架构实战营 模块五

第4课：负载均衡技巧

李运华

前阿里资深技术专家（P9）

教学目标

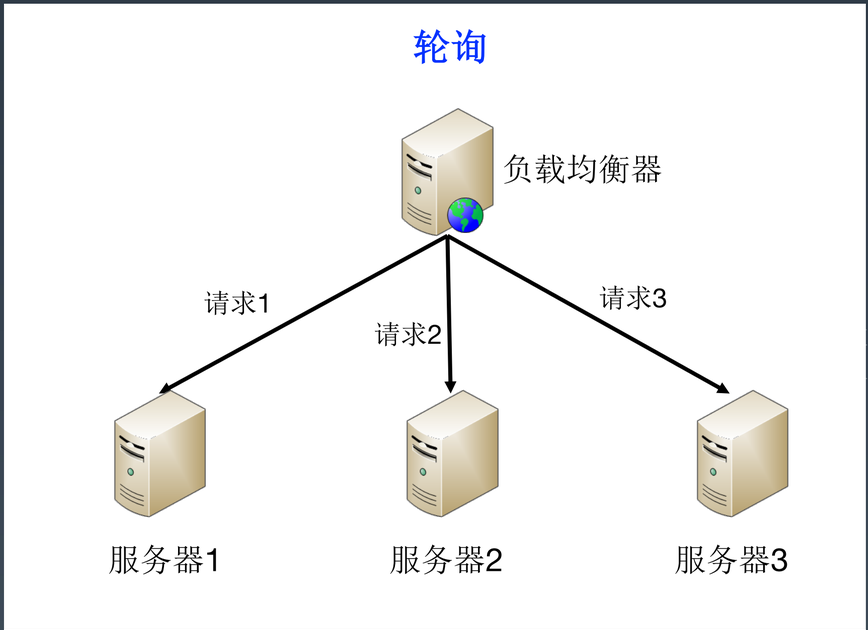
1. 掌握通用负载均衡算法和应用
2. 掌握常见业务负载均衡技巧

万变不离其宗！

目录

1. 通用负载均衡算法

负载均衡算法 - 轮询 & 随机



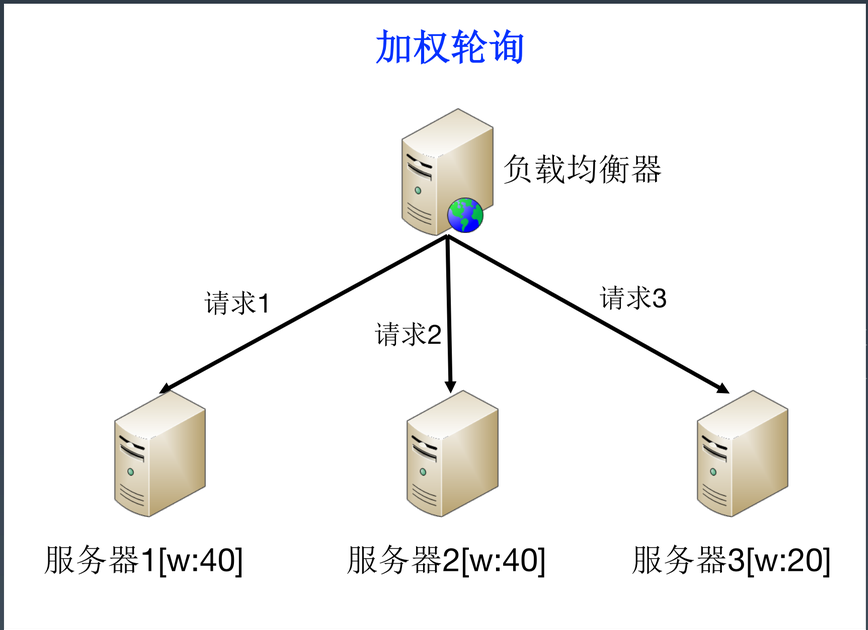
【基本原理】
轮询：将请求依次发给服务器。
随机：将请求随机发给服务器。

【适应场景】
通用，无状态的负载均衡。

【优缺点】
1. 实现简单；
2. 不会判断服务器状态，除非服务器连接丢失。

【问题场景】
1. 某个服务器当前因为触发了程序 bug 进入了死循环导致 CPU 负载很高，负载均衡系统是不感知的，还是会继续将请求源源不断地发送给它。
2. 集群中有新的机器是 32 核的，老的机器是 16 核的，负载均衡系统也是不关注的，新老机器分配的任务数是一样的。

负载均衡算法 - 加权轮询



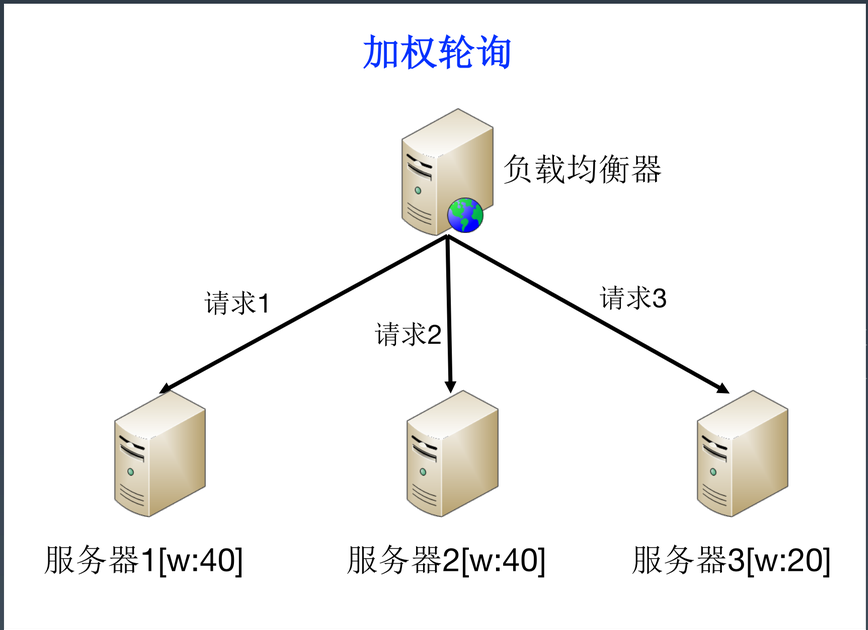
【基本原理】
按照预先配置的权重，将请求按照权重比例发送给不同的服务器。

【适应场景】
服务器的处理能力有差异，例如新老服务器搭配使用。

【优缺点】
1. 实现复杂，按照权重计算；
2. 不会判断服务器状态，除非服务器连接丢失；
3. 权重配置不合理可能导致过载。

【问题场景】
1. 2020采购的机器 CPU 核数是2019年采购的机器1倍，运维直接配置了2倍权重，结果导致新机器全部过载。

加权轮询算法



【算法1：权重 = 请求数量】

例如：给服务器1发送40个请求，然后再给服务器2发送40个请求，然后再给服务器3发送20个请求。

实现简单，但服务器资源利用可能不均衡，会出现毛刺现象，例如配置了[200, 50, 20]，但如果配置[2, 2, 1]的话，这种算法运行良好。

【算法2：权重概率】

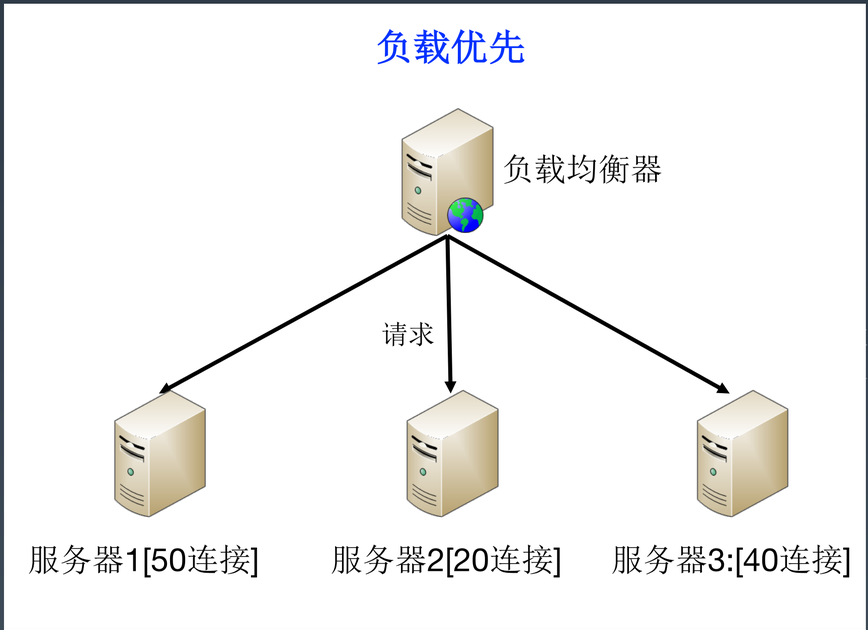
将所有服务器的权重加起来，然后计算各个服务器的分配概率，用随机数区间来做分配。

例如：服务器[0~39]，服务器2[40~79]，服务器3[80~99]，生成0~99的随机数，落入哪个区间就用那个服务器。

【算法3：权重动态调整】

Nginx 的实现，兼顾服务器故障后的慢启动，参考链接。

负载均衡算法 - 负载优先



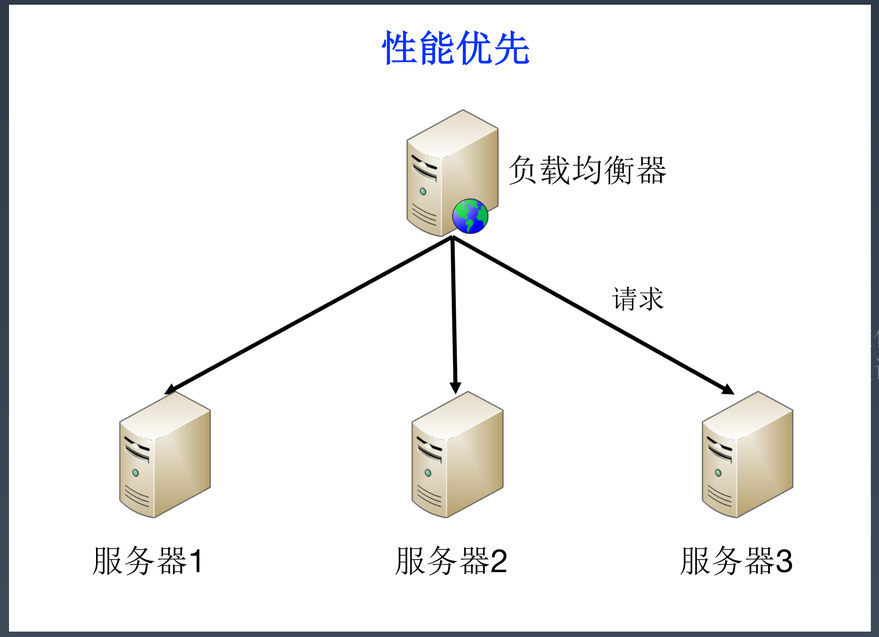
【基本原理】
负载均衡系统将任务分配给当前负载最低的服务器，这里的负载根据不同的任务类型和业务场景，可以用不同的指标来衡量。

【适应场景】
1. LVS 这种4层网络负载均衡设备，可以以“连接数”来判断服务器的状态，服务器连接数越大，表明服务器压力越大。
2. Nginx 这种7层网络负载系统，可以以“HTTP 请求数” 来判断服务器状态（Nginx 内置的负载均衡算法不支持这种方式，需要进行扩展）。

【优缺点】
1. 实现复杂，需要管理或者获取服务器状态；
2. 可以根据服务器状态进行负载均衡。

轮询算法应用广还是负载优先应用广？

负载均衡算法 - 性能优先



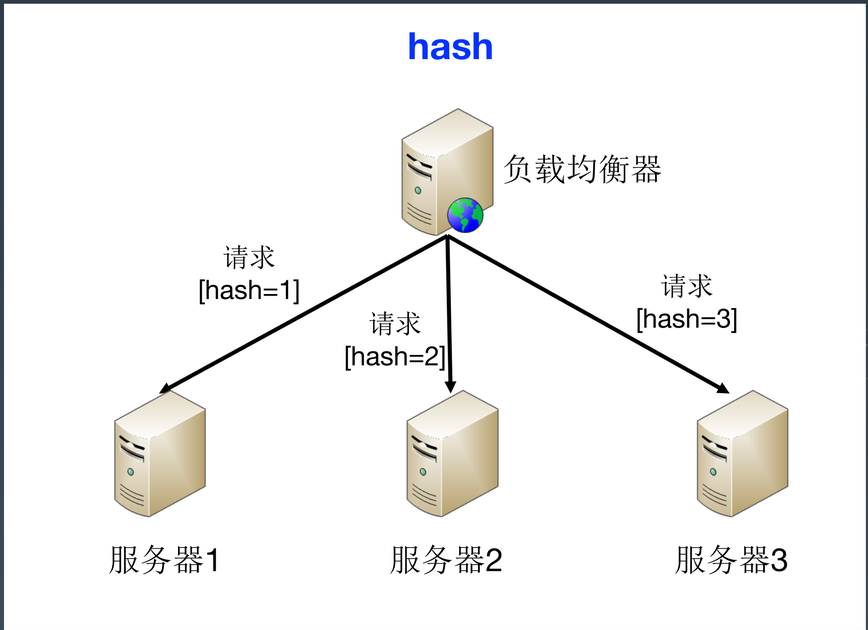
【基本原理】
负载均衡系统将任务分配给当前性能最好的服务器，主要是以响应时间作为性能衡量标准。

【适应场景】
Nginx 这种7层网络负载系统，可以以“HTTP 响应时间” 来判断服务器状态（Nginx 内置的负载均衡算法不支持这种方式，需要进行扩展）。

【优缺点】
1. 实现复杂，需要统计请求处理时间，需要耗费一定的 CPU 运算资源；
2. 可以根据性能进行负载均衡；
3. 如果服务器响应不经过负载均衡器，则不能应用这种算法。

一般会通过“采样”来统计，而不是统计每个请求

负载均衡算法 - Hash



【基本原理】
基于某个参数计算 Hash 值，将其映射到具体的服务器。

【适应场景】
1. 有状态的任务，例如购物车；
2. 任务是分片的，例如某个用户的请求只能在某台服务器处理。

【优缺点】
1. 实现简单；
2. 不会判断服务器状态，除非服务器连接丢失。

【常见 Hash 键】
1. 用户 IP 地址（session 场景）；
2. url（缓存场景）。

负载均衡算法举例 - Nginx

算法	基本实现	类型
round robin	依次将请求分配到各个后台服务器中，默认的负载均衡方式	轮询
weight	根据权重来分发请求到不同的机器中，指定轮询几率，weight 和访问比率成正比，用于后端服务器性能不均的情况。	加权轮询
IP_Hash	根据请求者 IP 的 Hash 值将请求发送到后台服务器中，可以保证来自同一 IP 的请求被打到固定的机器上，可以解决 session 问题。	Hash
url_Hash（第三方）	根据请求的 url 的 Hash 值将请求分到不同的机器中，当后台服务器为缓存的时候效率高。	Hash
fair（第三方）	根据后台响应时间来分发请求，响应时间短的分发的请求多。	性能优先

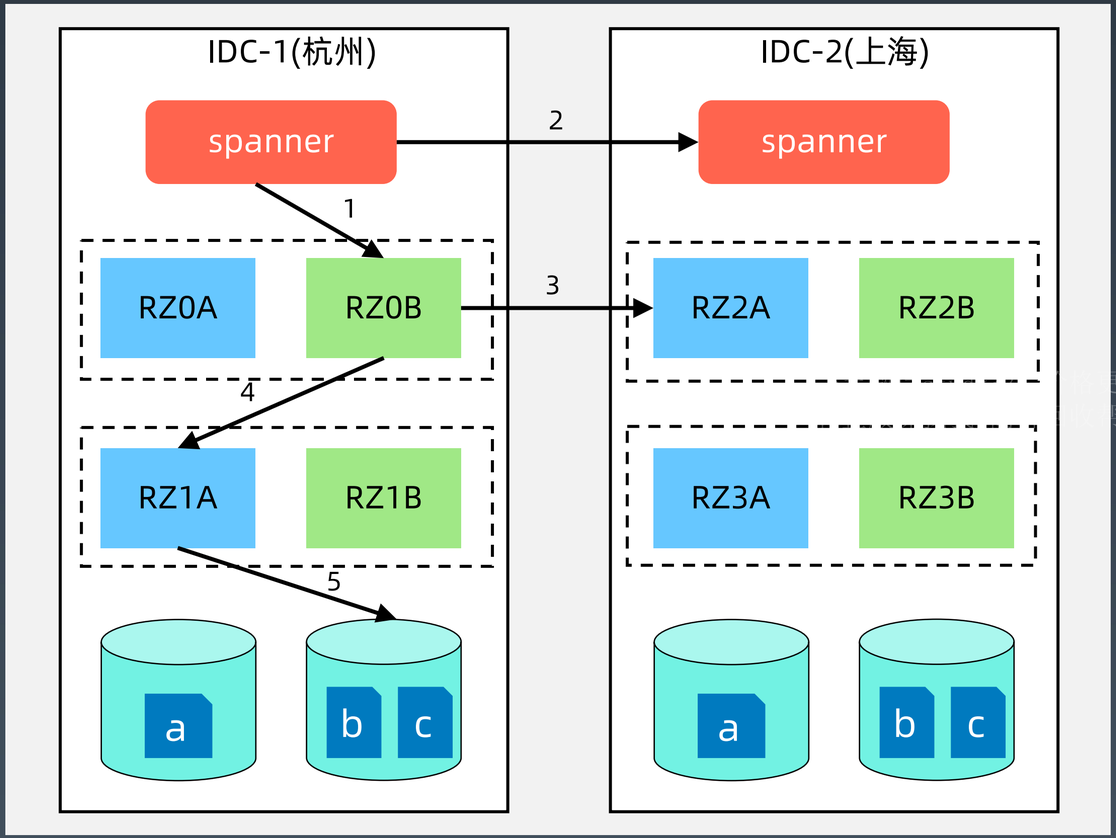
Nginx 为什么不实现基于连接数的负载优先算法？

负载均衡算法举例 - LVS

算法	基本实现	类型
Round robin	依次将请求分配到各个后台服务器中，默认的负载均衡方式	轮询
Weight Round Robin	LVS 会考虑每台服务器的性能，并给每台服务器添加一个权值，如果服务器A的权值为1，服务器 B 的权值为2，则调度器调度到服务器 B 的请求会是服务器 A 的两倍。权值越高的服务器，处理的请求越多。	加权轮询
最小连接 Least Connections	根据请求者 IP 的 Hash 值将请求发送到后台服务器中，可以保证来自同一 IP 的请求被打到固定的机器上，可以解决 session 问题。	负载优先
加权最小连接调度 Weight Least Connections	加权最小连接调度在调度新连接时尽可能使服务器的已建立连接数和其权值成比例。调度器可以自动问询真实服务器的负载情况，并动态地调整其权值。	负载优先
基于局部的最少连接 Locality-Based Least Connections	针对请求报文的目标 IP 地址的负载均衡调度，目前主要用于 Cache 集群系统。	负载优先
带复制的基于局部性的最少连接 Locality-Based Least Connections with Replication	针对目标 IP 地址的负载均衡，目前主要用于 Cache 集群系统，它与 LBLC 算法不同之处是它要维护从一个目标 IP 地址到一组服务器的映射，而 LBLC 算法维护从一个目标 IP 地址到一台服务器的映射。	负载优先
目标地址散列调度 Destination Hashing	先根据请求的目标 IP 地址，作为散列键 (Hash Key) 从静态分配的散列表找出对应的服务器，若该服务器是可用的且并未超载，将请求发送到该服务器，否则返回空。	Hash
源地址散列调度 Source Hashing	根据请求的源 IP 地址，作为散列键 (Hash Key) 从静态分配的散列表找出对应的服务器，若该服务器是可用的且并未超载，将请求发送到该服务器，否则返回空。	Hash
最短的期望的延迟调度 Shortest Expected Delay	基于 WLC 算法，计算出期望延迟来进行负载均衡。	性能优先
最少队列调度 Never Queue	无需队列，如果有 realserver 的连接数等于0就直接分配过去，不需要在进行 SED 运算。	性能优先

3. 业务级别负载均衡技巧

业务级别负载均衡介绍



通用负载均衡算法是基于请求的，业务级别的负载均衡是基于业务内容的，更灵活。
例如蚂蚁的 LDC 架构，腾讯的 SET 单元化架构。

案例：蚂蚁 LDC 架构，参考链接

【一次路由】

箭头1：对于应该在本 IDC 处理的请求，就直接映射到对应的 RZ 即可。
箭头2：不在本 IDC 处理的请求，Spanner 可以转发至其他 IDC 的 Spanner。

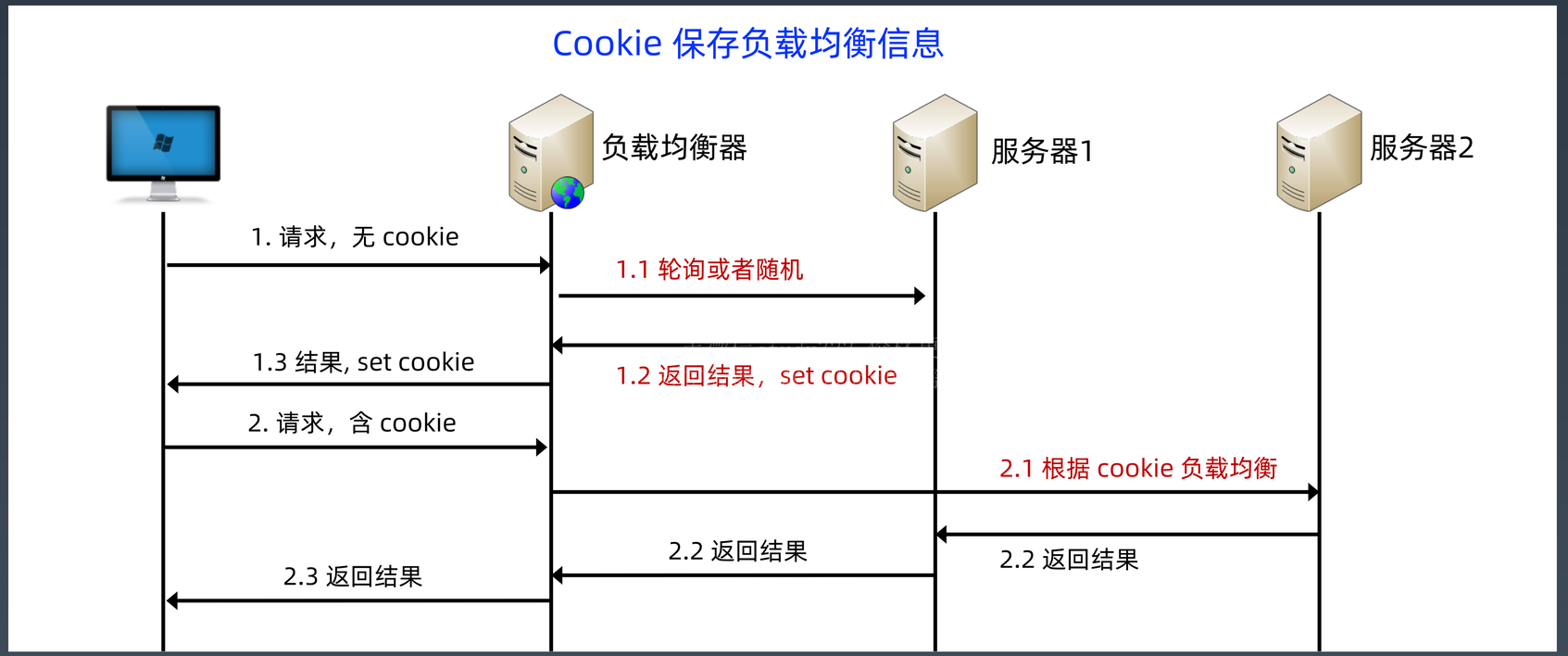
【二次路由】

有些场景来说，A 用户的一个请求可能关联了对 B 用户数据的访问，比如 A 转账给 B，A 扣完钱后要调用账务系统去增加 B 的余额。这时候就涉及到：
箭头3：跳转到其他 IDC 的 Rzone；
箭头4：跳转到本 IDC 的其他 Rzone。

【数据路由】

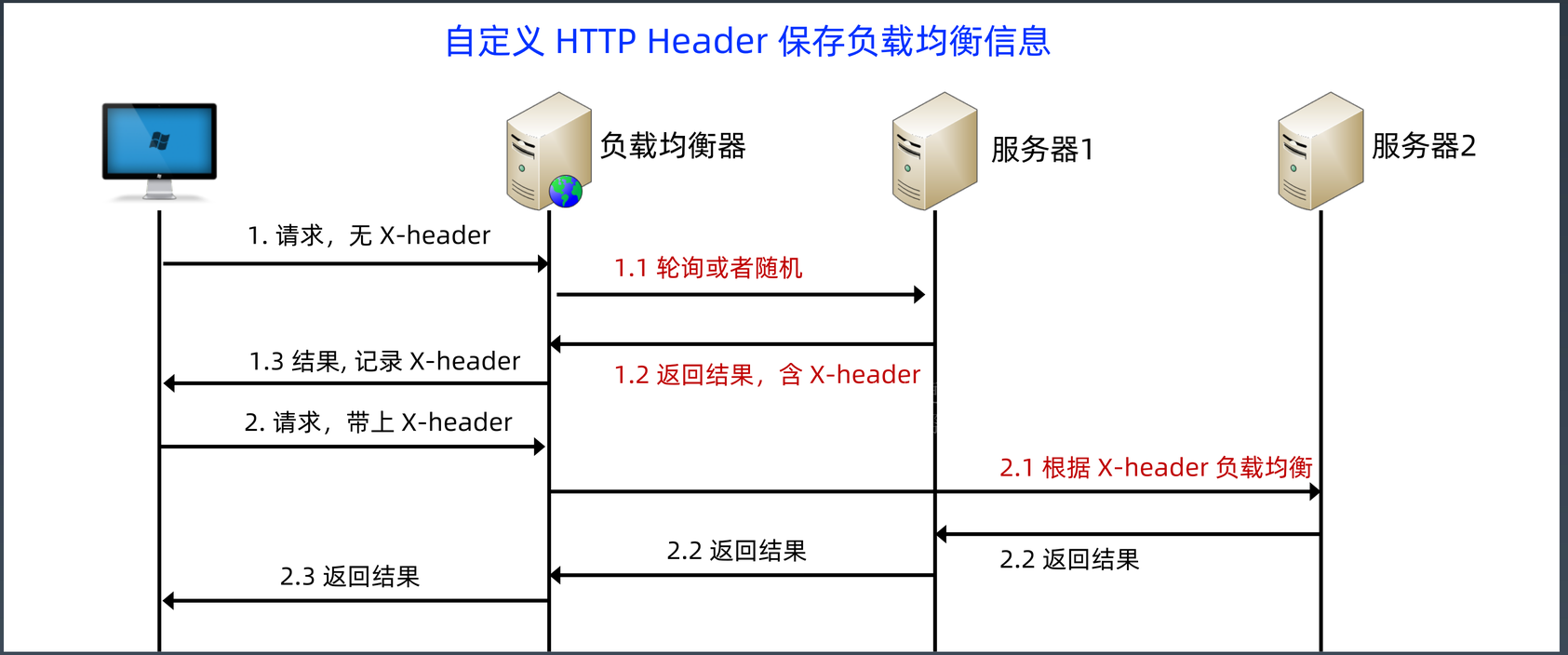
RZ 访问哪个数据库，是可以配置的，对应图中箭头5。

业务负载均衡技巧 - Cookie



应用场景：一般用于 session 保持，购物车，下单等场景。

业务负载均衡技巧 - 自定义 HTTP Header



可以通过 X- 来自定义 HTTP Header，可以服务器下发，也可以直接带上本地信息。这种用法被 IETF 在2012年6月发布的 RFC5548 中明确不推荐，虽然 RFC 已经不推荐使用带 X 前缀的 http 头部，但是不推荐不代表禁止，目前应用广泛。

应用场景：一般用于精细化的地理位置、机房级别、版本、平台、渠道等负载均衡，例如：
1）如果客户端位于加利福尼亚州山景城，则负载平衡器会添加 Header：X-Client-Geo-Location:US,Mountain View
2）X-Client-Version: 3.0.0, X-Client-Platform:iOS 11, X-Client-Channel:Huawei

业务负载均衡技巧 - HTTP query string

```
location / {
set $newIp "192.168.72.102:80";
if ($query_string ~ appversion=(1.4.0)){
set $newIp "192.168.72.102:8080";

}
proxy_pass "http://$newIp";
proxy_redirect off;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

}
```

【基本实现】

query string包含负载均衡信息，例如：

http://host:port/?userId=14167734

【优缺点】

1. 实现简单；
2. 对业务侵入较大，用这个还不如用 Cookie。

业务负载均衡技巧 - 服务器性能估算

接口性能

1. 线上业务服务器接口处理时间分布为 20~100ms；
2. 平均大约为 50ms；
3. 访问存储或者其它系统接口是主要的性能消耗点。

服务器性能

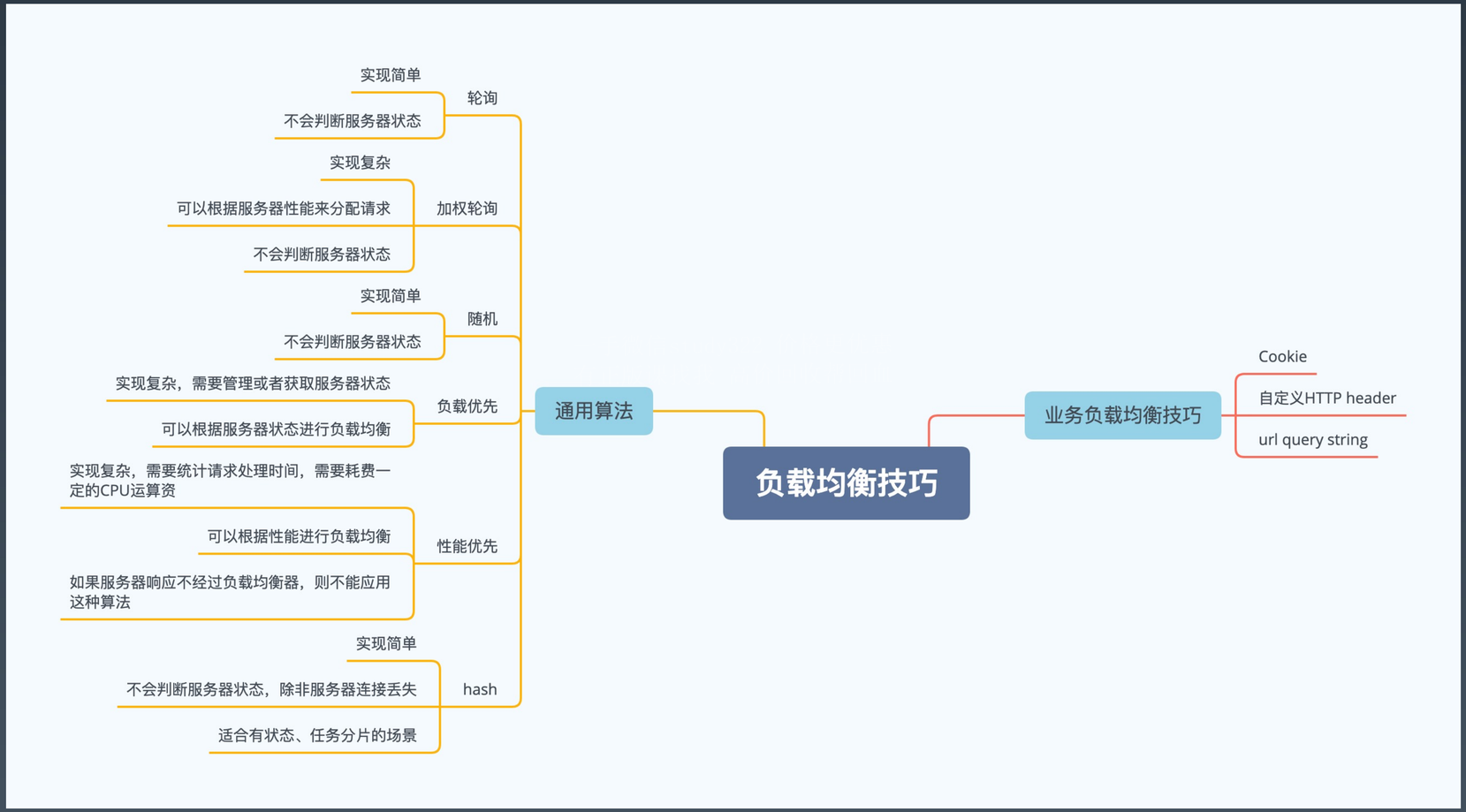
1. 线上单个服务器（32核）性能大约为 300~1000 TPS/QPS。

服务器数量

服务器数量 = (总 TPS+QPS) / 单个服务器性能。

CPU核数增加，是否能够线性提升处理能力？

本节思维导图


负载均衡技巧
通用算法
轮询
实现简单
不会判断服务器状态
加权轮询
实现复杂
可以根据服务器性能来分配请求
不会判断服务器状态
随机
实现简单
不会判断服务器状态
负载优先
实现复杂，需要管理或者获取服务器状态
可以根据服务器状态进行负载均衡
性能优先
实现复杂，需要统计请求处理时间，需要耗费一定的CPU运算资
可以根据性能进行负载均衡
如果服务器响应不经过负载均衡器，则不能应用这种算法
hash
实现简单
不会判断服务器状态，除非服务器连接丢失
适合有状态、任务分片的场景
业务负载均衡技巧
Cookie
自定义HTTP header
url query string

随堂测验

【判断题】

1. 轮询和随机太简单了，不推荐使用。
2. 加权轮询直接按照硬件配置来设置权重即可。
3. 负载优先和性能优先虽然看起来功能强大，但实现复杂，应用反而不多。
4. 自定义 HTTP header 已经被官方禁止，不能再使用。
5. Cookie 和 query string 相比，优先用 Cookie。

【思考题】

为什么看起来没那么强大的算法或者技术的应用反而会更广泛？类似的还有 TCP/IP（曾经的对手是 ATM），缓存算法中的 LRU（LFU、LRU-K……）。

Q&A

茶歇时间

八卦，趣闻，内幕.....

THANKS