

# Lab4: 性能测试 测试报告

---

•

## 1. 测试结果

### 1.1 测试用例1：登录 + 主页浏览详情

#### 1.1.1 测试流程

#### 1.1.2 测试结果分析

### 1.2 注册 + 登录

#### 1.2.1 测试详细流程

#### 1.2.2 测试结果与分析

### 1.3 登录 + 按类别查找商品

#### 1.3.1 测试流程

#### 1.3.2 测试结果分析

### 1.4 登录 + 加入购物车 + 下单

#### 1.4.1 测试流程

#### 1.4.2 测试结果分析

## 2. 网站的性能结果分析

### 2.1 架构

### 2.2 部署平台

### 2.3 资源文件大小

## 1. 测试结果

---

### 1.1 测试用例1：登录 + 主页浏览详情

#### 1.1.1 测试流程

1. 提前注册好多个用户，用于后面直接登录。这些用户的密码相等方便后面的参数化操作。

2. 使用 *Virtual User Generator*，对整个服务流程进行录制。

- 登录使用先前注册好的其中一个用户
- 因为使用Chrome录制，可能会访问google的一些相关网页，在录制的时候，需要开启代理。
- 录制选项里面，需要调整*Recording options*里面，基于*URL-based script*，否则在replay的时候会报错无法运行。
- 录制完成后，仅include和35.221.252.58相关的操作，exclude掉和谷歌相关的操作

录制好之后可以看到生成的脚本：

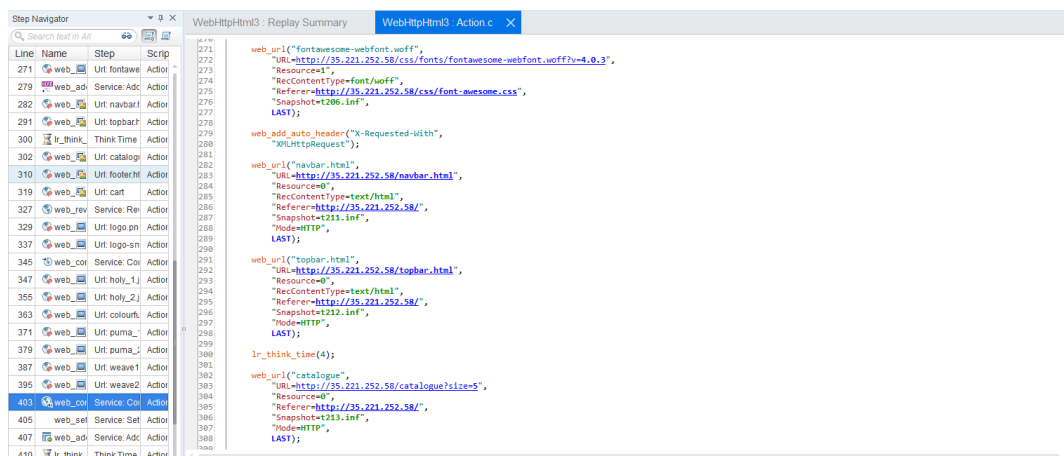


图1.1: 录制好的脚本

3. 本流程将全操作设为一个transaction，因此在前后加入 `lr_start_transaction` 和 `lr_end_transaction` 两部分脚本，给transaction起名为 `login and browse`。

(实际上在Controller运行的时候会自动把全文设为一个transaction，这里我自己做一个简单包装)

4. **参数化**：可以在脚本的最前端看到一个 `web_set_user` 的语句，这一部分是和登录相关的。此时，我们将用户名改为之前设计的多个用户的用户名。

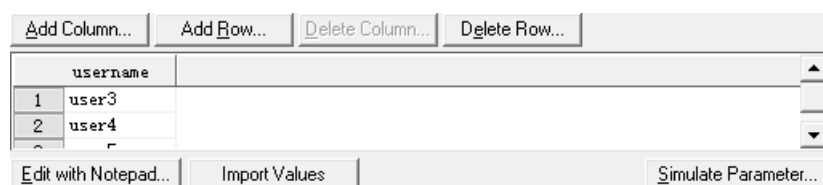


图1.2: 参数化

```
web_set_user("{username}",
lr_decrypt("5f003a65e3278d"),
"35.221.252.58:80");
```

图1.3: 参数化代码

到这一步，脚本基本上设置完成，尝试着去做replay操作，观察是否能够成功运行。

5. 打开Controller，调整并发Vuser的数量等参数，首先调整Vuser为5个：

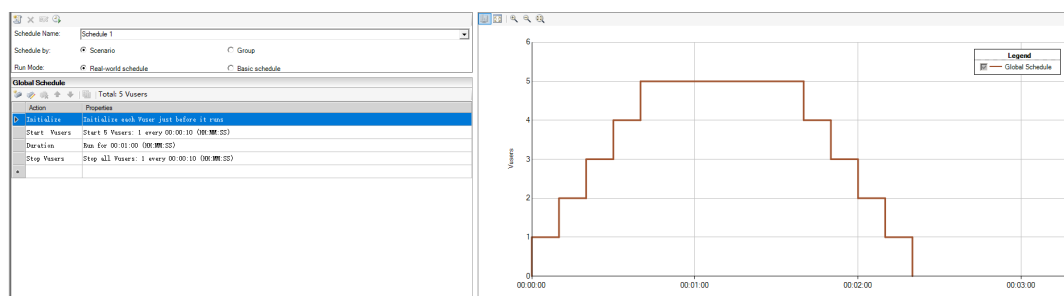


图1.4: Controller设置vuser数量

6. 点击按钮开始运行，稍加等待后得到结果：

(由于该服务器并没有跑在本地localhost，因此监控资源实际上没有太大意义)

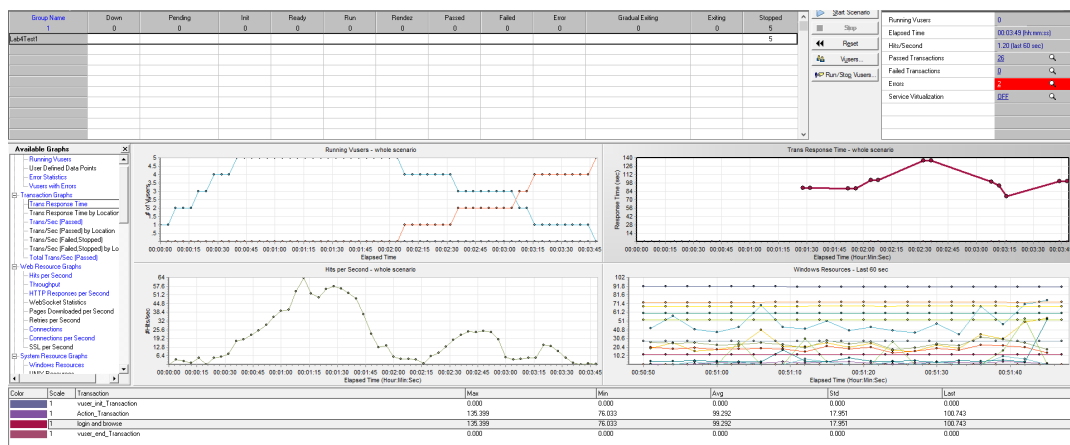


图1.5: Controller运行时截图

## 7. 使用Analysis，对测试结果进行关联分析：

(图中的规则变化曲线为Vuser的数量)

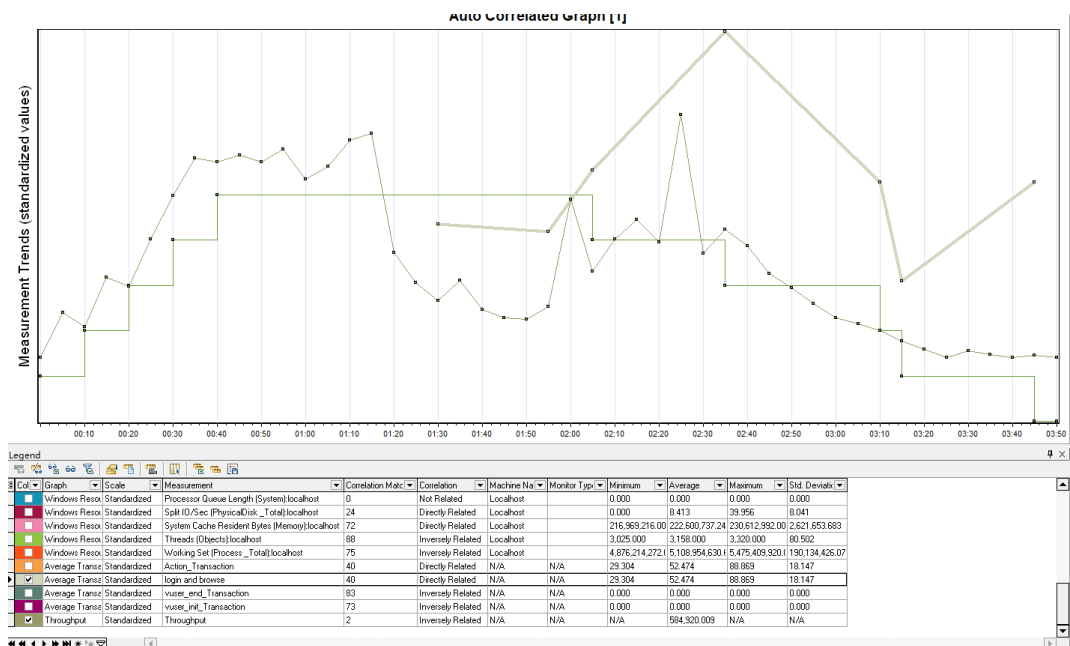


图1.6: Analysis生成的图片

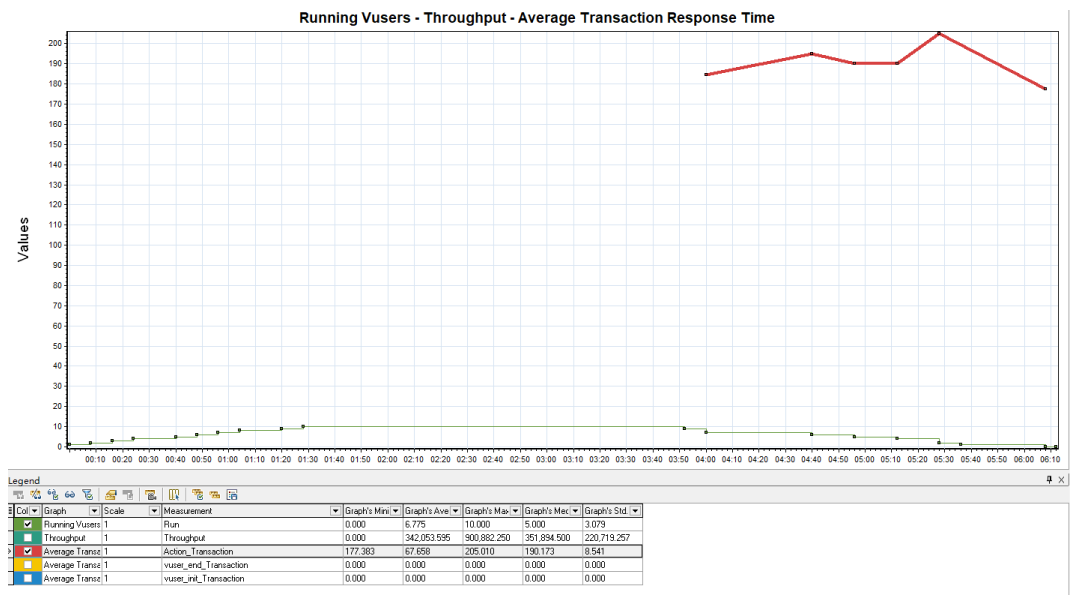
## 8. 变更Vuser的数量，回到第5步进行重复测试。

### 1.1.2 测试结果分析

#### 1. 当Vuser最大为5的时候：见上图

- 随着Vuser数量的增长，Throughput先是增加，之后开始下降并发生波动。说明对于一个Transaction来说，前半部分的文件下载量要大于后半部分。这里正好说明前半部分是在加载主页
- Transaction的响应时间成一个先增长，后下降的趋势，latency的增加说明5个Vuser已经给平台造成了一些压力。

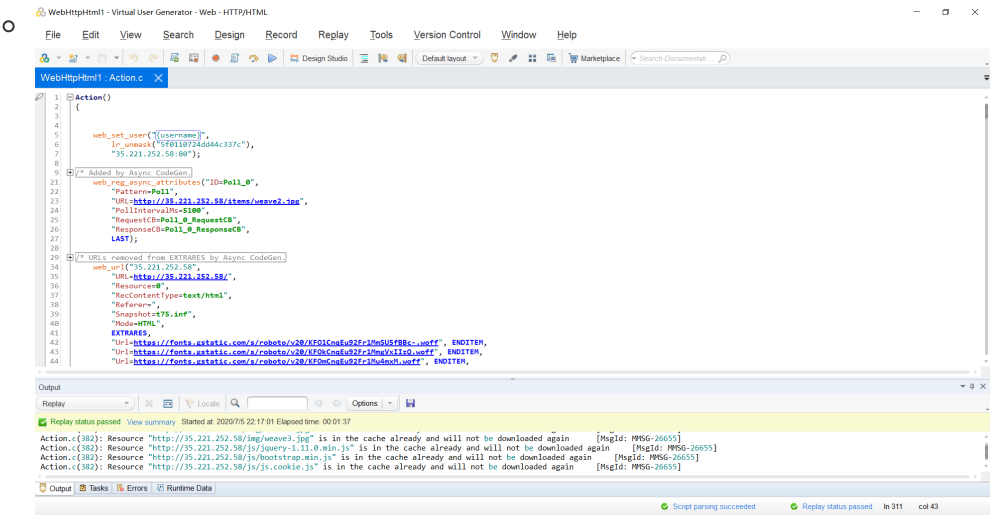
#### 2. 当Vuser最大为10的时候：



- 此时可以看到，相比之前测试而言，此时的平均响应时间发生了大量的增长，而且 throughput 也发生了下降，说明此时平台收到的压力已经比较明显。

### 1.2.1 测试详细流程

1. 使用Vuser Generator, 对整个服务流程进行录制。



2. 本流程将登录和注册设为两个 transaction，如下图所示。

```

lr_start_transaction("register");
web_custom_request("register",
    "URL=http://35.221.252.58/register",
    "Method=POST",
    "Resource=0",
    "RecContentType=application/json",
    "Referer=http://35.221.252.58/",
    "Snapshot=t81.inf",
    "Mode=HTML",
    "EncType=application/json; charset=utf-8",
    "Body={\"username\":\"{username}\",\"password\":\"12345\", \"email\":\"{email}\", \"firstName\":\"qq{username}\", \"lastName\":\"www{username}\"}",
    LAST);
lr_end_transaction("register", LR_AUTO);

```

```

lr_start_transaction("login");

web_url("login",
    "URL=http://35.221.252.58/login",
    "Resource=0",
    "RecContentType=text/html",
    "Referer=http://35.221.252.58/",
    "Snapshot=t97.inf",
    "Mode=HTML",
    LAST);

lr_end_transaction("login", LR_AUTO);

```

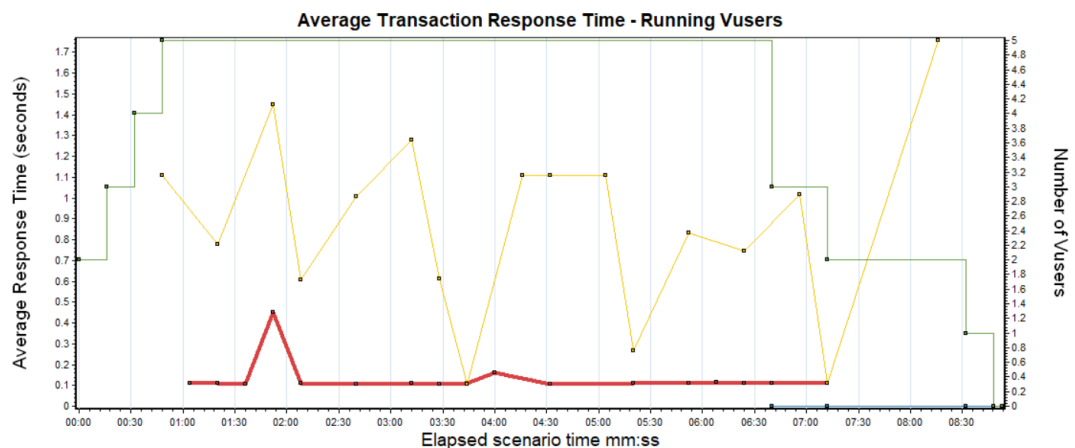
3. **参数化**：注册和登录的用户需要用不同的用户名以避免碰撞，于是需要进行参数化。将用户的邮箱和用户名参数化后，结果如上图。参数化的参数规则如下：

<pre>&lt;#&gt; email</pre> <pre>&lt;#&gt; username</pre>	Parameter type: Random Number Random range: Min: 1 Max: 10000 Sample value: email00001@user.com, email10000@user.com Number: email%05lu@user.com
<pre>&lt;#&gt; email</pre> <pre>&lt;#&gt; username</pre>	Parameter type: Random Number Random range: Min: 1 Max: 10000 Sample value: user00001, user10000 Number: user%05lu

4. 打开Controller，调整Vuser为5个，开始运行，一段时间后得到结果。

5. 使用Analysis，对测试结果进行关联分析：

(图中的绿色曲线为Vuser的数量)



## 1.2.2 测试结果与分析

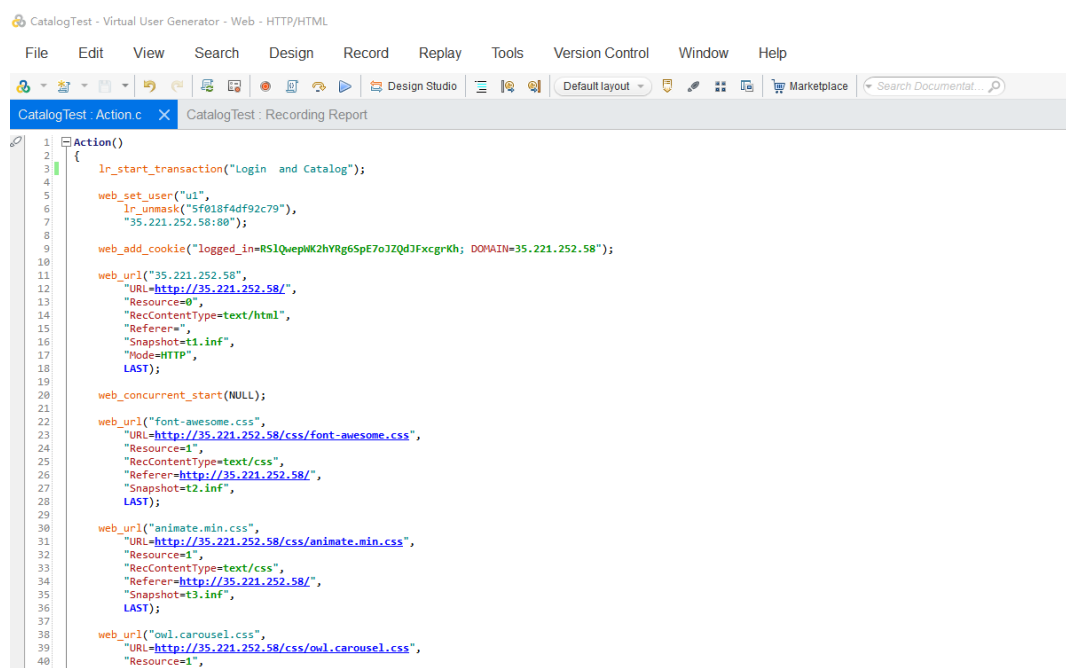
1. 当 Vuser 数量为 5 的时候，平台的注册操作依然有巨大的抖动，但登陆操作却基本稳定，初步判断是因为平台的写操作需要较长的时间才能提交。整体测试结果基本位于预期范围内，说明平台注册和登陆操作在较小的并发请求下能够正常工作。

## 1.3 登录 + 按类别查找商品

### 1.3.1 测试流程

1. 注册一个用户，用于后面登录。
2. 使用Virtual user Generator，对整个服务流程进行录制。
  - 登录使用先前注册好的其中一个用户
  - 因为使用Chrome录制，可能会访问google的一些相关网页，在录制的时候，需要开启代理。
  - 录制选项里面，需要调整Recording options里面，基于URL-based script，否则在replay的时候会报错无法运行。
  - 录制完成后，仅include和35.221.252.58相关的操作，exclude掉和谷歌相关的操作

录制好之后可以看到生成的脚本：



```

1  Action()
2  {
3      lr_start_transaction("Login and Catalog");
4
5      web_set_user("u1",
6          lr_unmask("5f018f4df92c79"),
7          "35.221.252.58:80");
8
9      web_add_cookie("logged_in=RS1QwepMK2hYRg6SpE7oJZQdJFxcgrKh; DOMAIN=35.221.252.58");
10
11      web_url("35.221.252.58",
12          "URL=http://35.221.252.58/",
13          "Resource=0",
14          "RecContentType=text/html",
15          "Referer=",
16          "Snapshot=t1.inf",
17          "Mode=HTTP",
18          LAST);
19
20      web_concurrent_start(NULL);
21
22      web_url("font-awesome.css",
23          "URL=http://35.221.252.58/css/font-awesome.css",
24          "Resource=1",
25          "RecContentType=text/css",
26          "Referer=http://35.221.252.58/",
27          "Snapshot=t2.inf",
28          LAST);
29
30      web_url("animate.min.css",
31          "URL=http://35.221.252.58/css/animate.min.css",
32          "Resource=1",
33          "RecContentType=text/css",
34          "Referer=http://35.221.252.58/",
35          "Snapshot=t3.inf",
36          LAST);
37
38      web_url("owl.carousel.css",
39          "URL=http://35.221.252.58/css/owl.carousel.css",
40          "Resource=1",
41          "RecContentType=text/css",
42          "Referer=http://35.221.252.58/",
43          "Snapshot=t4.inf",
44          LAST);
45
46      lr_end_transaction("Login and Catalog");
47  }
    
```

图3.1：脚本

3. 本流程将全操作设为一个transaction，因此在前后加入 lr\_start\_transaction 和 lr\_end\_transaction 两部分脚本，给transaction起名为 Login Catalog。

（实际上在Controller运行的时候会自动把全文设为一个transaction，这里我自己做一个简单包装）

4. 参数化：可以在脚本的最前端看到一个web\_set\_user的语句，这一部分是和登录相关的。此时，我们将用户名改为之前设计的多个用户的用户名。

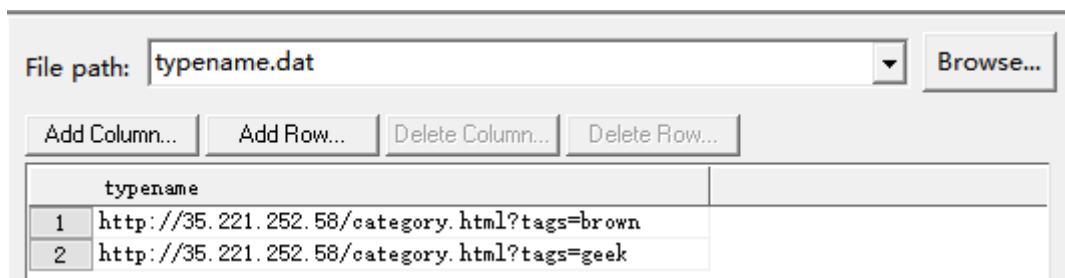


图3.2：类别参数化

```
web_url("category.html_4",
"URL={typename}",
"Resource=0",
"RecContentType=text/html",
"Referer=http://35.221.252.58/category.html?tags=geek",
"Snapshot=t123.inf",
"Mode=HTTP",
LAST);
```

图3.3：参数化相关代码

到这一步，脚本基本上设置完成，尝试着去做replay操作，观察是否能够成功运行。

5. 打开Controller，调整并发Vuser的数量等参数，首先调整Vuser为10个：

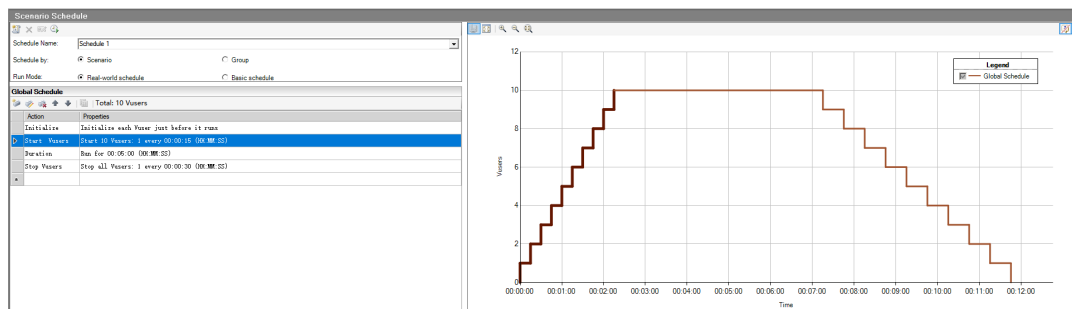


图3.4：vuser数量设置

6. 点击按钮开始运行，稍加等待后得到结果：

(由于该服务器并没有跑在本地localhost，因此监控资源实际上没有太大意义)

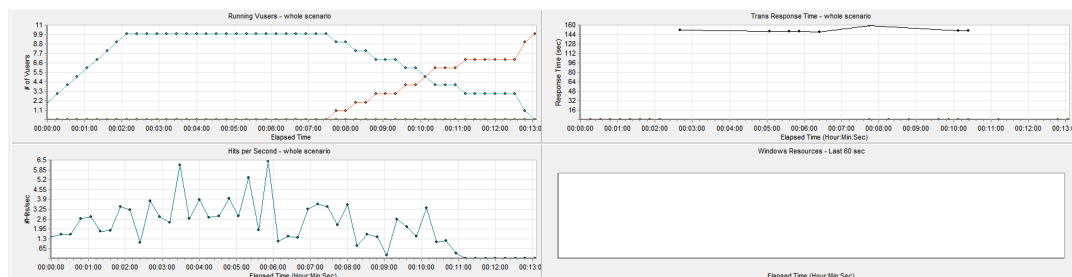


图3.5：Controller运行时截图

7. 使用Analysis，对测试结果进行关联分析：

(图中的规则变化曲线为Vuser的数量)

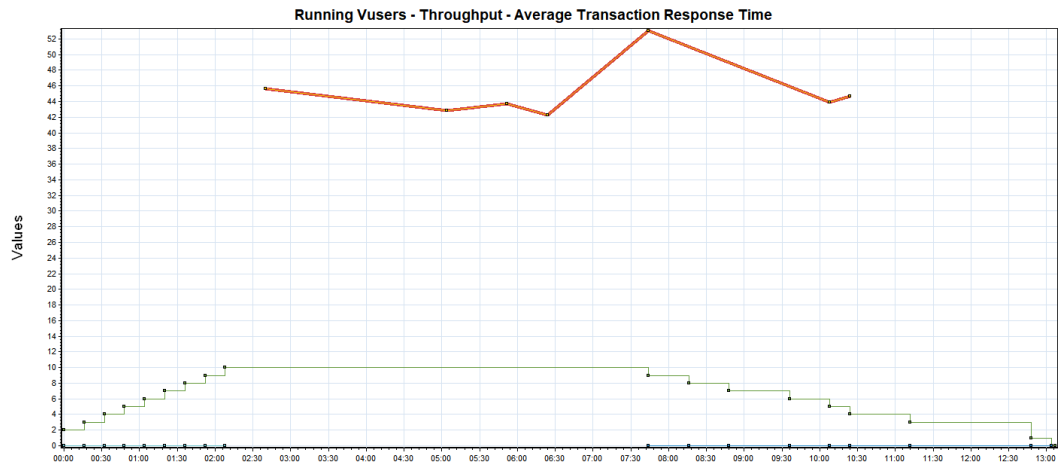


图3.6: Throughput - Avg.Latency - Running users

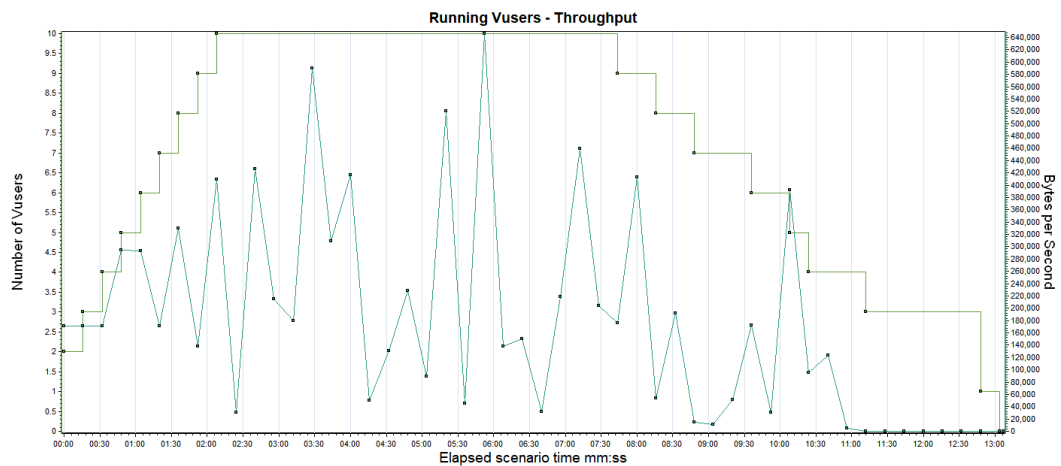


图3.7: Throughput - Running users

8. 变更Vuser的数量，回到第5步进行重复测试。

### 1.3.2 测试结果分析

1. 当Vuser最大为10的时候：见上图

- 随着Vuser数量的增长，Throughput的波动越来越剧烈且总体上呈下降趋势，。说明对于一个Transaction来说，前半部分的文件下载量要大于后半部分。这里正好说明前半部分是在加载主页
- Transaction的响应时间成一个先下降，后增长的趋势，latency的增加说明10个Vuser已经给平台造成的压力十分明显。

2. 当Vuser最大为4的时候：

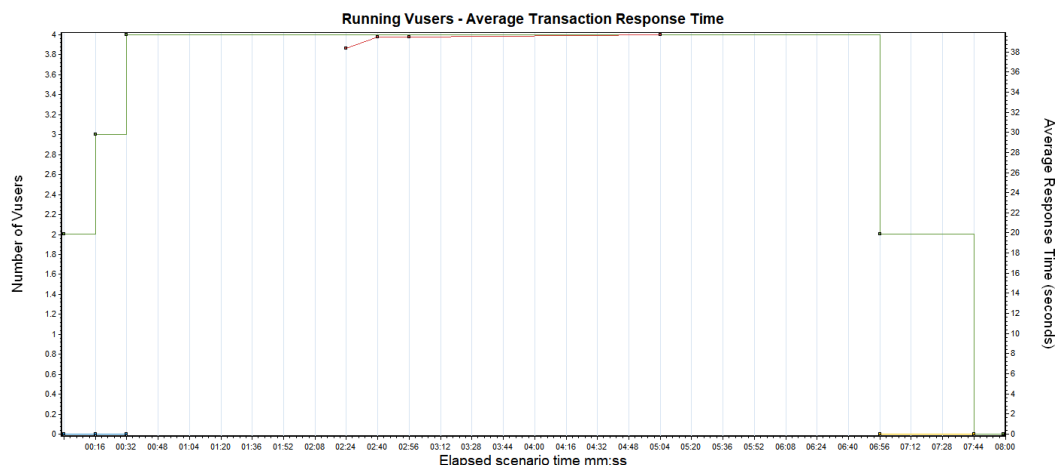


图3.8: Avg. Latency - Running users



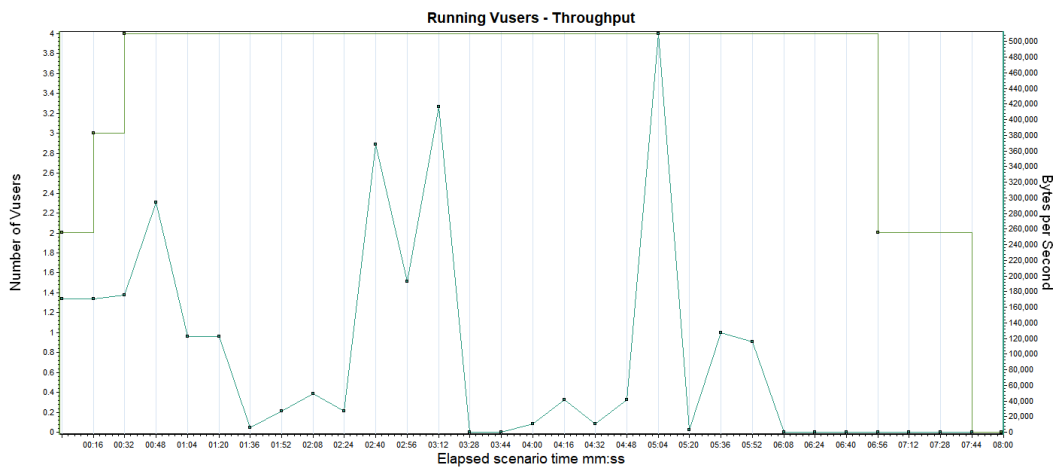


图3.9: Throughput - Throughput

- 此时可以看到，相比之前测试而言，由于并发用户数更少，此时的平均响应时间小了许多，而且throughput的抖动也更加平缓且总体上呈现比较平均的趋势，说明此时平台收到的压力比较小。

## 1.4 登录 + 加入购物车 + 下单

### 1.4.1 测试流程

1. 首先通过 *Virtual User Generator* 的录制脚本功能录制脚本。

- 加载完成后，在首页点击登录。
- 登录成功后，点击商品，进入商品详情列表，将商品加入购物车。
- 加入购物车后，添加收货地址、付款方式，点击下单
- 成功下单后，会进入用户的订单页面，查看订单是否成功。

2. 修改 *Virtual User Generator* 生成的脚本

- 将 *username* 参数化，这里需要使用 *vuserid* 的参数化配置，因为如果不同的 *vuser* 使用了同一个用户登录，进行下单操作，可能会造成并发问题。

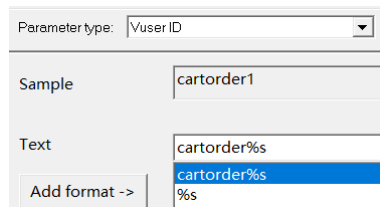


图4.1: 参数化设置

2. 获取用户登录的 session id

```
1 web_reg_save_param_ex(
2     "ParamName=sessionId",
3     "LB/IC=logged_in=",
4     "RB/IC=;",
5     "Ordinal=1",
6     SEARCH_FILTERS,
7     "Scope=cookies",
8     LAST);
```

3. 添加 transaction，分别添加 tx\_login、tx\_add\_card、tx\_add\_cart、tx\_order

3. 使用 *Controller* 设置 *vusers* 和运行场景

- 按照用例4，*vuser* 每 15 s 增加一个，直到 50 个 *vuser* 停止增加。每个 *vuser* 运行 5 分钟，运行过程中不断将商品添加购物车，添加下单，查看订单。运行一段时间后，*vuser* 每隔 10s 退出。

4. 使用 *Analysis* 对结果进行分析，设置SLA —— 用户数小于 20 时，每个事务的响应时间在 0.5s 以内；用户数介于 20 - 40 之间时，每个事务的响应时间在2s 以内；用户数大于40时，响应时间在 8s 以内。

The SLA status of the following measurements displayed over time. You can select a specific time range for each transaction in order to analyze the time rang

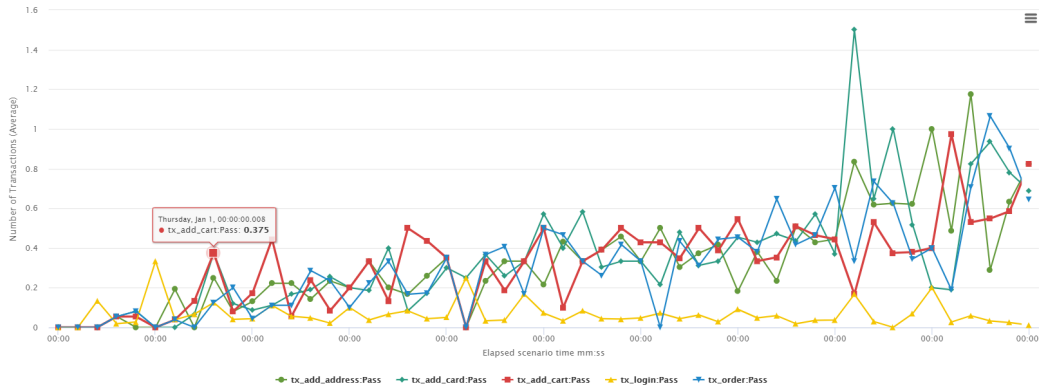
Measurement Name	Time Ranges																	
Application Under Test Errors	0	0	0	0	0	0	0	0	0	0	0+	0+	0+	0	0	0	0+	0+
tx_add_address																		
tx_add_card																		
tx_add_cart																		
tx_login																		
tx_order																		
	00:00:00	00:01:10	00:02:25	00:03:40	00:04:50	00:06:05	00:07:20	00:08:35	00:09:45	00:11:00	00:12:15	00:13:25	00:14:40	00:15:55	00:17:10	00:18:20	00:19:35	00:20:50

上图为 SLA 评测结果，50个vuser的情况下，性能比较正常

5. 使用 *Analysis* 对 TX 和 Running User 进行关联分析

Running Vusers - Transactions per Second

Granularity: 1 Second

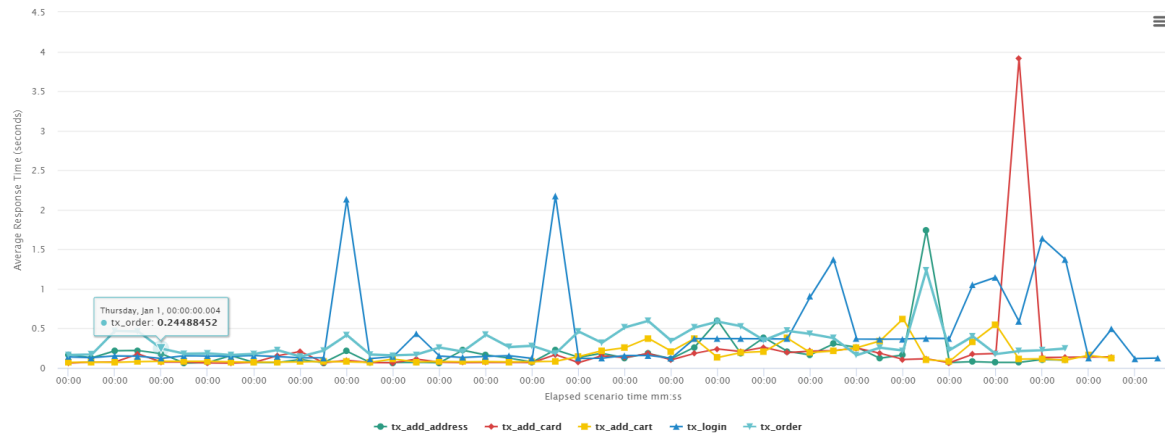


图：TPS

随着用户的上升，TPS 是呈上升趋势的。

Running Vusers - Average Transaction Response Time

Granularity: 1 Second



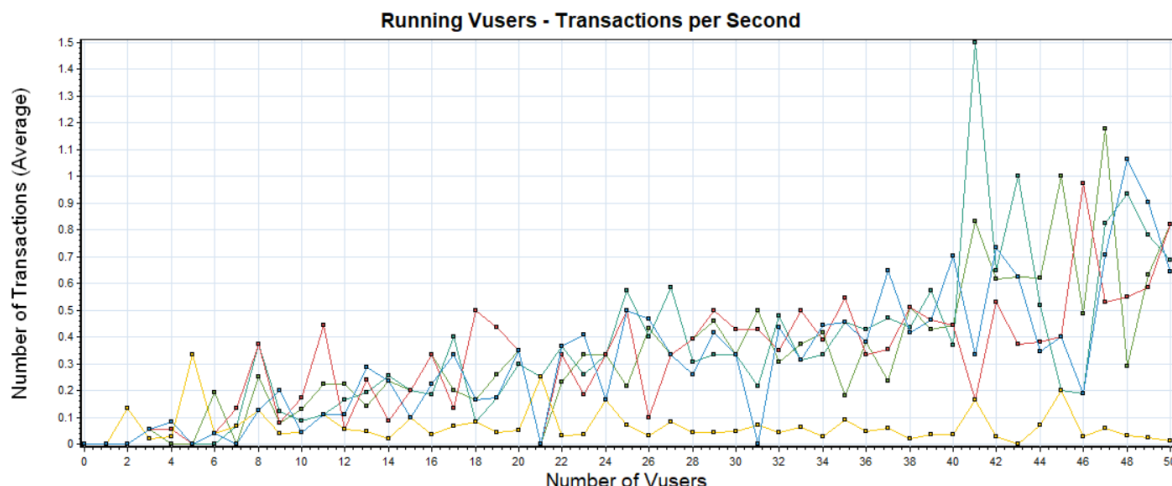
图：Avg. Transaction Response Time

每个事务的延时有波动，但是基本上都稳定在比较低的水平，说明系统暂未达到性能瓶颈。

### 1.4.2 测试结果分析

本次测试是在对系统的静态资源和架构进行优化之后再进行的测试，可以看到，系统可以响应50个 vuser 的并发请求，可见系统的性能有很大的提升。

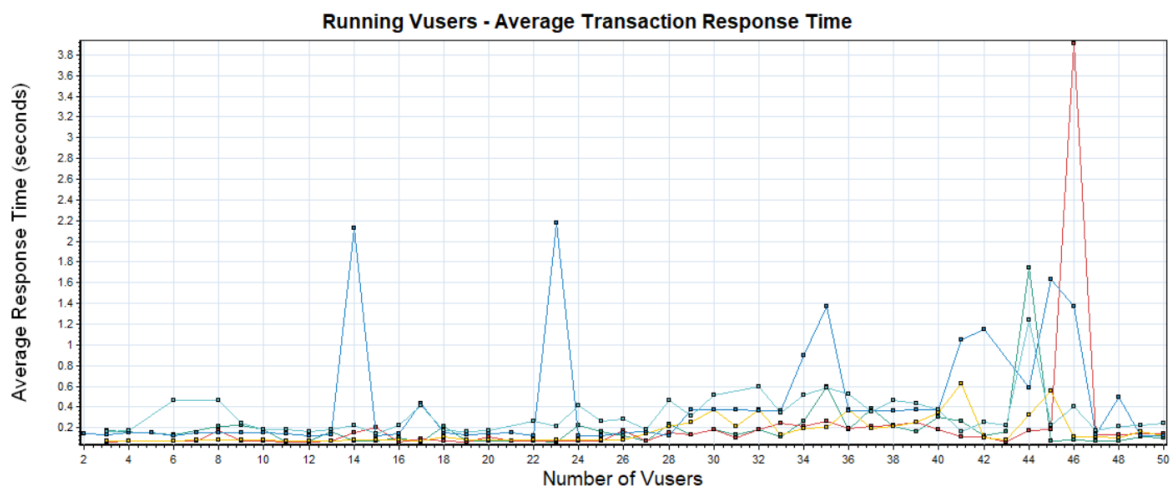
我们首先分析一下 TPS



可以看到，TPS 随着用户数量上升大致呈上升趋势，说明用户越多，系统未达到性能瓶颈的时候吞吐量是不断上升的。

在用户达到 40 的时候，TPS 出现持平的状态，这说明网站已经接近了极限，相信如果 继续增加 vuser数量的话，TPS 甚至会出现下降的趋势。

我们再看一下 Avg. Tx 延时



TX 的平均演示都比较低，再 vuser到达 30以后，每个事务的延时都略微上升。说明用户数量对与系统的性能起到了压力影响，但是并没有影响系统的服务能力

在用户到达 45 左右的时候，事务延迟上升明显，说明系统接近性能瓶颈了。如果再增加用户数，应该能看到 延时大幅度的上升甚至超时、错误发生。

## 2. 网站的性能结果分析

### 2.1 架构

刚开始时，我们直接使用测试计划中的架构部署被测网站，在使用浏览器访问时效果不是很差，感觉不出和其它网站的响应时间的差别，这主要是因为被测网站是个私人网站，只有几个用户，没有并发。

开始测试的时候，可以看到在 测试用例1 和 测试用例3 中吞吐量测试的时候 Vuser 的并发都被限制在了 10 以下。在测试执行过程中，也发现了 Controller 会报大量图 2.1.1 的错误。

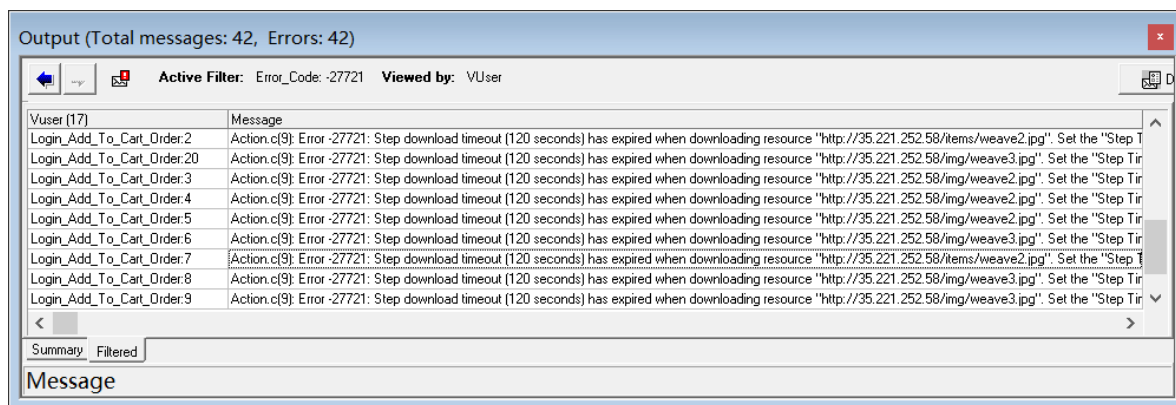


图2.1.1: Timeout

考虑到，我们原来的架构中，所有的请求都需要发送给 *frontend*，*frontend* 既需要通过 *express* 返回静态资源，有需要通过 *express* 注册号的路由返回动态资源，压力比较大，自然也会成为性能瓶颈。

于是，我们将静态资源和动态资源进行分离。如图2.1.2所示，我们使用 *nginx* 作为反向代理，将请求分成静态和动态，分别转发给原先的 *frontend* 处理动态请求，静态请求则发送给另一个 *nginx* 服务器来专门处理静态请求。

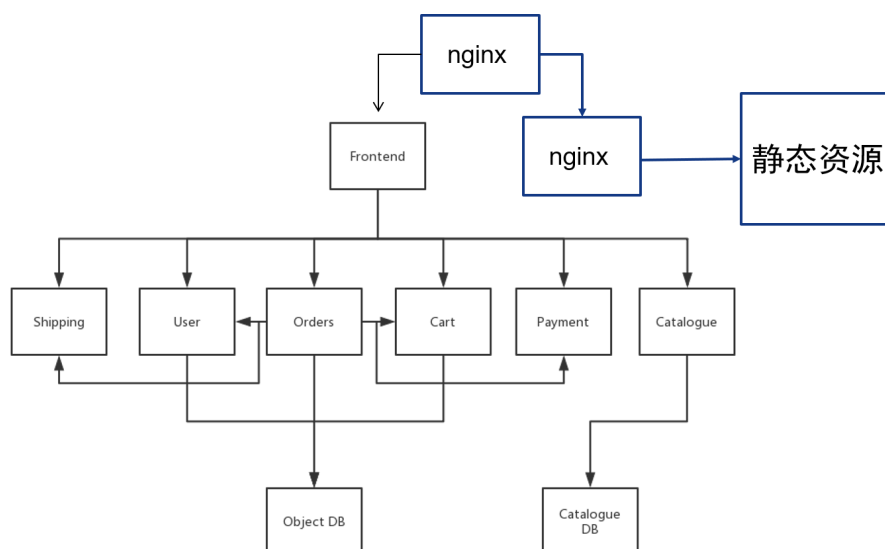


图2.1.2: 新架构

尽管换了新的架构，但是由于系统是微服务架构，并且使用的不是 *RPC* 通信，服务之间的通信开销也是性能瓶颈之一。

## 2.2 部署平台

由于我们将网站使用 *docker* 直接部署在云服务器上，没有对服务器的资源进行整合。在测试的时候，通过 *htop* 命令发现 *CPU*、*Memory* 资源利用率都非常低，基本上只有 10% 左右。

不过这也正说明了，我们部署的网站的性能瓶颈不是服务器性能，而是其他方面

## 2.3 资源文件大小

1. 当 *Vuser* 数量为 5 的时候，平台的注册操作依然有巨大的抖动，但登陆操作却基本稳定，初步判断是因为平台的写操作需要较长的时间才能提交。

考虑到一半注册请求不会有很大的并发量，因此整体测试结果基本位于预期范围内，说明平台注册和登陆操作在较小的并发请求下能够正常工作。