Performance Analysis: A Comparison between Apache and Your Typical Terrible C Server

Connor Smith

Computer Science Dept., Brigham Young University, Provo UT 84602

Abstract. Web servers are many in number and diverse in purpose, performance, and ease-of-use. We discuss the performance of a typical "hack-it-until-it-works" implementation of an HTTP server and compare how it performs to the Apache HTTP Server, the world's most widely used web server software[1].

1 Introduction

With the advent of the Internet came the need for users to easily exchange information. The creation of the web browser and web server together enabled the average person to "surf" for the first time. In 1989 Tim Berners-Lee proposed a new project that resulted in the first of both of these applications: the first browser, the WorldWideWeb; and the first web server: CERN httpd[2]. 25 years later, more professionally-developed web servers exist than anyone can name, and countless students have written their own simple implementation in a weekend. This paper aims to compare the performance between two extremes of the spectrum and quantify differences between the two. The Apache HTTP server will be used for reference and compared to Your Typical Terrible C Server™, or YTTCS (pronounced "yit-kss").

2 Related Work

Midgley's benchmarking guide[3] provides useful insight into the principles behind testing performance, common pitfalls, and methods of actually testing a web server. The guide starts by identifying key principles of HTTP benchmarking. The goal behind benchmarking is to obtain reliable data illustrating the quantitative behavior of the server while meeting certain criteria. The most obvious of these constraints is latency. The results from a study done by the Missouri University of Science and Technology[4] concluded that the average user will notice a delay of longer than 100ms, and after 1s, the user will begin to interrupt his thinking. Delays longer than 10 result in the user attempting to perform other tasks while waiting, or giving up entirely. Thus it is critical that the chosen server respond promptly to all requests, even during periods of high traffic. At the same time, no valid request should produce an error message. After all, it doesn't matter how quickly the server is responding to client requests if those responses don't contain the requested information. The third constraint listed is that performance needs to be measured over a randomized, diverse set of requests. This will better mimic "real-world" performance and bring the server's caching mechanism into play. Since YTTCS offers no special caching other than the operating system/architecture upon which it is running and in the interest of simplicity, we focus our attention on the first two criteria.

3 Experimental Setup

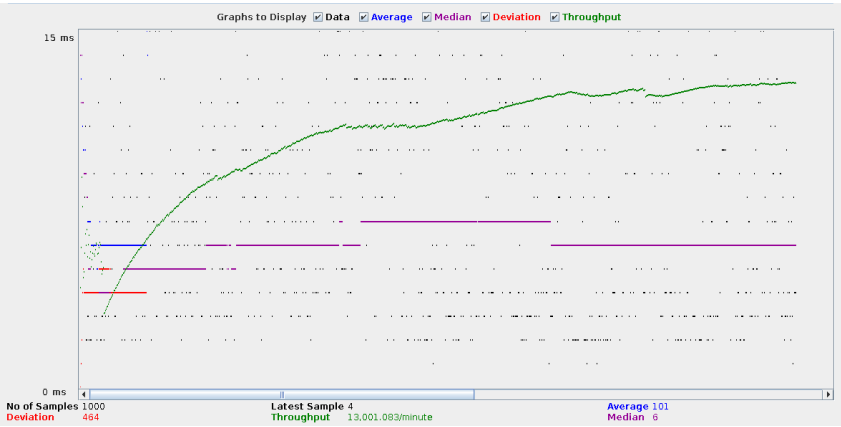
An important point emphasized in Midgley's guide is that the server and the software generating the test load should never be located on the same machine. Doing so would cause a conflict between two for CPU resources and jeopardize the accuracy of any results. In an ideal setting, the server should be running on a machine with no other load.

Respecting this consideration, YTTCS and Apache are located on one of the machines in BYU's CS department, and the load generating software is located on a separate machine in the department. We chose to primarily use JMeter for testing, as the department machines can only be accessed from within the department's network, and JMeter does not require root privileges to run. JMeter also has the added benefit of being a Java application, making it cross-compatible and easy to run from any machine.

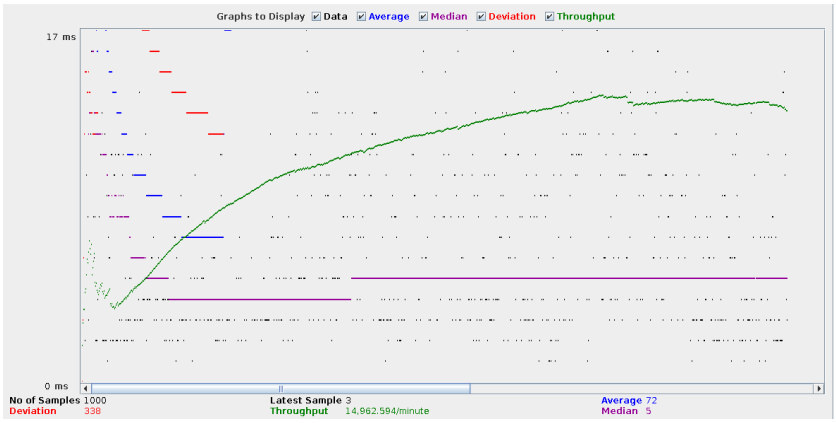
We tested only very simple HTTP requests. JMeter was configured to simulate 50 users making 20 requests, for a total of 1000 requests per test. We chose this configuration after experimenting with different parameters and finding that the preliminary results looked indistinguishable from other configurations. Performing 1000 requests only takes a couple seconds, however, and JMeter appeared to struggle to render more requests without starting to overwrite part of the graph.

4 Results

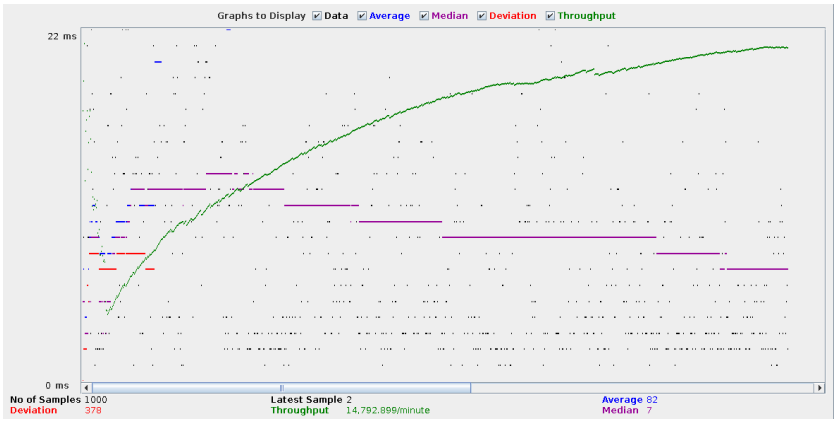
We first tested Apache. We ran each test five times in order to further eliminate random variation. As can be seen, the results are fairly consistent. Apache is able to keep a low latency throughout the tests, with a median of under 10ms each time. The mean is much higher, around 100ms, indicating that while most requests are being served almost immediately, a few are taking a disproportionate amount of time. The throughput of Apache is right around 15,000 requests per minute, or 250 per second, which is quite impressive.



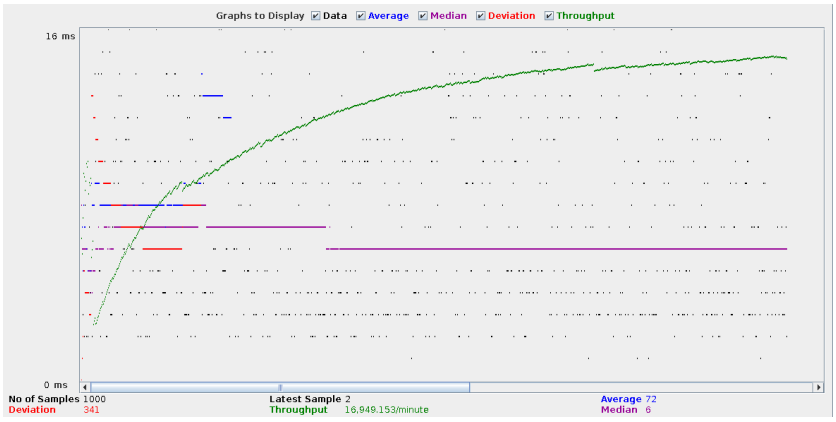
Apache Test 1



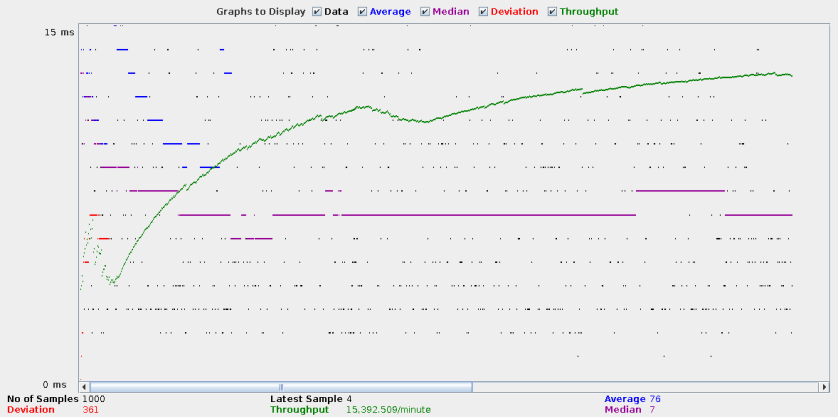
Apache Test 2



Apache Test 3



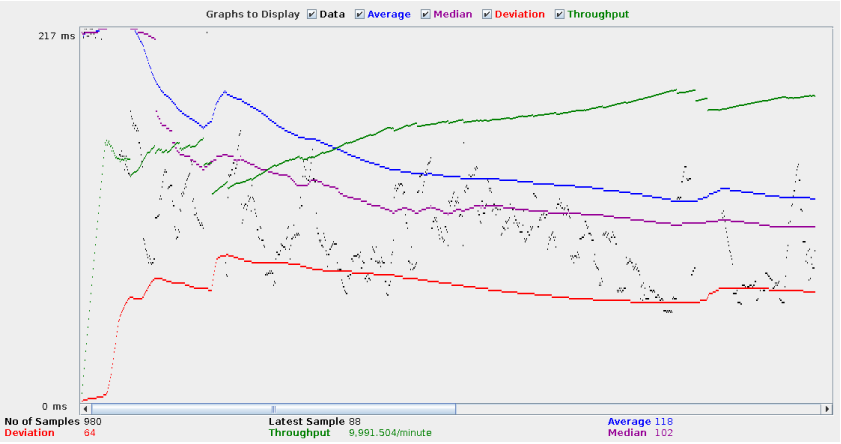
Apache Test 4



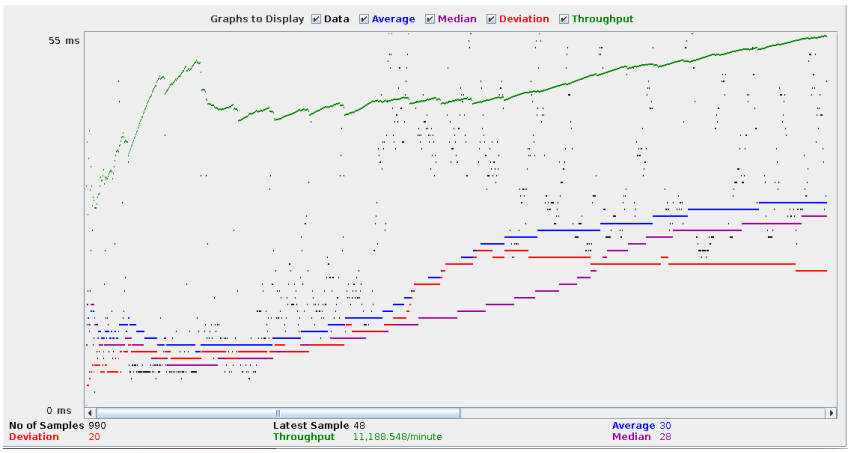
Apache Test 5

Next we ran the same test five times on YTTCS. The results for this server are markedly different than those for Apache. First, there is vastly more variation in response time and throughput, both within each test and among different iterations. The median response time ranged from 5 to 100ms. Likewise, the mean was between 20 and 120ms. Unlike Apache, the mean and median were very close each test, indicating that on the whole, the response time from one request to another was very similar. This can be seen in the much smaller values for deviation, which at its highest point was 160 for YTTCS and 465 for Apache. There is also a visual representation of this in the graphs. Most data points for YTTCS are clustered very closely to its neighbors, while those for Apache are split into uniform bands. We do not have conclusive evidence for why these bands appear, but suspect it may have to do with Apache giving special treatment to certain requests, or somehow load-balancing itself. YTTCS offers no such functionality; each request is simply served as soon as possible.

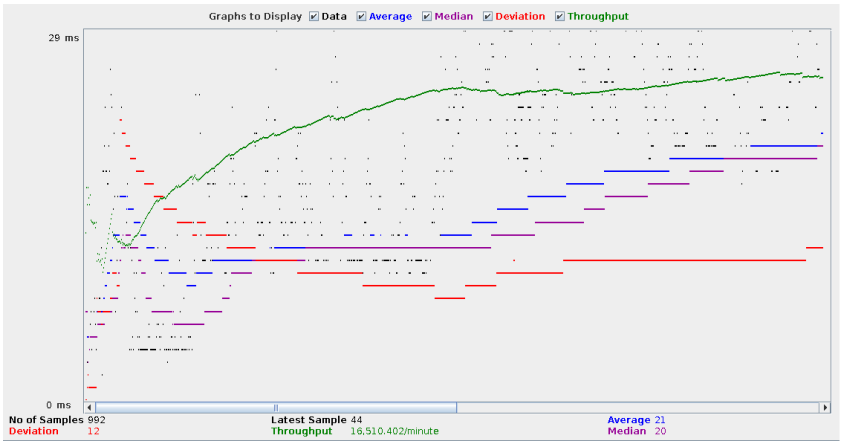
Interestingly, the overall throughput for Apache and YTTCS was very similar, with the former averaging around 13,000 responses per minute, or 215 per second. Given the implementation details of YTTCS, this is surprising and higher than expected.



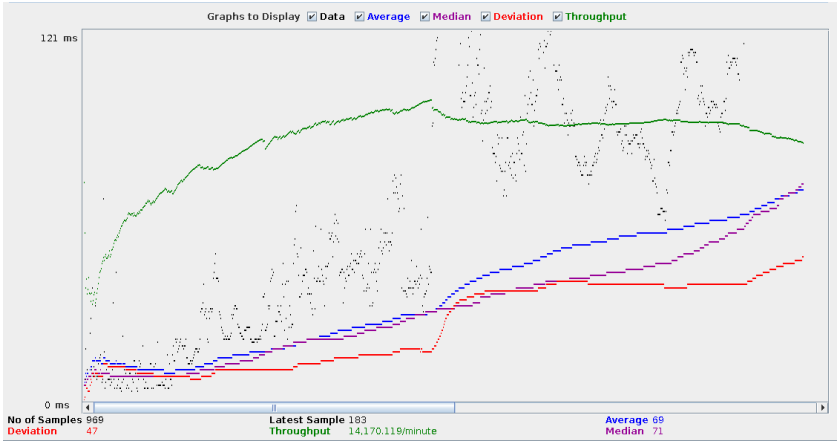
YTTCS Test 1



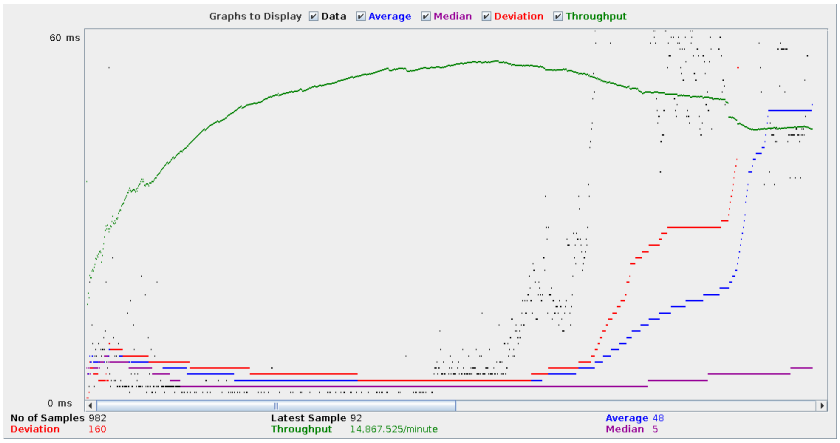
YTTCS Test 2



YTTCS Test 3



YTTCS Test 4



YTTCS Test 5

5 Conclusions

We expected much more of a difference between YTTCS and Apache than was observed. While there were several key differences in latency, performance was very similar between the two. As it stands now, file size appears to make no difference in performance on either server, although we suspect this is not the case for every instance. In addition, if more predictable worst-case response times are needed in a given application, YTTCS may actually prove to be the better choice. However, as a disclaimer we make no recommendation that this specific implementation be used in any application that requires any guarantee of stability whatsoever. In this case, Apache is the “tried-and-true” server that will suit a large variety of applications. In the future, we would like to expose both to more varied testing to discover where different interesting stress points manifest themselves for each.

6 Bibliography

1. .Apache HTTP Server. http://en.wikipedia.org/wiki/Apache_HTTP_Server

2. CERN httpd. http://en.wikipedia.org/wiki/CERN_httpd

3. Midgley, Julian T. J. The Linux HTTP Benchmarking HOWTO. http://www.xenoclast.org/doc/benchmark/HTTP-benchmarking-HOWTO/HTTP-benchmarking-HOWTO.html

4. Nah, Fui Hoon. A Study of Web Users’ Waiting Time.