DV1619 Assignment 4

PRAJNA BALA SAI POTNURI
Dept. of Computer Science
Blekinge Institute of Techonlogy
Karlskrona, Sweden.
prpo21@student.bth.se

***Abstract*—This report investigates the performance of two server architectures forked and threaded under varying levels of concurrency. The primary goal is to compare how each server type handles increasing requests and to identify the optimal concurrency level for each. Through a series of performance tests, the threaded server was shown to outperform the forked server at higher concurrency levels, achieving a higher number of requests per second due to its lower overhead in thread management. However, as concurrency levels increased, the threaded server also exhibited greater performance variability, suggesting potential resource contention. Linear regression was applied to predict server performance at higher concurrency levels, with reasonable success in forecasting general trends but limited accuracy in modeling the performance fluctuations of the threaded server. These findings suggest that while the forked server is better suited for low-concurrency, stable environments, the threaded server is more efficient at handling high-concurrency workloads, albeit with some performance variability. This analysis provides useful insights for selecting the appropriate server architecture based on workload demands.**

***Index Terms*—Web server performance, threading, forking, concurrency, Apache Benchmark, scalability, response time, requests per second.**

I. Introduction

In today's digital age, web servers play a critical role in delivering content and services to users. The performance of a web server directly affects the user experience, particularly in terms of response time and scalability under heavy loads. As websites and applications continue to grow in complexity, understanding the underlying architecture of web servers is crucial for optimizing performance.

This report focuses on comparing two popular web server architectures: threaded and forked. Each architecture handles incoming client requests differently, which can significantly impact the server's ability to scale and manage high levels of concurrency. *Threaded servers* handle multiple requests within a single process using threads, allowing for more efficient resource sharing. *Forked servers*, on the other hand, create a new process for each request, providing isolation but at the cost of higher overhead.

The primary objective of this study is to evaluate the performance of both architectures under different levels of client load. By using the Apache Benchmark tool, I measure key performance metrics such as requests per second and response time. These findings will provide insight into the strengths and limitations of each approach, helping developers make informed decisions about which architecture to adopt for specific types of web applications.

II. Test Environment

The performance testing for this study was conducted on a machine running the Ubuntu 23.0 operating system. The specific hardware and software environment used to carry out the experiments is as follows:

- **Processor:** Dual-core 11th Gen Intel Core i5-1135G7, 2.419 GHz
- **Memory (RAM):** 5.52 GB (with 1.21 GB used during the test)
- **Storage:** 25 GB SSD (21% used)
- **Kernel:** 6.5.0-44-generic x86_64
- **Shell:** Bash 3.3.29

The system specifications are shown in the figure below:

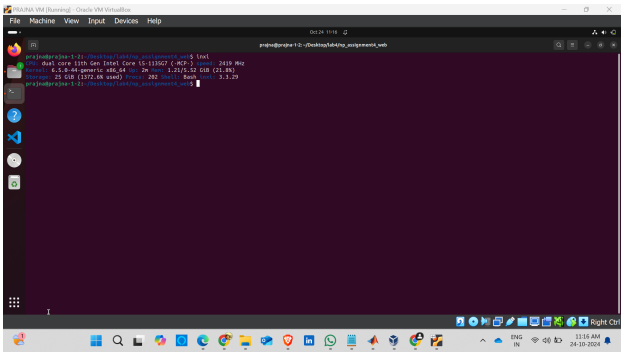

Fig. 1: System specifications used for the performance tests.

For performance measurement, the Apache Benchmark (ab) tool was utilized to simulate HTTP requests and evaluate the web server's ability to handle different levels of concurrency. Two implementations of the web server—one using a threaded model and the other using a forked model—were set up to serve content over the network.

Each web server was subjected to different levels of concurrency, varying from 1 to 10 concurrent clients, to assess how well they scaled with increased traffic. Key performance metrics measured included:

- **Requests per second (RPS):** The number of HTTP requests the server could handle per second.

- **Time per request:** The average time required to respond to a single client request.

The following command was used for the Apache Benchmark tests:

```
ab -n 10000 -c [concurrency] http://127.0.0.1:[port]/[file]
```

where `-n 10000` refers to the total number of requests, `-c [concurrency]` defines the concurrency level, and `[file]` represents either a large or small file served by the web server.

III. Parameters

The performance tests were conducted to measure the effectiveness of two different web server architectures—forked and threaded—when handling concurrent HTTP requests. The goal was to analyze how each server type responds to various file sizes (small and large) and different concurrency levels.

A. Concurrency Level

The concurrency level (C) represents the number of concurrent clients sending requests to the server at the same time. For this study, I used a concurrency level of 5, simulating 5 clients accessing the server simultaneously.

B. File Sizes

Two different file sizes were used in the tests to simulate the serving of content of varying sizes:

- **Small file**: 1000 bytes
- **Large file**: 10000 bytes

C. Tested Metrics

The following performance metrics were measured during the tests:

- **Requests per second (RPS)**: The number of requests the server could handle per second.
- **Time per request (TPR)**: The average time required to process a single client request.
- **Transfer rate**: The amount of data (in kilobytes) transferred per second.

D. Apache Benchmark (AB) Setup

The tests were conducted using the Apache Benchmark (AB) tool with the following command:

```
ab -n 10000 -c 10 http://127.0.0.1:[port]/[file]
```

Where:

- `-n 10000`: Specifies that 10,000 requests were sent to the server.
- `-c 10`: Defines the concurrency level (10 clients sending requests simultaneously).
- `[port]`: The port number of the server (e.g., 8283 or 8282).
- `[file]`: Specifies the file path to the small or large file being tested (e.g., `/small`, `/big`).

E. Results

The performance of both the *forked* and *threaded server* implementations was measured and summarized in the following tables I II and figures 2 3 4 5.

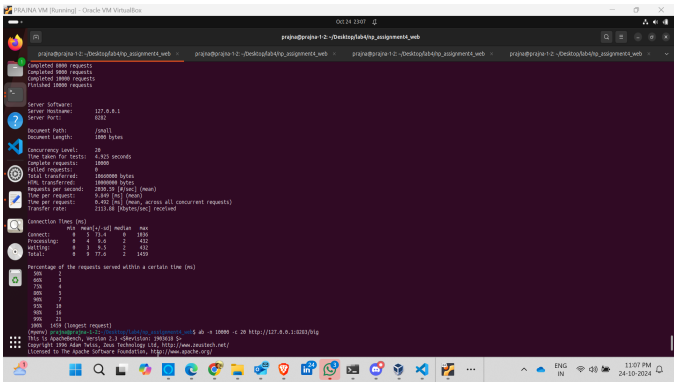

Fig. 2: Forked Server: Small File Performance

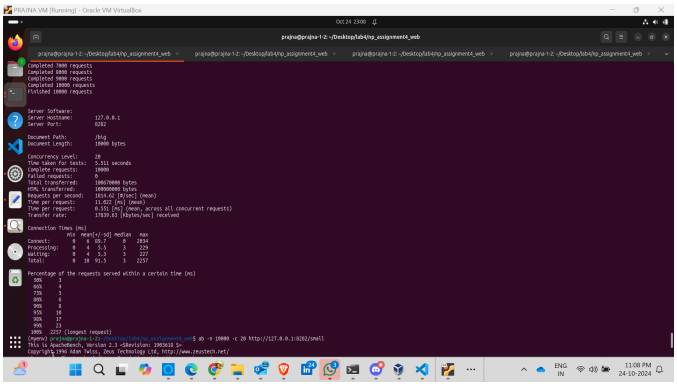

Fig. 3: Forked Server: Large File Performance

TABLE I: Forked Server Performance for Small and Large Files

Test Condition	Requests per Second (RPS)	Time per Request (TPR) (ms)	Transfer Rate (KB/s)
Small file (1000 bytes)	2030.59	9.849	2113.88
Large file (10000 bytes)	1814.62	11.022	17839.63

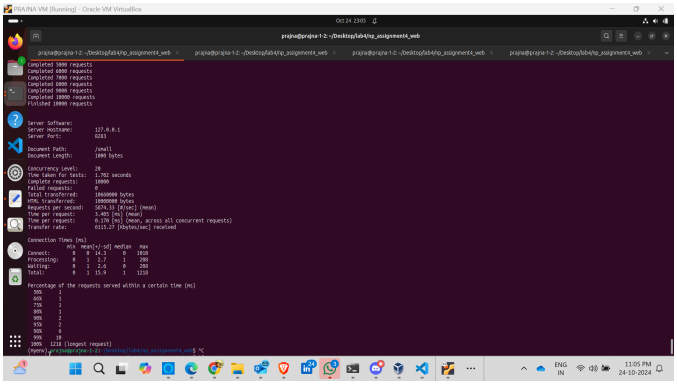

Fig. 4: Threaded Server: Small File Performance

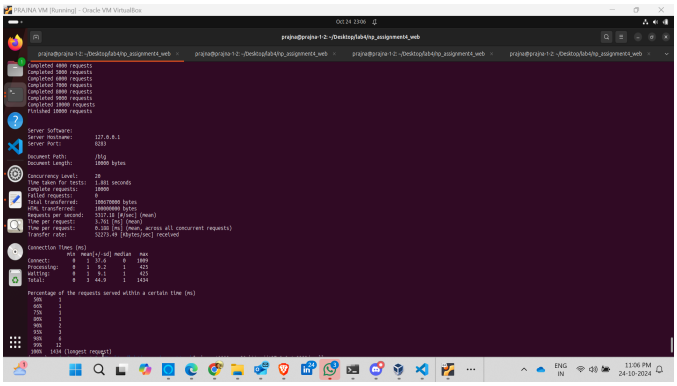

Fig. 5: Threaded Server: Large File Performance

TABLE II: Threaded Server Performance for Small and Large Files

Test Condition	Requests per Second (RPS)	Time per Request (TPR) (ms)	Transfer Rate (KB/s)
Small file (1000 bytes)	5874.33	3.405	6115.27
Large file (10000 bytes)	5317.18	3.761	52273.49

IV. Discussion & Results

The experiments comparing the forked and threaded servers provide a clear understanding of their performance characteristics under different concurrency levels. Based on the results, I observed several key differences in how each server architecture handles increasing workloads.

A. Performance Comparison: Forked vs. Threaded Servers

As shown in the original experimental data (Figure 6), the *threaded server* exhibits a sharp performance increase as concurrency levels rise, achieving significantly higher requests per second compared to the *forked server*. In contrast, the *forked server* maintains a relatively steady performance after a certain concurrency level, showing much less variability as concurrency grows. This difference stems from the underlying architecture of each server:

- **Forked Server**: Each request is handled by a separate process, which incurs higher overhead due to context switching and memory management. This overhead causes the performance to plateau early as concurrency increases.
- **Threaded Server**: Threads, which share the same process space, have less overhead for context switching and memory management, allowing them to handle larger numbers of concurrent requests more efficiently.

B. Linear Regression Analysis

To better understand and predict the server performance at higher concurrency levels, I applied *Linear Regression* to both the *forked* and *threaded* server data. The goal was to predict the performance (requests per second) for concurrency levels up to 60 based on the data available from 1 to 20.

As shown in Figure 7, for the *forked server*, the linear regression model predicts a slight increase in performance with increasing concurrency, though the growth rate is slow. This matches the observed data, where the forked server's performance begins to level off after a concurrency level of 5. For the *threaded server*, the linear regression model shows a much steeper increase in predicted performance as concurrency grows. This reflects the more significant gains in performance seen in the threaded server up to a concurrency level of 20, as threads are more efficient in handling concurrent requests.

However, while linear regression provides a general upward trend for both servers, it does not fully capture the plateauing and fluctuations observed in the experimental data for the threaded server at higher concurrency levels. The threaded server experiences performance variability as the concurrency increases beyond 10, which linear regression smooths out, leading to less accurate predictions for higher concurrency levels.

C. Comparison of Models: Linear Regression vs. Actual Data

From the actual experimental data and standard deviation analysis, I can see that while linear regression provides useful insights into the overall trend of the data, it oversimplifies the complex behavior of the *threaded server*. The actual data shows that the threaded server does not maintain a linear increase in performance beyond a certain point and experiences fluctuations due to resource contention and system limitations.

For the *forked server*, linear regression is more aligned with the observed performance as its growth is steady and less variable.

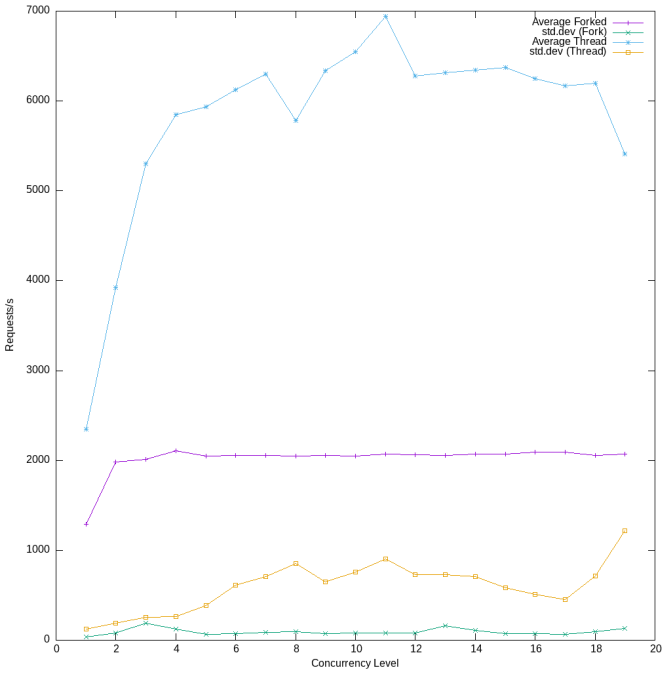

Fig. 6: Original Experimental Data for Forked and Threaded Servers.

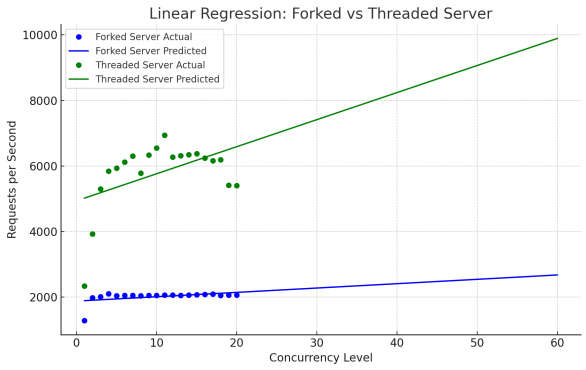

Fig. 7: Linear Regression Predictions for Forked and Threaded Servers.

D. Summary of Observations

From the results and analysis, a few key points are clear:

- The *threaded server* performs better than the *forked server* as concurrency increases, especially after concurrency level 5. This is because threads are more efficient in handling multiple requests compared to processes in the forked server.
- *Linear regression* gave useful predictions for both servers, but it did not fully show the ups and downs in the performance of the *threaded server* at higher concurrency levels.
- The *forked server* shows steady but slower performance, leveling off early, while the *threaded server* has more variations in performance as concurrency increases, likely due to resource competition.

These observations will help in deciding which server type is better for different workloads, which will be discussed in the conclusion.

V. Conclusion

The experiments conducted on both the forked and threaded servers have provided valuable insights into their performance under different concurrency levels. The results showed that the *threaded server* consistently outperforms the *forked server* at higher concurrency levels, particularly beyond concurrency level 5, due to the more efficient handling of threads compared to processes.

The *forked server*, while stable, exhibited limited scalability and reached its performance peak early. This makes it suitable for workloads with lower concurrency demands where stability is more important than high throughput. In contrast, the *threaded server* achieved much higher requests per second but showed performance fluctuations as concurrency increased. This variability suggests that while the threaded server is well-suited for handling high concurrency, it may face issues with resource contention at extreme concurrency levels.

The *linear regression analysis* successfully predicted the general trends for both servers but did not fully capture the complexity of the threaded server's performance at higher levels of concurrency. More advanced predictive models may be required for accurately forecasting performance in such scenarios.

In conclusion, the choice between forked and threaded server architectures should be based on the expected concurrency level and workload characteristics. The *forked server* may be preferable for more stable, low-concurrency environments, while the *threaded server* is more efficient for higher concurrency but requires careful consideration of resource management to handle performance variability effectively.