UNIVERSITÀ DI PAVIA

Enterprise Digital Infrastructure Course

Web Services Performance

Andrea Alberti

Department of Computer Engineering - Data Science

University of Pavia, Italy

Contact: andrea.alberti01@universitadipavia.it

GitHub: GitHub: https://github.com/AndreaAlberti07/Web-Services-Performance.git

July 2, 2023

Contents

Abstract

This report presents an analysis of web services, focusing on their implementation aspects and the impact on their performance. The study examines several factors that influence performance, including the number of parallel TCP connections, caching mechanisms, HTTP versions, and warm-up time allocation. The primary metric used to investigate the impact of these factors on the performance is the page load time (PLT). Additionally, the study aims to compare the performance of different websites under various conditions, utilizing metrics such as Requests Per Second (RPS), Time Per Request (TPR) and its Standard Deviation (SD). Several tools were employed for data collection and analysis, including the Apache Benchmark, the Web Development Tools integrated into Mozilla Firefox, and the h2load tool. The experiments were performed multiple times within the same day to ensure accurate and consistent results. The study highlights the optimal number of parallel TCP connections, showing the importance of caching mechanisms and the superiority of newer HTTP versions. Furthermore, the study identifies the potential impact of warm-up time allocation on web service performance. Overall, this research contributes to the

understanding and improvement of web service implementation and performance.

List of Figures

List of Tables

1 Introduction

Internet technologies play a crucial role in modern society, facilitating seamless communication, information exchange, and collaboration over the internet. The continuous advancement of these technologies is shaping the way we live, work, and interact, transforming various industries and providing new opportunities for growth and connectivity. One of the most important branches of Internet Technologies are the Web Services. This report is aimed at the analysis of the performance of these services, focusing on their implementation aspects and the impact on their performance.

2 Experimental Setup

- **Why**: The first goal involves investigating the impact of the number of parallel connections, caching policies and HTTP version on the PLT. The second goal is to compare different websites in different conditions to asses their performance.
- **Which**: The used websites are listed in table 1.
- **What**: The metrics used are the PLT for the first goal, while RPS and TPR are used for the second one.
- **Where**: The experiments are conducted with a Macbook Pro 14, equipped with a Virtualized Ubuntu 22.04 LTS. The browser is *Mozilla Firefox* and the internet connection is a FTTH 200/20 Mbps provided by Vodafone IT.
- **How**: The used tools are the *Apache Benchmark*, the *Web Dev. Tools* built-in in Mozilla Firefox and *h2load tool*.
- **When**: The experiments are conducted several times in the same day to provide reliable results.

Website	URL
Apple	`https://www.apple.com/store`
Harvard	`https://www.harvard.edu/`
MIT	`https://www.mit.edu/`
V.News	`https://www.vallesabbianews.it/`
GZ	`https://www.grafichezorzi.it/`
V.Spluga	`https://vallespluga.it/`
Unitus	`http://www.unitus.it/`
MIT gov	`https://www.mit.gov.it/`
Istat	`https://www.istat.it/`
OffWhite	`https://www.off---white.com/`

Table 1: List of tested websites

3 Web Services

The increasing frequency of user interactions with websites has brought to light the criticality of website performance in delivering a satisfactory user experience. Among the key performance factors, page load time holds significant importance. To optimize this aspect, an option is to focus on managing the number of parallel connections exploitable between client and server, and implementing effective caching policies. The ultimate objective is to minimize page load time (PLT), thus fostering user engagement and ensuring an enhanced user experience.

3.1 Parallel TCP

The number of parallel connections refers to a browser's ability to establish simultaneous connections to retrieve multiple resources concurrently. Increasing parallel connections can potentially expedite page load times, but it may also strain server resources and lead to network congestion. Specifically:

- **Fewer Parallel Connections**: If the number of parallel connections is set too low, it can result in slower page load times. With fewer connections, the browser fetches resources sequentially, causing delays as it waits for each request to complete before initiating the next one.
- **Default Number of Parallel Connections**: Most modern browsers allow around six parallel connections per host by default. This number hit a balance between resource fetching and the server's load. With this setting, the browser can fetch multiple resources simultaneously, improving the overall page load performance.
- **Higher Number of Parallel Connections**: Increasing the number of parallel connections can potentially speed up page load times. By allowing more simultaneous connections, the browser can fetch a greater number of resources in parallel, reducing the overall loading time. However, an excessively high value may overwhelm the server and cause network congestion.

This section presents a practical investigation into the influence of the number of parallel connections on the Page Load Time (PLT). Various websites (as listed in Table 2) were tested using different numbers of parallel connections to determine the configurations that benefit the most from parallel TCP connections.

Website	Number of Objects	Size (MB)
Vallesabbia	118	5.01
Grafiche	65	5.31
Vallespluga	151	20.03
Unitus	72	1.74
Gov	218	9.98
Istat	134	11.33

Table 2: Websites details

Before each measurement, the browser cache was cleared and the "disable cache" option in Mozilla Web Developer Tools was enabled. The selection of tested sites was conducted carefully to utilize the HTTP/1.1 protocol, which allows for the utilization of multiple parallel TCP connections. This choice aimed to highlight the potential benefits that can be derived from employing such connections. Each website was tested using different numbers of TCP connections, specifically 1, 5, 10, and 15. To ensure accurate measurements, the reported Page Load Time (PLT) represents the average value obtained from five data points collected under the same experimental conditions. It is important to

note that the actual number of TCP connections established depends on the server configuration, as the client merely expresses its preference. To ensure proper interpretation of the results, the number of opened TCP connections was verified using Wireshark, and these details are provided in Table 3.

Website	N. TCPs Ask	N. TCPs Estab.
Vallesabbia	15	15
Grafiche	15	13
Vallespluga	15	15
Unitus	15	15
Gov	15	15
Istat	15	15

Table 3: Number of TCP connections asked and established

Analysis of Figure 1 reveals a decrease in PLT as the number of connections increases. However, this trend only persists until a certain threshold (in this case, 5 connections), beyond which the PLT remains relatively constant. The specific reasons for this behavior are not apparent from the conducted experiments, as the servers configuration remains unknown, resembling a black box. Notably, the website that derives the most benefit from concurrent TCP connections is *Istat*. This advantage is not solely attributable to the size of the website or the number of objects it contains, as other websites with larger sizes exhibit a different behavior. Therefore, further investigations are necessary to identify the underlying factors contributing to this phenomenon. Overall, it can be concluded that the number of parallel TCP connections significantly impacts the PLT. However, the benefits tend to plateau as the number of connections increases excessively. The default choice of six concurrent TCP connections made by the browser appears to be a reasonable selection based on the findings of this study.

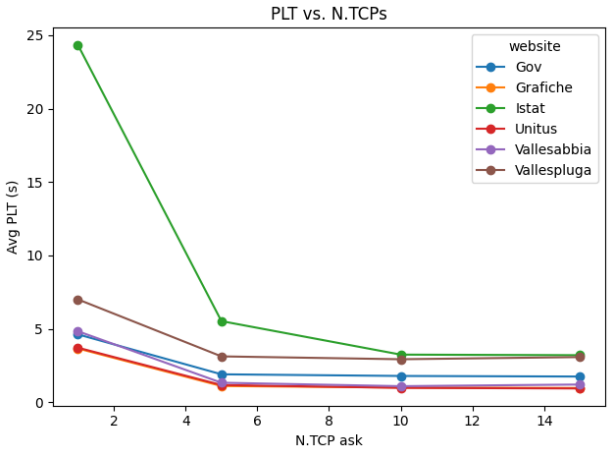

Figure 1: Page Load Time (PLT) vs. N.TCP

3.2 Caching Policy

Caching policies, controlled through HTTP headers, significantly impact page load times by reducing the need for repeated resource requests. Effective caching policies leverage client-side (browser caching) and intermediate server storage (proxy caching).

Proxy caching involves storing copies of web resources at intermediate proxy servers, which can then serve subsequent requests for the same resources without contacting the origin server. This reduces network latency and bandwidth usage.

Browser caching involves storing web resources locally on the client's device, allowing subsequent page visits to be loaded faster without the need to re-download the same resources.

Implementation: the implementation of caching policies is facilitated through the use of the 'Cache-Control' header in HTTP responses. This header allows servers to communicate specific caching directives to clients and intermediaries. It provides instructions such as the maximum time a resource should be considered fresh ('max-age') and whether the resource can be cached by intermediate proxies ('public') or the browser ('private'). To ensure the validity of cached resources, several headers are utilized in caching strategies. The 'Expires' header specifies the date and time after which the resource should be considered stale. The 'Last-Modified' header indicates the date and time of the resource's last modification. Additionally, the 'Etag' header provides a unique identifier for the resource. These headers enable three main non-mutually exclusive strategies for validating cached resources: 'Expiration', 'Validation', and 'Heuristic'. The **Expiration** and **Heuristic** strategies generally result in faster response times, as they rely on pre-determined expiration dates or heuristics to determine the freshness of the cached resources. On the other hand, the **Validation** strategy requires a round-trip to the server to confirm the validity of the cached resources, resulting in slower response times.

3.2.1 Caching mechanisms

To evaluate the impact of caching policies on page load times, various websites with different caching strategies leveraging the browser cache were tested. The results, as depicted in Figure 2, clearly demonstrate the benefits of caching. Enabling caching led to a significant reduction in page load times across all tested websites.

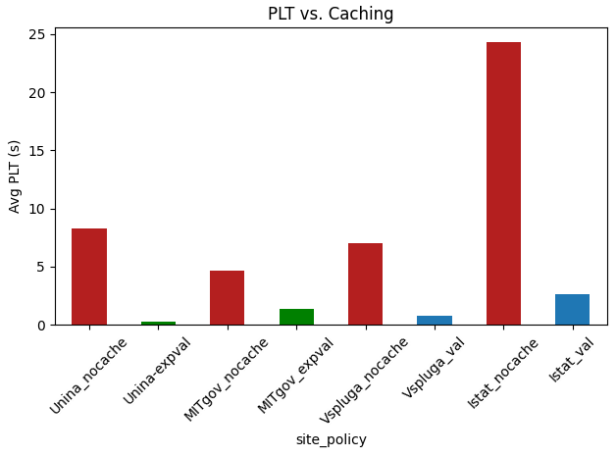

Figure 2: Caching policies of the tested websites

Interestingly, when comparing the specific methods used to validate cached resources (validation and exp+validation), no noticeable differences were observed (Figure 3). This observation can be attributed to two factors. Firstly, websites can implement different caching policies for various resources, making it challenging to discern distinct impacts on page load times. Secondly, the performance gap between the 'Expiration' and 'Validation' strategies primarily arises when the resources are cached and not yet expired, allowing the 'Expiration' strategy to be leveraged. However, verifying this fact requires a thorough examination of individual resources, which is not easily feasible.

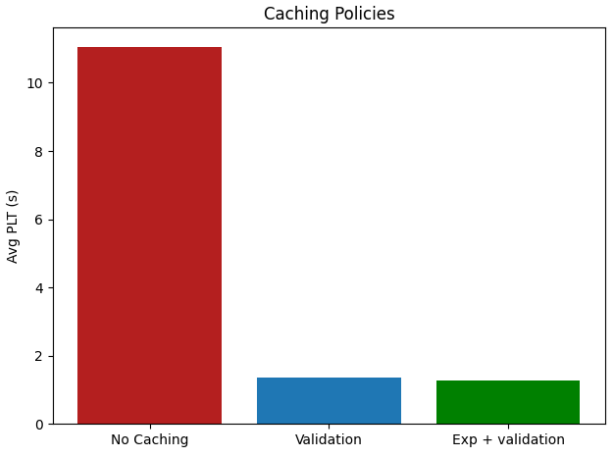

Figure 3: Page Load Time (PLT) vs. Caching Policy

In conclusion, even though the 'Expiration', 'Validation', and 'Heuristic' strategies have their advantages and trade-offs, the overall performance benefits of caching are evident.

3.2.2 Caching and HTTP version

Over the years, multiple versions of the HTTP protocol have been released, each introducing new features and improvements. HTTP/1.1 brought advancements such as persistent connections, pipelining, and caching. These enhancements aimed to enhance the efficiency of resource delivery. Building on this foundation, HTTP/2 introduced significant improvements, including binary framing, multiplexing, server push, and header compression. These features further optimized the protocol's performance. HTTP/3, the latest version, introduced a new transport protocol called QUIC, which is based on UDP (User Datagram Protocol), as opposed to the previous versions that relied on TCP (Transmission Control Protocol). This change in the underlying transport protocol offers potential benefits such as improved latency, better congestion control, and enhanced security. Despite these protocol advancements, the fundamental caching system remains unchanged and relies on headers. To practically assess the performance impact of different HTTP versions and the influence of caching, a website (Off-White) was subjected to testing. The website was tested using different versions of the HTTP protocol, with caching enabled and disabled. The experimental results are depicted in Figure 4.

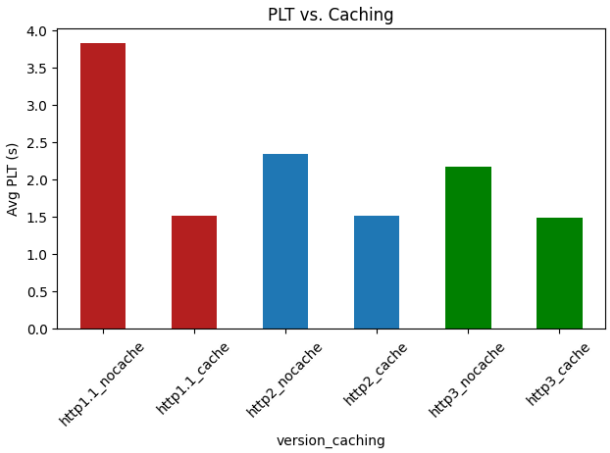

Figure 4: Page Load Time (PLT) vs. HTTP version and caching

As anticipated, enabling caching resulted in a substantial decrease in page load times for all tested versions of the HTTP protocol. The uniformity of loading times with caching enabled suggests that the caching mechanism played a more prominent role in influencing performance than the specific protocol version being used. This underscores the significance of efficient caching strategies in optimizing overall page load times.
Another interesting observation is depicted in Figure 5. As expected, each new protocol version outperforms the previous one. This performance improvement is not dependent on caching mechanism and it is attributable to the above discussed enhancements brought by each new version.

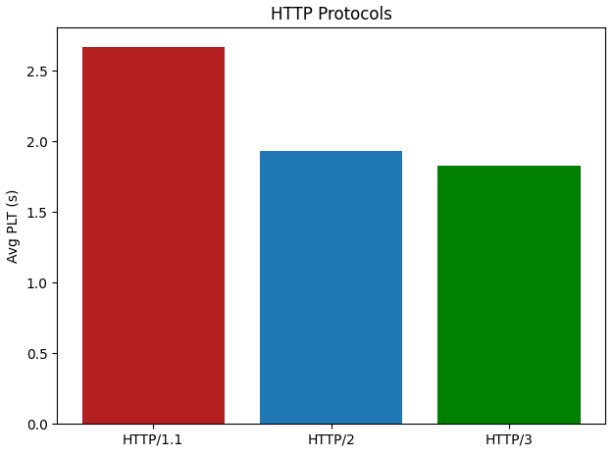

Figure 5: Page Load Time (PLT) vs. HTTP version

3.3 Performance

This section presents a comparative analysis of the performance of three distinct websites. To gain insights into their performance under different conditions, two key parameters (Figure 6) were varied throughout the experiments: the number of requests sent to each website and the concurrency level. It is important to note that these parameters' values were chosen conscientiously, taking ethical considerations into account to prevent overloading the websites. All experiments were conducted using identical conditions, as described in Section 2. The used scripts can be consulted

in Appendix A.

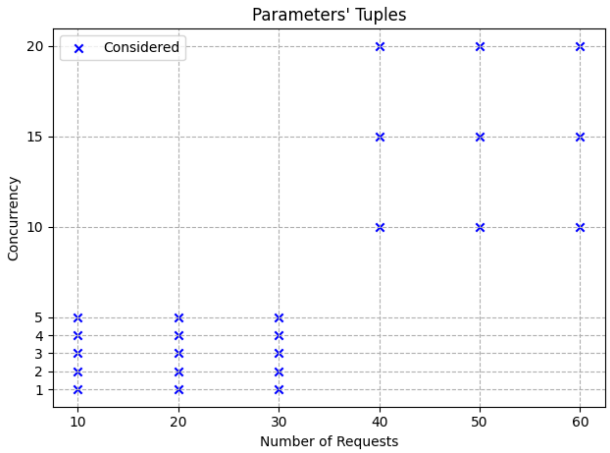

Figure 6: Concurrency and RPS combinations

It is worth mentioning that the client-side cache was intentionally disabled to ensure that the experiments accurately captured the website's performance without any caching effect. All the results were gathered using the *Apache Benchmark* tool, and the metric used to compare the websites is the RPS (Requests Per Second). Three analysis are presented: the first one consider the impact on the RPS of the concurrency level, the second one the impact of the number of requests, and the last the overall RPS achieved by each website.

3.3.1 Concurrency VS RPS

The concurrency is intended as the number of concurrent clients hitting the website at the same time. As the concurrency level increases, the website's server is required to process multiple requests concurrently. This can result in a higher RPS, as more requests are being handled simultaneously. However, there may be limits to the server's capacity to handle concurrent requests efficiently. If the server is properly configured and equipped to handle the increased concurrency, it may exhibit a linear or near-linear increase in RPS as the concurrency level rises. However, there may be a point of diminishing returns where further increasing the concurrency level does not lead to a significant increase in RPS or may even negatively impact the server's performance. This can occur if the server becomes overwhelmed with the high concurrency, leading to delays and increased response times. The results in Figure 7 depict the RPS (averaged across number of requests) for each tuple (website, concurrency level). The Requests Per Second increase with the concurrency in a less than linear way until a threshold (10 in this case). After that point, 'MIT' and 'Apple' showed some performance degradation, suggesting they might start struggling to handle the increased concurrency. 'Harvard' instead, showed a constant increasing trend. It is worth to mention that the RPS values are influenced by many other factors, such as the server's load at the time of the experiment, that may have affected the results.

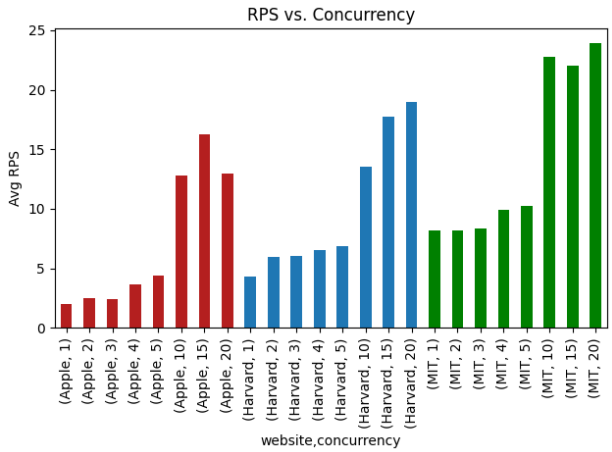

Figure 7: Concurrency VS RPS

3.3.2 Number of requests VS RPS

The number of requests sent to a website in a given time interval is another important factor that can affect the website's performance. As the number of requests increases, the server is forced to handle an higher volume of traffic and the considerations about the RPS behavior may not change from the ones discussed above. The results in Figure 8 show the RPS (averaged across concurrency level) for each tuple (website, number of requests). As expected there is a positive correlation between the number of requests and the RPS, with the websites 'Apple' and 'MIT' showing a trend quite similar to the one observed in the concurrency analysis. 'Harvard' instead, shows an odd behavior with 50 requests. According to Figure 13, the justification is a dip in the RPS value for 50 requests and 10 concurrency level. The reasons for this behavior are not clear, therefore a further investigation is necessary.

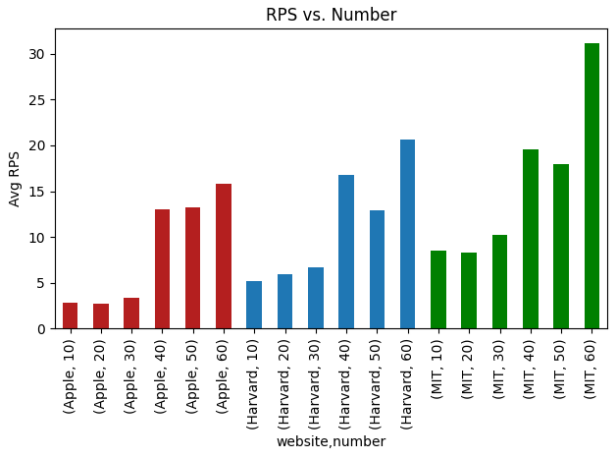

Figure 8: Number of requests VS RPS

3.3.3 Overall RPS

In conclusion to asses a overall performance of the websites, the RPS values for each website were averaged across all the concurrency levels and number of requests. The results are shown in Figure 9. The website achieving the highest average RPS is 'MIT', followed by 'Harvard' and 'Apple'.

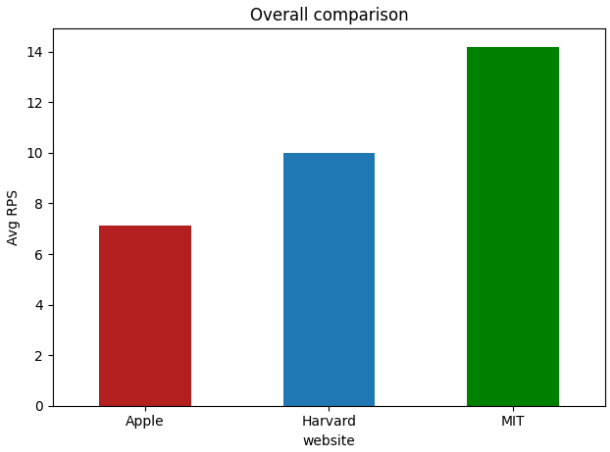

Figure 9: Overall RPS

3.4 Warm-up time impact

A server hosting a website does not need to operate at its maximum capacity continuously. With proper design and optimization, a server can determine when it is necessary to operate at full power and when it can operate in a more energy-efficient manner. This approach helps to save energy and reduce operational costs. However, when transitioning from a lower-power state to full capacity, the server requires a warm-up time to allocate resources and reach its optimal performance level. During this warm-up period, the server undergoes initialization processes, to prepare itself for handling incoming requests effectively. The warm-up time might impact in a non-negligible way the performance of the server.

High Warm-up Time A longer warm-up time allows the server to effectively initialize and allocate its resources before the measurement begins. As a result, when the measurement period starts, the server is already operating at a high capacity level, enabling it to deliver optimal performance.

Low Warm-up Time Conversely to previous situation, a shorter warm-up time can lead the measurement to start before the server reaches its full potential resulting in lower performance.

It is important to note that the impact of warm-up time on server performance can vary due to various factors, such as the specific server workload, network conditions, and other environmental variables. As a result, the previously mentioned scenarios regarding the relationship between warm-up time and server performance should be interpreted with caution. In this section, we test the warm-up time impact on different websites performance. For each website different warm-up times were tested with different concurrency levels (Table 4).

Parameter	Values
Concurrency	1, 2, 4, 6, 8, 10
Warm-up (s)	2, 4, 6, 8, 10

Table 4: Parameters warm-up test

Each experiment was carried out for a duration of 20 seconds, which includes both the actual measurement period and the warm-up time. The used scripts can be consulted in the Appendix A. During these experiments, several metrics were recorded to evaluate the performance of the server. The metrics include Requests Per Second (RPS), which measures the number of requests processed by the server within a second, Time Per Request (TPR), which calculates the average time taken by the server to respond to a single request, and TPR Standard Deviation (SD), which provides an indication of the variability or dispersion of the measured values. Due to logistic reasons, in this case a 100Mb/s 4G connection from Brescia (BS) was utilized throughout the experiments.
In the following sections the results of the experiments are presented and discussed with the help of summarizing charts. To provide an enhanced perspective of the findings, detailed charts are inserted in the Appendix A.

3.4.1 Warm-up time and RPS

In Figure 10, a consistent pattern is observed across all the websites examined. The Requests Per Second (RPS) metric initially increases as the warm-up time is extended, reaching a peak at approximately 6 seconds. However, the delta between the peak and the neighbors is negligible. Beyond the peak, a slight decrease in RPS can be observed. The observed behavior can be attributed to several factors. As public websites, the load on these platforms experiences significant variations, resulting in fluctuations in the measured results. The complex interaction between user traffic, server capacity, and network conditions can contribute to the observed patterns in RPS. However, the precise reasons for the slight decrease in RPS after the peak are not definitively established and require further investigation. In conclusion, in the analyzed websites, the RPS metric is not significantly impacted by the warm-up time.

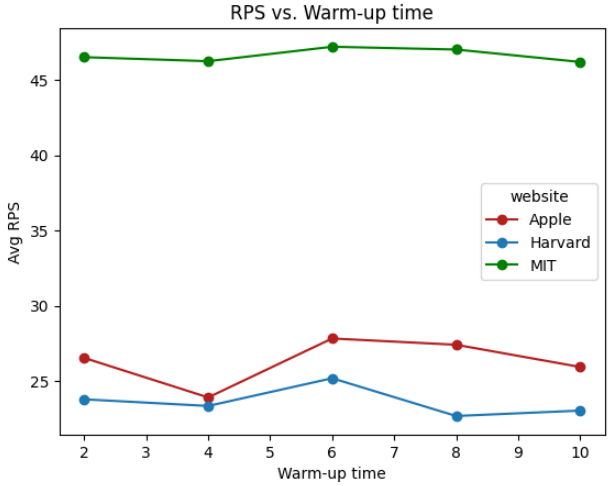

Figure 10: RPS for different warm-up times

3.4.2 Warm-up time, TPR and SD

In Figure 11, the Time Per Request (TPR) metric is shown for different warm-up times. In this case no evident patterns are observed. It is possible to conclude that in the exper-

iments no correlation appears between the warm-up time and the TPR.

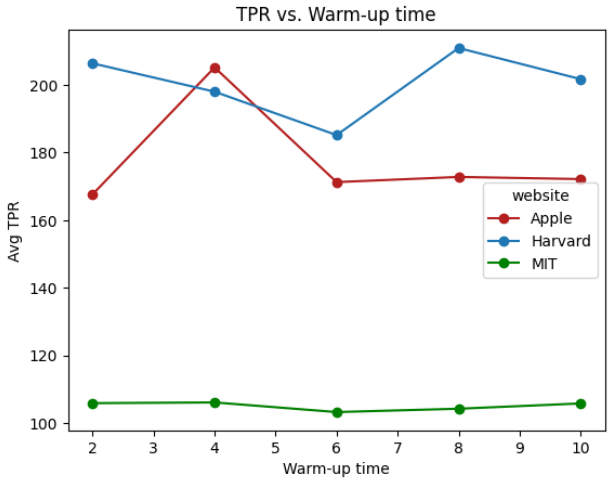

Figure 11: TPR for different warm-up times

Figure 12 presents the TPR Standard Deviation (SD) metric for different warm-up times. Although the pattern may not be as pronounced, there appears to be a discernible trend. By dividing the warm-up times into two groups, namely 2 to 5 seconds and 6 to 10 seconds, it becomes apparent that the SD is higher in the first group. The results indicate that a longer warm-up time has a positive impact on the server's performance stability. Indeed, by conceding to the server enough time to fully initialize and adapt to the workload, a longer warm-up time facilitates a more stable and predictable performance.

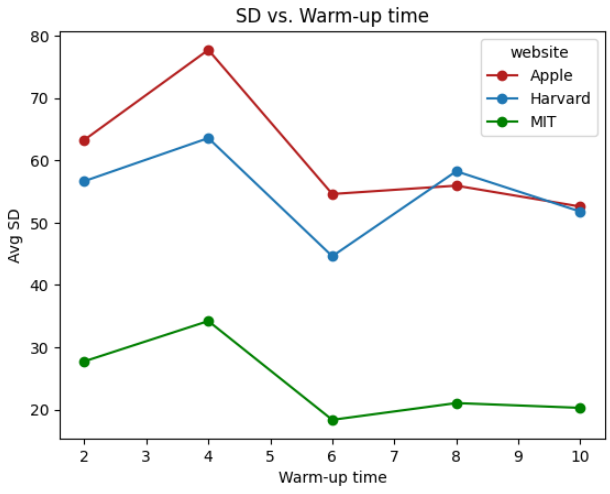

Figure 12: TPR SD for different warm-up times

In conclusion, while the direct impact of warm-up time on the analyzed websites was not significant within the limitations of the available data, some noteworthy insights into its influence on the RPS and SD metrics were observed. Specifically, a warm-up time of 6 seconds, which accounted for approximately 30% of the total measurement time, appeared to be associated with a peak in RPS and a decrease in the TPR SD. These findings suggest that allocating a sufficient warm-up period for servers can have positive implications for their performance.

3.5 Conclusions

The findings of this study demonstrate that the number of parallel TCP connections has a significant impact on page load time, although there is a diminishing return as the number of connections increases excessively. Based on the results, the default choice of six concurrent TCP connections made by browsers appears to be a reasonable selection.

Additionally, caching mechanisms, including the 'Expiration,' 'Validation,' and 'Heuristic' strategies, consistently contribute to improved performance. The results clearly indicate the overall benefits of caching, highlighting its importance in reducing page load time and enhancing the user experience.

The analysis of different HTTP versions revealed that each new version outperforms its predecessor, regardless of the caching mechanism employed. These performance improvements can be attributed to the enhancements introduced in each new version, emphasizing the significance of utilizing the latest HTTP protocols.

Regarding the performance analysis of the analyzed websites, it can be concluded that the fastest website is 'MIT,' followed by 'Harvard' and 'Apple.'

Although the direct impact of warm-up time on the analyzed websites was found to be not highly relevant, primarily due to limited available data, notable observations were made regarding its influence on the RPS (Requests Per Second) and TPR SD (Standard Deviation) metrics. Specifically, it was observed that a warm-up time of 6 seconds, comprising approximately 30% of the total measurement time, resulted in a peak in RPS and a decrease in TPR SD. These findings suggest that allocating an adequate warm-up period for servers can yield positive implications for their overall performance.

In conclusion, this study provides valuable insights into various aspects of web technologies. The optimal number of parallel TCP connections, the importance of caching mechanisms, the superiority of newer HTTP versions, and the potential impact of warm-up time allocation for the websites performance assessment.

A Appendix

Appendix A: Detailed Charts

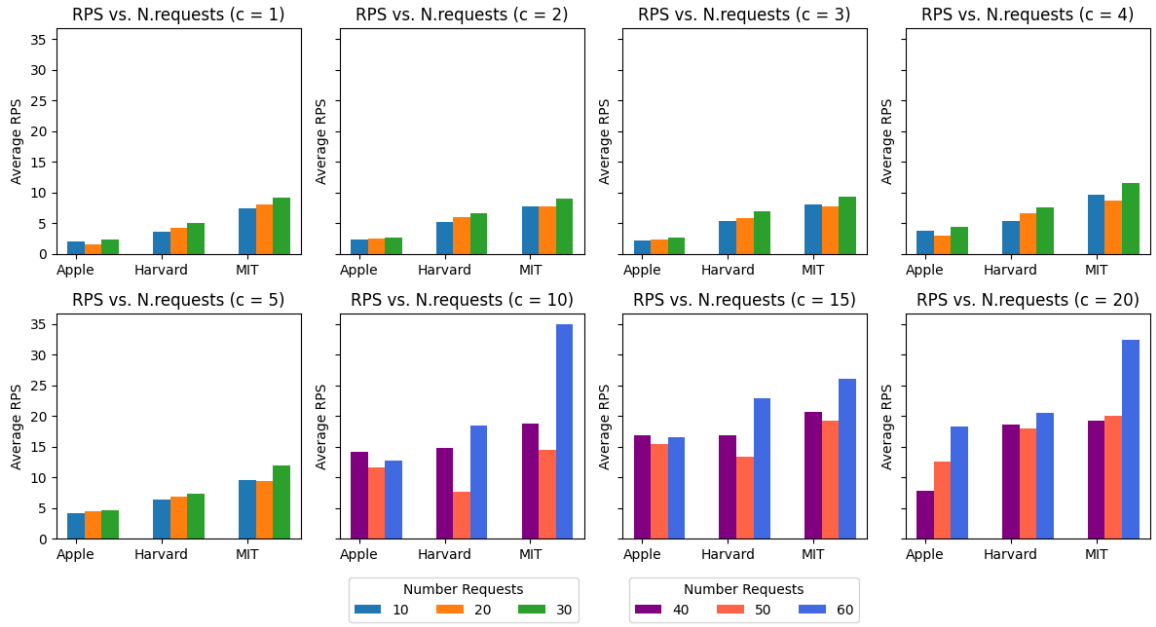

Figure 13: Detailed analysis RPS vs Number of requests and Concurrency level

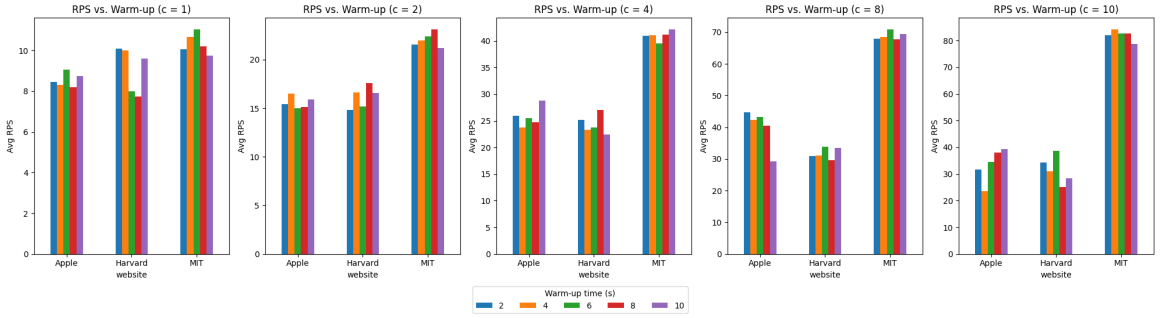

Figure 14: RPS and Warm-up time for different concurrency levels

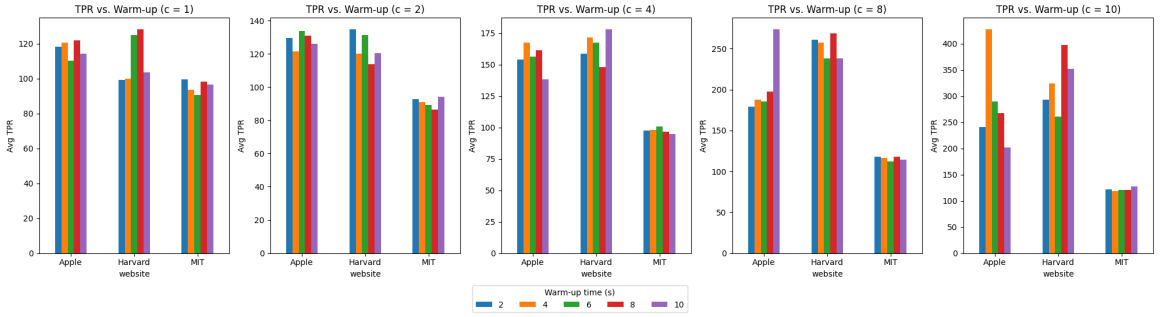

Figure 15: TPR and Warm-up time for different concurrency levels

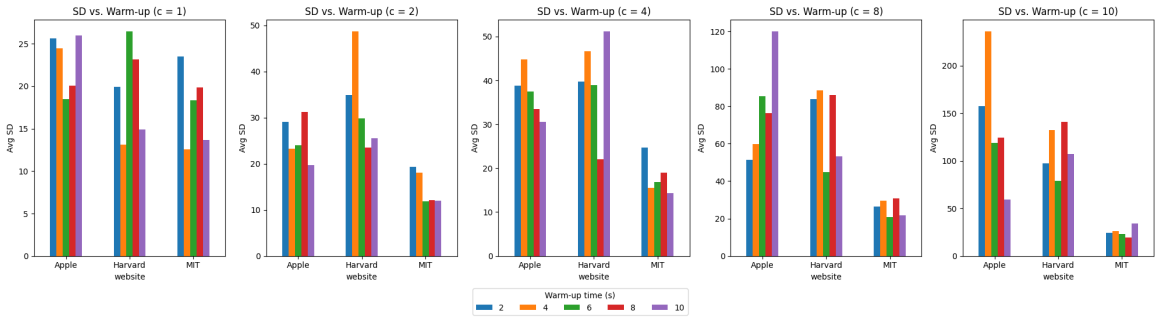

Figure 16: SD and Warm-up time for different concurrency levels

Appendix B: Scripts

```bash
output_file="ab_perf_results.txt"

for website in "${websites[@]}"; do
    for concurrency in "${concurrency_levels[@]}"; do
        for req in "${requests[@]}"; do
            ab -n "$req" -c "$concurrency" "$website" >> "$output_file"
            sleep 5
        done
    done
done
```

Figure 17: Bash script for ab experiments

```bash
output_file="5_h2load_results.txt"

for website in "${websites[@]}"; do
    for concurrency in "${concurrency_levels[@]}"; do
        for warm_up_time in "${warm_up_times[@]}"; do
            echo "Testing: Website: $website, Concurrency: $concurrency, Warm-up: $warm_up_time" >> "$output_file"
            h2load --warm-up-time="$warm_up_time" -c "$concurrency" -D "$duration" "$website" >> "$output_file"
            echo >> "$output_file"
            echo >> "$output_file"
            sleep 5
        done
    done
done
```

Figure 18: Bash script for h2load experiments