软件体系结构实验报告

实验四：可拓展 Web 架构

姓 名: 王 皓 冉
学 号: 191220107

软件体系结构
(春季, 2022)

南京大学
计算机科学与技术系

2022 年 4 月 7 日

摘 要

对于可拓展的 Web 架构，需要考虑以下的几点问题：

1. 使用虚拟化技术/容器技术便于服务的扩展
2. 使用负载均衡技术实现水平扩展
3. 使用缓存服务器减少主服务器负载
4. 扩展之后的用户会话状态保持

实验四在实验三实现的 WebPOS 的基础上：

- 将该服务系统容器化
- 使用 HAProxy 提供的负载均衡功能对容器进行水平扩展
- 处理缓存丢失问题和会话保持问题

1 实验环境

使用 Ubuntu Server 20.04 操作系统，运行在 VirtualBox 下。网络环境采用仅主机（Host-Only）网络，宿主机 IP 为 192.168.137.1，虚拟机 IP 为 192.168.137.2。宿主机分配了 4 个物理核心和 4G 内存给虚拟机，对于网络请求来说计算性能可能并不是瓶颈。

2 容器化

2.1 容器的构建和运行

使用 Docker 进行容器化，构建 Docker 镜像所使用的 Dockerfile 如下：

```
FROM maven:3.8.4-eclipse-temurin-11
COPY settings.xml /usr/share/maven/conf/settings.xml
COPY aw04-wanghr64.tar.gz /
WORKDIR /
RUN tar -xzvf aw04-wanghr64.tar.gz
WORKDIR /aw04-wanghr64
RUN mvn install
CMD mvn clean spring-boot:run
```

即基于 maven 官方提供的 Docker 镜像进行构建。考虑到构建 Docker 镜像时对目录的复制并不能保证目录下文件结构的保持，因此这里首先将工程目录进行打包，在构建的过程中进行解压操作。为了使得镜像在每一次运行时不需要重新下载依赖，在构建过程中首先对大部分的依赖进行了下载。

进行构建之后得到的 Docker 镜像信息如下：

```
wanghr@ubuntu20:~/SA/lab04$ sudo docker image ls
REPOSITORY    TAG                       IMAGE ID       CREATED        SIZE
webpos        latest                    04364cdcaf3a   4 days ago     690MB
maven         3.8.4-eclipse-temurin-11  b2f8191268e3   4 weeks ago    536MB
wanghr@ubuntu20:~/SA/lab04$
```

运行该镜像的命令如下：

```
wanghr@ubuntu20:~/SA/lab04$ sudo docker run -d -p 8080:8080 webpos
62edd97109bfc0124f64160e9240f46884db6e875219c589da27cef413b71a80
wanghr@ubuntu20:~/SA/lab04$
```

在宿主机的浏览器中即可访问虚拟机容器中运行中的网页前端服务。

2.2 压力测试

使用 JMeter 进行压力测试，测试配置的结构如下：

1. 线程组
   使用 100 个线程，Ramp-Up 时间为 1 秒，循环次数为 30 次。

   (a) HTTP 请求
   服务器名称为 192.168.137.2，端口号为 8080，HTTP 请求为 GET，
   路径为/add?pid=12800420（即请求添加一个商品）

   (b) 查看结果树

   (c) 汇总报告

压力测试的汇总结果如下：

表 1: 测试结果汇总 1

Label	**样本数量**	**平均值**	**最小值**	**最大值**	**标准偏差**
HTTP 请求	3000	431	13	1859	293.29

表 2: 测试结果汇总 2

异常%	**吞吐量**	**接受 KB/sec**	**发送 KB/sec**	**平均字节数**
0	218.21356	7860.16	29.62	36885

3 水平拓展

3.1 HAProxy 配置

首先在后台运行 6 个之前构造的 WebPOS 镜像，分别映射到虚拟机的 8081-8086 端口。

```
CONTAINER ID   IMAGE     PORTS
3dea148bd8f9   webpos    0.0.0.0:8086->8080/tcp, :::8086->8080/tcp
ed3b15d0fd5e   webpos    0.0.0.0:8085->8080/tcp, :::8085->8080/tcp
db1b8b606a98   webpos    0.0.0.0:8084->8080/tcp, :::8084->8080/tcp
f53b7271c112   webpos    0.0.0.0:8083->8080/tcp, :::8083->8080/tcp
963f3c2c40c8   webpos    0.0.0.0:8082->8080/tcp, :::8082->8080/tcp
1b77d1fd791e   webpos    0.0.0.0:8081->8080/tcp, :::8081->8080/tcp
```

编辑 HAProxy 的配置文件/etc/haproxy/haproxy.cfg，添加以下内容：

```
frontend WebPOS_frontend
        bind 0.0.0.0:8080
        mode http
        default_backend WebPOS_backends

backend WebPOS_backends
        mode http
        balance leastconn
        server localhost:8081 localhost:8081
        server localhost:8082 localhost:8082
        server localhost:8083 localhost:8083
        server localhost:8084 localhost:8084
        server localhost:8085 localhost:8085
        server localhost:8086 localhost:8086
```

最后重启 haproxy.service 服务。

在宿主机进行浏览器的访问测试，192.168.137.2:8080 地址可以访问 WebPOS（效果和之前容器化一样，因此不再赘述）

3.2 压力测试

使用与之前单容器的压力测试一样的测试配置，得到结果如下：

表 3: 测试结果汇总 1

Label	样本数量	平均值	最小值	最大值	标准偏差
HTTP 请求	3000	369	6	1380	193.41

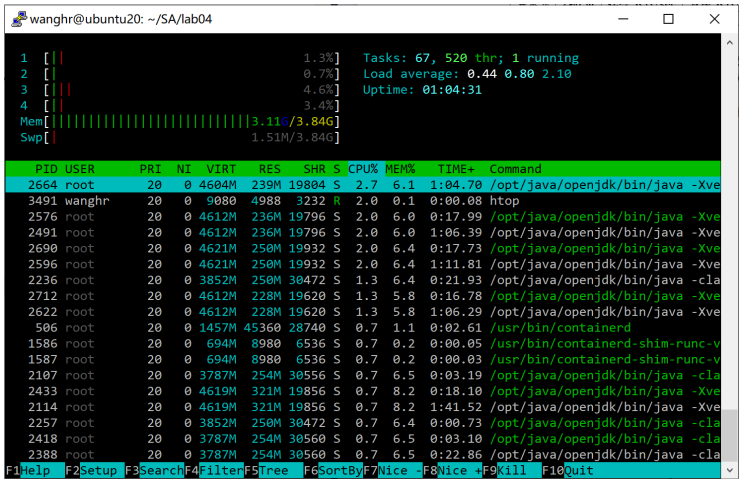
表 4: 测试结果汇总 2

异常%	吞吐量	接受 KB/sec	发送 KB/sec	平均字节数
0	245.56	8824.90	33.33	36803.5

可以看出，虽然平均值相较于之前的单容器服务可能有所下降，但是标准偏差作为稳定性的衡量指标，有着一定的提升。同时吞吐量、接受速率和发送速率也有一定的提升。

但是使用负载均衡的效果并没有好于预期。吞吐量、接受速率和发送速率的提升幅度只有 12%。

本来考虑可以通过进一步地添加容器，使用更多的后端进行负载均衡。但是仅仅 6 个容器就已经让虚拟机环境的内存资源不堪重负。也许无限制地进行水平拓展，在计算上并没有太多的压力（还是取决于用户的请求强度），但是闲置状态的内存压力就已经很大了。



上图中可以看出，分配的 4G 内存已经被占用了 3.11G。更大规模的水平扩展已经几乎不可能。

4 缓存机制

4.1 环境配置

以 7000 端口的 Redis 节点为例，每一个 Redis 节点的配置文件 redis.conf 的内容如下：

```
port 7000
cluster-enabled yes
cluster-config-file nodes.conf
cluster-node-timeout 5000
appendonly yes
```

修改 pom.xml 文件，添加 Redis 相关的依赖：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-cache</artifactId>
</dependency>
<dependency>
    <groupId>redis.clients</groupId>
    <artifactId>jedis</artifactId>
</dependency>
```

在 WebPosApplication.java 的 main 函数前，添加以下注解：

```
@EnableCaching
```

具体需要进行缓存的数据对象，是在 JD.java 中向京东服务器请求的 products。因此在 getProducts() 函数前添加注解如下：

```
@Cacheable(value = "products")
```

4.2 压力测试

由于 docker 配合 redis 的网络配置比较复杂，因此这里使用 maven 将 spring-boot 应用打包，直接在虚拟机的环境中不使用容器，在后台同时运行 6 个 WebPos 实例。

shell 脚本文件如下：

```
java -jar target/webpos-0.0.1-SNAPSHOT.jar --server.port=8081 &
java -jar target/webpos-0.0.1-SNAPSHOT.jar --server.port=8082 &
java -jar target/webpos-0.0.1-SNAPSHOT.jar --server.port=8083 &
java -jar target/webpos-0.0.1-SNAPSHOT.jar --server.port=8084 &
java -jar target/webpos-0.0.1-SNAPSHOT.jar --server.port=8085 &
java -jar target/webpos-0.0.1-SNAPSHOT.jar --server.port=8086 &
```

测试结果的汇总如下：

表 5: 测试结果汇总 1

Label	**样本数量**	**平均值**	**最小值**	**最大值**	**标准偏差**
HTTP 请求	3000	237	5	3590	399.91

表 6: 测试结果汇总 2

异常%	**吞吐量**	**接受 KB/sec**	**发送 KB/sec**	**平均字节数**
0	304.7	10525.53	40.17	35377.5

可以看出，在单纯的水平扩展和负载均衡的基础上，性能又得到了进一步的提升。（实验过程中由于仅主机网络出现了错误，这里将虚拟机的网络类型改为了 NAT。考虑到 NAT 的额外开销，实际的性能表现应该会更好。）

5 会话保持机制

5.1 环境配置

修改 pom.xml 文件，添加 Redis 相关的依赖：

```
<dependency>
    <groupId>org.springframework.session</groupId>
    <artifactId>spring-session-data-redis</artifactId>
</dependency>
```

在 WebPosApplication.java 的 main 函数前，添加以下注解：

```
@EnableRedisHttpSession
```

修改 PosController.java，添加 session 相关的代码。修改后的结果如下：

```
@Controller
public class PosController {

    @Autowired
    private HttpSession session;

    private PosService posService;

    private Cart getCart() {
        Cart cart = (Cart) session.getAttribute("cart");
        if (cart == null) {
            cart = new Cart();
            this.saveCart(cart);
        }
        return cart;
    }

    public void saveCart(Cart cart) {
        session.setAttribute("cart", cart);
    }

    ......

    @GetMapping("/add")
    public String addByGet(@RequestParam(name = "pid") String pid, Model model) {
        saveCart(posService.add(getCart(), pid, 1));
        model.addAttribute("products", posService.products());
        model.addAttribute("cart", getCart());
        return "index";
    }
}
```

5.2 测试

使用两个不同的浏览器，访问 WebPos 服务，并添加不同的商品，得到结果如下：