文 负载均衡算法及手段 (/a/1190000004492447)

xixicat (https://segmentfault.com/u/xixicat) 3 天前发布

> 分布式理论系列

- 从ACID到CAP到BASE (https://segmentfault.com/a/1190000004468442)
- 2PC到3PC到Paxos到Raft到ISR (https://segmentfault.com/a/1190000004474543)
- 复制、分片和路由 (https://segmentfault.com/a/1190000004485355)
- 副本更新策略 (https://segmentfault.com/a/1190000004480546)
- 负载均衡算法及手段 (https://segmentfault.com/a/1190000004492447)

序

本文主要讲述负载均衡的一些基本东西。

相关知识点

冷备与热备

- 冷备份(cool standby)，指配备平时不运行的备用设备，当运行设备发生故障时，使用备用设备替换。
- 热备份(hot standby)，指在设备运行的同时运行备用设备，当运行设备发生故障时，能够自动替换备用设备。

fail-over与fail-back

- fail-over，在空余结构中，停止运行设备，使用备用设备进行工作的过程称为替换，英文称为fail-over或者switch-over。
- fail-back，替换后再次恢复到原来的运行设备，也就是从运行状态的备用设备再切换到原来的运行设备的过程，称为回退，英文称为fail-back或switch-back。

冗余类型

- 1.主备方式(Active-Standby)
  准备两台路由器，其中一台作为正常运行业务的活跃设备(active)，也可以称为主设备(master)或者首要设备(primary)。另一台作为发生故障时替换的备用设备(standby)，也可以称为备机(backup)、从设备(slave)、必要设备(secondary)。活跃设备和备用设备必须共享关于设备的设置信息。
- 2.双活方式(Active-Active)
  准备两台路由器，其中一台作为首要设备(primary)，另一台作为次要设备(secondary)，二者同时运行来组成冗余结构。这种方式可以通过与负载均衡设备并用或者设置DNS、客户端一侧的路由信息来达到负载均衡的目的。
- 3.集群方式(Cluster)
  在主备方式或双活方式中，使用3台以上的硬件协同组成冗余结构的方式。

L2与L3交换机

- L2基于数据链路层，L3基于网络层，具有IP分组与路由选择功能。
- L2可以通过使用VLAN分割广播域，但终端之间的数据帧交换必须位于同一VLAN范围内。 不同VLAN上的终端如有相互通信需求，则必须使用路由器。
- L3交换机与路由器均可以实现跨VLAN路由，但L3交换机多用于在以太网构筑的Intranet内部转发分组，而路由器则大多作为连接互联网与Intranet内网之间的网关来使用。

L4与L7交换机

- L4交换机
  能够支持到TCP层级访问控制的交换机，称为L4交换机。
- L7交换机
  能够基于HTTP和HTTPS这类应用层L7参数进行负载均衡等操作的产品，称为L7交换机。

负载均衡器

可以是专用设备，也可以是在通用服务器上运行的应用程序。 分散请求到拥有相同内容或提供相同服务的服务器。 专用设备一般只有以太网接口，可以说是多层交换机的一种。 负载均衡器一般会被分配虚拟IP地址，所有来自客户端的请求都是针对虚拟IP地址完成的。 负载均衡器通过负载均衡算法将来自客户端的请求转发到服务器的实际IP地址上。

负载均衡算法

```
private Map<String,Integer> serverMap = new HashMap<String,Integer>(){{
        put("192.168.1.100",1);
        put("192.168.1.101",1);
        put("192.168.1.102",4);
        put("192.168.1.103",1);
        put("192.168.1.104",1);
        put("192.168.1.105",3);
        put("192.168.1.106",1);
        put("192.168.1.107",2);
        put("192.168.1.108",1);
        put("192.168.1.109",1);
        put("192.168.1.110",1);
    }};
```

1.随机算法

- Random
  随机，按权重设置随机概率。在一个截面上碰撞的概率高，但调用量越大分布越均匀，而且按概率使用权重后也比较均匀，有利于动态调整提供者权重。

```
public void random(){
        List<String> keyList = new ArrayList<String>(serverMap.keySet());
        Random random = new Random();
        int idx = random.nextInt(keyList.size());
        String server = keyList.get(idx);
        System.out.println(server);
    }
```

- WeightRandom

```
public void weightRandom(){
        Set<String> keySet = serverMap.keySet();
        List<String> servers = new ArrayList<String>();
        for(Iterator<String> it = keySet.iterator();it.hasNext();){
            String server = it.next();
            int weithgt = serverMap.get(server);
            for(int i=0;i<weithgt;i++){
                servers.add(server);
            }
        }
        String server = null;
        Random random = new Random();
        int idx = random.nextInt(servers.size());
        server = servers.get(idx);
        System.out.println(server);
    }
```

2.轮询及加权轮询

- 轮询(Round Robbin)
  当服务器群中各服务器的处理能力相同时，且每笔业务处理量差异不大时，最适合使用这种算法。 轮循，按公约后的权重设置轮循比率。存在慢的提供者累积请求问题，比如：第二台机器很慢，但没挂，当请求调到第二台时就卡在那，久而久之，所有请求都卡在调到第二台上。

```
private Integer pos = 0;
public void roundRobin(){
        List<String> keyList = new ArrayList<String>(serverMap.keySet());
        String server = null;
        synchronized (pos){
            if(pos > keyList.size()){
                pos = 0;
            }
            server = keyList.get(pos);
            pos++;
        }
        System.out.println(server);
    }
```

- 加权轮询(Weighted Round Robbin)
  为轮询中的每台服务器附加一定权重的算法。比如服务器1权重1，服务器2权重2，服务器3权重3，则顺序为1-2-2-3-3-3-1-2-2-3-3-3- .....

```
public void weightRoundRobin(){
        Set<String> keySet = serverMap.keySet();
        List<String> servers = new ArrayList<String>();
        for(Iterator<String> it = keySet.iterator();it.hasNext();){
            String server = it.next();
            int weithgt = serverMap.get(server);
            for(int i=0;i<weithgt;i++){
               servers.add(server);
            }
        }
        String server = null;
        synchronized (pos){
            if(pos > keySet.size()){
                pos = 0;
            }
            server = servers.get(pos);
            pos++;
        }
        System.out.println(server);
    }
```

3.最小连接及加权最小连接

- 最少连接(Least Connections)
  在多个服务器中，与处理连接数(会话数)最少的服务器进行通信的算法。即使在每台服务器处理能力各不相同，每笔业务处理量也不相同的情况下，也能够在一定程度上降低服务器的负载。
- 加权最少连接(Weighted Least Connection)
  为最少连接算法中的每台服务器附加权重的算法，该算法事先为每台服务器分配处理连接的数量，并将客户端请求转至连接数最少的服务器上。

4.哈希算法

- 普通哈希

```
public void hash(){
        List<String> keyList = new ArrayList<String>(serverMap.keySet());
        String remoteIp = "192.168.2.215";
        int hashCode = remoteIp.hashCode();
        int idx = hashCode % keyList.size();
        String server = keyList.get(Math.abs(idx));
        System.out.println(server);
    }
```

- 一致性哈希
  一致性Hash，相同参数的请求总是发到同一提供者。当某一台提供者挂时，原本发往该提供者的请求，基于虚拟节点，平摊到其它提供者，不会引起剧烈变动。

5.IP地址散列

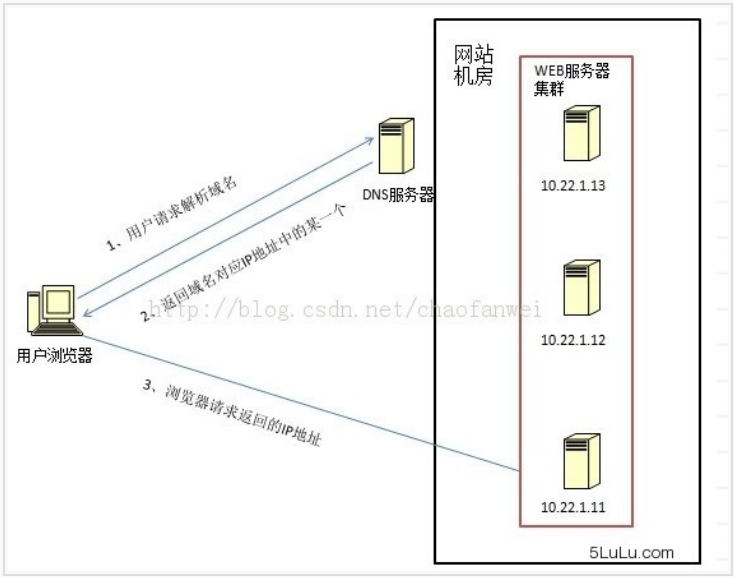
通过管理发送方IP和目的地IP地址的散列，将来自同一发送方的分组(或发送至同一目的地的分组)统一转发到相同服务器的算法。当客户端有一系列业务需要处理而必须和一个服务器反复通信时，该算法能够以流(会话)为单位，保证来自相同客户端的通信能够一直在同一服务器中进行处理。

6.URL散列

通过管理客户端请求URL信息的散列，将发送至相同URL的请求转发至同一服务器的算法。

负载均衡算法的手段(DNS->数据链路层->IP层->Http层)

1、DNS域名解析负载均衡(延迟)



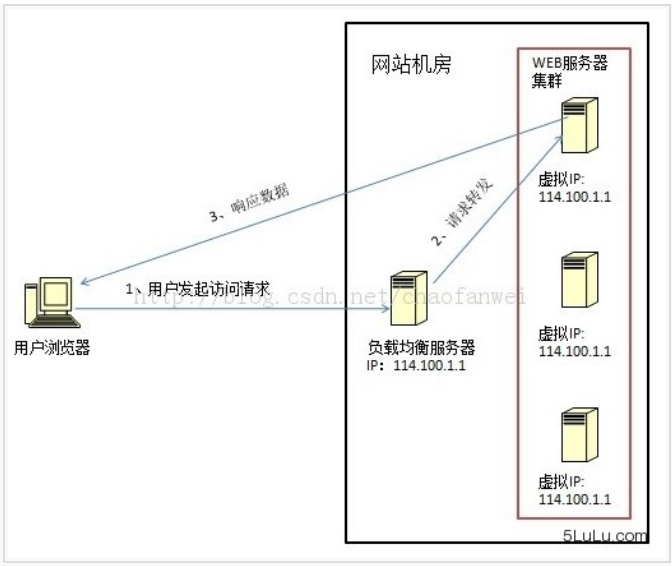
利用DNS处理域名解析请求的同时进行负载均衡是另一种常用的方案。在DNS服务器中配置多个A记录，如：www.mysite.com IN A 114.100.80.1、www.mysite.com IN A 114.100.80.2、www.mysite.com IN A 114.100.80.3.

每次域名解析请求都会根据负载均衡算法计算一个不同的IP地址返回，这样A记录中配置的多个服务器就构成一个集群，并可以实现负载均衡。

DNS域名解析负载均衡的优点是将负载均衡工作交给DNS，省略掉了网络管理的麻烦，缺点就是DNS可能缓存A记录，不受网站控制。

事实上，大型网站总是部分使用DNS域名解析，作为第一级负载均衡手段，然后再在内部做第二级负载均衡。

2、 数据链路层负载均衡(LVS)



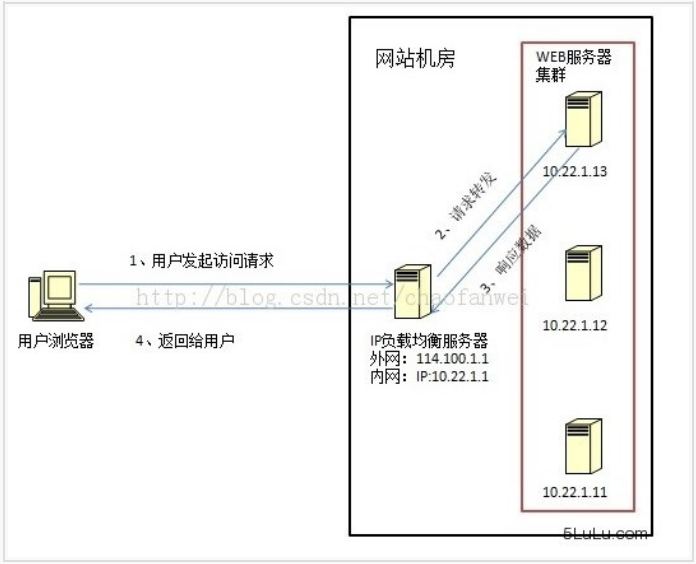
数据链路层负载均衡是指在通信协议的数据链路层修改mac地址进行负载均衡。

这种数据传输方式又称作三角传输模式，负载均衡数据分发过程中不修改IP地址，只修改目的的mac地址，通过配置真实物理服务器集群所有机器虚拟IP和负载均衡服务器IP地址一样，从而达到负载均衡，这种负载均衡方式又称为直接路由方式（DR）.

在上图中，用户请求到达负载均衡服务器后，负载均衡服务器将请求数据的目的mac地址修改为真是WEB服务器的mac地址，并不修改数据包目标IP地址，因此数据可以正常到达目标WEB服务器，该服务器在处理完数据后可以经过网管服务器而不是负载均衡服务器直接到达用户浏览器。

使用三角传输模式的链路层负载均衡是目前大型网站所使用的最广的一种负载均衡手段。在linux平台上最好的链路层负载均衡开源产品是LVS(linux virtual server)。

3、IP负载均衡(SNAT)



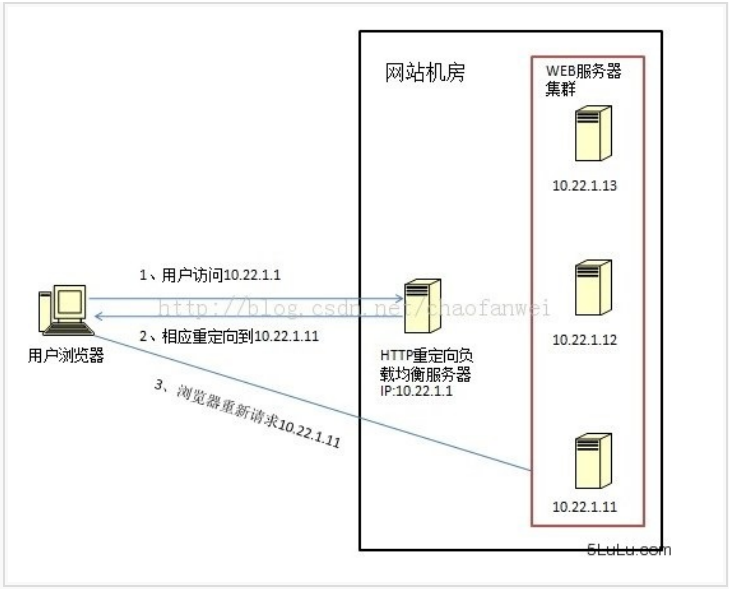
IP负载均衡：即在网络层通过修改请求目标地址进行负载均衡。

用户请求数据包到达负载均衡服务器后，负载均衡服务器在操作系统内核进行获取网络数据包，根据负载均衡算法计算得到一台真实的WEB服务器地址，然后将数据包的IP地址修改为真实的WEB服务器地址，不需要通过用户进程处理。真实的WEB服务器处理完毕后，相应数据包回到负载均衡服务器，负载均衡服务器再将数据包源地址修改为自身的IP地址发送给用户浏览器。

这里的关键在于真实WEB服务器相应数据包如何返回给负载均衡服务器，一种是负载均衡服务器在修改目的IP地址的同时修改源地址，将数据包源地址改为自身的IP，即源地址转换（SNAT），另一种方案是将负载均衡服务器同时作为真实物理服务器的网关服务器，这样所有的数据都会到达负载均衡服务器。

IP负载均衡在内核进程完成数据分发，较反向代理均衡有更好的处理性能。但由于所有请求响应的数据包都需要经过负载均衡服务器，因此负载均衡的网卡带宽成为系统的瓶颈。

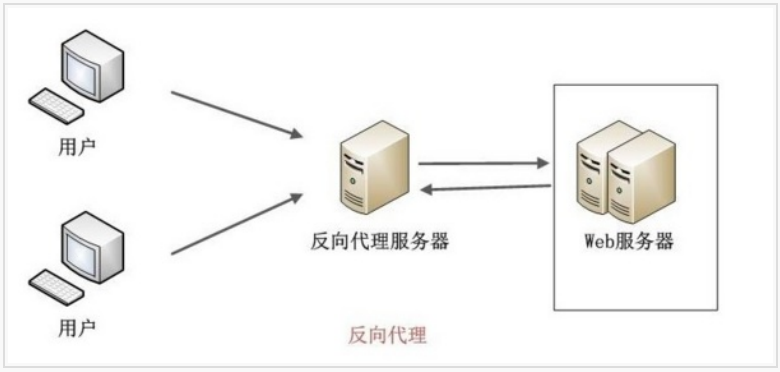
4、HTTP重定向负载均衡(少见)



HTTP重定向服务器是一台普通的应用服务器，其唯一的功能就是根据用户的HTTP请求计算一台真实的服务器地址，并将真实的服务器地址写入HTTP重定向响应中（响应状态吗302）返回给浏览器，然后浏览器再自动请求真实的服务器。

这种负载均衡方案的优点是比较简单，缺点是浏览器需要每次请求两次服务器才能拿完成一次访问，性能较差；使用HTTP302响应码重定向，可能是搜索引擎判断为SEO作弊，降低搜索排名。重定向服务器自身的处理能力有可能成为瓶颈。因此这种方案在实际使用中并不见多。

5、反向代理负载均衡(nginx)

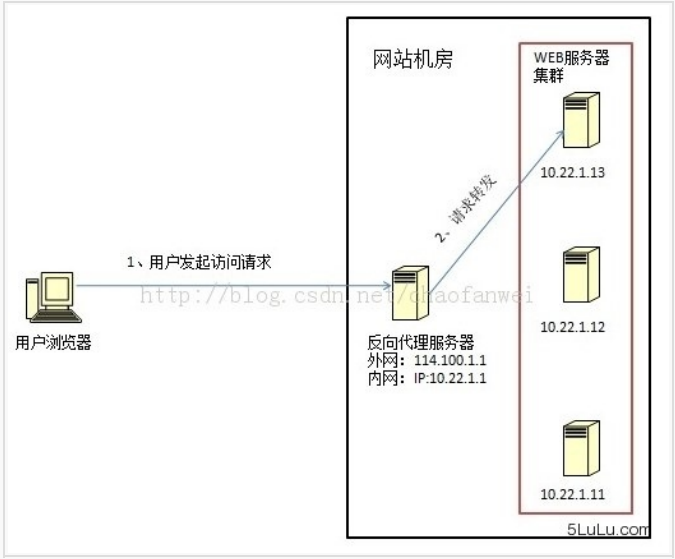


传统代理服务器位于浏览器一端，代理浏览器将HTTP请求发送到互联网上。
而反向代理服务器则位于网站机房一侧，代理网站web服务器接收http请求。

反向代理的作用是保护网站安全，所有互联网的请求都必须经过代理服务器，相当于在web服务器和可能的网络攻击之间建立了一个屏障。

除此之外，代理服务器也可以配置缓存加速web请求。当用户第一次访问静态内容的时候，静态内存就被缓存在反向代理服务器上，这样当其他用户访问该静态内容时，就可以直接从反向代理服务器返回，加速web请求响应速度，减轻web服务器负载压力。

另外，反向代理服务器也可以实现负载均衡的功能。



由于反向代理服务器转发请求在HTTP协议层面，因此也叫应用层负载均衡。优点是部署简单，缺点是可能成功系统的瓶颈。

参考

- 图解网络硬件 (http://book.douban.com/subject/25919428/)
- 分布式环境中的负载均衡策略 (http://codemacro.com/2014/08/25/lb-policy/)
- 负载均衡调度算法 (http://mp.weixin.qq.com/s?__biz=MzAxNzA1ODY2OA==&mid=201368320&idx=1&sn=de49e9836b7c34711db41c3e789de028&scene=1#rd)
- Tengine ngx_http_sysguard_module 过载保护模块使用 (http://luojianlong.blog.51cto.com/4412415/1382463)
- Nginx 简单的负载均衡配置示例[原创] (http://zyan.cc/post/306/)

3 天前发布 (/a/1190000004492447)

你可能感兴趣的文章

[原]量化投资教程:基于Spark的ADMM分布式算法在组合优化中的应用 (https://segmentfault.com/a/1190000004501379) 31 浏览

Apache Spark 的一些浅见。 (https://segmentfault.com/a/1190000003799830) 2 收藏，361 浏览

ECUG Con 邀您共议服务端开发最深度实践 (https://segmentfault.com/a/1190000004338952) 89 浏览

讨论区

本文隶属于专栏

xixicat (https://segmentfault.com/blog/xixicat)

spring boot , docker and so on

xixicat (https://segmentfault.com/u/xixicat)
作者