Implementation and Performance Testing of Different HTTP Servers

Ziheng Zhuang
ziheng.zhuang@mail.utoronto.ca

Xinyan He
xinyan.he@mail.utoronto.ca

Abstract

In this report, we will introduce implementation of three different HTTP servers, Simple HTTP server, Persistence Server, Pipelined Server and their corresponding performance

1 Implementation Details for each server

1.1 Simple Server

For each connection, simple server would buffer request, and once see end of http header notation `\r\n\r\n` in the buffer, extract the request string from buffer. We have a Then we parse request string and try to get corresponding resource. If there is If-Modified-Since in the header, simple server would check for the modify date before reading file because some large file takes time to read. Then, construct an responseGenerator class given response status, set several attributes and body, generate string given ResponseGenerator, and send it back. After send finishes, simple server would close the connection because it does not support persistence.

1.2 Persistent Server

Compared to simple server, persistence server would check the Connection attribute in HTTP header. If it sees Keep-Alive, then persistence server would not close the connection after sending the response. Also it sets content-Length in response header to help client recognize the length of the body. Persistence Server should be faster than simple server because it does not require multiple connection for multiple requests

1.3 Pipelined Server

Compared to persistence server, pipelined server would not wait for send to finish before handling the next request. To do this, in each request handling, pipelined server would generate the repsonse string, then initiate another thread to send the date, and jump immediately to handle the next request. Once the connection is about to be terminated, pipelined server would wait for all data to be sent before close the connection. Theoretically pipelined server should be faster than persistence server, but in practice we observed a performance decrease for pipelined server. We believe there might be several reasons that causes the situation:

1. Pipelined server uses new threads and need sync operations such as locking, so the overhead of synchronization would hide the performance increase of overlapping sending and receiving.

2. Pipelining is also a feature of the client. The client would send without getting the response in nature like Chrome, so our pipelined server could only overlap sending with receiving on the server side, and the performance increase of doing so is not significant compared to the overlapping nature of TCP protocol.

Preprint. Under review.

2 Reproduce Instruction

1. The server was implemented using C++ language, and we will compile the file first

```
simple-server git:(master) make
clang++ -std=c++17 SimpleServer.cc -o SimpleServer
clang++ -std=c++17 PersistentServer.cc -o PersistentServer
clang++ -std=c++17 PipelinedServer.cc -o PipelinedServer
```

2. All the three servers would take port number and http root path, here all the web pages for our servers are located in **http** directory. We will take Simple server as an example and use **http** as our http root path , 8000 as our port number

```
./SimpleServer 8000 http
```

3. Our server has been turn on after the previous command, you can open a browser and type **localhost:8080/** and plus the html file you are interested to go to. For example, the following url would lead us to the index.html. And if you type some web page which not stored in our **http** directory, you should see **HTTP/1.1 404 Not Found** in terminal

```
localhost:8080/index.html
```

4. For reproducing the conditional get case, we can test with the following command for transmitting a header containing **If-Modified-Since**

```
echo 'GET /index.html HTTP/1.1\r\nConnection: keep-alive\r\nIf-Modified-Since:
Mon, 06 Dec 2022 03:54:33 GMT\r\n\r\n' | nc localhost 8000
```

And we should get a result as the following, the response should be 'HTTP/1.1 304 Not Modified'

```
HTTP/1.1 304 Not Modified
Last-Modified: Mon, 06 Dec 2021 14:56:55 GMT
Content-Type: text/plain
```

3 Performance Testing

We will perform three tests here to analyze the performance for our HTTP server. The detailed results of performance testing are in test_data.txt.

Table 1: Port number we used here for different servers

Dataset	Port number
Apache Server	8080
Simple Server	8000
Persistent Server	8001
Pipelined Server	8002

3.1 Apache Benchmark Tool

The first tool we use to test the loading performance for servers is Apache Benchmark

```
Usage: ab [options] [http[s]://]hostname[:port]/path
Options are:
    -n requests     Number of requests to perform
    -c concurrency  Number of multiple requests to make
```

Our first test will be sending 100 requests in total, and with maximum 10 requests running concurrently. The command of this is as follows.

```
ab -n 100 -c 10  http://127.0.0.1:8080/
```

Table 2: Performance result through Apache Benchmark Tool

Sever	Requests per second	Time per request (ms)	Transfer rate (Kbytes/sec)
Apache Server	12374.71	0.808	3069.51
Simple Server	11392.12	0.878	1012.39
Persistent Server	13322.68	0.751	1338.27
Pipelined Server	12014.90	0.832	1290.66

For the second test, we will reduce the concurrent connection to 1

```
ab -n 100 -c 1  http://127.0.0.1:8080/
```

Table 3: Performance result through Apache Benchmark Tool

Sever	Requests per second	Time per request (ms)	Transfer rate (Kbytes/sec)
Apache Server	5062.78	0.198	1255.81
Simple Server	4173.80	0.240	370.91
Persistent Server	4217.27	0.237	453.03
Pipelined Server	3214.30	0.311	345.29

3.2 httperf - HTTP performance measurement tool

Here we will employ httperf to sends the HTTP requests,

```
httperf --num-conns=3000 --timeout=5 --server=127.0.0.1 --uri=/index.html  --port=8080
```

Table 4: Performance result through httperf

Sever	Request rate	Net I/O (KB/s)
Apache Server	7141.8	2141.1
Simple Server	4202.3	993.1
Persistent Server	4316.8	1104.5
Pipelined Server	3717.7	951.2

4 Performance Result Analysis

4.1 Comparison between Apache Server and Servers we implemented

One thing we found interesting about Apache Server is that compared to the servers we implemented, Apache Server have a high variation in the result of request rate. So sometimes it could perform better while sometimes it perform worse. This is probably due to the complexity of features of Apache Server, there are many feature implementation that would affect the performance of Apache Server, such as the feature Multiple Request Processing modes, which is event-based and threaded, the overhead of synchronization may sometimes affect the performance. Different feature may affect the performance of server at a different level at a certain time, leading a high variation on Apache server's performance.

4.2 Comparison between Simple Server and Persistent Server

We can see from Table2 that compared to Simple Server, the Persistent Server can handle more requests within a second. The reason for that is for Persistent Server, the extra feature we add is handling persistent connections while for Simple Server we only handle Non-Persistent connection. For Non-Persistent connection, we have to set up a TCP connection for every request, which takes extra RTT for every request. While for Persistent connection, we only have to set up a TCP connection at the beginning and we would leave it open until the process ends. Therefore, within the same time, more requests would be handled by our Persistent Server compared to the Simple Server.

4.3 Comparison between Persistent Server and Pipelined Server

We can see from Table2 that Persistent Server also perform better than Pipelined Server. According to our assumption, Pipelined Server should behave better at this case, since for Pipelined Server HTTP request can be sent without waiting for previous request's response. Thus we can handle more requests within the same time for Pipelined Server. However in reality, we may have the issue for buffering and context switching overhead of pipelining, leading to the increasing single request's latency. We can also notice from the results when our concurrent connection number reduced to 1 and httperf test case that the result of Pipelined Server become even worse than Simple Server, this is due to the reason that while the concurrent connection number decrease, the advantage of multiple threads would be decrease, while the disadvantage for overhead issue would become a trival part now. Therefore, we would expect the performance of Persistent Server becomes even worse.

5 Limitations and improvement

5.1 Improvement ideas on Pipelined Server

Current design is to overlap [receving+parsing, reading files] with [sending]. We could further divide the server stage into overlap [receiving+parsing], [reading files] and [sending], because in some scenarios, reading files from disk is the bottleneck. Just imagine how CPU pipelines instructions.

5.2 Limitations on Conditional Get

Due to time constraint, we are only able to implement one conditional get feature for http requests. We implemented If-Modified-Since feature. There are other conditional get features, but all of them have the same idea (use time or etag value to check cache validity). If server find the modify time of the file is earlier than requested modify time, server would not send the resource back. Instead send 304 not modified to tell client that cache is valid. Also, If-Modified-Since is the one used by most browsers, so with this feature we could enable caching on browsers, which is the most common usage of our http server.

6 Video Presentation

We have uploaded the video to Youtube and here is the link `https://www.youtube.com/watch?v=R7Jmp4DzBBc`