EDI: Third Lab Report

Aiman Al Masoud - 502044

June 9, 2022

Abstract

1 Parallel Connections

Page Load Time (PLT) is one of the most critical performance evaluation metrics for a website, as it correlates directly with user experience, taking into account both the time to download the html of a page, as well as the time to download the whole set of its embedded objects, as a browser would typically do when directed to navigate to a page.

A good Page Load Time in 2022 is between 1 and 2 seconds, but it depends, and having an even lower Page Load Time is desirable; users obviously tend to abandon websites that take more time to load than their competitors. [1]

1.1 HTTP/1 vs HTTP/2

The first experiment performed here pertains to the effect of increased parallel connections on Page Load Time. As power users will know, modern web browsers allow their users to peek into various configuration settings and tweak them, this is especially true of Mozilla Firefox, where the user can access various technical settings by typing in 'about:config' into the navigation bar and hitting enter. More specifically, one of these configuration parameters is called 'network.http.max-persistent-connections-per-server', and as the name suggests, it governs the maximum amount of parallel persistent TCP connections that the browser is allowed to open for any particular web server on the internet. [2]

By default, the value for this option is set to '6', but can increasing it arbitrarily lead to an overall improvement in the performance of the web browser when loading pages?

Here comes to play a very important distinction between websites, and that is difference between HTTP/1 and HTTP/2: the first two major versions of the ubiquitous Hypertext Transfer Protocol. There are various ways in which HTTP/2 tried to improve on HTTP/1, but what is of specific interest to us in this particular inquiry is that: websites running on HTTP/1 tend to favor connections to multiple separate servers, to dowload the objects of a page in parallel, and overcome the limitation imposed by browsers on the number of parallel connections to a single server, a technique known as Domain Sharding [3].

HTTP/2 helped overcome this very necessity, by introducing Streams that could convey multiple concurrent (and bi-directional) HTTP requests within a single persistent TCP connection; thus eliminating the need to connect to separate resource servers, and even eliminating the overhead associated to having to open multiple TCP connections with them. This is something that, at least in the long run, will render the technique of Domain Sharding obsolete.

So, to go back to the original question: resource intensive HTTP/1 websites may benefit from an increased number of parallel TCP connections, as they only support a single HTTP request per TCP connection. On the other hand, HTTP/2 websites support multiple concurrent HTTP requests over the same TCP connection, hence, from a theoritical viewpoint, it shouldn't make much of a difference to them if the client supports a greater number of TCP connections.

1.2 Methodology and experimental setup

The experiment was performed in practice, by measuring the PLTs of 6 different websites (3 HTTP/1 and 3 HTTP/2 websites) under two different conditions: 1 and 6 maximum connections per server.

The browser in question was Firefox, and the number of maximum connections was set tweaking the aforementioned: "network.http.max-persistent-connections-per-server" parameter.

Although the results aren't exhaustive, they clearly show that HTTP/1 websites fare much better when granted a higher number of connections per sever.

		Average PLT (seconds)
HTTP Version	**Connections**	
1	**1**	3.017778
	6	2.212222
2	**1**	3.178889
	6	3.486667

Table 1: ...

2 Question 2

2.1 Methodology and experimental setup

3 Apache Benchmark

There are several automated CLI benchmarking tools to measure the performance of webservers, and Apache Benchmark (AB) is one of them. Unfortunately, AB doesn't support the second version of the HTTP protocol, but it provides several options to test the performance of HTTP/1 servers, and especially Apache webservers.

3.1 Methodology and experimental setup

The idea behind this experiment was to test the effects of the concurrency (-c) and keep-alive (-k) options of Apache Benchmark, on the performance of 3 websites, when issuing a fixed amount of requests (20).

The concurrency (-c) option regulates the number of multiple requests to perform at a time; since the servers in questions are HTTP/1 only servers, concurrent requests will have to be performed over multiple TCP connections.

The keep-alive option (-k) specifies whether to request the usage of the HTTP KeepAlive feature (aka: persistent connections), which means that a single TCP connection will be re-used for multiple sequential HTTP requests, thus reducing the overhead of TCP connection creation [4]. Obviously, since this is HTTP/1, concurrent requests cannot share the same TCP connection, as already stated.

Each website was tested with the following 4 commands:

1. ab -c 1 -n 20 http://example.com/

   20 requests, one at a time, without reusing the TCP connection.

2. ab -c 10 -n 20 http://example.com/

   20 requests, 10 at a time, without reusing TCP connections.

3. ab -k -n 20 http://example.com/

   20 requests, one at a time, reusing the TCP connection.

4. ab -c 10 -k -n 20 http://example.com/

   20 requests, 10 at a time, reusing TCP connections.

The first command (-c 1 and no -k) is expected to be the slowest, as the requests are all performed sequentially, and, for each request, a new TCP connection must be opened, with all of the overhead associated to that costly operation.

The last command instead (-c 10 and k), is expected to be the fastest, as it combines the benefits from issuing multiple parallel requests, with the benefits of reusing already opened TCP connections.

Sure enough, these are the average times that it took to run these tests:

Options	Average Time (seconds)
-c1	5.191
-c1 && -k	3.194
-c10	1.036
-c10 && -k	0.939

Table 2: ...

4 Question 4

4.1 Methodology and experimental setup

5 References

1. https://www.hobo-web.co.uk/your-website-design-should-load-in-4-seconds/
2. https://www.computerworld.com/article/2541429/hacking-firefox--the-secrets-of-about
   html?page=5
3. https://blog.stackpath.com/glossary-domain-sharding/
4. https://httpd.apache.org/docs/2.4/programs/ab.html#synopsis