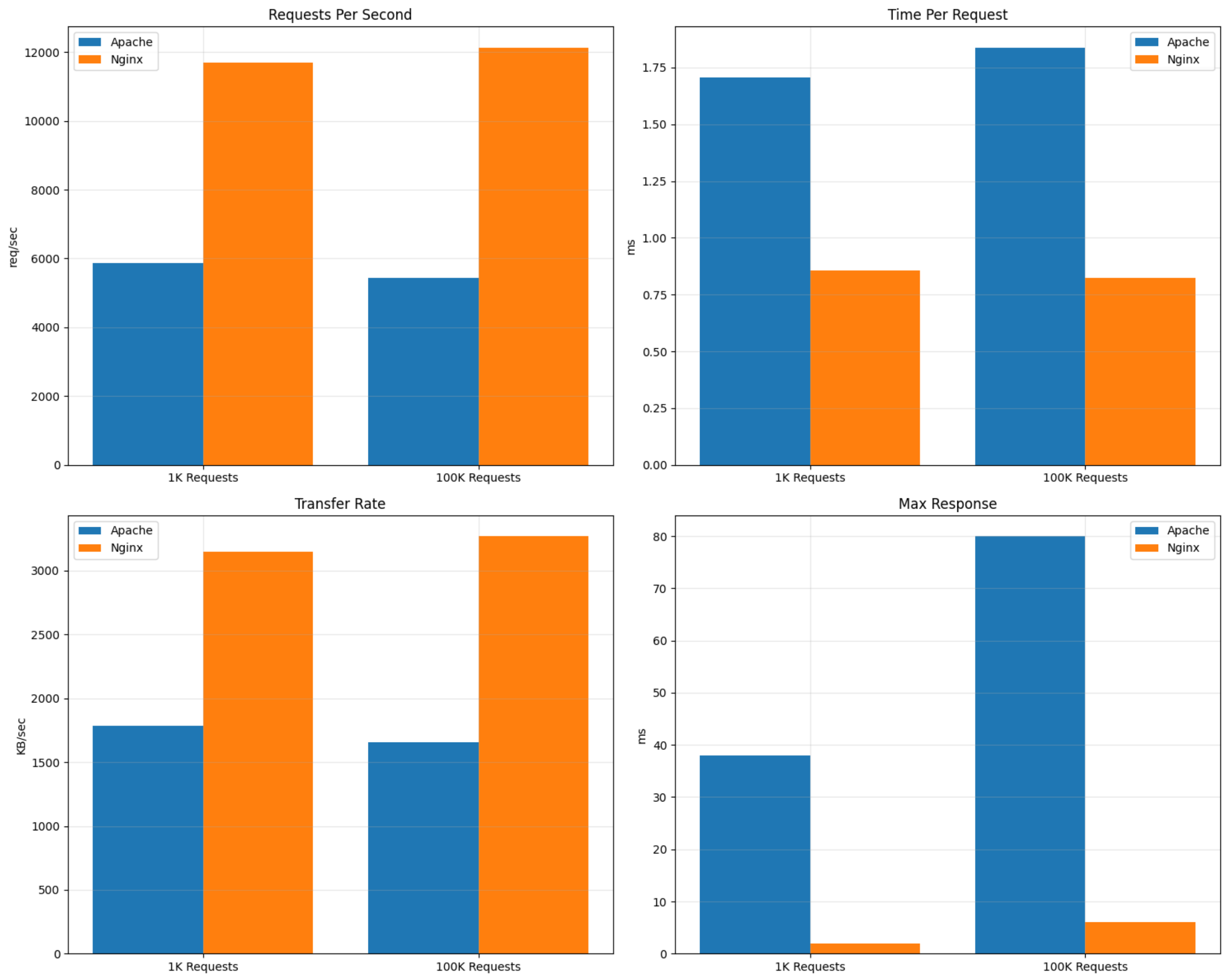

Figure 1: Performance comparison between Apache and Nginx

2.1 Apache

Apache is a popular open-source web server that is widely used on the internet. It is known for its stability, security, and flexibility. Apache is highly configurable and can be extended with a wide variety of modules to support different functionalities, such as server-side scripting, caching, and load balancing.

In terms of performance, Apache is generally considered to be a solid performer, but it may not be the fastest web server available. Its performance can be affected by a number of factors, including the hardware it is running on, the configuration settings, and the type of traffic it is serving.

2.2 Nginx

Nginx is another popular open-source web server that has gained significant popularity in recent years. It is known for its high performance, scalability, and efficiency. Nginx is particularly well-suited for handling high traffic loads and serving static content. It is also commonly used as a reverse proxy and load balancer. Nginx's performance is generally considered to be very good, especially for handling concurrent connections and serving static content. It is designed to use minimal resources and can handle a large number of requests with low latency.

2.3 Performance Analysis

The performance metrics displayed in Figure 2 definitively demonstrate Nginx's superior performance compared to Apache. The benchmark results reveal that Nginx consistently outperforms Apache across all key metrics: achieving higher requests per second, maintaining lower time per request, delivering high transfer rates, and most notably, exhibiting significantly reduced maximum response times. These empirical results strongly support Nginx's reputation as the more efficient web server solution.

The choice between Apache and Nginx depends on the specific needs of the application. Apache is a versatile and feature-rich web server that is well-suited for a wide range of applications. Nginx is a high-performance web server that is particularly well-suited for handling high traffic loads and serving static content.

3 Web Servers

Python vs. JavaScript

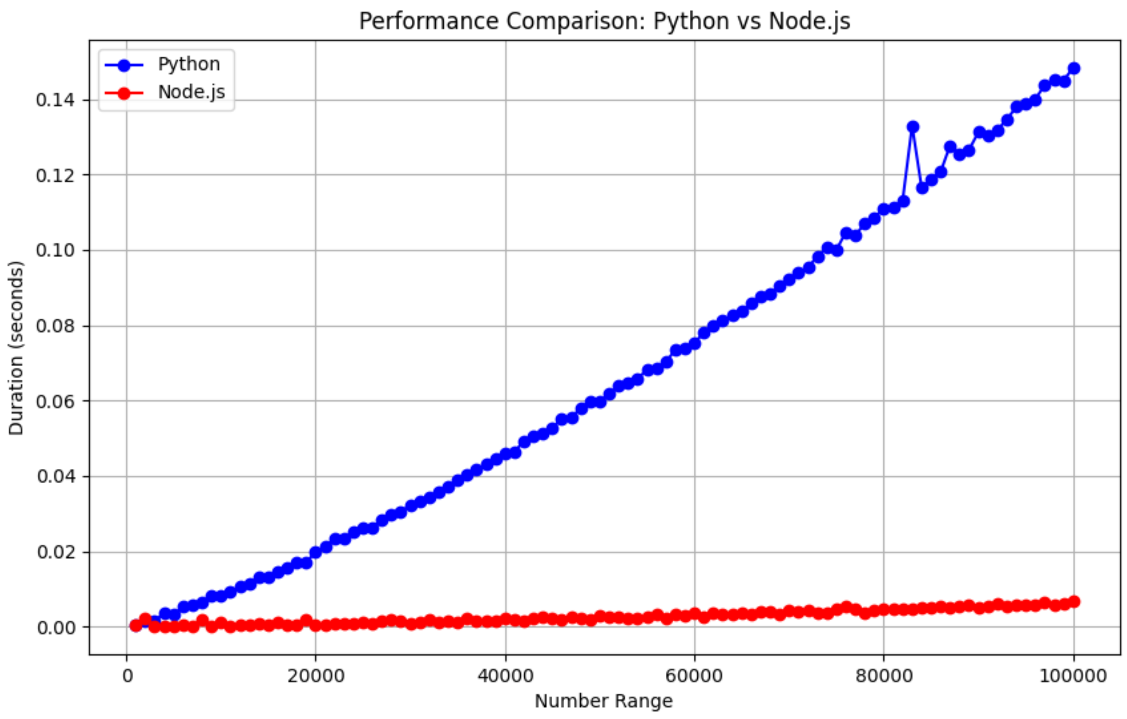

Figure 2: Performance comparison between Python and JavaScript

3.1 Python

The Python code defines a function `is_prime()` that checks if a number is prime and a function `find_primes()` that generates a list of prime numbers up to a given limit. It then runs a test to find primes within various ranges (1,000 to 200,000 in steps of 1,000) and measures the time taken for each range. The results, including the range, number of primes found, and duration, are saved in a CSV file named "python_results.csv".

3.2 JavaScript

The JavaScript code mirrors the Python code in functionality. It defines `isPrime()` and `findPrimes()` functions with the same logic. It also iterates through ranges (1,000 to 200,000 in steps of 1,000) to find prime numbers and records the time taken. The results are stored in a CSV file named "nodejs_results.csv".

3.3 Performance Analysis

Based on the provided graph in Figure 2, it's evident that Node.js significantly outperforms Python in this prime number finding task. Here's a breakdown:

- **Efficiency:** Node.js consistently demonstrates a much faster execution time across all number ranges. This efficiency could be attributed to Node.js's non-blocking, event-driven architecture, which might handle the iterative prime checking more effectively than Python's sequential approach.
- **Scalability:** As the number range increases, the performance gap between Node.js and Python widens further. This trend suggests that Node.js scales better with larger computational tasks, making it potentially more suitable for CPU-intensive operations.
- **Consistency:** Node.js exhibits a smoother performance curve, indicating more consistent execution times. Python, while generally slower, shows some fluctuations in its performance, particularly around the 80,000 number range where a spike is observed.

Overall, the results highlight Node.js's superior performance in this specific scenario. However, it's important to remember that performance can be influenced by various factors, including hardware, specific code implementation, and the nature of the task itself.

4 Conclusions

This study provided a comparative analysis of web server performance, highlighting the strengths of Nginx over Apache in handling high request volumes and delivering content efficiently. The evaluation of programming languages revealed Node.js as a significantly faster option compared to Python for CPU-intensive tasks like prime number generation, likely due to its event-driven, non-blocking architecture. However, it is crucial to acknowledge that performance is multifaceted and influenced by various factors beyond the scope of this study.

Key findings include:

- Nginx outperforms Apache in handling high request volumes and efficient content delivery.

- Node.js is significantly faster than Python for CPU-intensive tasks due to its event-driven, non-blocking architecture.

Potential bottlenecks:

- Network latency, particularly in geographically diverse user bases, can significantly impact response times.
- CPU and memory limitations during peak traffic or computationally demanding operations could lead to performance degradation.
- Efficiency of data storage and retrieval mechanisms, as well as database architecture, can become bottlenecks if not properly optimized.

Recommendations to mitigate these issues:

- Implement a cloud-based infrastructure with auto-scaling capabilities for dynamic resource adjustment based on real-time demand.
- Employ a Content Delivery Network (CDN) to minimize latency by caching content closer to users.
- Optimize databases using techniques such as indexing and query optimization to improve data access speeds.

Code-level optimizations:

- In Python, leverage asynchronous programming and libraries like NumPy for numerical computations.
- In Node.js, manage asynchronous operations carefully and utilize memory efficiently.
- Use profiling tools to identify performance bottlenecks within the code for targeted optimizations.

It is important to recognize that the optimal configuration is highly dependent on the specific application and its usage patterns. Continuous monitoring and performance testing are essential to identify and address bottlenecks as they emerge. By combining infrastructure optimization, code-level enhancements, and ongoing performance analysis, a robust and scalable web application can be achieved.