

# Lab2 test report

Team: 一朵花三根草

## 1. 实验概要

使用网络编程知识，从零开始实现一个基于 HTTP/1.1 的 HTTP 服务器，并尝试使用从课堂上中学习的高并发编程技能来保证 web 服务器的性能。最后对服务器进行并发测试，比较不同环境下服务器的性能。

### 1.1 程序输入

无。

### 1.2 程序输出

服务器端：输出建立连接和请求类型的信息

客户端：显示 HTTP 响应报文

### 1.3 性能指标

服务器每秒可以处理多少 HTTP 请求

### 1.4 实验环境

使用实验中共有 2 个不同的实验环境：ENV1 和 ENV2。

ENV1: linux 内核版本为 Linux ubuntu 4.4.0-148-generic #174~14.04.1-Ubuntu SMP; 2GB 内存; CPU 型号为 Intel(R) Core(TM) i5-6300HQ CPU @ 2.30GHz, 共有 1 个物理 CPU; 每个物理 CPU 有 4 个物理核心, 共有 4 个物理核心; 不使用超线程技术。

ENV2-1: linux 内核版本为 Linux ubuntu 4.15.0-96-generic #97~16.04.1-Ubuntu SMP; 2GB 内存; CPU 型号为 Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz, 共 2 个物理 CPU; 每个物理 CPU 有 1 个物理核心, 共有 2 个物理核心; 不使用超线程技术。

ENV2-2: linux 内核版本为 Linux ubuntu 4.15.0-96-generic #97~16.04.1-Ubuntu SMP; 2GB 内存; CPU 型号为 Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz, 共 2 个物理 CPU; 每个物理 CPU 有 2 个物理核心, 共有 4 个物理核心; 不使用超线程技术。

### 1.5 代码实现版本

本次实验中实现的是基础版本，可以处理客户端的 GET、POST 请求，对其他请求返回

501 未实现错误，每个 TCP 连接在同一时间只有一个请求。使用了线程池来增加并发性，线程池中线程的数目在启动服务器时由--number-thread 参数确定。

## 2. 性能测试

使用 apache 完成性能测试。

### 2.1 改变服务器及其环境，测试每秒可以处理多少 HTTP 请求

#### 2.1.1 改变线程池中的线程数(ENV1)

-n 5000 (Number of requests to perform for the benchmarking session.)

-c 200 (Number of multiple requests to perform at a time. )

对 index.html 进行请求，记录大小为 1652 字节。

先提前解释一下以下 apache 测试结果中的内容：

erver Software: //被测试的服务器所用的软件信息

Server Hostname: //被测主机名

Server Port://被测主机的服务端口号

Document Path: //请求的具体文件

Document Length://请求的文件的大小

Concurrency Level: //并发级别，也就是并发数，请求中-c 参数指定的数量

Time taken for tests: //本次测试总共花费的时间

Complete requests: //本次测试总共发起的请求数量

Failed requests://失败请求的数量。因网络原因或服务器性能原因，发起的请求并不一定全部成功，通过该数值和 Complete requests 相除可以计算请求的失败率，作为测试结果的重要参考。

Total transferred: //总共传输的数据量，指的是 ab 从被测服务器接收到的总数据量，包括 index.html 的文本内容和请求头信息。

HTML transferred://从服务器接收到的 index.html 文件的总大小，等于 Document Length\*Complete requests  
=227 bytes\*100=22700 bytes

Requests per second://平均(mean)每秒完成的请求数：QPS，这是一个平均值

Time per request: (mean) //从用户角度看，完成一个请求所需要的时间（因用户数量不止一个，服务器完成 10 个请求，平均每个用户才接收到一个完整的返回，所以该值是下一项数值的 10 倍。）

Time per request: (mean, across all concurrent requests)// 服务器完成一个请求的时间。

Transfer rate: [Kbytes/sec] received //网络传输速度。对于大文件的请求测试，这个值很容易成为系统瓶颈所在。要确定该值是不是瓶颈，需要了解客户端和被测服务器之间的网络情况，包括网络带宽和网卡速度等信息。

Connection Times (ms)					
	min	mean[+/-sd]	median	max	
Connect:	0	14 116.9	0	1032	
Processing:	1	41 17.2	41	95	
Waiting:	1	40 17.1	41	94	
Total:	9	55 118.7	42	1096	

这几行组成的表格主要是针对响应时间也就是第一个 Time per request 进行细分和统计。一个请求的响应时间可以分成网络链接 (Connect)，系统处理 (Processing) 和等待 (Waiting) 三个部分。表中 min 表示最小值；mean 表示平均值；[+/-sd] 表示标准差 (Standard Deviation)，也称均方差 (mean square error)；median 表示中位数；max 当然就是表示最大值了。

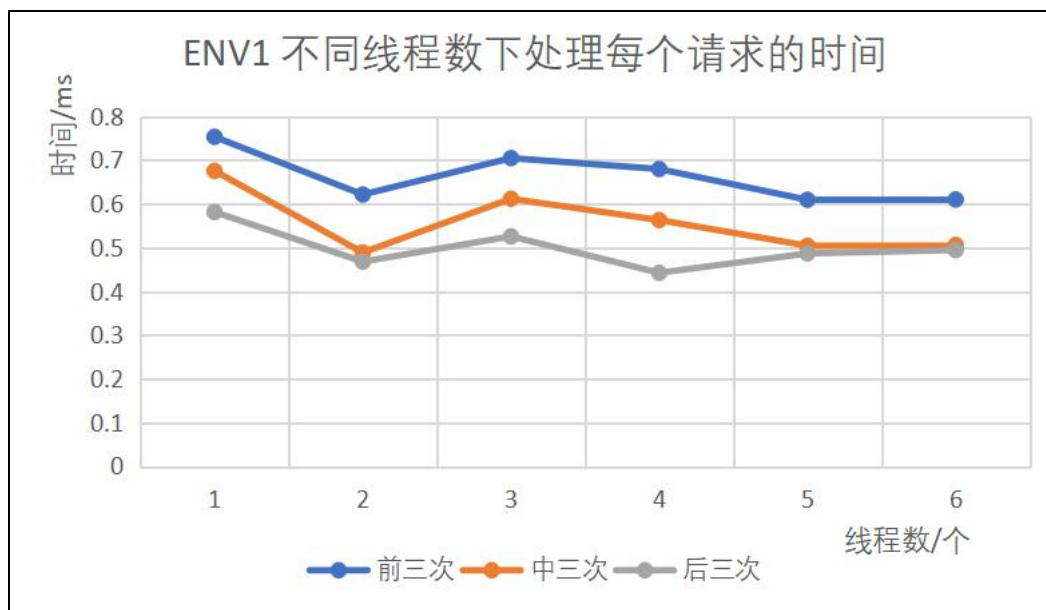
需要注意的是表中的 Total 并不等于前三行数据相加，因为前三行的数据并不是在同一个请求中采集到的，可能某个请求的网络延迟最短，但是系统处理时间又是最长的。所以 Total 是从整个请求所需要的时间的角度来统计的。这里可以看到最慢的一个请求花费了 27.773s，这个数据可以在下面的表中得到验证。

Percentage of the requests served within a certain time (ms)	
50%	42
66%	49
75%	54
80%	57
90%	64
95%	70
98%	79
99%	1050
100%	1096 (longest request)

表二：这个表第一行表示有 50% 的请求都是在 121ms 内完成的。以此类推，90% 的请求是小于等于 1207ms 的。刚才我们看到响应时间最长的那个请求是 27773ms，那么显然所有请求 (100%) 的时间都是小于等于 27773 毫秒的，也就是表中最后一行的数据肯定是时间最长的那个请求 (longest request)。

ENV1 下改变线程数目处理每个请求所需时间的变化：

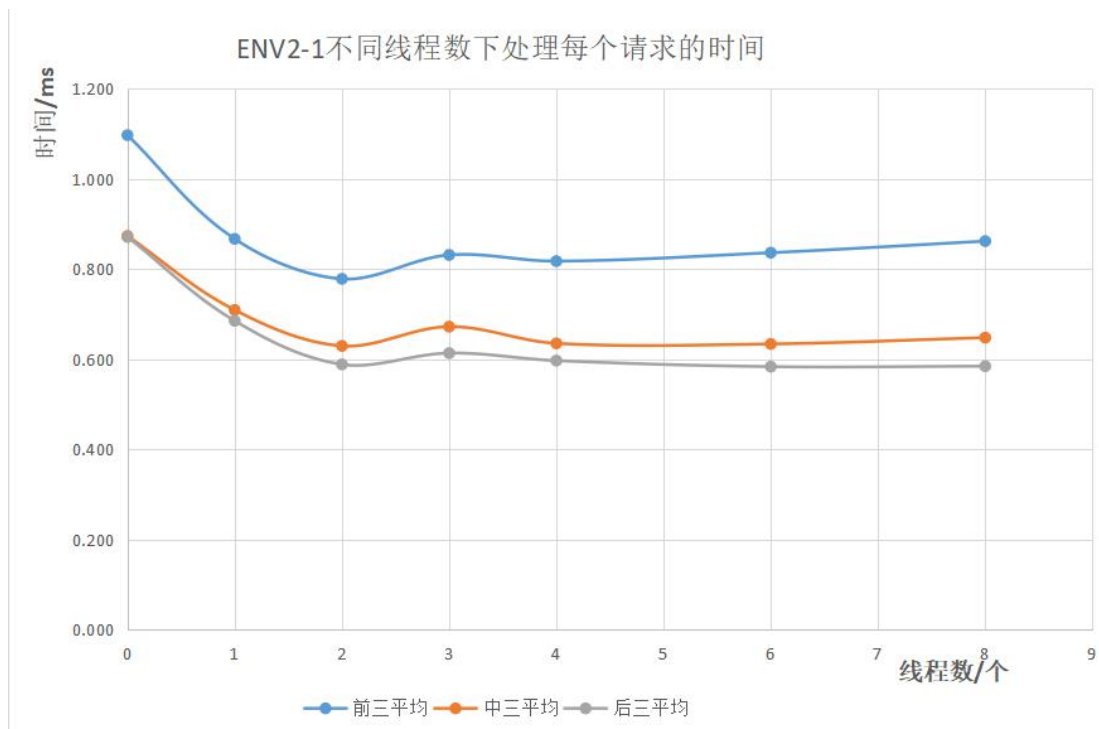
ThreadNum	时间/ms							
	1	2	3	4	5	前三平均	中三平均	后三平均
1	0.821	0.504	0.540	0.423	0.442	0.622	0.489	0.468
2	0.737	0.710	0.668	0.458	0.452	0.705	0.612	0.526
4	0.772	0.758	0.510	0.422	0.396	0.680	0.563	0.443
6	0.756	0.542	0.531	0.441	0.489	0.610	0.505	0.487
8	0.800	0.538	0.493	0.488	0.503	0.610	0.506	0.495



### 2.1.2 改变线程池中的线程数和 CPU 核数(ENV2)

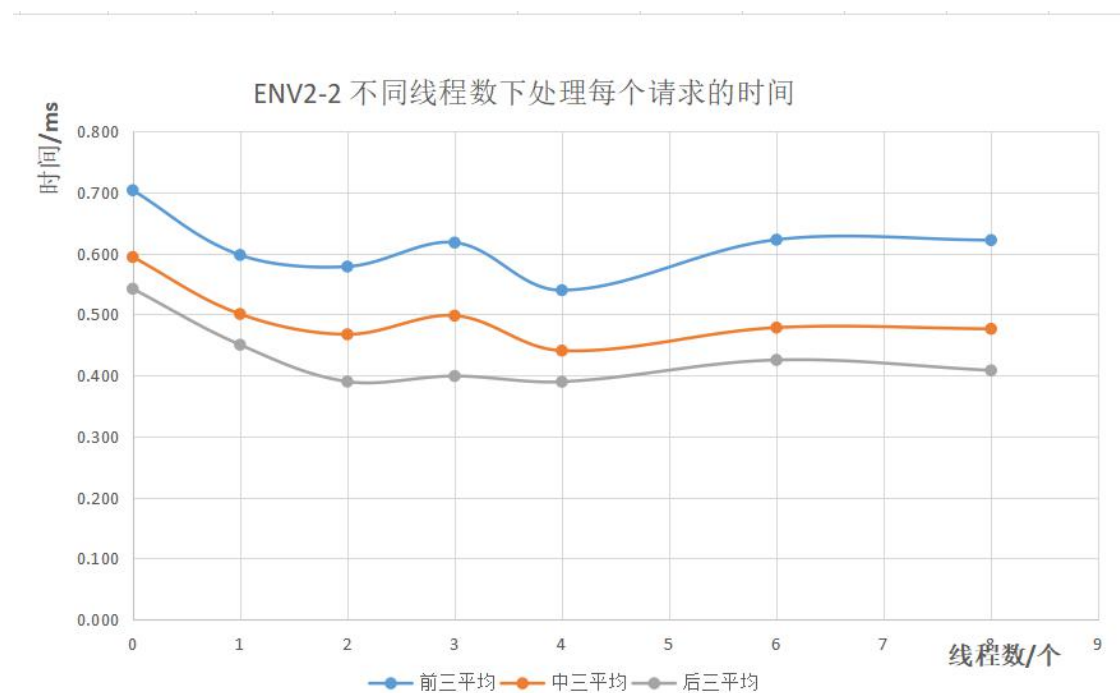
ENV2-1 下改变线程数目处理每个请求所需时间的变化:

ThreadNum	时间/ms							
	1	2	3	4	5	前三平均	中三平均	后三平均
0	1.544	0.871	0.875	0.873	0.865	1.097	0.873	0.871
1	1.254	0.735	0.611	0.781	0.663	0.867	0.709	0.685
2	1.054	0.661	0.619	0.608	0.538	0.778	0.629	0.588
3	1.029	0.783	0.681	0.552	0.607	0.831	0.672	0.613
4	1.087	0.706	0.659	0.540	0.591	0.817	0.635	0.597
6	1.172	0.705	0.631	0.565	0.554	0.836	0.634	0.583
8	1.192	0.693	0.700	0.550	0.503	0.862	0.648	0.584



ENV2-2 下改变线程数目处理每个请求所需时间的变化：

ThreadNum	时间/ms							
	1	2	3	4	5	前三平均	中三平均	后三平均
0	0.872	0.693	0.545	0.544	0.537	0.703	0.594	0.542
1	0.771	0.551	0.470	0.481	0.400	0.597	0.501	0.450
2	0.725	0.599	0.411	0.392	0.367	0.578	0.467	0.390
3	0.708	0.718	0.427	0.349	0.421	0.618	0.498	0.399
4	0.673	0.552	0.394	0.376	0.399	0.540	0.441	0.390
6	0.936	0.521	0.410	0.504	0.362	0.622	0.478	0.425
8	0.860	0.583	0.422	0.424	0.379	0.622	0.476	0.408



`number-thread` 为 0 时，是一个动态上线的线程池。

同是 4 核下，ENV2-2 下比 ENV1 下性能更好一些。另外从图表中看出：

1. 启动服务器后进行多次测试，请求响应时间呈下降趋势（因为第一次测试时有创建连接的开销）。
2. `--number-thread` 为 2 时基本已经达到瓶颈，后续再增加线程数量，对请求的响应时间没有什么减少，甚至还会小幅度上升。