Performance Testing

by Devashish Shivamraj Bhatoolaul and Elysia Monde Zhi Yong

In this report, our team aims to compare the performance of three different versions of an HTTP server that we implemented. Notably, these implementations include—**a simple HTTP server**, **a persistent HTTP server**, and **a pipelined HTTP server**. Our team also chose to benchmark our results by comparing these three different server versions to the very popular **Apache HTTP server**.

To realize our goals, we first thoroughly detail the differences in each of the four servers, then we provide our performance hypotheses—how we think each server is going to behave. Finally, we present our test cases, why we chose them, the results each server gave us for each test, and if the conclusions we can draw from these results align with our original hypotheses.

1. Differences in our Servers

Simple HTTP Server	As the name suggests, this server is the most basic among the four. It also serves as the base for the **Persistent and Pipelined Server** versions. As no persistence and pipelining is implemented, each time a client requests an object from the server, it must form a new TCP connection—a client can only request one object at a time! The server does however implement concurrent requests from multiple clients—a fork() new process call is made to handle each client in the server respectively.

Persistent HTTP Server	This server extends the **Simple HTTP Server** by adding persistence—a client is able to request (and our server is able to receive) multiple objects from the same TCP connection. This way, the server and client do not lose precious time re-instating a new connection to trade objects. We should note that to efficiently manage memory in our server, we implemented a five-second timeout; if a client has not requested anything in the past five seconds, close their socket! This is a bit cut-throat, but it makes sure that we do not leave too many file descriptors open. We know that we only need one file descriptor for the server's socket, and one process can have at most 1024 file descriptors open, so alternatively, we could have made sure that no more than 1023 connections were active on the server. However, we chose the five-second timeout method for simplicity.
Pipelined HTTP Server	Similarly, this HTTP server extends the **Persistent HTTP Server** by implementing pipelining—multiple HTTP requests can be sent by a client without first getting the response for each individual request. The server then makes sure to send their responses in the order of the client's requests. To implement this, we simply put each client's requests into a buffer, read from it, processed each request, then sent them back in the same order right after each was finished processing. We should also note that we set the buffer for each connected client to hold at most 50K bytes of data, so depending on the HTTP request length, this is the upper limit for the amount of messages the client can request at once.
Apache HTTP Server	You could read more about this popular HTTP server here: https://en.wikipedia.org/wiki/HTTP_pipelining. From what it seems, Apache is an HTTP Server that implements the persistence we described, but no pipelining.

2. Performance Hypotheses

Simple HTTP Server	Since a new TCP connection is needed for each client request, we are hypothesizing that this server is going to obviously perform the worst when the server needs to respond by sending many objects to the client for one request
Persistent HTTP Server and Pipelined HTTP Server	As no new TCP connection is now needed to get multiple objects, we are hypothesizing that when our server needs to send many files to a client, the **Persistent and Pipelined Servers** will perform miles better than the **Simple HTTP Server** (of course, again, only when a client needs more than one object from the server). Also, we think the **Pipelined Server** is slightly going to outperform the **Persistent Server** as the client does not need to wait for a response before sending multiple requests!
Apache HTTP Server	As this is a commercial piece of software, we believe that the Apache server is going to outperform each of our three servers—it is probably much more optimized. However, since this server is essentially performing the exact same tasks as ours, we believe that the performance difference is not going to be too drastic...

3. Test Cases

To test each of our four servers, we are using an automated tool—`httperf`. `httperf` can be used to send batch-requests to a server, much like an automated client.

We should note that we are configuring `httperf` to request a page on our HTTP servers that requires many files to load—`test_websites/www.cs.toronto.edu/~ylzhang/csc258/memes.html`. This test assures that we truly see the performance differences between the **Simple HTTP Server** and the others, since we explained how needing multiple objects is a challenge for the Simple HTTP Server—we need to constantly form new TCP connections, unlike with the other three servers. Additionally, we are creating no more than 100 connections: `--num-conn=100`, with each persistence-enabled server, since we found that this is a safe limit to not experience significant test delays. Finally, for each persistence-enabled server, we have selected three specific tests: one test where we try to make at most 10 requests on each of our 100 connections, another with 20 requests, and another with 30. For the Simple HTTP Server, since we cannot send multiple requests on the same connection, we opt to send the number of requests * number of connections for each tests (i.e. `test-1` for persistence-enabled servers is 100 connections with 10 requests on each connection, so this corresponds to 1000 connections on the Simple HTTP Server). As this number of requests goes up, we should start to notice more performance differences in the servers. In summary, the `httperf` commands or instructions for these three tests are as follows—assuming each HTTP server is running on `localhost` at port `18000`, with root directory as `test_websites` :

test-1:

```
httperf --server localhost --port 18000 --uri /www.cs.toronto.edu/~ylzhang/csc258/memes.html --rate 10 --num-conn 100 --num-call 10 (or --rate 1000 --num-conn 1000 --num-call 1 for Simple HTTP Server) --timeout 5
```

test-2:

```
httperf --server localhost --port 18000 --uri /www.cs.toronto.edu/~ylzhang/csc258/memes.html --rate 10 --num-conn 100 --num-call 20 (or --rate 2000 --num-conn 2000 --num-call 1 for Simple HTTP Server) --timeout 5
```

test-3:

```
httperf --server localhost --port 18000 --uri /www.cs.toronto.edu/~ylzhang/csc258/memes.html --rate 10 --num-conn 100 --num-call 10 (or --rate 3000 --num-conn 3000 --num-call 1 for Simple HTTP Server) --timeout 5
```

Let us now see what results each of these three tests generated for the four servers...

4. Results

test-1

Simple HTTP Server	`Total: connections 1000 requests 1000 replies 1000 test-duration 1.719 s`
	
	`Connection rate: 581.9 conn/s (1.7 ms/conn, <=10 concurrent connections)`
	`Connection time [ms]: min 0.2 avg 4.4 max 1009.9 median 0.5 stddev 63.5`
	`Connection time [ms]: connect 4.0`
	`Connection length [replies/conn]: 1.000`
	
	`Request rate: 581.9 req/s (1.7 ms/req)`
	`Request size [B]: 107.0`
	
	`Reply rate [replies/s]: min 0.0 avg 0.0 max 0.0 stddev 0.0 (0 samples)`
	`Reply time [ms]: response 0.4 transfer 0.0`
	`Reply size [B]: header 90.0 content 4148.0 footer 0.0 (total 4238.0)`
	`Reply status: 1xx=0 2xx=1000 3xx=0 4xx=0 5xx=0`
	
	`CPU time [s]: user 0.71 system 1.00 (user 41.3% system 58.1% total 99.4%)`
	`Net I/O: 2468.9 KB/s (20.2*10^6 bps)`
	
	`Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0`
	`Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0`

Persistent HTTP Server	`Total: connections 100 requests 1000 replies 1000 test-duration 9.902 s`
	
	`Connection rate: 10.1 conn/s (99.0 ms/conn, <=1 concurrent connections)`
	`Connection time [ms]: min 0.9 avg 1.2 max 7.1 median 1.5 stddev 0.7`
	`Connection time [ms]: connect 0.0`
	`Connection length [replies/conn]: 10.000`
	
	`Request rate: 101.0 req/s (9.9 ms/req)`
	`Request size [B]: 107.0`
	
	`Reply rate [replies/s]: min 100.0 avg 100.0 max 100.0 stddev 0.0 (1 samples)`
	`Reply time [ms]: response 0.1 transfer 0.0`
	`Reply size [B]: header 90.0 content 4148.0 footer 0.0 (total 4238.0)`
	`Reply status: 1xx=0 2xx=1000 3xx=0 4xx=0 5xx=0`
	
	`CPU time [s]: user 4.99 system 4.88 (user 50.4% system 49.3% total 99.7%)`
	`Net I/O: 428.5 KB/s (3.5*10^6 bps)`
	
	`Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0`
	`Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0`

Pipelined HTTP Server	Total: connections 100 requests 1000 replies 1000 test-duration 9.902 s Connection rate: 10.1 conn/s (99.0 ms/conn, <=1 concurrent connections) Connection time [ms]: min 0.9 avg 1.2 max 2.2 median 1.5 stddev 0.3 Connection time [ms]: connect 0.0 Connection length [replies/conn]: 10.000 Request rate: 101.0 req/s (9.9 ms/req) Request size [B]: 107.0 Reply rate [replies/s]: min 100.0 avg 100.0 max 100.0 stddev 0.0 (1 samples) Reply time [ms]: response 0.1 transfer 0.0 Reply size [B]: header 90.0 content 4148.0 footer 0.0 (total 4238.0) Reply status: 1xx=0 2xx=1000 3xx=0 4xx=0 5xx=0 CPU time [s]: user 5.02 system 4.85 (user 50.7% system 49.0% total 99.7%) Net I/O: 428.5 KB/s (3.5*10^6 bps) Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0 Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
Apache HTTP Server	Total: connections 100 requests 1000 replies 1000 test-duration 9.902 s Connection rate: 10.1 conn/s (99.0 ms/conn, <=1 concurrent connections) Connection time [ms]: min 0.9 avg 1.3 max 4.3 median 1.5 stddev 0.4 Connection time [ms]: connect 0.0 Connection length [replies/conn]: 10.000 Request rate: 101.0 req/s (9.9 ms/req) Request size [B]: 110.0 Reply rate [replies/s]: min 100.0 avg 100.0 max 100.0 stddev 0.0 (1 samples) Reply time [ms]: response 0.1 transfer 0.0 Reply size [B]: header 254.0 content 4148.0 footer 0.0 (total 4402.0) Reply status: 1xx=0 2xx=1000 3xx=0 4xx=0 5xx=0 CPU time [s]: user 5.32 system 4.57 (user 53.7% system 46.1% total 99.9%) Net I/O: 445.0 KB/s (3.6*10^6 bps) Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0 Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

test-2

Simple HTTP Server	Total: connections 2000 requests 2000 replies 2000 test-duration 1.869 s Connection rate: 1070.2 conn/s (0.9 ms/conn, <=43 concurrent connections) Connection time [ms]: min 0.2 avg 18.6 max 1224.1 median 0.5 stddev 134.4 Connection time [ms]: connect 17.7 Connection length [replies/conn]: 1.000 Request rate: 1070.2 req/s (0.9 ms/req) Request size [B]: 107.0 Reply rate [replies/s]: min 0.0 avg 0.0 max 0.0 stddev 0.0 (0 samples) Reply time [ms]: response 0.9 transfer 0.0 Reply size [B]: header 90.0 content 4148.0 footer 0.0 (total 4238.0) Reply status: 1xx=0 2xx=2000 3xx=0 4xx=0 5xx=0 CPU time [s]: user 0.51 system 1.35 (user 27.1% system 72.1% total 99.2%) Net I/O: 4541.0 KB/s (37.2*10^6 bps) Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0 Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
Persistent HTTP Server	Total: connections 100 requests 2000 replies 2000 test-duration 9.903 s Connection rate: 10.1 conn/s (99.0 ms/conn, <=1 concurrent connections) Connection time [ms]: min 1.5 avg 2.1 max 11.3 median 1.5 stddev 1.2 Connection time [ms]: connect 0.0 Connection length [replies/conn]: 20.000 Request rate: 202.0 req/s (5.0 ms/req) Request size [B]: 107.0 Reply rate [replies/s]: min 200.0 avg 200.0 max 200.0 stddev 0.0 (1 samples) Reply time [ms]: response 0.1 transfer 0.0 Reply size [B]: header 90.0 content 4148.0 footer 0.0 (total 4238.0) Reply status: 1xx=0 2xx=2000 3xx=0 4xx=0 5xx=0 CPU time [s]: user 5.25 system 4.61 (user 53.0% system 46.6% total 99.5%) Net I/O: 857.0 KB/s (7.0*10^6 bps) Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0 Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Pipelined HTTP Server	Total: connections 100 requests 2000 replies 2000 test-duration 9.903 s Connection rate: 10.1 conn/s (99.0 ms/conn, <=1 concurrent connections) Connection time [ms]: min 1.6 avg 2.6 max 10.0 median 2.5 stddev 1.6 Connection time [ms]: connect 0.0 Connection length [replies/conn]: 20.000 Request rate: 202.0 req/s (5.0 ms/req) Request size [B]: 107.0 Reply rate [replies/s]: min 200.0 avg 200.0 max 200.0 stddev 0.0 (1 samples) Reply time [ms]: response 0.1 transfer 0.0 Reply size [B]: header 90.0 content 4148.0 footer 0.0 (total 4238.0) Reply status: 1xx=0 2xx=2000 3xx=0 4xx=0 5xx=0 CPU time [s]: user 5.20 system 4.66 (user 52.5% system 47.1% total 99.6%) Net I/O: 857.0 KB/s (7.0*10^6 bps) Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0 Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
Apache HTTP Server	Total: connections 100 requests 2000 replies 2000 test-duration 9.903 s Connection rate: 10.1 conn/s (99.0 ms/conn, <=1 concurrent connections) Connection time [ms]: min 1.6 avg 2.1 max 4.1 median 2.5 stddev 0.4 Connection time [ms]: connect 0.0 Connection length [replies/conn]: 20.000 Request rate: 202.0 req/s (5.0 ms/req) Request size [B]: 110.0 Reply rate [replies/s]: min 200.0 avg 200.0 max 200.0 stddev 0.0 (1 samples) Reply time [ms]: response 0.1 transfer 0.0 Reply size [B]: header 254.0 content 4148.0 footer 0.0 (total 4402.0) Reply status: 1xx=0 2xx=2000 3xx=0 4xx=0 5xx=0 CPU time [s]: user 5.24 system 4.64 (user 52.9% system 46.9% total 99.8%) Net I/O: 889.9 KB/s (7.3*10^6 bps) Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0 Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

test-3

Simple HTTP Server	Total: connections 3000 requests 3000 replies 3000 test-duration 2.060 s Connection rate: 1456.2 conn/s (0.7 ms/conn, <=199 concurrent connections) Connection time [ms]: min 0.2 avg 65.8 max 1987.7 median 1.5 stddev 261.8 Connection time [ms]: connect 56.5 Connection length [replies/conn]: 1.000 Request rate: 1456.2 req/s (0.7 ms/req) Request size [B]: 107.0 Reply rate [replies/s]: min 0.0 avg 0.0 max 0.0 stddev 0.0 (0 samples) Reply time [ms]: response 9.4 transfer 0.0 Reply size [B]: header 90.0 content 4148.0 footer 0.0 (total 4238.0) Reply status: 1xx=0 2xx=3000 3xx=0 4xx=0 5xx=0 CPU time [s]: user 0.15 system 1.83 (user 7.4% system 88.8% total 96.2%) Net I/O: 6178.9 KB/s (50.6*10^6 bps) Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0 Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
Persistent HTTP Server	Total: connections 100 requests 3000 replies 3000 test-duration 9.904 s Connection rate: 10.1 conn/s (99.0 ms/conn, <=1 concurrent connections) Connection time [ms]: min 2.1 avg 3.7 max 11.6 median 3.5 stddev 1.7 Connection time [ms]: connect 0.0 Connection length [replies/conn]: 30.000 Request rate: 302.9 req/s (3.3 ms/req) Request size [B]: 107.0 Reply rate [replies/s]: min 300.0 avg 300.0 max 300.0 stddev 0.0 (1 samples) Reply time [ms]: response 0.1 transfer 0.0 Reply size [B]: header 90.0 content 4148.0 footer 0.0 (total 4238.0) Reply status: 1xx=0 2xx=3000 3xx=0 4xx=0 5xx=0 CPU time [s]: user 5.07 system 4.76 (user 51.2% system 48.0% total 99.3%) Net I/O: 1285.2 KB/s (10.5*10^6 bps) Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0 Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Pipelined HTTP Server	`Total: connections 100 requests 3000 replies 3000 test-duration 9.904 s`
	
	`Connection rate: 10.1 conn/s (99.0 ms/conn, <=1 concurrent connections)`
	`Connection time [ms]: min 2.2 avg 3.7 max 13.7 median 2.5 stddev 1.9`
	`Connection time [ms]: connect 0.0`
	`Connection length [replies/conn]: 30.000`
	
	`Request rate: 302.9 req/s (3.3 ms/req)`
	`Request size [B]: 107.0`
	
	`Reply rate [replies/s]: min 300.0 avg 300.0 max 300.0 stddev 0.0 (1 samples)`
	`Reply time [ms]: response 0.1 transfer 0.0`
	`Reply size [B]: header 90.0 content 4148.0 footer 0.0 (total 4238.0)`
	`Reply status: 1xx=0 2xx=3000 3xx=0 4xx=0 5xx=0`
	
	`CPU time [s]: user 5.02 system 4.84 (user 50.7% system 48.9% total 99.6%)`
	`Net I/O: 1285.3 KB/s (10.5*10^6 bps)`
	
	`Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0`
	`Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0`

Apache HTTP Server	`Total: connections 100 requests 3000 replies 3000 test-duration 9.903 s`
	
	`Connection rate: 10.1 conn/s (99.0 ms/conn, <=1 concurrent connections)`
	`Connection time [ms]: min 2.3 avg 3.1 max 5.7 median 3.5 stddev 0.6`
	`Connection time [ms]: connect 0.0`
	`Connection length [replies/conn]: 30.000`
	
	`Request rate: 302.9 req/s (3.3 ms/req)`
	`Request size [B]: 110.0`
	
	`Reply rate [replies/s]: min 300.0 avg 300.0 max 300.0 stddev 0.0 (1 samples)`
	`Reply time [ms]: response 0.1 transfer 0.0`
	`Reply size [B]: header 254.0 content 4148.0 footer 0.0 (total 4402.0)`
	`Reply status: 1xx=0 2xx=3000 3xx=0 4xx=0 5xx=0`
	
	`CPU time [s]: user 5.27 system 4.61 (user 53.2% system 46.6% total 99.8%)`
	`Net I/O: 1334.8 KB/s (10.9*10^6 bps)`
	
	`Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0`
	`Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0`

We should note that every test returned the **200 OK** status code response. This is as intended, and assures that the performance of each of our servers are tested as fairly as possible (i.e. no bad status codes slows down any of the servers, or skews the results).

Let us now move on to analyzing this data for our conclusion...

5. Conclusion—How the Differences in our Servers Affected Their Performance

What Went as Expected:

As expected, the **Persistent, Pipelined, and Apache HTTP Servers** completely outperformed the **Simple HTTP Server**. When looking at the connection times for the lightest test: `test-1`, the Simple HTTP Server's connection times are as follows:

```
Connection time [ms]: min 0.2 avg 4.4 max 1009.9 median 0.5 stddev 63.5.
```

On the other hand, the three other servers have on average:

```
Connection time [ms]: min 0.9 avg 1.23 max 4.5 median 1.5 stddev 0.46.
```

Again, this is to be expected as the Simple HTTP server must form a new TCP connection for every new requests that it makes for an object. Similarly, we see this effect amplify as the number of requests goes up in `test-2` and `test-3`!

However, what's interesting—and almost unexpected here is how the minimum connection time for the Simple HTTP Server is 0.5 ms faster than the average minimum time for the other servers. Upon further inspection, we can deduce here that this might be the case because a particular, perhaps lightweight request, was able to get a server response quicker on the the Simple Server. But, even the median connection time for this Simple Server is a full millisecond faster than the average median connection time for the other servers! It must be the case that great outliers skew the Simple Server's connection time—the standard deviation: `stddev`, confirms this. Nonetheless, the maximum connection time for Simple Server is almost 995 ms greater than the average of the other servers, and the average connection time is also much slower—clearly showing that persistent connection allows for faster handoffs on average, which is most important for any server.

Next (also as expected) the **Apache** Server slightly outperformed the **Persistent** and **Pipelined** Server. For the final and most rigorous test for our web servers with 3000 requests, we see that for the Apache Server:

```
Connection time [ms]: min 2.3 avg 3.1 max 5.7 median 3.5 stddev 0.6.
```

Now, here are the connection times for the Persistent Server:

```
Connection time [ms]: min 2.2 avg 3.7 max 13.7 median 2.5 stddev 1.9.
```

Finally, here we see the connection times for the Pipelined Server:

```
Connection time [ms]: min 2.1 avg 3.7 max 11.6 median 3.5 stddev 1.7.
```

On average, the results for the Apache Server seem to be just slightly faster than Persistent and Pipelined. However, notice that the `max` connection time, and the `stddev` are much better than our two server implementations. This signals that the Apache Server is much more consistent, speed-wise, when it comes to answering these client requests. Again, this points back to and confirms what we hypothesized—since the Apache Server is a commercial piece of software, it will be much more optimized as an HTTP Server, but since our servers are essentially doing the same task, the difference in performance should not be too drastic!

Where our Hypotheses Failed:

In our hypotheses, we mentioned:

> "[W]e think the **Pipelined Server** is slightly going to outperform the **Persistent Server** as the client does not need to wait for a response before sending multiple requests!"

However, upon inspecting our performance tests, we see that this is not exactly the case!

Take a look at `test-2`...

For Persistent Server:

```
Connection time [ms]: min 1.5 avg 2.1 max 11.3 median 1.5 stddev 1.2.
```

For Pipelined Server:

```
Connection time [ms]: min 1.6 avg 2.6 max 10.0 median 2.5 stddev 1.6.
```

Here, we see Persistent outperformed Pipelined.

In contrast, for `test-3`...

For Persistent Server:

```
Connection time [ms]: min 2.2 avg 3.7 max 13.7 median 2.5 stddev 1.9.
```

For Pipelined Server:

```
Connection time [ms]: min 2.1 avg 3.7 max 11.6 median 3.5 stddev 1.7.
```

Here, we see Pipelined outperformed Persistent.

As a reminder, `test-2` and `test-3` differ by about 1000 requests. Our team suggests that maybe because the Pipelined Server requires a bit more of an overhead to process—parsing multiple client requests at once, instead of sending them back each one-by-one, this could have contributed to the Pipelined Server being slightly slower than the Persistent Server in `test-2`. In this case, when the requests to overhead ratio is smaller like in `test-3` (where we send 3000 requests, instead of 2000), Pipelined might have the chance to be faster than Persistent. Through this performance testing, we see that theory and practice seem to align most of the time, but as with everything, there exists some interesting outliers.