湖南大學

HUNAN UNIVERSITY

云计算性能分析报告

实 验 名 称：实验二

组长　姓名：田志鹏-201808010404

学生　姓名：沈祥振-201808010427

学生　姓名：谭江浩-201808010405

学生　姓名：罗凯洋-201808010527

完 成 时 间：2021年5月5日

信息科学与工程学院

一、实验概要

超文本传输协议（HTTP）是应用层的一个协议，是万维网生态系统的核心。HTTP 最初的用途是传输文本和图像，但 Web 服务模型的发展需要大量与 Web 相关的协议和组件来建立运行于 Web 浏览器里的工具。

HTTP 使用 CS 模型：HTTP 客户端打开与 HTTP 服务器的网络连接，并发送 HTTP 请求消息。HTTP 消息是简单的格式化数据块。

所有 HTTP 消息分为两种：请求消息和响应消息。

请求消息请求来自 Web 服务器的操作。响应消息将请求的结果返回给客户端。 请求和响应消息都具有相同的基本消息结构。

库网址：https://github.com/tzphh/CloudComputing_Labs.git

最终实验结果路径：CloudComputingLabs/Lab2/

1. **HTTP 服务器功能说明**

   每个 TCP 连接只能同时发送一个请求。客户端等待响应，当客户端获得响应时，也许将 TCP 连接重新用于新请求（或使用新的 TCP 连接）。这也是普通 HTTP 服务器支持的内容。

   a. 处理 HTTP GET 请求
   b. 处理 HTTP POST 请求
   c. 其他请求不处理

2. **编译背景**

   Socket C++

3. **使多线程来提高并发性**

   多线程设计：

   每连接一个客户端，就创建一个线程解决所有该客户端发出的请求。在创建线程前上锁，进入线程回调函数后解锁。

4. **指定参数**

   程序应启用长选项以接受参数并在启动期间指定这些参数。它们是--ip，--port。

   a. --ip- 指定服务器 IP 地址。
   b. --port -选择 HTTP 服务器侦听传入连接的端口。
      如果未指定端口号，则默认为端口 8888。

5. **运行 HTTP 服务器**

   创建套接字 sockfd，并绑定 bind 套接字，监听 listen 套接字。循环接受不同的客户端，每连接一个客户端则创建一个线程，客户端关闭则线程关闭。最后关闭监听的套接字。

   运行方法：./httpserver（默认 ip 为 127.0.0.1，port 为 8888）

   若要更改 ip 和 port，则可指定参数更改。

   ```
   root@ubuntu:/home/wayyzt64/lab2# ./httpserver --ip 127.0.0.1 --port 8888
   ```

6. **请求处理大致思路**

   a. 处理 GET 请求 判断是报文是 GET 请求：请求路径与 html 页面文件相对应→回复 200 OK 和文件的 完整内容；否则回复 404 Not Found。
   b. 处理 POST 请求
      判断报文是 POST 请求：判断 URL 为/Post_show，并且键为 Name 和 ID→则回复 200 OK 和 Name-ID 对；否则回复 404 Not Found。
   c. 其他方法
      回复 501 Not Implemented

7. 实验环境

2GB 内存；Intel(R) Core(TM) i5-9400 CPU @ 2.90GHz

共有 2 个核心 CPU；不使用超线程技术。

```
wayyzt64@ubuntu:~/lab2$ lscpu
Architecture:            x86_64
CPU op-mode(s):          32-bit, 64-bit
Byte Order:              Little Endian
CPU(s):                  2
On-line CPU(s) list:     0,1
Thread(s) per core:      1
Core(s) per socket:      1
Socket(s):               2
NUMA node(s):            1
Vendor ID:               GenuineIntel
CPU family:              6
Model:                   158
Model name:              Intel(R) Core(TM) i5-9400 CPU @ 2.90GHz
Stepping:                10
CPU MHz:                 2904.004
BogoMIPS:                5808.00
Hypervisor vendor:       VMware
Virtualization type:     full
L1d cache:               32K
L1i cache:               32K
L2 cache:                256K
L3 cache:                9216K
NUMA node0 CPU(s):       0,1
```

二、性能测试

通过更改同时发送请求到服务器的并发客户端数，测试服务器处理指定个数 HTTP 请求的时间大小

GET

client=1

```
./get 127.0.0.1 8888 10 耗时sec= 0.03 秒
./get 127.0.0.1 8888 100 耗时sec= 0.15 秒
./get 127.0.0.1 8888 1000 耗时sec= 0.42 秒
./get 127.0.0.1 8888 2000 耗时sec= 0.71 秒
./get 127.0.0.1 8888 3000 耗时sec= 1.16 秒
./get 127.0.0.1 8888 4000 耗时sec= 1.40 秒
./get 127.0.0.1 8888 5000 耗时sec= 1.57 秒
./get 127.0.0.1 8888 6000 耗时sec= 2.10 秒
./get 127.0.0.1 8888 7000 耗时sec= 2.34 秒
./get 127.0.0.1 8888 8000 耗时sec= 2.39 秒
./get 127.0.0.1 8888 9000 耗时sec= 2.68 秒
./get 127.0.0.1 8888 10000 耗时sec= 3.12 秒
```

client=2

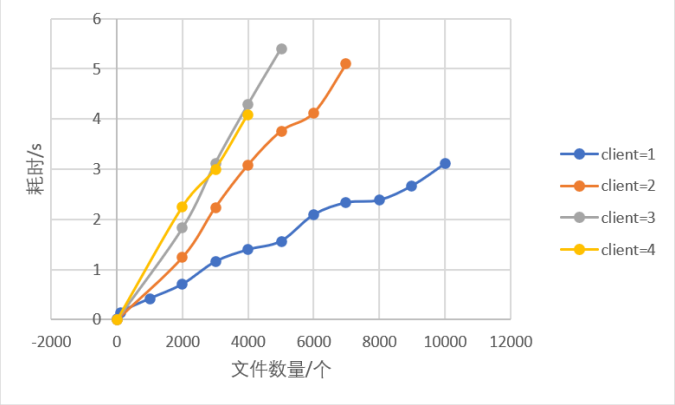
```
./get 127.0.0.1 8888 2000 耗时sec= 1.37 秒 耗时sec= 1.13 秒
./get 127.0.0.1 8888 3000 耗时sec= 2.36 秒 耗时sec= 2.13 秒
./get 127.0.0.1 8888 4000 耗时sec= 3.01 秒 耗时sec= 3.18 秒
./get 127.0.0.1 8888 5000 耗时sec= 3.84 秒 耗时sec= 3.69 秒
./get 127.0.0.1 8888 6000 耗时sec= 4.18 秒 耗时sec= 4.08 秒
./get 127.0.0.1 8888 7000 耗时sec= 5.05 秒 耗时sec= 5.17 秒
```

client=3

```
./get 127.0.0.1 8888 2000
耗时sec= 1.71 秒 耗时sec= 1.98 秒 耗时sec= 1.81 秒
./get 127.0.0.1 8888 3000
耗时sec= 3.06 秒 耗时sec= 3.27 秒 耗时sec= 3.06 秒
./get 127.0.0.1 8888 4000
耗时sec= 4.25 秒 耗时sec= 4.41 秒 耗时sec= 4.24 秒
./get 127.0.0.1 8888 5000
耗时sec= 5.42 秒 耗时sec= 5.41 秒 耗时sec= 5.38 秒
```

client=4

```
./get 127.0.0.1 8888 2000
耗时sec= 1.89 秒 耗时sec= 2.49 秒 耗时sec= 2.39 秒 耗时sec= 2.24 秒
./get 127.0.0.1 8888 3000
耗时sec= 2.40 秒 耗时sec= 3.41 秒 耗时sec= 2.96 秒 耗时sec= 3.25 秒
./get 127.0.0.1 8888 4000
耗时sec= 3.42 秒 耗时sec= 4.27 秒 耗时sec= 4.52 秒 耗时sec= 4.16 秒
```



POST

client=1

```
./post 127.0.0.1 8888 10 耗时sec= 0.01 秒
./post 127.0.0.1 8888 20 耗时sec= 0.01 秒
./post 127.0.0.1 8888 100 耗时sec= 0.06 秒
./post 127.0.0.1 8888 1000 耗时sec= 0.34 秒
./post 127.0.0.1 8888 2000 耗时sec= 0.71 秒
./post 127.0.0.1 8888 3000 耗时sec= 1.09 秒
./post 127.0.0.1 8888 5000 耗时sec= 1.48 秒
./post 127.0.0.1 8888 6000 耗时sec= 2.32 秒
./post 127.0.0.1 8888 7000 耗时sec= 2.38 秒
./post 127.0.0.1 8888 8000 耗时sec= 2.92 秒
./post 127.0.0.1 8888 9000 耗时sec= 2.53 秒
./post 127.0.0.1 8888 10000 耗时sec= 3.33 秒
./post 127.0.0.1 8888 30000 耗时sec= 10.27 秒
./post 127.0.0.1 8888 50000 耗时sec= 15.47 秒
./post 127.0.0.1 8888 100000 耗时sec= 31.61 秒
```

client=2

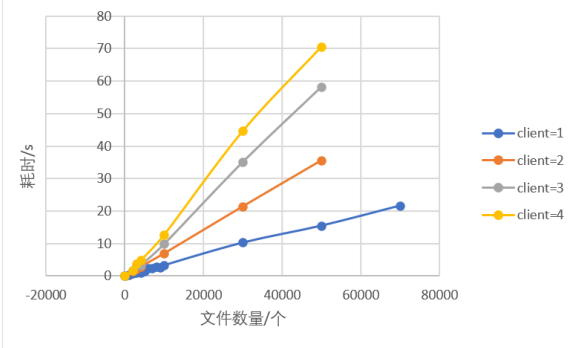
```
./post 127.0.0.1 8888 2000 耗时sec= 1.41 秒 耗时sec= 1.32 秒
./post 127.0.0.1 8888 3000 耗时sec= 2.33 秒 耗时sec= 2.18 秒
./post 127.0.0.1 8888 4000 耗时sec= 2.92 秒 耗时sec= 2.93 秒
./post 127.0.0.1 8888 10000 耗时sec= 7.01 秒 耗时sec= 6.99 秒
./post 127.0.0.1 8888 30000 耗时sec= 21.51 秒 耗时sec= 21.43 秒
./post 127.0.0.1 8888 50000 耗时sec= 35.65 秒 耗时sec= 35.65 秒
```

client=3

```
./post 127.0.0.1 8888 2000
耗时sec= 1.61 秒 耗时sec= 1.94 秒 耗时sec= 1.77 秒
./post 127.0.0.1 8888 3000
耗时sec= 2.86 秒 耗时sec= 3.13 秒 耗时sec= 2.90 秒
./post 127.0.0.1 8888 4000
耗时sec= 3.47 秒 耗时sec= 3.54 秒 耗时sec= 3.01 秒
./post 127.0.0.1 8888 10000
耗时sec= 9.87 秒 耗时sec= 10.17 秒 耗时sec= 9.61 秒
./post 127.0.0.1 8888 30000
耗时sec= 34.54 秒 耗时sec= 35.65 秒 耗时sec= 35.24 秒
./post 127.0.0.1 8888 50000
耗时sec= 58.35 秒 耗时sec= 58.52 秒 耗时sec= 58.00 秒
```

client=4

```
./post 127.0.0.1 8888 2000
耗时sec= 1.50 秒 耗时sec= 1.97 秒 耗时sec= 1.99 秒 耗时sec= 1.30 秒
./post 127.0.0.1 8888 3000
耗时sec= 3.53 秒 耗时sec= 3.80 秒 耗时sec= 3.99 秒 耗时sec= 3.71 秒
./post 127.0.0.1 8888 4000
耗时sec= 4.73 秒 耗时sec= 4.89 秒 耗时sec= 5.01 秒 耗时sec= 4.65 秒
./post 127.0.0.1 8888 10000
耗时sec= 12.12 秒 耗时sec= 13.14 秒 耗时sec= 12.84 秒 耗时sec= 12.78 秒
./post 127.0.0.1 8888 30000
耗时sec= 44.63 秒 耗时sec= 45.20 秒 耗时sec= 44.36 秒 耗时sec= 44.51 秒
./post 127.0.0.1 8888 50000
耗时sec= 70.52 秒 耗时sec= 70.51 秒 耗时sec= 71.36 秒 耗时sec= 70.08 秒
```



如上图，不论是 GET 还是 POST，当只有一个客户端发送请求时服务器处理速度最快，随着客户端数量增多，发送请求时处理速度逐渐下降。

测试结果贴近理论实际，对于双核 CPU 来说，客户端越多即线程越多，增加的上下文

切换越多，使最终处理请求个数的速度越慢。
实际测量时，get 请求最多只能测量到 10000 个请求，之后会遇到服务器端的”too many open files”，而 post 可测到 100000 个请求还有富余。
对于 Too many open files 的错误，通常意味着“文件描述符”不足，它一般会发生在“创建线程”、“创建 socket”、“打开文件”这种场景下，通过修改 ulimit 可以使问题得到初步缓解，但 get 和 post 的差距还是存在。

后来又查阅了 get 和 post 的区别：

对于 GET 方式的请求，浏览器会把 http header 和 data 一并发送出去，服务器响应 200（返回数据）；而对于 POST，浏览器先发送 header，服务器响应 100 continue，浏览器再发送 data，服务器响应 200 ok（返回数据）。

GET 只需要汽车跑一趟就把货送到了，而 POST 得跑两趟。GET 请求，通常用来获取静止数据，例如简单的网页和图片。POST 请求通常用来获取的数据，取决于我们发给服务器的数据，例如用户名和密码。GET 请求 URL 的长度是受限制的，POST 无限制。

所以猜测服务器可以处理更多 post 请求的原因，可能是单次传输负载量较小而且限制稍小。