软件测试上机报告

第三次上机作业

学　　院___智能与计算学部__
专　　业___软件工程_____
队伍名称___Stayhome team _
成　　员___王兆盟、陶柏安__
成员学号___3017218072、3017218070__
年　　级___2017 级_ _ __
班　　级___软件工程一班____

1、 Experiment Requirement

1.1 Overall Goal：

Install the virtual machine and a set of LAMP (Linux + Apache + MySQL + PHP) system to be tested, recommend ECShop (http://www.ecshop.com), conduct jmeter stress test based on this, and get jmeter test report after the test, and get CIMN (CPU, IO, memory and network) performance of Linux server according to sysstat.

1.2 Specific Goal:

1) The screenshot of the result of “top” command in Linux
2) The screenshot of visiting of B/S system
3) The screenshot of jmeter Testplan
4) Beanshell code
5) The Aggregate Report Result after running jmeter
6) The screenshot of the performance of server after running jmeter
7) If any error occurs, briefly descript the error and analysis it.

2、 Experiment Process

2.1 The division of group labor:

We have two group members in our group, Wang Zhaomeng and Tao Boan.

Wang is responsible for the installation of the LAMP system and develop a B/S system on it, which designed to be the target testing system and monitor the system during the testing process and do the statistics.

On the other hand, Tao is responsible for the testing process, who conducted the jmeter test plan of the stress test and developed a Beanshell script and ultimately drew the Aggregate Report Result of the testing.

2.2 The B/S system to be tested:

We rented a cloud server of Tecent (https://cloud.tencent.com/) which was configured as 1 core CPU, 2GB RAM, 1Mbps bandwidth and with Linux Ubuntu16.04 as the operating system.

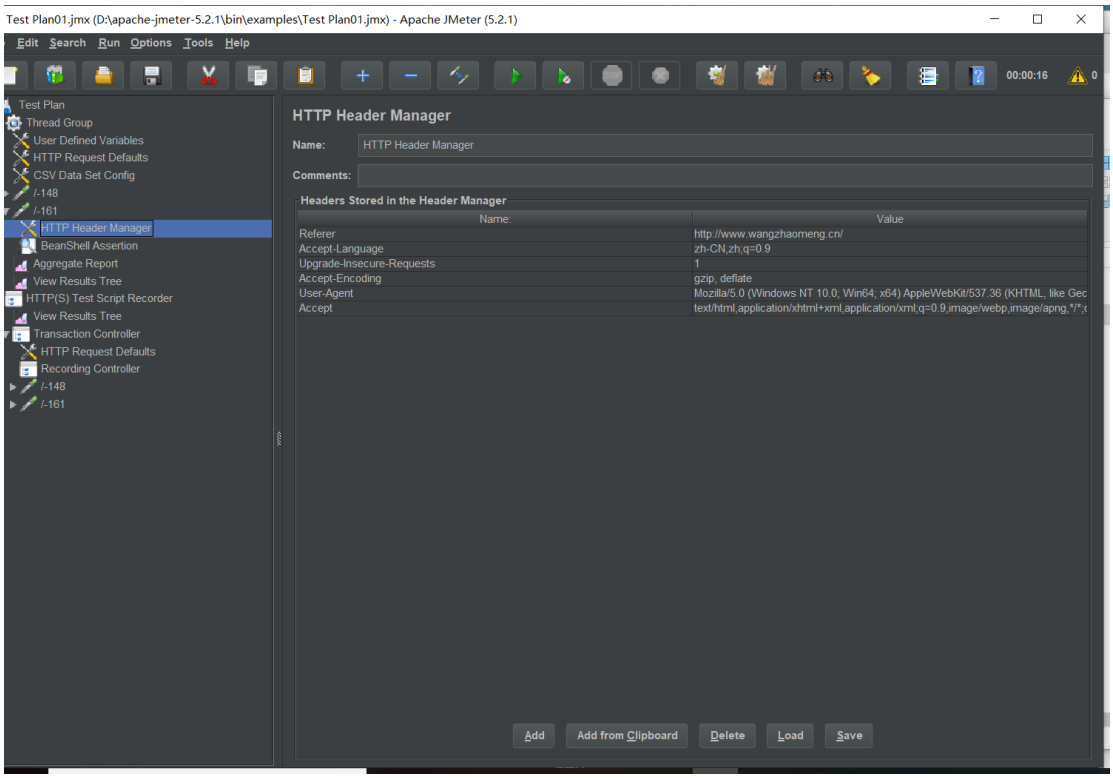
Then we installed the Apache2 as the Web Server, PHP 7.2 as the programming language of the Web application and the MySQL as the database of the backend.

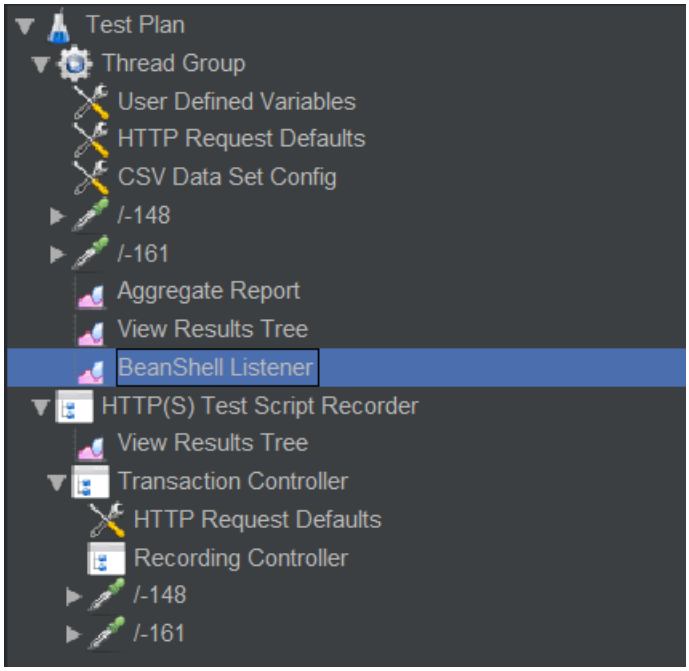
We adopted the WordPress which provides an easy and convenient way to build a blog system based on PHP by offering the framework of the Web application. The blog is a typical browser/server system totally satisfied the requirement of the experiment. The URL of the blog is shown below: http://www.wangzhaomeng.cn/

The screenshot is shown below as well:

2.3 The test process by using jmeter:

Firstly, in accordance with the course on imooc.com, we designed the thread group added the aggregate report and source tree and recorded the testing process by using HTTP(S) Test Script Recorder. The entire test plan is shown below:





Combined the CSV Data Set Config we wrote the Beanscript to test if the result shown on the Web consistent with the the information in the CSV. There is a search bar at the bottom of the blog and it supposes to display all related information on the blog after clicking the search button. So we decided to test the search bar via Beanscript assertion.

The search bar is shown below:

The result after searching:

The Beanscript is shown below:

Script (variables: ctx vars props sampleEvent sampleResult log prev)

Script:

```
java.util.regex.Pattern p=java.util.regex.Pattern.compile("s={\\d+}");
java.util.regex.Matcher m=p.matcher(bsh.args[0]);
boolean found = m.find();
if(found){
    if(!m.group(1).equals(bsh.args[0]))
        Failure = true;
        FailureMessage = m.group(1) + "<>" + bsh.args[1];
    }
}
else Failure = true;
```

Then we conducted the stress test in order to test the performance of the Web application under high currency situation. We configured the parameter and executed 5*10、50*20 test plan. The aggregate report are shown below:

5*10:

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received K...	Sent KB/sec
/-148	50	762	559	1463	1548	2746	113	2746	0.00%	3.2/sec	68.68	1.48
/-161	50	661	519	1199	1271	2431	121	2431	0.00%	3.2/sec	61.18	1.57
TOTAL	100	712	555	1381	1676	2431	113	2746	0.00%	6.3/sec	128.47	3.02

50*20:

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB...	Sent KB/sec
/-148	1000	9017	3180	13420	40053	114721	82	271699	2.00%	2.2/sec	46.53	1.01
/-161	1000	7460	2720	10786	25843	134307	86	216160	1.40%	2.2/sec	41.67	1.08
TOTAL	2000	8239	2940	12056	33271	133319	82	271699	1.70%	4.4/sec	88.01	2.08

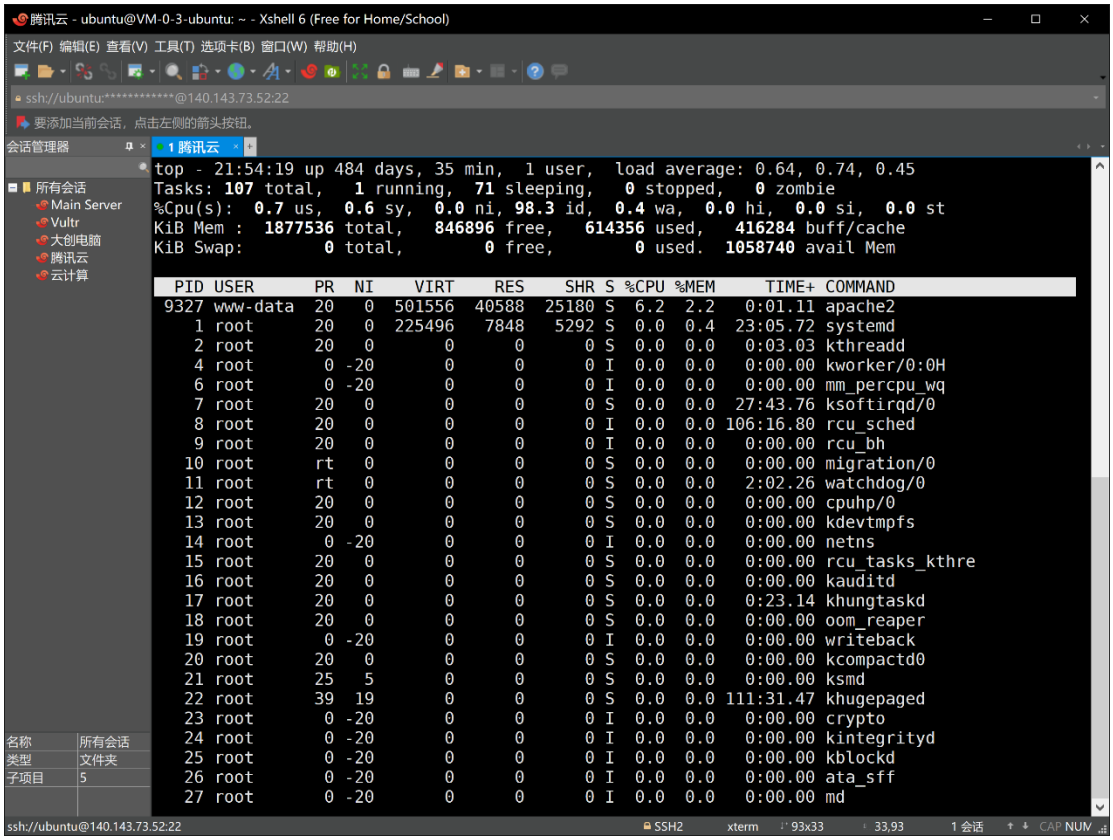
3、Experiment Result

This section mainly demonstrate the performance of the server during the stress test process.

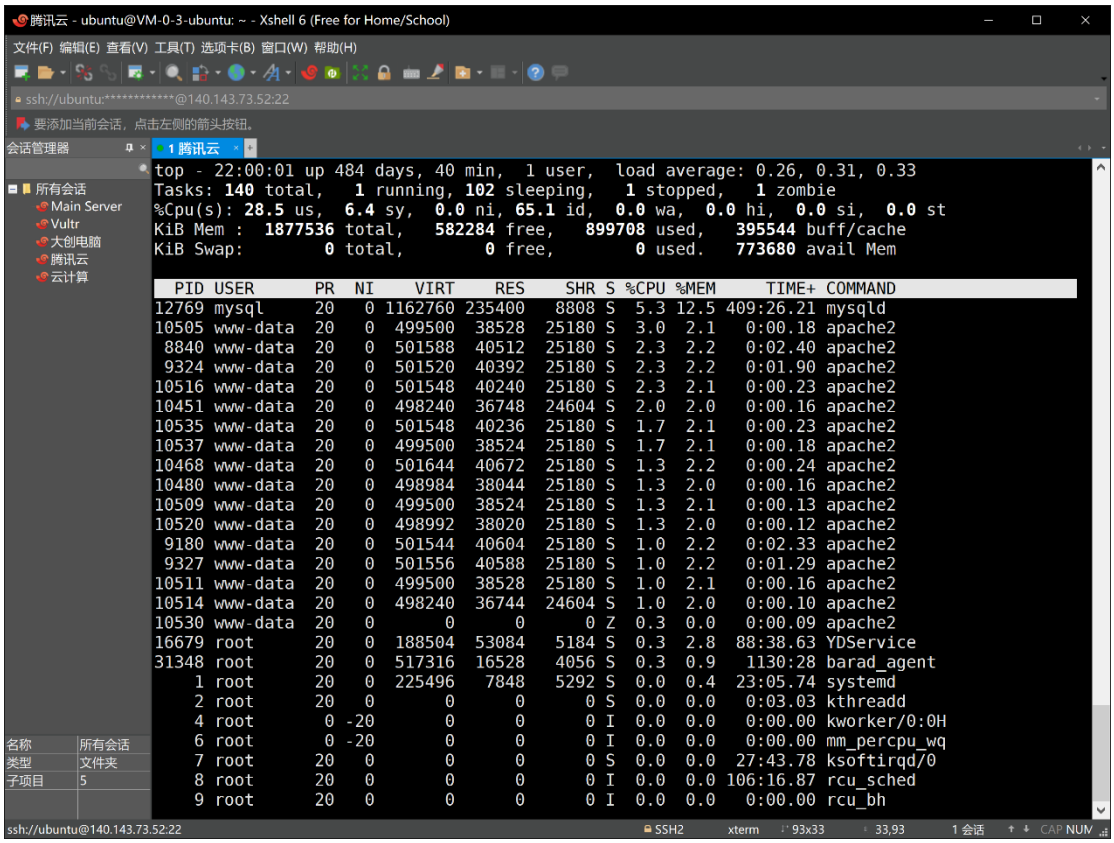
The “top” command help to show the usage of the system source like CPU and memory shared by each task.

The result are shown below:

before the test:



after the test:



Then we install sysstat to monitor the performance of the server.

By using “sar -u 1 5” we can analysis the usage of CPU each second in totally 5 seconds.

```
ubuntu@VM-0-3-ubuntu:~$ sar -u 1 5
Linux 4.15.0-29-generic (VM-0-3-ubuntu)         04/01/2020      _x86_64_        (1 CPU)

09:55:22 PM     CPU     %user     %nice   %system   %iowait    %steal     %idle
09:55:23 PM     all      0.99      0.00      0.99      0.00      0.00     98.02
09:55:24 PM     all      0.00      0.00      1.96      0.00      0.00     98.04
09:55:25 PM     all      0.00      0.00      0.00      0.00      0.00    100.00
09:55:26 PM     all      0.00      0.00      1.01      0.00      0.00     98.99
09:55:27 PM     all      0.00      0.00      0.00      1.02      0.00     98.98
Average:        all      0.20      0.00      0.80      0.20      0.00     98.80
```

By using “sar -r 1 5” we can analysis the usage of memory each second in totally 5 seconds.

```
ubuntu@VM-0-3-ubuntu:~$ sar -r 1 5
Linux 4.15.0-29-generic (VM-0-3-ubuntu)         04/01/2020      _x86_64_        (1 CPU)

10:01:37 PM kbmemfree   kbavail kbmemused  %memused kbbuffers  kbcached  kbcommit   %commit
kbactive   kbinact   kbdirty
10:01:38 PM    771420    960356   1106116     58.91     58592    245228   1901288    101.27
  776876    129080       160
10:01:39 PM    770192    959240   1107344     58.98     58592    245240   1901300    101.27
  777372    128640       168
10:01:40 PM    770228    959300   1107308     58.98     58592    245244   1901300    101.27
  777892    128196       172
10:01:41 PM    781216    970288   1096320     58.39     58592    245244   1884648    100.38
  767908    128196       172
10:01:42 PM    781220    970296   1096316     58.39     58592    245248   1885680    100.43
  768160    127956       172
Average:       774855    963896   1102681     58.73     58592    245241   1894843    100.92
  773642    128414       169
```

By using “sar -b 1 5” we can analysis the I/O throughput each second in totally 5 seconds.

```
ubuntu@VM-0-3-ubuntu:~$ sar -b 1 5
Linux 4.15.0-29-generic (VM-0-3-ubuntu)         04/01/2020      _x86_64_        (1 CPU)

10:01:17 PM       tps      rtps      wtps   bread/s   bwrtn/s
10:01:18 PM      1.98      0.00      1.98      0.00    118.81
10:01:19 PM      6.12      6.12      0.00     48.98      0.00
10:01:20 PM      0.00      0.00      0.00      0.00      0.00
10:01:21 PM     23.47      2.04     21.43    171.43    195.92
10:01:22 PM      0.00      0.00      0.00      0.00      0.00
Average:         6.28      1.62      4.66     43.72     63.16
```

By using “sar -n DEV 1 5” we can analysis the network throughput by Ethernet adapter each second in totally 5 seconds.

```
ubuntu@VM-0-3-ubuntu:~$ sar -n DEV 1 5
Linux 4.15.0-29-generic (VM-0-3-ubuntu)         04/01/2020      _x86_64_        (1 CPU)

09:56:26 PM     IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
 %ifutil
09:56:27 PM vethcd7c6fc      0.00      0.00      0.00      0.00      0.00      0.00      0.00
      0.00
09:56:27 PM        lo      0.00      0.00      0.00      0.00      0.00      0.00      0.00
      0.00
09:56:27 PM      eth0      5.05      3.03      0.42      0.39      0.00      0.00      0.00
      0.00
09:56:27 PM   docker0      0.00      0.00      0.00      0.00      0.00      0.00      0.00
      0.00

09:56:27 PM     IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
 %ifutil
09:56:28 PM vethcd7c6fc      0.00      0.00      0.00      0.00      0.00      0.00      0.00
      0.00
09:56:28 PM        lo      0.00      0.00      0.00      0.00      0.00      0.00      0.00
      0.00
09:56:28 PM      eth0      7.07      8.08      0.51      1.86      0.00      0.00      0.00
      0.00
09:56:28 PM   docker0      0.00      0.00      0.00      0.00      0.00      0.00      0.00
      0.00
```

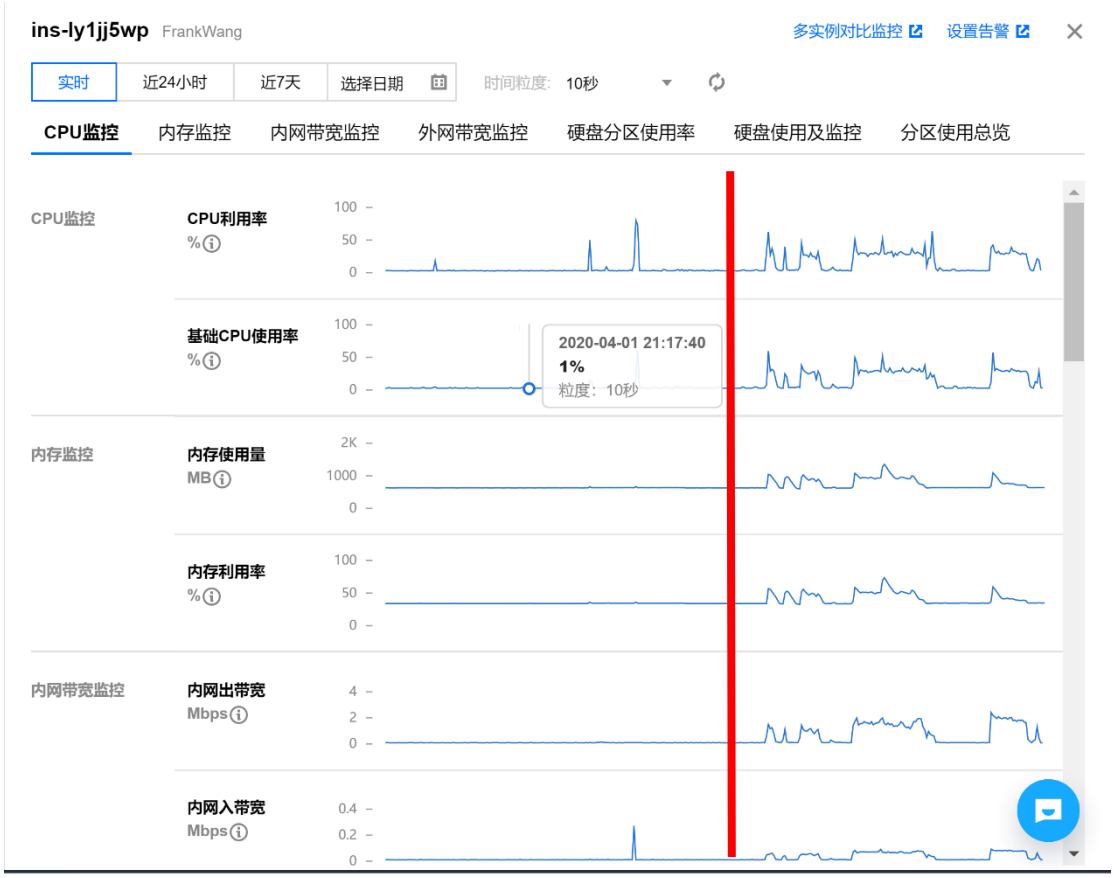
```
09:56:28 PM     IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
 %ifutil
09:56:29 PM vethcd7c6fc      0.00      0.00      0.00      0.00      0.00      0.00      0.00
      0.00
09:56:29 PM        lo      0.00      0.00      0.00      0.00      0.00      0.00      0.00
    0.00
09:56:29 PM      eth0      9.09      9.09      0.61      2.11      0.00      0.00      0.00
    0.00
09:56:29 PM   docker0      0.00      0.00      0.00      0.00      0.00      0.00      0.00
    0.00

09:56:29 PM     IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
 %ifutil
09:56:30 PM vethcd7c6fc      0.00      0.00      0.00      0.00      0.00      0.00      0.00
      0.00
09:56:30 PM        lo      0.00      0.00      0.00      0.00      0.00      0.00      0.00
    0.00
09:56:30 PM      eth0      3.00      5.00      0.15      1.24      0.00      0.00      0.00
    0.00
09:56:30 PM   docker0      0.00      0.00      0.00      0.00      0.00      0.00      0.00
    0.00

09:56:30 PM     IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
 %ifutil
09:56:31 PM vethcd7c6fc      0.00      0.00      0.00      0.00      0.00      0.00      0.00
      0.00
09:56:31 PM        lo      0.00      0.00      0.00      0.00      0.00      0.00      0.00
    0.00
09:56:31 PM      eth0      6.06     14.14      0.32      2.52      0.00      0.00      0.00
    0.00
09:56:31 PM   docker0      0.00      0.00      0.00      0.00      0.00      0.00      0.00
    0.00

Average:        IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
 %ifutil
Average:    vethcd7c6fc      0.00      0.00      0.00      0.00      0.00      0.00      0.00
      0.00
Average:           lo      0.00      0.00      0.00      0.00      0.00      0.00      0.00
    0.00
Average:         eth0      6.05      7.86      0.40      1.62      0.00      0.00      0.00
    0.00
Average:      docker0      0.00      0.00      0.00      0.00      0.00      0.00      0.00
    0.00
```

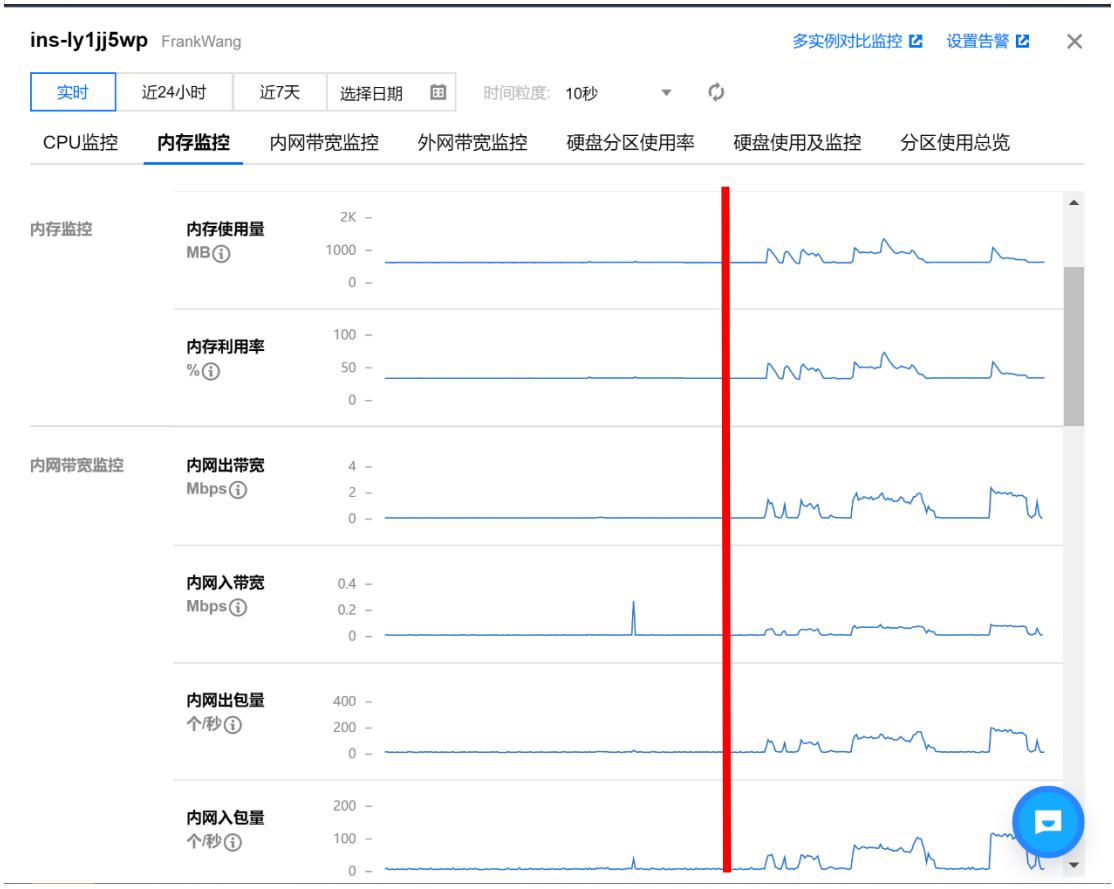
By using “sar -n SOCKET 1 5” we can analysis the network throughput by Socket of IPV4 each second in totally 5 seconds.

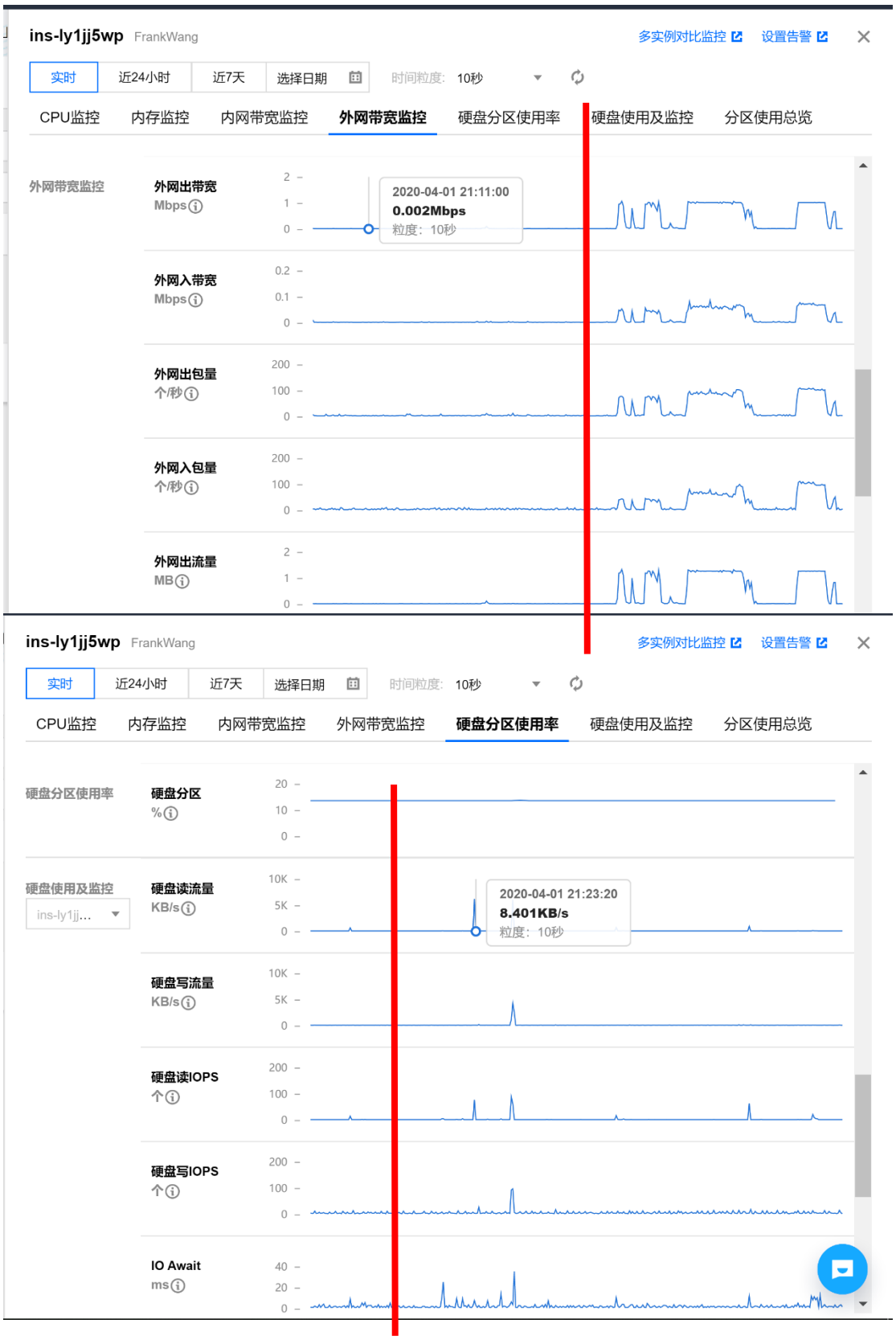
```
ubuntu@VM-0-3-ubuntu:~$ sar -n SOCK 1 5
Linux 4.15.0-29-generic (VM-0-3-ubuntu)         04/01/2020      _x86_64_        (1 CPU)

10:03:00 PM    totsck    tcpsck    udpsck    rawsck   ip-frag    tcp-tw
10:03:01 PM       198        13         6         0         0        25
10:03:02 PM       202        14         6         0         0        25
10:03:03 PM       198        13         6         0         0        24
10:03:04 PM       198        13         6         0         0        24
10:03:05 PM       198        13         6         0         0        23
Average:          199        13         6         0         0        24
```

In order to figured it out more directly we visited the dashboard of the server provide by Tecent. In the dashboard, it presents us a visualization of the performance of the server by drawing the statistic graph that, we believe, is very straightforward to estimate.

The screenshot are shown below(the start point are shown as the red line added in the graph):


ins-ly1jj5wp FrankWang
多实例对比监控
设置告警
实时
近24小时
近7天
选择日期
时间粒度: 10秒
CPU监控
内存监控
内网带宽监控
外网带宽监控
硬盘分区使用率
硬盘使用及监控
分区使用总览
CPU监控
CPU利用率
%
基础CPU使用率
%
2020-04-01 21:17:40
1%
粒度: 10秒
内存监控
内存使用量
MB
内存利用率
%
内网带宽监控
内网出带宽
Mbps
内网入带宽
Mbps


ins-ly1jj5wp FrankWang
多实例对比监控
设置告警
实时
近24小时
近7天
选择日期
时间粒度: 10秒
CPU监控
内存监控
内网带宽监控
外网带宽监控
硬盘分区使用率
硬盘使用及监控
分区使用总览
内存监控
内存使用量
MB
内存利用率
%
内网带宽监控
内网出带宽
Mbps
内网入带宽
Mbps
内网出包量
个/秒
内网入包量
个/秒



From those above we can see the dramatic increase of source usage after the stress test begin.

In the end, we can declare that, fortunately, we did not encounter any unexpected error in the experiment.