# Apache 2.4.1 vs. Nginx: A comparison under real online applications

**NetEase, Inc.**

Email: wangbin579@gmail.com

2012.12.5

# Table of Contents

# 1 Introduction

Apache is the most popular web server for a long time, while Nginx is the fastest-growing web server now. As Apache 2.4 claims to be on par or faster than Nginx according to Jim Jagielski at ApacheCon 2011, some comparison tests have been performed between Apache and Nginx. However, none of them are under the real online environments.

To compare their real performances, we conducted a stress test on our online ad (advertisement) delivery system with Apache 2.4.1 and Nginx. As one of the most famous IT companies in China (NetEase), its ad delivery server cluster receives about one million ad requests per minute. To guarantee Apache and Nginx be tested under the same environments, we used TCPCopy to copy the online flow to two target test servers (one deployed with Apache proxy and the other with Nginx proxy), while the online system was running. Our results are different from Jim's claim. Nginx is better than Apache 2.4.1 both in its throughput and CPU load in our real online application.

In this document, we will introduce how we set up our test environment, our results and discussion and how we optimize Apache.

# 2 Test procedure with TCPCopy

Before presenting our test procedure, we have to introduce an important tool TCPCopy briefly, which is used to copy online ad requests to Apache and Nginx proxy servers in our test. TCPCopy is an online TCP duplication tool mainly developed by NetEase and also contributed by developers in other companies (such as Taobao). Currently, it has been widely used in companies in China (such as NetEase, Taobao and Sina) for performance testing, smoke testing, regression testing, live testing, etc. As most applications on the internet are based on TCP protocol, TCPCopy could be widely used for applications based on HTTP, memcached, POP3, etc. TCPCopy utilizes online request packets on the online source server and transmits them to a target server in real-time, so the requests sent to the target server are almost the same as the online requests from end-users. Thus, it brings the complexity of online environments to a target server.

TCPCopy consists of two components: TCPCopy client (*tcpcopy*) and TCPCopy server (*intercept*). TCPCopy client should be deployed on the online server and it copies requests and sends them to a target server. TCPCopy server should be deployed on the target server, and it receives requests from the online server, processes them and then sends response headers to the online server. More detailed introduction of TCPCopy could be seen in TCPCopy manual

(https://github.com/downloads/wangbin579/tcpcopy/TCPCopy_0.6.5_Manual.pdf).

In our test, we deployed TCPCopy client (*tcpcopy*) on online ad delivery servers to copy live ad delivery requests to two proxy servers: A and B. Apache 2.4.1 proxy was deployed on test server A, while Nginx 1.1.17 proxy was deployed on test server B. TCPCopy server (*intercept*) was deployed on both two proxy servers to intercept responses and return response headers to online *tcpcopy*. Proxy servers would then transmit these requests to the test ad delivery server C through HTTP proxy. Access log information would be recorded both on one of the online ad delivery servers and the test ad delivery server, which would be used to compare their throughputs. Figure 1 shows the architecture of our test environment.

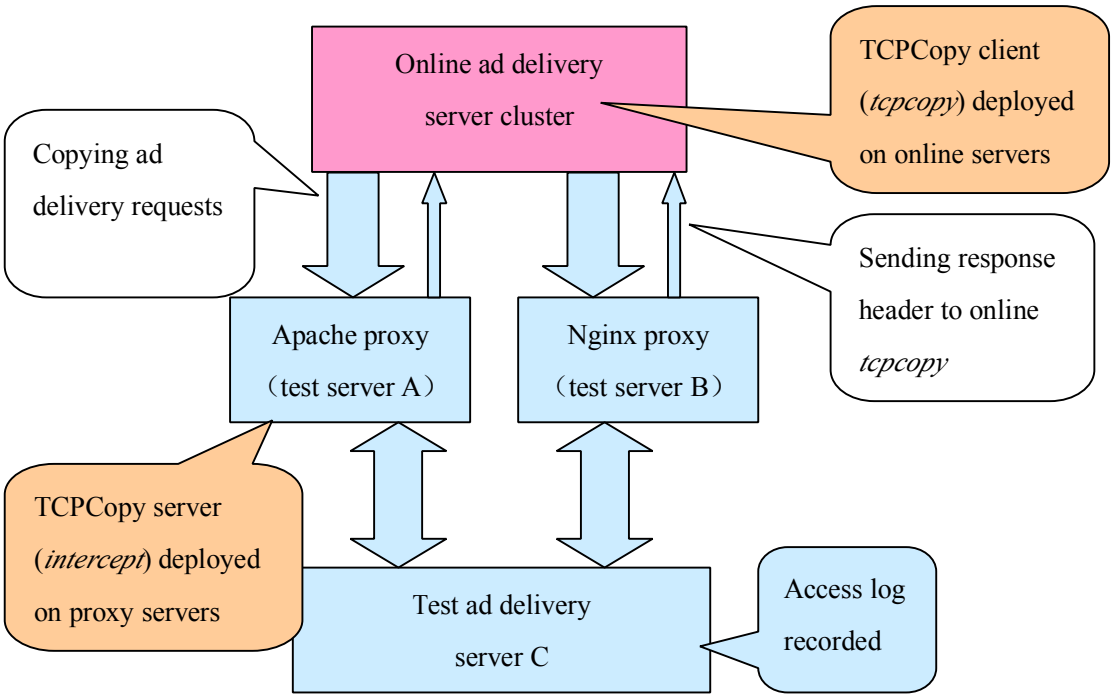


**Figure 1. Architecture of our test environment with TCPCopy**

**We wanted to know whether Nginx could be better than Apache even when Apache is optimized. So we set MaxRequestWorkers of Apache to be 8000, which was expected to perform well in our experiments. Also we used optimized configuration of sysctl on Apache 2.4.1 proxy server. How we optimize Apache can be seen in Section 4. As for Nginx, we utilized the frequently used Nginx configuration and default sysctl configuration on Nginx proxy server.**

More detailed information about the test setup is presented in the following.

**The software and hardware information of two proxy test servers are as follows:**

[test server A]$ uname –a

Linux 2.6.18-164.el5 #1 SMP Tue Aug 18 15:51:48 EDT 2009 x86_64 x86_64 x86_64 GNU/Linux

[test server B]$ uname -a

Linux 2.6.18-164.el5 #1 SMP Tue Aug 18 15:51:48 EDT 2009 x86_64 x86_64 x86_64 GNU/Linux

[test server A]$ grep "model\ name" /proc/cpuinfo

model name : AMD Opteron(tm) Processor 246

model name : AMD Opteron(tm) Processor 246

[test server B]$ grep "model\ name" /proc/cpuinfo

model name : AMD Opteron(tm) Processor 246

model name : AMD Opteron(tm) Processor 246

[test server A]$ free -m

| | total | used | free | shared | buffers | cached |
|---|---|---|---|---|---|---|
| Mem: | 4044016 | 912084 | 3131932 | 0 | 117016 | 140240 |
| -/+ buffers/cache: | | 654828 | 3389188 | | | |
| Swap: | 4192924 | 0 | 4192924 | | | |

[test server B]$ free –m

| | total | used | free | shared | buffers | cached |
|---|---|---|---|---|---|---|
| Mem: | 4044016 | 663196 | 3380820 | 0 | 48376 | 186752 |
| -/+ buffers/cache: | | 428068 | 3615948 | | | |
| Swap: | 4192924 | 0 | 4192924 | | | |

[test server A]$ cat /etc/security/limits.conf

| | | | |
|---|---|---|---|
| * | soft | nofile | 65536 |
| * | hard | nofile | 65536 |

[test server B]$ cat /etc/security/limits.conf

| | | | |
|---|---|---|---|
| * | soft | nofile | 65536 |
| * | hard | nofile | 65536 |

**The configuration of Nginx proxy server is as follows:**

```
worker_processes    4;
worker_rlimit_nofile 65536;
events {
        use    epoll;
        epoll_events    4096;
        worker_connections    8192;
        accept_mutex off;
}
sendfile              on;
```

```
access_log off;
keepalive_timeout 12;
upstream ad_test_system{
        server ad_delivery_test_server_IP_address:18080;
        keepalive 1024;
}
server {
         listen          18080;
         location   /{
                  proxy_http_version 1.1;
                  proxy_set_header Connection keepalive;
                  proxy_pass http:// ad_test_system;
        }
 }
```

**The configuration of Apache 2.4.1 proxy server is as follows:**

```
ListenBackLog 5000
ServerLimit 100
StartServers 100
ThreadLimit 100
ThreadsPerChild 100
MinSpareThreads 500
MaxSpareThreads 1000
MaxRequestWorkers 8000
MaxRequestsPerChild   10000
KeepAliveTimeout 12
Listen 18080

LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_connect_module modules/mod_proxy_connect.so
LoadModule proxy_http_module modules/mod_proxy_http.so
…
#close access_log
#CustomLog "logs/access_log" common
```

ProxyPass / http:// ad_delivery_test_server_IP_address:18080/

EnableSendfile on

**The configuration of sysctl.conf on Apache proxy server is as follows:**

net.core.wmem_max = 16777216

net.core.rmem_max = 16777216

net.core.rmem_default = 65536

net.core.wmem_default = 65536

net.ipv4.tcp_mem = 32768 196608 8388608

net.ipv4.tcp_rmem = 32768 196608 8388608

net.ipv4.tcp_wmem = 32768 196608 8388608

net.ipv4.tcp_fin_timeout = 3

net.ipv4.tcp_syncookies = 0

net.ipv4.ip_local_port_range = 32768 61000

net.ipv4.tcp_max_syn_backlog = 262144

net.ipv4.tcp_max_tw_buckets = 5000

net.core.netdev_max_backlog = 30000

net.ipv4.tcp_no_metrics_save = 1

net.core.somaxconn = 262144

net.ipv4.tcp_max_orphans = 262144

net.ipv4.tcp_synack_retries = 2

net.ipv4.tcp_syn_retries = 2

net.ipv4.tcp_tw_reuse = 1

net.ipv4.tcp_tw_recycle = 1

# 3 Results & Discussion (Nginx vs. optimized Apache)

We compared the performances of Apache proxy (with optimized sysctl and MaxRequestWorkers 8000) and Nginx proxy (default sysctl).

## 3.1.Results

When we copied live flow from **one online server**, the results can be seen in Figure 2.

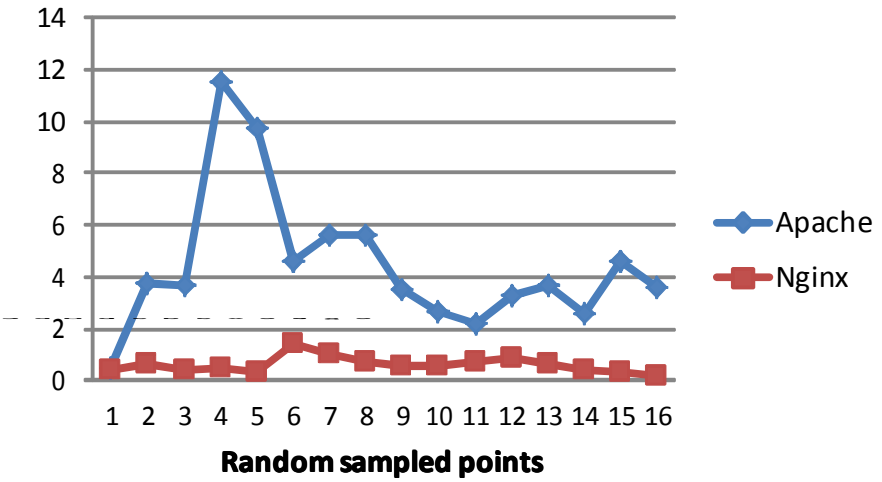


**Figure 2 CPU load of Apache and Nginx with live flow from one online server**

Requests successfully transmitted by Apache and Nginx are as follows.

[online ad delivery server]# grep '27/Mar/2012:10:21' access.log |wc -l：100507

[test ad delivery server]# grep '27/Mar/2012:10:21' access.log |grep 'Apache server IP address'|wc -l：100205

[test ad delivery server]# grep '27/Mar/2012:10:21' access.log | grep 'Nginx server IP address'|wc -l：100263

**We can see that the throughputs of Apache and Nginx are similar with live flow from one online server, although Nginx performs a little better. However, as for the CPU load average, Nginx is much better.**

**While this test uses live flow from one online server, it demonstrates the performances of Apache and Nginx under real online applications.**

We increased the stress by copying live flow from **two online servers**. The results can be seen in Figure 3.

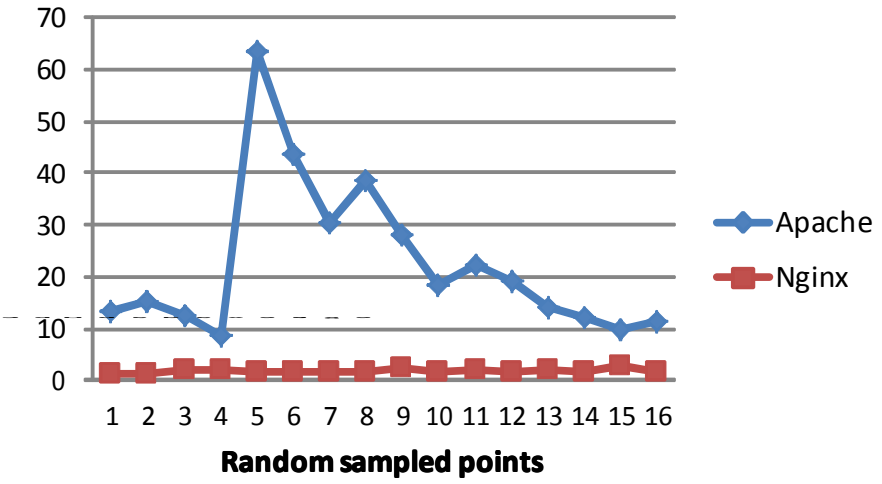


**Figure 3. CPU load of Apache and Nginx with live flow from two online servers**

Requests successfully transmitted by Apache and Nginx are as follows.

[online ad delivery server]# grep '27/Mar/2012:10:40' access.log |wc -l：102601

[test ad delivery server]# grep '27/Mar/2012:10: 40' access.log |grep 'Apache server IP address'|wc -l：72204

[test ad delivery server]# grep '27/Mar/2012:10: 40' access.log | grep 'Nginx server IP address '|wc -l：203438

**Under the same online stress from two online servers, the request loss rate becomes high for Apache proxy, while it is still low for Nginx proxy. Meanwhile, the CPU load of Nginx is better than that of Apache.**

**As the request loss rate is already high for Apache with live flow from two online servers, we did not continue our stress test with more online servers.**

### 3.2.Discussion

Our results show that Nginx is better than Apache 2.4.1. There may be two reasons. On the one hand, the memory Apache occupies is too large, and this probably leads to the frequent exchange of swap space and memory. On the other hand, Apache is based on process model or thread model, while Nginx is based on asynchronous event model. When the stress is large, the schedule cost of Apache may be large and thus less time may be spent on the processing of requests.

Our test is performed under real online applications and we believe the results would be valuable, though the test could be improved in the following aspects.

First, only one online server records access.log for maintenance reasons. Although LVS (*Linux Virtual Server*) is adopted for load balance, we can't get the exact request number. The request number and request loss rate may be not exactly accurate when more than one online server is adopted.

Second, currently only one test ad delivery server is used in the test, a better choice would be using two test ad delivery servers: one to process requests from Nginx proxy, the other to process requests from Apache proxy.

Third, our tests with one online server and two online servers are performed at different times. As the online stress may be different at different times, the request number of two online servers may not be exactly double that of one server.

# 4 How we optimize Apache?

In this section, we evaluated how parameter optimization would influence Apache 2.4.1. We first examined MaxRequestWorkers, and then tested sysctl.

## 4.1. MaxRequestWorkers 5000

We modified the configuration file of Apache as follows.

ListenBackLog 5000

ServerLimit 50

StartServers 50

ThreadLimit 100

ThreadsPerChild 100

MinSpareThreads 500

MaxSpareThreads 1000

MaxRequestWorkers 5000

MaxRequestsPerChild 10000

KeepAliveTimeout 12

Listen 18080

To get the throughput, CPU load and request loss rate of Apache under different stress situations, we stress tested our apache proxy server by copying live flow from different numbers of online servers (1 to 3) with TCPCopy.

The results can be seen in Figure 4. The throughput increases when the request number increases at the beginning, and decreases afterward. CPU load has a similar trend. However, the request loss rate increases as the stress increases.

The requests successfully transmitted by Apache proxy per second for one server, two servers, and three servers are: 1524, 2702, and 2190. The average CPU loads with live flow from one online server, two servers, and three servers are: 2.336, 3.66, and 3.455. The request loss rates for one server, two servers, and three servers are: 0.35%, 4.2%, and 47%. When requests from three online servers arrive, only half requests are transmitted successfully by Apache proxy.

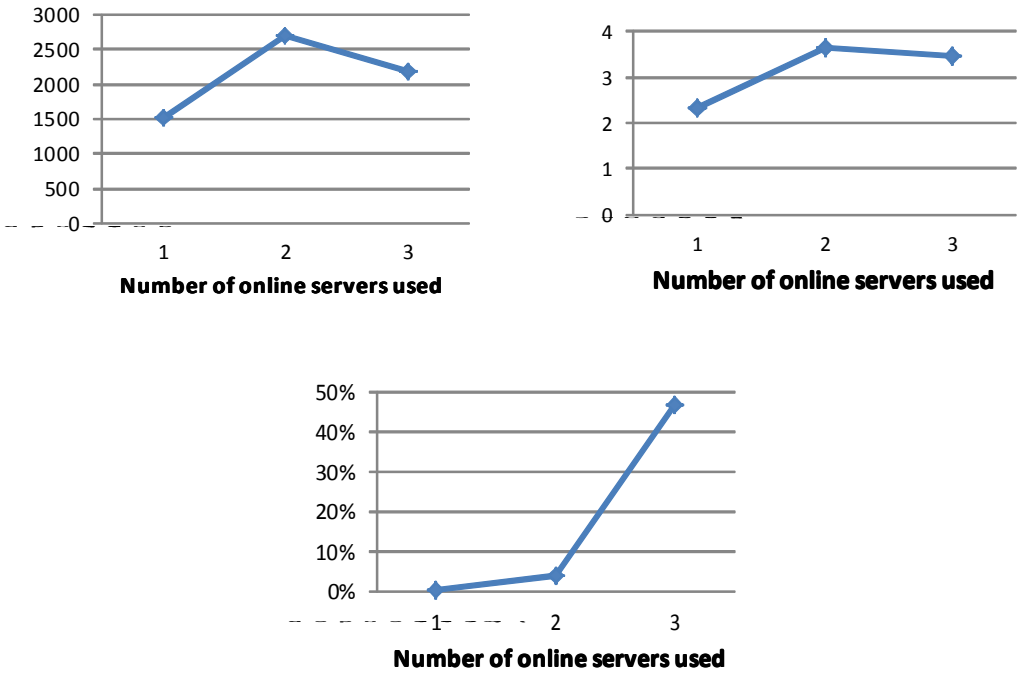


**Figure 4. Throughput, CPU load and request loss rate of Apache proxy with live flow from different numbers of online servers when MaxRequestWorkers is set to 5000.**

With live flow from one online server, Apache error_log prompted the following warning.

scoreboard is full, not at MaxRequestWorkers

With live flow from two online servers, Apache reports the following warning.

server reached MaxRequestWorkers setting, consider raising the MaxRequestWorkers setting

With live flow from three online servers, Apache also report the above warning. And the state of TCP is as follows.

```
netstat -n | awk '/^tcp/ {++state[$NF]} END {for(key in state) print key,"\t",state[key]}'
```

| | |
|---|---|
| TIME_WAIT | 3765 |
| CLOSE_WAIT | 1288 |
| FIN_WAIT1 | 1946 |
| FIN_WAIT2 | 301 |
| ESTABLISHED | 10244 |
| SYN_RECV | 45732 |
| CLOSING | 580 |
| LAST_ACK | 1014 |

Thus, **Apache would not break down when dealing with more requests than MaxRequestWorkers can handle. With a proper MaxRequestWorkers value, Apache could be protected from large live flow.**

### 4.2. MaxRequestWorkers 8000

We then modified MaxRequestWorkers of Apache to be 8000. The results can be seen in Figure 5. Similar to MaxRequestWorkers 5000, **the throughputs and CPU loads of MaxRequestWorkers 8000 both decrease at some point. However, the throughput of MaxRequestWorkers 8000 is larger.**

The requests successfully transmitted by Apache proxy per second for one server, two servers, three servers, four servers and five servers are: 1208, 2319, 3363, 4065 and 3547. The average CPU loads with live flow from one server, two servers, three servers, four servers and five servers are: 1.11, 4.62, 12.12, 23.33 and 18.66. The request loss rates for one server, two servers, three servers, four servers and five servers are: 0.57%, 0%, 0.7%, 0.87% and 30%.

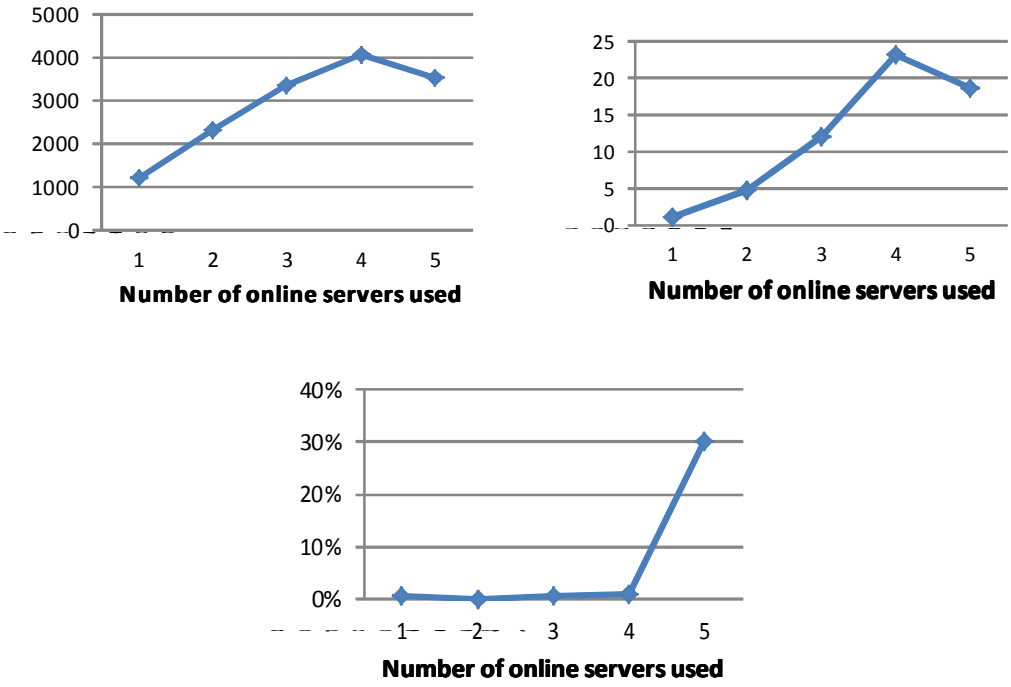


**Figure 5. Throughput, CPU load, and request loss rate of Apache proxy with live flow from different numbers of online servers when MaxRequestWorkers is set to 8000.**

With live flow from **one** online server, Apache error_log did not report any warnings.

With live flow from **two** online servers, Apache error_log did not report any warnings.

With live flow from **three** online servers, Apache still didn't report any warnings. While the throughput increases 50%, the CPU load increases 162%. **Thus, when we increase MaxRequestWorkers, the throughput of Apache could be increased, while the CPU load would deteriorate rapidly.**

The state of TCP is as follows.

```
netstat -n | awk '/^tcp/ {++state[$NF]} END {for(key in state) print key,"\t",state[key]}'
    TIME_WAIT          4335
    CLOSE_WAIT         1383
    FIN_WAIT1          43
    FIN_WAIT2          32
    ESTABLISHED        15585
    SYN_RECV           1699
    CLOSING            247
    LAST_ACK           284
```

With live flow from **four** online servers, Apache began to report warnings. It suggests increasing MaxRequestWorkers. **The Apache performance was even worse with average CPU load 23.33.**

```
server reached MaxRequestWorkers setting, consider raising the MaxRequestWorkers setting
scoreboard is full, not at MaxRequestWorkers
```

The state of TCP is as follows.

```
netstat -n | awk '/^tcp/ {++state[$NF]} END {for(key in state) print key,"\t",state[key]}'
    TIME_WAIT          3846
    CLOSE_WAIT         1778
    FIN_WAIT1          204
    FIN_WAIT2          125
    ESTABLISHED        17113
    SYN_RECV           1986
    CLOSING            564
    LAST_ACK           1480
```

With live flow from **five** online servers, **t**he request loss rate becomes high, and the throughput is lower than before. CPU load also decreases. Thus, **although request number increases rapidly, Apache does not crash and still provides service to part of users.**

The state of TCP is as follows.

```
    netstat -n | awk '/^tcp/ {++state[$NF]} END {for(key in state) print key,"\t",state[key]}'
    TIME_WAIT          3833
    CLOSE_WAIT         1810
    FIN_WAIT1          4973
```

| | |
|---|---|
| SYN_SENT | 1 |
| FIN_WAIT2 | 643 |
| ESTABLISHED | 10122 |
| SYN_RECV | 29742 |
| CLOSING | 1002 |
| LAST_ACK | 2119 |

### 4.3.sysctl

The above test is based on optimized sysctl setting. When default sysctl value is adopted, How Apache behaves?

The following figures give the comparison of default sysctl and optimized sysctl. Obviously, Apache performs better when using optimized sysctl.

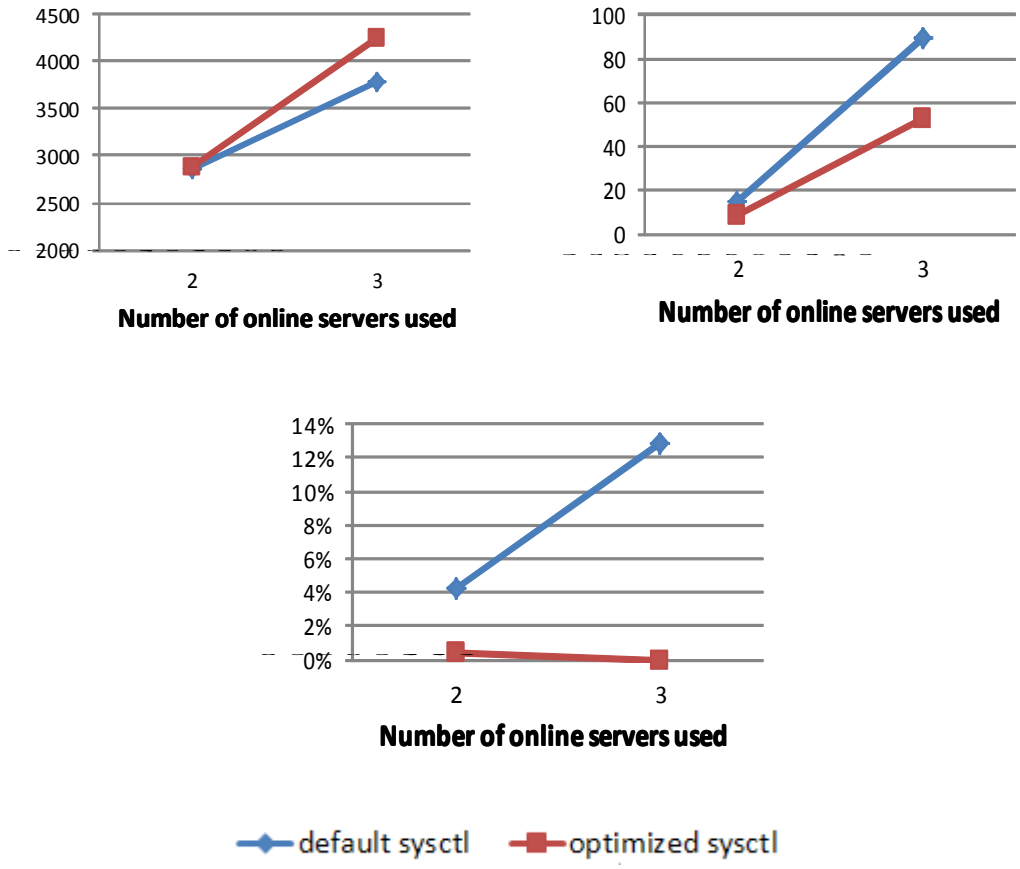


**Figure 6. Throughput, CPU load, and request loss rate of Apache proxy with optimized sysctl and default sysctl.**