Atitit 高性能架构之道 attilax著 艾龙 著

[1. 应用服务与数据隔离 2](#_Toc24112)

[2. 负载均衡你问题 2](#_Toc14806)

[2.1. 用户的请求由谁来转发到到具体的应用服务器 2](#_Toc11736)

[2.2. 有什么转发的算法 2](#_Toc16679)

[2.3. 应用服务器如何返回用户的请求 2](#_Toc27815)

[2.4. 用户如果每次访问到的服务器不一样，那么如何维护session的一致性 2](#_Toc24462)

[3. 负载均衡 2](#_Toc30099)

[3.1. http重定向。 推荐简单，直接js搞定负载均衡 3](#_Toc14980)

[3.2. DNS域名解析负载均衡 3](#_Toc7460)

[3.3. 反向代理服务器。 3](#_Toc19865)

[3.4. IP层负载均衡。 4](#_Toc11561)

[3.5. 数据链路层负载均衡。 4](#_Toc3549)

[4. 负载均衡　2、第二个问题即是集群调度算法问题，常见的调度算法有10种。 5](#_Toc14087)

[5. 负载均衡问题解决 7](#_Toc15722)

[5.1. 3、第三个问题是集群模式问题，一般3种解决方案： 7](#_Toc14734)

[5.2. 4、第四个问题是session问题，一般有4种解决方案： 7](#_Toc6247)

[6. 使用数据库连接池和线程池 9](#_Toc16000)

[7. Cache redis 前端cache等 9](#_Toc3836)

[7.1. 1、后台应用层和数据库层的缓存 9](#_Toc29987)

[7.2. 2、页面缓存 9](#_Toc7182)

[8. 全文索引 数据库全文索引 与文件全文索引 10](#_Toc15001)

[9. Msa 10](#_Toc1730)

[10. 读写分离集群 主从复制 10](#_Toc17882)

[11. Cdn动静分离 11](#_Toc3340)

[12. 跟高性能数据库 oracle取代mysql比如 11](#_Toc7287)

[13. 更高性能的存储引擎 11](#_Toc10208)

[14. Nosql 更高性能数据库 11](#_Toc24218)

[15. 存储过程 提升单台数据库能力 11](#_Toc23995)

[16. 单表分区 11](#_Toc11228)

[17. 分布式存储 11](#_Toc32411)

[17.1. 业务拆分垂直拆分 分库 11](#_Toc12261)

[17.2. 水平拆分 按照时间维度推荐 ，用户地理维度等。 11](#_Toc15527)

[18. 参考资料 12](#_Toc7409)

[18.1. 浅谈web网站架构演变过程 - 语不惊人死不休 - CSDN博客.mhtml 12](#_Toc28430)

# 应用服务与数据隔离

# 负载均衡你问题

**系统演变到这里，将会出现下面四个问题**：

## 用户的请求由谁来转发到到具体的应用服务器

## 有什么转发的算法

## 应用服务器如何返回用户的请求

## 用户如果每次访问到的服务器不一样，那么如何维护session的一致性

# 负载均衡

## ****http重定向****。 推荐简单，直接js搞定负载均衡

1. HTTP重定向就是应用层的请求转发。用户的请求其实已经到了HTTP重定向负载均衡服务器，服务器根据算法要求用户重定向，用户收到重定向请求后，再次请求真正的集群

　　　　　　优点：简单。

　　　　　　缺点：性能较差。

## ****DNS域名解析负载均衡****

1. 。DNS域名解析负载均衡就是在用户请求DNS服务器，获取域名对应的IP地址时，DNS服务器直接给出负载均衡后的服务器IP。

　　　　　　优点：交给DNS，不用我们去维护负载均衡服务器。

　　　　　　缺点：当一个应用服务器挂了，不能及时通知DNS，而且DNS负载均衡的控制权在域名服务商那里，网站无法做更多的改善和更强大的管理。

## ****反向代理服务器****。

1. 在用户的请求到达反向代理服务器时（已经到达网站机房），由反向代理服务器根据算法转发到具体的服务器。常用的apache，nginx都可以充当反向代理服务器。

　　　　　　优点：部署简单。

　　　　　　缺点：代理服务器可能成为性能的瓶颈，特别是一次上传大文件。

## ****IP层负载均衡****。

1. 在请求到达负载均衡器后，负载均衡器通过修改请求的目的IP地址，从而实现请求的转发，做到负载均衡。

　　　　　　优点：性能更好。

　　　　　　缺点：负载均衡器的宽带成为瓶颈。

## ****数据链路层负载均衡****。

1. 在请求到达负载均衡器后，负载均衡器通过修改请求的mac地址，从而做到负载均衡，与IP负载均衡不一样的是，当请求访问完服务器之后，直接返回客户。而无需再经过负载均衡器。

　　2、第二个问题即是集群调度算法问题，常见的调度算法有10种。

　　　　1、**rr 轮询调度算法**。顾名思义，轮询分发请求。

　　　　　　优点：实现简单

　　　　　　缺点：不考虑每台服务器的处理能力

　　　　2、**wrr 加权调度算法**。我们给每个服务器设置权值weight，负载均衡调度器根据权值调度服务器，服务器被调用的次数跟权值成正比。

　　　　　　优点：考虑了服务器处理能力的不同

　　　　3、**sh 原地址散列**：提取用户IP，根据散列函数得出一个key，再根据静态映射表，查处对应的value，即目标服务器IP。过目标机器超负荷，则返回空。

　　　　4、**dh 目标地址散列**：同上，只是现在提取的是目标地址的IP来做哈希。

　　　　　　优点：以上两种算法的都能实现

# 负载均衡　2、第二个问题即是集群调度算法问题，常见的调度算法有10种。

　　　　1、**rr 轮询调度算法**。顾名思义，轮询分发请求。

　　　　　　优点：实现简单

　　　　　　缺点：不考虑每台服务器的处理能力

　　　　2、**wrr 加权调度算法**。我们给每个服务器设置权值weight，负载均衡调度器根据权值调度服务器，服务器被调用的次数跟权值成正比。

　　　　　　优点：考虑了服务器处理能力的不同

　　　　3、**sh 原地址散列**：提取用户IP，根据散列函数得出一个key，再根据静态映射表，查处对应的value，即目标服务器IP。过目标机器超负荷，则返回空。

　　　　4、**dh 目标地址散列**：同上，只是现在提取的是目标地址的IP来做哈希。

　　　　　　优点：以上两种算法的都能实现同一个用户访问同一个服务器。

　　　　5、**lc 最少连接**。优先把请求转发给连接数少的服务器。

　　　　　　优点：使得集群中各个服务器的负载更加均匀。

　　　　6、**wlc 加权最少连接**。在lc的基础上，为每台服务器加上权值。算法为：（活动连接数\*256+非活动连接数）÷权重 ，计算出来的值小的服务器优先被选择。

　　　　　　优点：可以根据服务器的能力分配请求。

　　　　7、**sed 最短期望延迟**。其实sed跟wlc类似，区别是不考虑非活动连接数。算法为：（活动连接数+1)\*256÷权重，同样计算出来的值小的服务器优先被选择。

　　　　8、**nq 永不排队**。改进的sed算法。我们想一下什么情况下才能“永不排队”，那就是服务器的连接数为0的时候，那么假如有服务器连接数为0，均衡器直接把请求转发给它，无需经过sed的计算。

　　　　9、**LBLC 基于局部性的最少连接**。均衡器根据请求的目的IP地址，找出该IP地址最近被使用的服务器，把请求转发之，若该服务器超载，最采用最少连接数算法。

　　　　10、**LBLCR 带复制的基于局部性的最少连接**。均衡器根据请求的目的IP地址，找出该IP地址最近使用的“服务器**组**”，注意，并不是具体某个服务器，然后采用最少连接数从该组中挑出具体的某台服务器出来，把请求转发之。若该服务器超载，那么根据最少连接数算法，在集群的**非**本服务器组的服务器中，找出一台服务器出来，加入本服务器组，然后把请求转发之。

　　3、第三个问题是集群模式问题，一般3种解决方案：

　　　　1、**NAT**：负载均衡器接收用户的请求，转发给具体服务器，服务器处理完请求返回给均衡器，均衡器再重新返回给用户。

　　　　2、**DR**：负载均衡器接收用

# 负载均衡问题解决

## 3、第三个问题是集群模式问题，一般3种解决方案：

　　　　1、**NAT**：负载均衡器接收用户的请求，转发给具体服务器，服务器处理完请求返回给均衡器，均衡器再重新返回给用户。

　　　　2、**DR**：负载均衡器接收用户的请求，转发给具体服务器，服务器出来玩请求后直接返回给用户。需要系统支持IP Tunneling协议，难以跨平台。

　　　　3、**TUN**：同上，但无需IP Tunneling协议，跨平台性好，大部分系统都可以支持。

## 4、第四个问题是session问题，一般有4种解决方案：

　　　　1、**Session Sticky**。session sticky就是把同一个用户在某一个会话中的请求，都分配到固定的某一台服务器中，这样我们就不需要解决跨服务器的session问题了，常见的算法有ip\_hash法，即上面提到的两种散列算法。

　　　　　　优点：实现简单。

　　　　　　缺点：应用服务器重启则session消失。

　　　　2、**Session Replication**。session replication就是在集群中复制session，使得每个服务器都保存有全部用户的session数据。

　　　　　　优点：减轻负载均衡服务器的压力，不需要要实现ip\_hasp算法来转发请求。

　　　　　　缺点：复制时宽带开销大，访问量大的话session占用内存大且浪费。

　　　　3、**Session数据集中存储**：session数据集中存储就是利用数据库来存储session数据，实现了session和应用服务器的解耦。

　　　　　　优点：相比session replication的方案，集群间对于宽带和内存的压力减少了很多。

　　　　　　缺点：需要维护存储session的数据库。

　　　　4、**Cookie Base**：cookie base就是把session存在cookie中，有浏览器来告诉应用服务器我的session是什么，同样实现了session和应用服务器的解耦。

　　　　　　优点：实现简单，基本免维护。

　　　　　　缺点：cookie长度限制，安全性低，宽带消耗。

**值得一提的是**：

　　nginx目前支持的负载均衡算法有wrr、sh

# 使用数据库连接池和线程池

# Cache redis 前端cache等

## 1、****后台应用层和数据库层的缓存****

　　随着访问量的增加，逐渐出现了许多用户访问同一部分内容的情况，对于这些比较热门的内容，没必要每次都从数据库读取。我们可以使用缓存技术，例如可以使用google的开源缓存技术guava或者使用memcacahe作为应用层的缓存，也可以使用redis作为数据库层的缓存。

## 2、****页面缓存****

　　除了数据缓存，还有页面缓存。比如使用HTML5的localstroage或者cookie。

**优点**：

* 减轻数据库的压力
* 大幅度提高访问速度

**缺点**：

* 需要维护缓存服务器
* 提高了编码的复杂性

# 全文索引 数据库全文索引 与文件全文索引

# Msa

# 读写分离集群 主从复制

那么如何实现数据库的读写分离呢？目前的思路将数据库进行主从拆分，所有的写操作操作主库，所有的读操作操作从库，对主库的更新操作会通过binlog同步到从库上，从而在从库也可以拿到最新的数据。如此一来，读写不再互相阻塞，性能至少提升1倍以上。就MySQL而言，主从热备的功能可以通过cobar、mycat之类的框架来完成。

**解决问题方案**：

1. 我们可以使用MYSQL自带的master+slave的方式实现主从复制。
2. 采用第三方数据库中间件，例如mycat。mycat是从cobar发展而来的，而cobar是阿里开源的数据库中间件，后来停止开发。mycat是国内比较好的mysql开源数据库分库分表中间件。

# Cdn动静分离

# 跟高性能数据库 oracle取代mysql比如

# 更高性能的存储引擎

# Nosql 更高性能数据库

# 存储过程 提升单台数据库能力

# 单表分区

# 分布式存储

## 业务拆分垂直拆分 分库

## 水平拆分 按照时间维度推荐 ，用户地理维度等。

# 参考资料

## 浅谈web网站架构演变过程 - 语不惊人死不休 - CSDN博客.mhtml