

一篇有趣的负载均衡算法实现

OSC开源社区 昨天

以下文章来源于程序猿阿朗，作者达西呀



程序猿阿朗

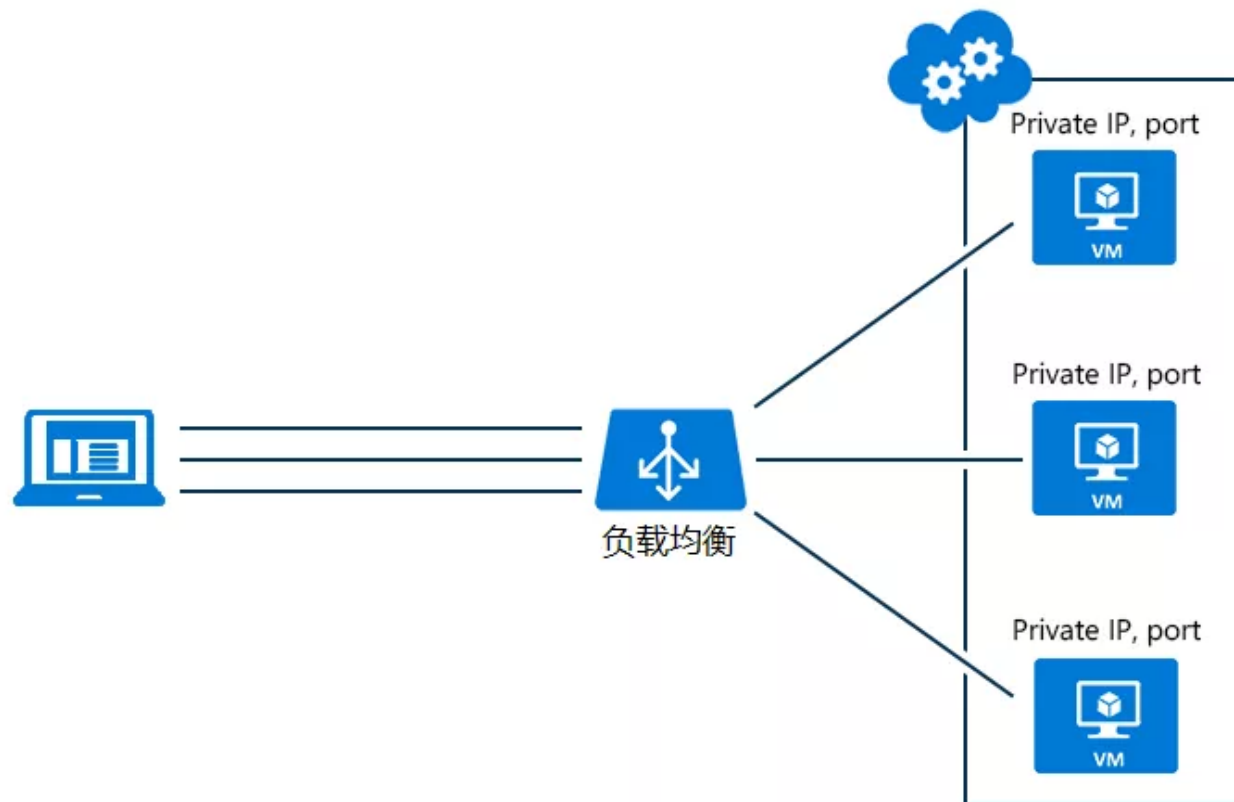
Hello world :) 一线技术工具人，认认真真写篇文章。



负载均衡（Load balancing）是一种在多个计算机（网络、CPU、磁盘）之间均匀分配资源，以提高资源利用的技术。使用负载均衡可以最大化服务吞吐量，可能最小化响应时间，同时由于使用负载均衡时，会使用多个服务器节点代单点服务，也提高了服务的可用性。

负载均衡的实现可以软件可以硬件，硬件如大名鼎鼎的 F5 负载均衡设备，软件如 NGINX 中的负载均衡实现，又如 Springcloud Ribbon 组件中的负载均衡实现。

如果看到这里你还不知道负载均衡是干嘛的，那么只能放一张图了，毕竟没图说个啥。



负载均衡要做到在多次请求下，每台服务器被请求的次数大致相同。但是实际生产中，可能每台机器的性能不同，我们会希望性能好的机器承担的请求更多一些，这也是正常需求。

如果这样说下来你看不懂，那我就再举个例子好了，一排可爱的小熊（服务器）站好。



这时有人（用户）要过来打脸（请求访问）。



那么怎么样我们才能让这每一个可爱的小熊被打的次数大致相同呢？

又或者熊 4 比较胖，抗击打能力是别人的两倍，我们怎么提高熊 4 被打的次数也是别人的两倍呢？

又或者每次出手的力度不同，有重有轻，恰巧熊 4 总是承受这种大力度啪啪打脸，熊 4 即将不省熊事，还要继续打它吗？

这些都是值得思考的问题。

说了那么多，口干舌燥，我双手已经饥渴难耐了，迫不及待的想要撸起代码了。

1. 随机访问

上面说了，为了负载均衡，我们必须保证多次出手后，熊 1 到熊 4 被打次数均衡。比如使用随机访问法，根据数学上的概率论，随机出手次数越多，每只熊被打的次数就会越相近。代码实现也比较简单，使用一个随机数，随机访问一个就可以了。

```
/** 服务器列表 */
private static List<String> serverList = new ArrayList<>();
static {
    serverList.add("192.168.1.2");
    serverList.add("192.168.1.3");
    serverList.add("192.168.1.4");
    serverList.add("192.168.1.5");
}

/**
 * 随机路由算法
 */
public static String random() {
    // 复制遍历用的集合，防止操作中集合有变更
    List<String> tempList = new ArrayList<>(serverList.size());
    tempList.addAll(serverList);
    // 随机数随机访问
    int randomInt = new Random().nextInt(tempList.size());
    return tempList.get(randomInt);
}
```

因为使用了非线程安全的集合，所以在访问操作时操作的是集合的拷贝，下面几种轮询方式中也是这种思想。

写一个模拟请求方法，请求10w次，记录请求结果。

```
public static void main(String[] args) {  
    HashMap<String, Integer> serverMap = new HashMap<>();  
    for (int i = 0; i < 20000; i++) {  
        String server = random();  
        Integer count = serverMap.get(server);  
        if (count == null) {  
            count = 1;  
        } else {  
            count++;  
        }  
        // 记录  
        serverMap.put(server, count);  
    }  
    // 路由总体结果  
    for (Map.Entry<String, Integer> entry : serverMap.entrySet()) {  
        System.out.println("IP:" + entry.getKey() + " · 次数：" + entry.getValue());  
    }  
}
```

运行得到请求结果。

```
IP:192.168.1.3 · 次数：24979  
IP:192.168.1.2 · 次数：24896  
IP:192.168.1.5 · 次数：25043  
IP:192.168.1.4 · 次数：25082
```

每台服务器被访问的次数都趋近于 2.5w，有点负载均衡的意思。但是随机毕竟是随机，是不能保证访问次数绝对均匀的。

2. 轮询访问

轮询访问就简单多了，拿上面的熊1到熊4来说，我们一个接一个的啪啪 - 打脸，熊1打完打熊2，熊2打完打熊3，熊4打完打熊1，最终也是实现了被打均衡。但是保证均匀总是要付出代价的，随机访问中需要随机，轮询访问中需要什么来保证轮询呢？

```
/** 服务器列表 */
private static List<String> serverList = new ArrayList<>();
static {
    serverList.add("192.168.1.2");
    serverList.add("192.168.1.3");
    serverList.add("192.168.1.4");
    serverList.add("192.168.1.5");
}
private static Integer index = 0;

/**
 * 随机路由算法
 */
public static String randomOneByOne() {
    // 复制遍历用的集合，防止操作中集合有变更
    List<String> tempList = new ArrayList<>(serverList.size());
    tempList.addAll(serverList);
    String server = "";
    synchronized (index) {
        index++;
        if (index == tempList.size()) {
            index = 0;
        }
        server = tempList.get(index);
    }
    return server;
}
```

由代码里可以看出来，为了保证轮询，必须记录上次访问的位置，为了让在并发情况下不出现问题，还必须在使用位置记录时进行加锁，很明显这种互斥锁增加了性能开销。

依旧使用上面的测试代码测试10w次请求负载情况。

```
IP:192.168.1.3 · 次数 : 25000
IP:192.168.1.2 · 次数 : 25000
IP:192.168.1.5 · 次数 : 25000
IP:192.168.1.4 · 次数 : 25000
```

3. 轮询加权

上面演示了轮询方式，还记的一开始提出的熊4比较胖抗击打能力强，可以承受别人2倍的挨打次数嘛？上面两种方式都没有体现出来熊 4 的这个特点，熊 4 窃喜，不痛不痒。但是熊 1 到 熊 3 已经在崩溃的边缘，不行，我们必须要让胖着多打，能者多劳，提高整体性能。

```
/** 服务器列表 */
private static HashMap<String, Integer> serverMap = new HashMap<>();
static {
    serverMap.put("192.168.1.2", 2);
    serverMap.put("192.168.1.3", 2);
    serverMap.put("192.168.1.4", 2);
    serverMap.put("192.168.1.5", 4);
}
private static Integer index = 0;

/**
 * 加权路由算法
 */
public static String oneByOneWithWeight() {
    List<String> tempList = new ArrayList();
    HashMap<String, Integer> tempMap = new HashMap<>();
    tempMap.putAll(serverMap);
    for (String key : serverMap.keySet()) {
        for (int i = 0; i < serverMap.get(key); i++) {
```

```
        tempList.add(key);
    }
}
synchronized (index) {
    index++;
    if (index == tempList.size()) {
        index = 0;
    }
    return tempList.get(index);
}
}
```

这次记录下了每台服务器的整体性能，给出一个数值，数值越大，性能越好。可以承受的请求也就越多，可以看到服务器 192.168.1.5 的性能为 4，是其他服务器的两倍，依旧 10 w 请求测试。

```
IP:192.168.1.3 · 次数：20000
IP:192.168.1.2 · 次数：20000
IP:192.168.1.5 · 次数：40000
IP:192.168.1.4 · 次数：20000
```

192.168.1.5 承担了 2 倍的请求。

4. 随机加权

随机加权的方式和轮询加权的方式大致相同，只是把使用互斥锁轮询的方式换成了随机访问，按照概率论来说，访问量增多时，服务访问也会达到负载均衡。

```
/** 服务器列表 */
private static HashMap<String, Integer> serverMap = new HashMap<>();
```



```
static {
    serverMap.put("192.168.1.2", 2);
    serverMap.put("192.168.1.3", 2);
    serverMap.put("192.168.1.4", 2);
    serverMap.put("192.168.1.5", 4);
}
/**
 * 加权路由算法
 */
public static String randomWithWeight() {
    List<String> tempList = new ArrayList();
    HashMap<String, Integer> tempMap = new HashMap<>();
    tempMap.putAll(serverMap);
    for (String key : serverMap.keySet()) {
        for (int i = 0; i < serverMap.get(key); i++) {
            tempList.add(key);
        }
    }
    int randomInt = new Random().nextInt(tempList.size());
    return tempList.get(randomInt);
}
```

依旧 10 w 请求测试, 192.168.1.5 的权重是其他服务器的近似两倍,

```
IP:192.168.1.3 · 次数 : 19934
IP:192.168.1.2 · 次数 : 20033
IP:192.168.1.5 · 次数 : 39900
IP:192.168.1.4 · 次数 : 20133
```

5. IP-Hash

上面的几种方式要么使用随机数，要么使用轮询，最终都达到了请求的负载均衡。但是也有一个很明显的缺点，就是同一个用户的多次请求很有可能不是同一个服务进行处理的，这时问题来了，如果你的服务依赖于 session，那么因为服务不同，session 也会丢失，不是我们想要的，所以出现了一种根据请求端的 ip 进行哈希计算来决定请求到哪一台服务器的方式。这种方式可以保证同一个用户的请求落在同一个服务上。

```
private static List<String> serverList = new ArrayList<>();
static {
    serverList.add("192.168.1.2");
    serverList.add("192.168.1.3");
    serverList.add("192.168.1.4");
    serverList.add("192.168.1.5");
}

/**
 * ip hash 路由算法
 */
public static String ipHash(String ip) {
    // 复制遍历用的集合，防止操作中集合有变更
    List<String> tempList = new ArrayList<>(serverList.size());
    tempList.addAll(serverList);
    // 哈希计算请求的服务器
    int index = ip.hashCode() % serverList.size();
    return tempList.get(Math.abs(index));
}
```

6. 总结

上面的四种方式看似不错，那么这样操作下来真的体现了一开始说的负载均衡吗？答案是不一定的。就像上面的最后一个提问。

又或者每次出手的力度不同，有重有轻，恰巧熊 4 总是承受这种大力度啪啪打脸，熊 4 即将不省熊事，还要继续打它吗？

服务器也是这个道理，每次请求进行的操作对资源的消耗可能是不同的。比如说某些操作它对 CPU 的使用就是比较高，也很正常。**所以负载均衡有时不能简单的通过请求的负载来作为负载均衡的唯一依据**。还可以结合服务的当前连接数量、最近响应时间等维度进行总体均衡，总而言之，就是为了达到资源使用的负载均衡。

----- END -----


一个开源项目的自我介绍

关注视频号“开开开源”

带你了解更多开源知识





 觉得不错，请点个在看呀

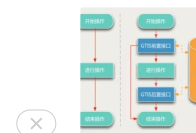
[阅读原文](#)

喜欢此内容的人还喜欢

分布式系统互斥性与幂等性问题的分析与解决

顶级架构师

云原生应用发布组件 Triton 开源之旅



分布式实验室



Hadoop 生态里，为什么 Hive 活下来了？

InfoQ

