序

近期在对阿里云服务器做压力测试,因为webbench ,ab两个工具的压力测试结果和loadrunner,jemer的压测结果相关太远,有上百倍的差距,也是让我们百思不得其解,非常干扰思路.所以对ab,webbench做了简单测试~

## 压测工具简介:

|  |  |  |
| --- | --- | --- |
| 工具 | 简介 | 级别 |
| Ab | ab - Apache HTTP server benchmarking tool | 轻量级(比webbench稍强大) |
| Webbench | webbench - simple forking web benchmark | 轻量级 |
| jmeter | Apache JMeter是Apache组织开发的基于Java的压力测试工具 | 中级 |
| loadrunner | HP研发,收费版,但国内早破解泛滥,有window(6G),linux(1.6G)版本 | 重量级/专业级 |

------loadrunner几尽为国内测试人员的专用压测工具,他的高度灵活性,数据分析功能,图表展示,用户行为模拟等功能非常强大,专业压测还是以loadrunner为准------

## 并发1000概念理解

这里需再强调下并发的概念,以并发1000为例:

这里的并发1000是针对用户来说,而非服务器每秒并发请求数.即每秒有1000用户(每秒有多次或多种行为)同时对服务器发起请求

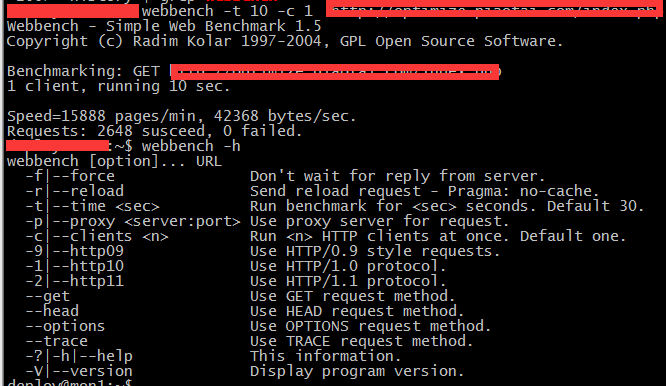
## Webbench 每秒请求数测试

-t 指定压测时间

-c 指定并发用户数(非请求数)

-f 零等待服务器响应

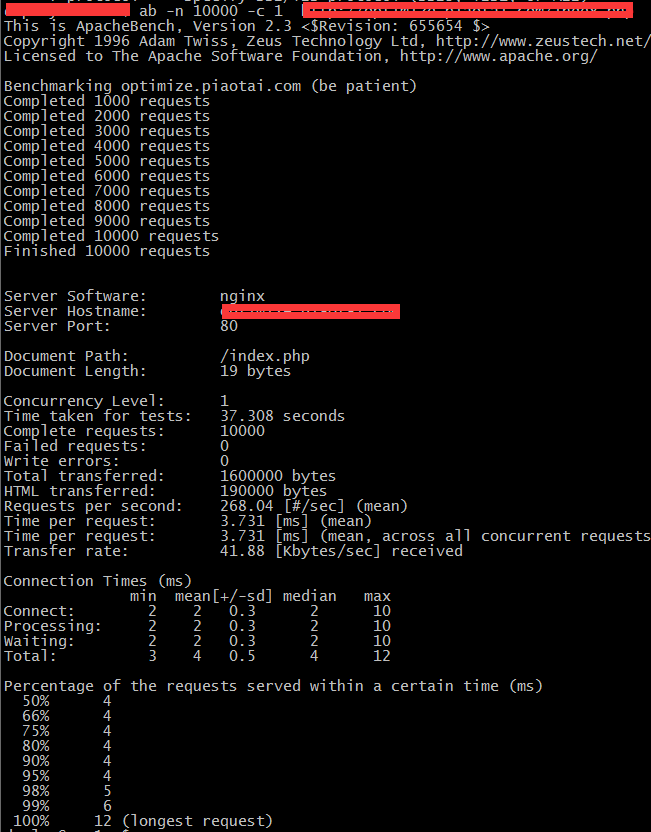
如下图做了简单的性能压测.可以看出webbench模拟一个client相当于每秒有264.8个请求,，如果并发压-c 1000clients,相当于每秒有1000\*264.8请求~想想也是相当凶残~~



## Ab每秒请求数测试

-n 设置请求总数

-c 设置并发client数量



如图可以看出ab模拟一个client相当于每秒有268.04有个请求,，如果并发压-c 1000clients,相当于每秒有1000\*268.04请求

## ab/webbench如何模拟用户行为

业界一直认为loadrunner是最专业的,ab,webbench相比之下哪里不专业了呢,测试同学的说法是webbench,ab只进行了2次握手就离开继续下一次新请求,这里个人觉得不靠谱,我们模拟一次测试验证下.这里有遇到一个问题,如果去抓取一个用户数据包.ab的功能就有大显身手了,只压一个数据包.

ab -n 1 -c 1 -k 'http://optimize.piaotai.com/index.php' 同时在被压测机抓包

### http请求数据流解析

#### 抓包命令

tcpdump -i eth0 ip host web-yv4 and port ! 22

tcpdump -i eth0 ip host 10.169.11.99 and 10.171.215.112 and dst port 80

#### tcpdump flags

|  |  |  |
| --- | --- | --- |
| **TCP Flag** | **Flag in tcpdump** | **Flag Meaning** |
| SYN | s | Syn packet, a session establishment request. The first part of any TCP connection. |
| ACK | ack | Ack packet, used to acknowledge the receipt of data from the sender. May appear in conjunction with other flags. |
| FIN | f | Finish flag, used to indicate the sender’s intention to terminate the connection to the receiving host. |
| RESET | r | Indicates the sender’s intention to immediately abort the existing connection. |
| PUSH | p | Signals the immediate push of data from the sending host to the receiving host. For interactive applications such as telnet, the main issue is the quickest response time, which this “push” flag signals. |
| URGENT | urg | Urgent data should take precedence over other data. For example, a Ctrl-C to terminate a FTP download. |
| Placeholder | . | If the connection does not have a syn, finish, reset, or push flag set, this placefolder flag will be found after the destination port. Note that it also appears in conjunction with the ack flag. |

SYN表示建立连接，  
FIN表示关闭连接，  
ACK表示响应，  
PUSH表示有 DATA数据传输，  
RST表示连接重置。

码即tcp标志位，有6种标示：SYN(synchronous建立联机) ACK(acknowledgement 确认) PSH(push传送) FIN(finish结束) RST(reset重置) URG(urgent紧急)Sequence number(顺序号码) Acknowledge number(确认号码)

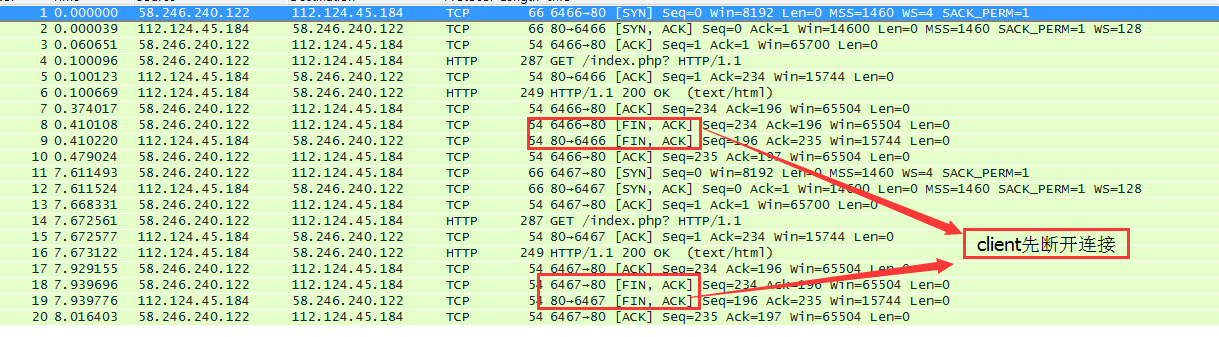
如下范例是一条完整的http数据流分析

|  |
| --- |
| 19:28:31.951855 IP 58.246.240.122.63798 > 112.124.45.184.http: Flags [S], seq 3672193843, win 8192, options [mss 1460,nop,wscale 2,nop,nop,sackOK], length 0  ***//client58.246.240.122通过63798向WebServer 112.124.45.184发起一条序列号为3672193843,win size为8192,没有数据的SYNC请求***  ***//第一次握手***  19:28:31.951889 IP 112.124.45.184.http > 58.246.240.122.63798: Flags [S.], seq 38994743, ack 3672193844, win 14600, options [mss 1460,nop,nop,sackOK,nop,wscale 7], length 0  ***//WerServer回应ack收到并发送一条seq为38994743***  ***//第二次握手***  19:28:32.207276 IP 58.246.240.122.63801 > 112.124.45.184.http: Flags [S], seq 3586463872, win 8192, options [mss 1460,nop,wscale 2,nop,nop,sackOK], length 0  ***//(应该)前两次握手失败,重新发起一次新一轮握手请求***  ***//第一次握手***  19:28:32.207304 IP 112.124.45.184.http > 58.246.240.122.63801: Flags [S.], seq 1159592009, ack 3586463873, win 14600, options [mss 1460,nop,nop,sackOK,nop,wscale 7], length 0  ***//第二次握手***  19:28:32.361241 IP 58.246.240.122.63798 > 112.124.45.184.http: Flags [.], ack 1, win 16425, length 0  ***//第三次握手,三次握手至此结束***  ***//client回应收到***  19:28:32.361403 IP 58.246.240.122.63798 > 112.124.45.184.http: Flags [P.], seq 1:350, ack 1, win 16425, length 349  ***//client发起ack应答并PUSH一条长度为349大小数据的请求***  19:28:32.361419 IP 112.124.45.184.http > 58.246.240.122.63798: Flags [.], ack 350, win 123, length 0  ***//webserver应答seq为350的请求.***  19:28:32.361945 IP 112.124.45.184.http > 58.246.240.122.63798: Flags [P.], seq 1:241, ack 350, win 123, length 240  ***//webserver应答并PUSH一条长度为240大小的数据***  19:28:32.383558 IP 58.246.240.122.63801 > 112.124.45.184.http: Flags [.], ack 1, win 16425, length 0  ***//client应答ack1表示收到数据***  19:28:32.613585 IP 58.246.240.122.63798 > 112.124.45.184.http: Flags [.], ack 241, win 16365, length 0  ***//client应答ack241表示收到数据***  19:28:37.384847 IP 58.246.240.122.63801 > 112.124.45.184.http: Flags [F.], seq 1, ack 1, win 16425, length 0  ***//client应答ack1并起一条seq为1的FIN断开请求***  ***//第一次挥手***  ***//tcp状态置为timewait状态***  19:28:37.384949 IP 112.124.45.184.http > 58.246.240.122.63801: Flags [F.], seq 1, ack 2, win 115, length 0  ***//webserver回应收到seq为1的请求并也发起一条seq为1的FIN断开请求***  ***//第二次挥手***  ***//tcp状态置为close\_wait状态***  19:28:37.445546 IP 58.246.240.122.63801 > 112.124.45.184.http: Flags [.], ack 2, win 16425, length 0  ***//client应答表示收到seq为1(这里的ack=1是syn+1)的请求***  ***//第三次挥手***  ***//tcp状态置为closed状态***  19:28:42.418048 IP 58.246.240.122.63798 > 112.124.45.184.http: Flags [.], seq 349:350, ack 241, win 16365, length 1  19:28:42.418073 IP 112.124.45.184.http > 58.246.240.122.63798: Flags [.], ack 350, win 123, options [nop,nop,sack 1 {349:350}], length 0  ***//第四次挥手***  ***//server发起一*** |

我们具体来看下ab/webbench访问的过程请求,我们会发现是webserver先发起断开请求,



## loadrunner用户行为测试



Ok至此,webbench/ab 和 loadrunner有用户行为分析完毕,大家可能也有留意到webserver在这过程中断开请求的行为是不一样的.粗心的同学就会有下文的悲剧,满世界在找webserver端有TIMEWAIT的问题~~

# 压测工具小结

1. Webbench/ab默认使用http1.0协议
2. loadrunner默认使用http1.1协议
3. webbench/ab一个client每秒的请求数可达到270左右
4. 专业的压测工具建议选择loadrunner/jmeter
5. Webserver针对ab/webbench/loadrunner/jmeter的响应是完全不一样的
6. 了解并发的概念

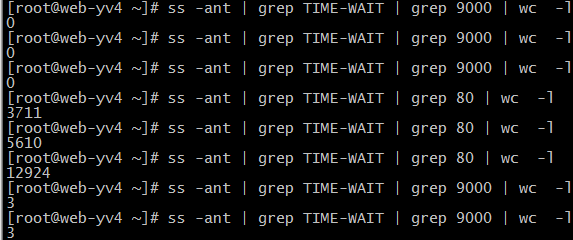
-----------------对于压测工具,了解到如上程度也差不多了~~但要想更深入,下文必看~~

上面的这些介绍,相信对压测工具已经有所了解,但我们压测过程中遇到的问题却远没有解决.

1. webserver为什么刚开始请求就有timewait状态?
2. 一台机器的实际并发竟然有多少?
3. pm.max\_children 和 worker\_processes 究竟多少性能会达到最佳
4. yaf框架相对原生php环境性能空间有多少亏损

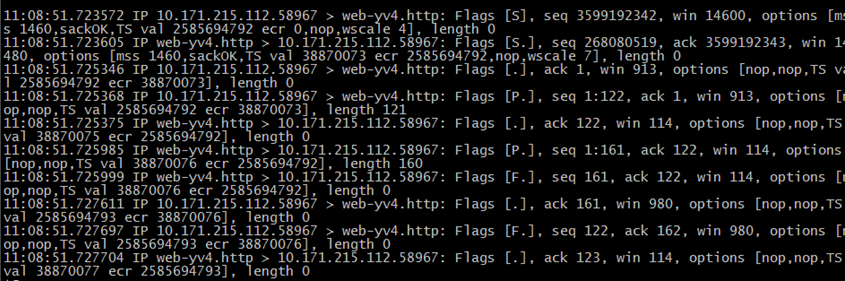
# 终于找到压测刚开始webserver有很多TIME-WAIT的问题了？

大家知道,nginx在针对请求如果开启keepalived\_timeout,在超时范围内是不会主动断开请求连接, 且一般是请求发起方会先发起FIN主动断开连接,那我们服务器上那么多的TIMEWAIT是哪里来的呢？刚开始压测就发现服务器有近1.3w的TIME-WAIT状态.而且这些状态不是后端php的,是80端口的



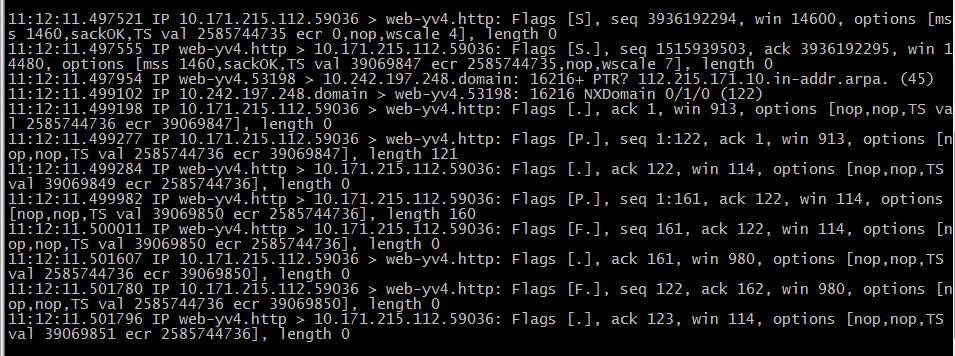
## 验证keepalived\_timeout生效问题

### Keepalived\_time = 0时的数据包请求过程



我们可以看出是webserver主动发起FIN包断开连接.这个是正常的

### Keepalived\_time = 60时的数据包请求过程

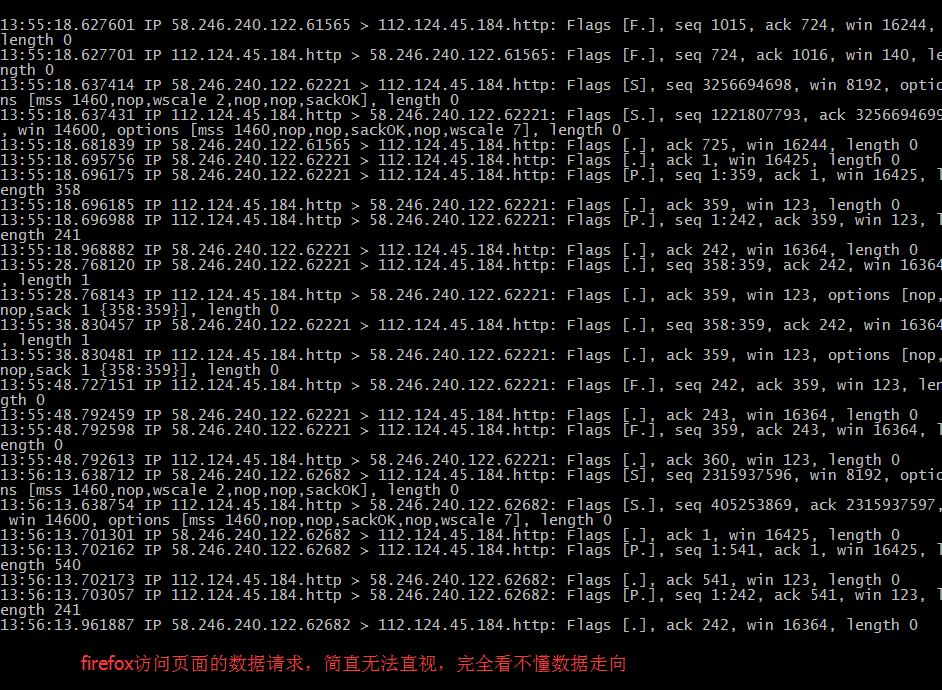


还是webserver主动断开请求~~,这个也太不正常了,难道是keepalived一直没有生效或者是nginx重启失败(安全期间一直用的是restart,没有用reload)？

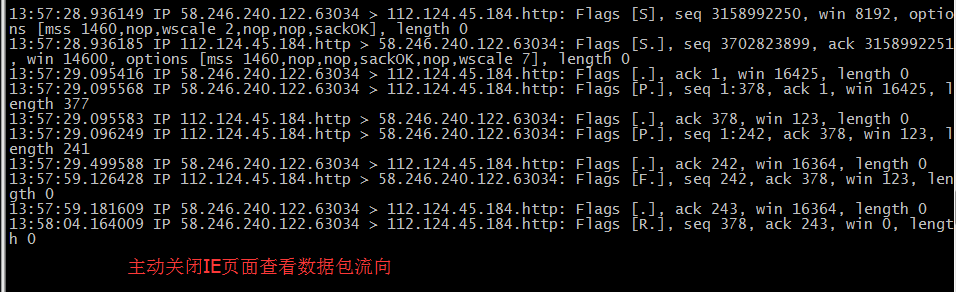
### 手动浏览器访问

#### Firefox访问测试

抓出来的数据一片混乱~~,经测试同学专业建议,改用IE访问测试.



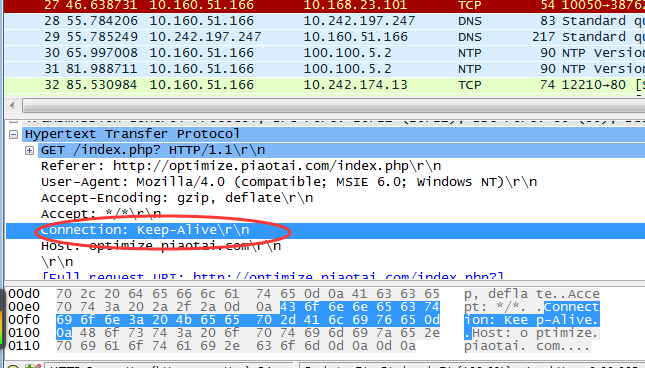
#### IE访问测试





经如上几轮测试,大家应该可以认定keepalived是***生效的***

### 抓包分析ab/webbench请求



Tcp抓取ab/webbench的数据包分析也是有keepalived包头,但keepalived是不生效的,但为什么不生效呢??...

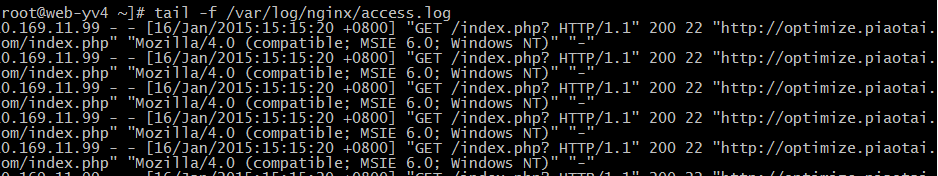
虽然问题根源没有找到,但经过测试,我们最少发现

1. IE访问和FIREFOX访问的区别,IE相对更”干净”些;
2. 不管NGINX keepalived\_timeout = 0还是60,使用ab/webbench的情况下都是webserver先断开服务器链接
3. 正常访问的方式,keepalived\_timeout是生效的

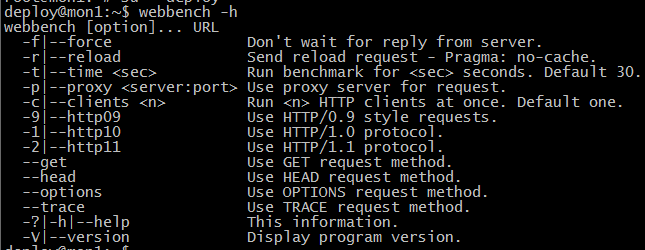
### 内鬼-Http协议

久查无果,也是相当伤脑筋. 大师说:在没有思路的时候多出去走走. 果不其然,放松之后头脑也慢慢变清醒,有一个关键事件一直没有关注---日志---！！一直沉浸在问题数据流层面,同时也是因为压测数据量很大,对nginx日志的关注度略有降低,迅速拎来日志粗看就发现问题了~webbench/ab和loadrunner访问请求是完全不一样的. webbench/ab用的是 http 1.0的协议, loadrunner走的是http1.1的协议~~~~~~





Ab/webbench是否支持http1.1协议呢~ 再看webbench/ab help说明



1. Webbench - 支持;
2. ab - “It does not implement HTTP/1.x fully”;

=== 🡺 专业的压测工具还是用loadrunner吧~~~

#### http1.0/1.1

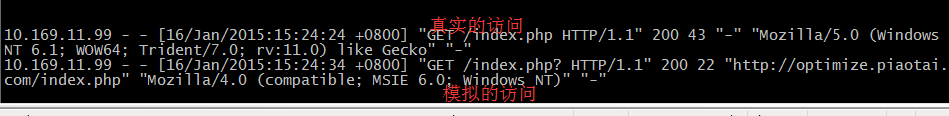
那http1.0和http1.1究竟有什么区别呢,导致服务器的行为有如此大的区别?

|  |
| --- |
| 一个WEB站点每天可能要接收到上百万的用户请求，为了提高系统的效率，HTTP 1.0规定浏览器与服务器只保持短暂的连接，浏览器的每次请求都需要与服务器建立一个TCP连接，服务器完成请求处理后立即断开TCP连接，服务器不跟踪每个客户也不记录过去的请求 |

在tcp/ip 3次握手4次挥手中,主动发送FIN状态要求断开连接方TCP状态会被置为TIMEWAIT状态,被断开方TCP状态会被置为CLOSEWAIT状态.正因为这样的原因,所以每次压测都会出现在keepalived-time超时时长内webserver就会有TIMEWAIT状态.为了验证,我们改用loadrunner来测试.

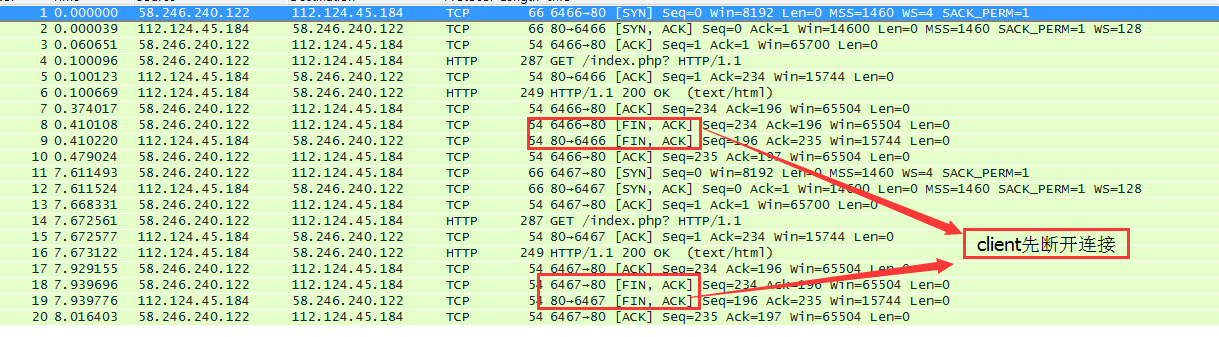
## 确认loadrunner访问原理

1. 先对比nginx访问日志,我们会发现,webbench调用的是服务器本地的浏览器内核来访问甚至可以定义用什么浏览器来访问

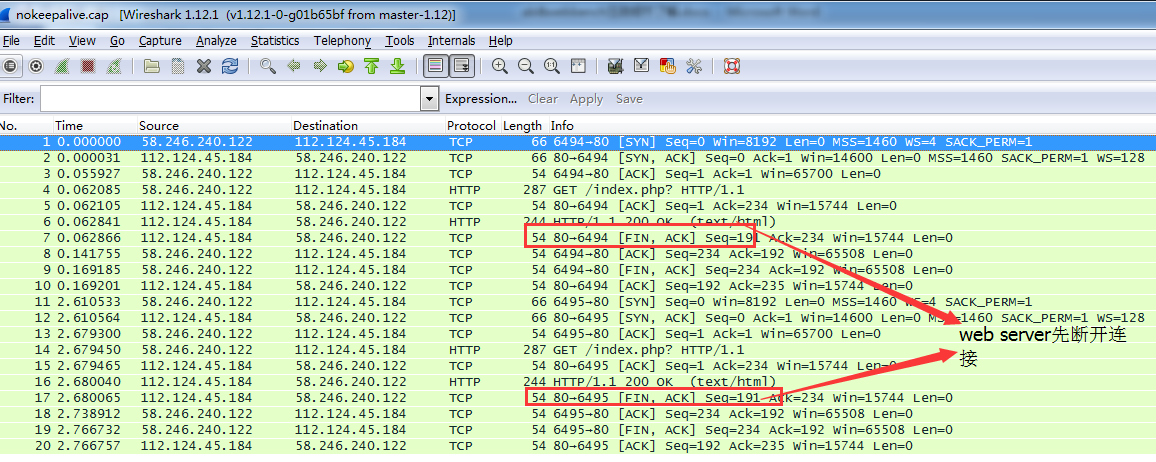


1. Nginx访问日志没有异样的情况下我们用tcpdump来查看一次完整的http请求是否和正常访问的方式一样。

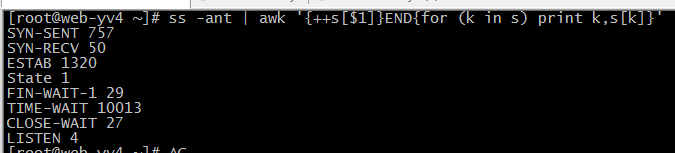
2.1> 设置nginx中keepalived-timeout = 30s,tcpdump抓取loadrunner来看数据流向。我们看到是client先断开连接



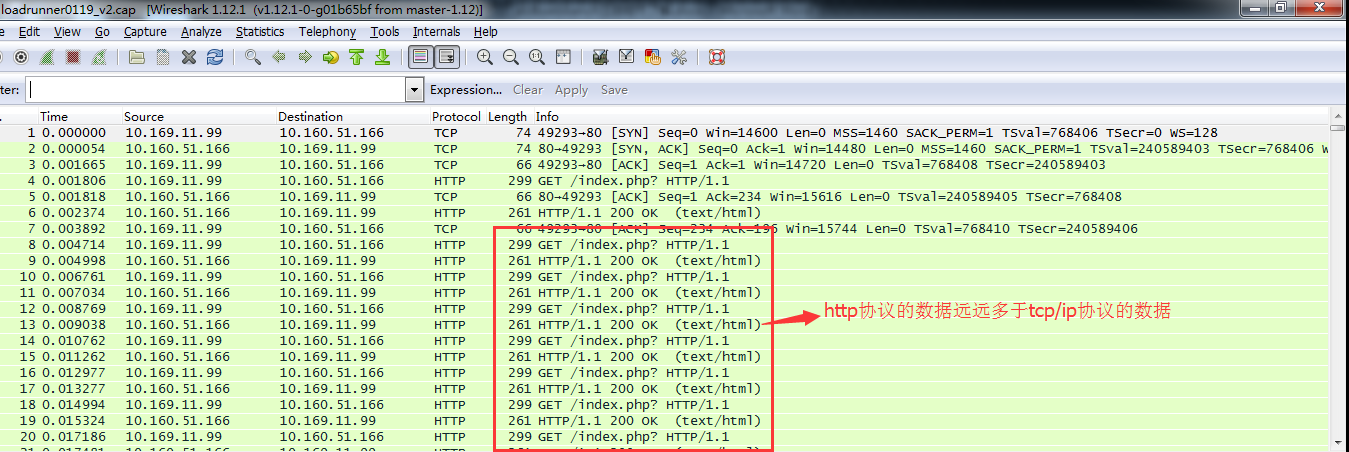
2.2> 如果Keepalived-timeout = 0会不会是webserver先断开连接呢？对的,看下图tcpdump抓到的数据



再来看服务器各状态汇总,在高并发状态下,每种状态都有



### http请求数远多于tcp/ip请求



在一次http压包测中我们发现服务器请求中http请求数远多于tcp/ip协议请求数.这个是什么原因呢？难道loadrunner访问模拟的是浏览器有cache的这种方式,只有第一次访问是全新访问,后面的访问全部都是f5刷新方式的访问?

### f5和ctrl+f5的区别



**从上面这个图我们也可以看到,f5的请求和ctrl+f5的请求是有很大区别的。对服务器的压力也一定相关巨大！并发的概念中是每次请求都是全新请求还是第一次请求是全新请求就可以了？**

## 小结

1. Webbench/ab 因为和loadrunner运行方式不同,所以导致服务器的有大量的TIMEWAIT存在
2. 专业的压测工具请用loadrunner
3. http1.1和http1.0是两种实现方式完全不同的协议
4. loadrunner每次的访问都是全新的请求访问
5. 对并发请求的概念有新的质疑(全新访问还是第一次访问全新的呢？)

如何迅速定位合作伙伴的协作能力