**redis集群方案-一致性hash算法**

转载 2016年08月18日 20:05:27

前奏

集群的概念早在 Redis 3.0 之前讨论了，3.0 才在源码中出现。Redis 集群要考虑的问题：

1. 节点之间怎么据的同步，如何做到数据一致性。一主一备的模式，可以用 Redis 内部实现的主从备份实现数据同步。但节点不断增多，存在多个 master 的时候，同步的难度会越大。
2. 如何做到负载均衡？请求量大的时候，如何将请求尽量均分到各个服务器节点，负载均衡算法做的不好会导致雪崩。
3. 如何做到平滑拓展？当业务量增加的时候，能否通过简单的配置即让新的 Redis 节点变为可用。
4. 可用性如何？当某些节点鼓掌，能否快速恢复服务器集群的工作能力。
5. ……

一个稳健的后台系统需要太多的考虑。

也谈一致性哈希算法（consistent hashing）

一致性hash算法可以参考着这篇完整一起看，比较容易理解：<http://blog.csdn.net/cywosp/article/details/23397179>

背景

通常，业务量较大的时候，考虑到性能的问题（索引速度慢和访问量过大），不会把所有的数据存放在一个 Redis 服务器上。这里需要将一堆的键值均分存储到多个 Redis 服务器，可以通过：

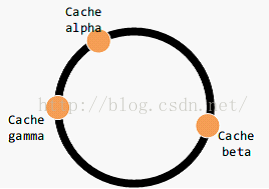
target = hash(key)\%N，其中 target 为目标节点，key 为键，N 为 Redis 节点的个数哈希取余的方式会将不同的 key 分发到不同的服务器上。

但考虑如下场景：

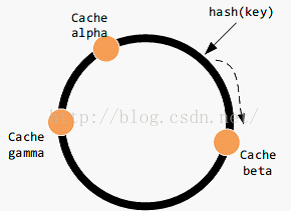
1. 业务量突然增加，现有服务器不够用。增加服务器节点后，依然通过上面的计算方式：hash(key)%(N+1) 做数据分片和分发，但之前的 key 会被分发到与之前不同的服务器上，导致大量的数据失效，需要重新写入（set）Redis 服务器。
2. 其中的一个服务器挂了。如果不做及时的修复，大量被分发到此服务器请求都会失效。

这也是两个问题。

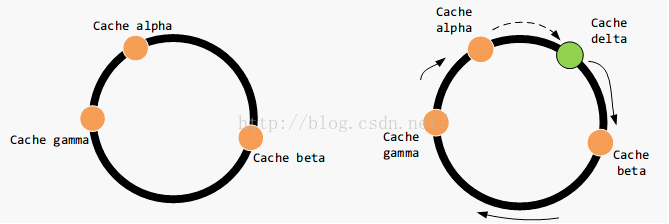
一致性哈希算法



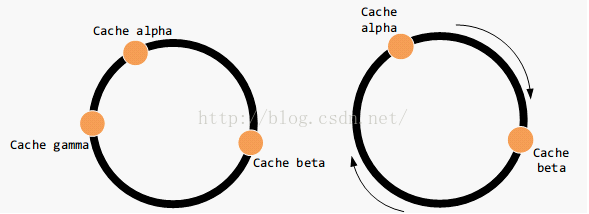
设定一个圆环上 0-2^3̂2-1 的点，每个点对应一个缓存区，每个键值对存储的位置也经哈希计算后对应到环上节点。但现实中不可能有如此多的节点，所以倘若键值对经哈希计算后对应的位置没有节点，那么顺时针找一个节点存储它。



考虑增加服务器节点的情况，该节点顺时针方向的数据仍然被存储到顺时针方向的节点上，但它逆时针方向的数据被存储到它自己。这时候只有部分数据会失效，被映射到新的缓存区。

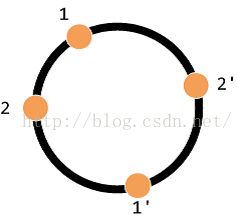


考虑节点减少的情况。该缺失节点顺时针方向上的数据仍然被存储到其顺时针方向上的节点，设为 beta，其逆时针方向上的数据会被存储到 beta 上。同样，只有有部分数据失效，被重新映射到新的服务器节点。



这种情况比较麻烦，上面图中 gamma 节点失效后，会有大量数据映射到 alpha 节点，最怕 alpha 扛不住，接下去 beta 也扛不住，这就是多米诺骨牌效应;)。这里涉及到数据平衡性和负载均衡的话题。数据平衡性是说，数据尽可能均分到每个节点上去，存储达到均衡。

虚拟节点简介



将多个虚拟节点对应到一个真实的节点，存储可以达到更均衡的效果。之前的映射方案为：

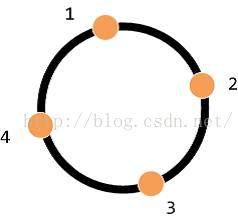
key -> node

中间多了一个层虚拟节点后，多了一层映射关系：

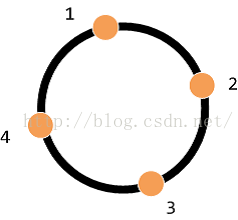
key -> <virtual node> -> node

为什么需要虚拟节点

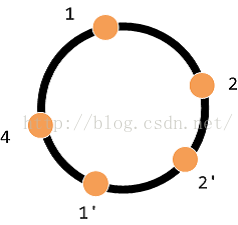
虚拟节点的设计有什么好处？假设有四个节点如下：



节点 3 突然宕机，这时候原本在节点 3 的数据，会被定向到节点 4。在三个节点中节点 4 的请求量是最大的。这就导致节点与节点之间请求量是不均衡的。



为了达到节点与节点之间请求访问的均衡，尝试将原有节点 3 的数据平均定向到到节点 1,2,4. 如此达到负载均衡的效果，如下：



总之，一致性哈希算法是希望在增删节点的时候，让尽可能多的缓存数据不失效。

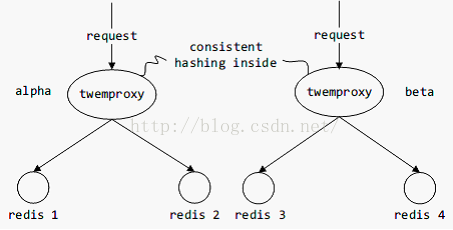
怎么实现？

一致性哈希算法，既可以在客户端实现，也可以在中间件上实现（如 proxy）。在客户端实现中，当客户端初始化的时候，需要初始化一张预备的 Redis 节点的映射表：hash(key)=> . 这有一个缺点，假设有多个客户端，当映射表发生变化的时候，多个客户端需要同时拉取新的映射表。

另一个种是中间件（proxy）的实现方法，即在客户端和 Redis 节点之间加多一个代理，代理经过哈希计算后将对应某个 key 的请求分发到对应的节点，一致性哈希算法就在中间件里面实现。可以发现，twemproxy 就是这么做的。

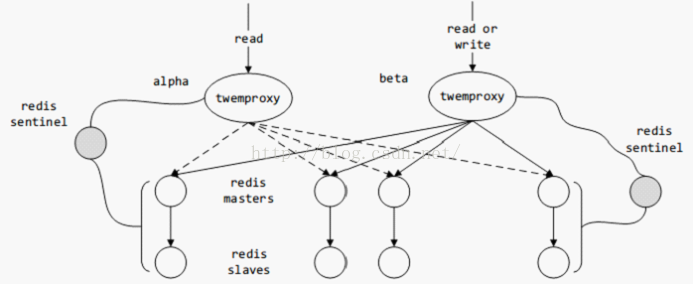
twemproxy - Redis 集群管理方案

twemproxy 是 twitter 开源的一个轻量级的后端代理，兼容 redis/memcache 协议，可用以管理 redis/memcache 集群。



twemproxy 内部有实现一致性哈希算法，对于客户端而言，twemproxy 相当于是缓存数据库的入口，它无需知道后端的部署是怎样的。twemproxy 会检测与每个节点的连接是否健康，出现异常的节点会被剔除；待一段时间后，twemproxy 会再次尝试连接被剔除的节点。

通常，一个 Redis 节点池可以分由多个 twemproxy 管理，少数 twemproxy 负责写，多数负责读。twemproxy 可以实时获取节点池内的所有 Redis 节点的状态，但其对故障修复的支持还有待提高。解决的方法是可以借助 redis sentinel 来实现自动的主从切换，当主机 down 掉后，sentinel 会自动将从机配置为主机。而 twemproxy 可以定时向 redis sentinel 拉取信息，从而替换出现异常的节点。



twemproxy 的更多细节，这里不再做深入的讨论。

Redis 官方版本支持的集群

最新版本的 Redis 也开始支持集群特性了，再也不用靠着外援过日子了。基本的思想是，集群里的每个 Redis 都只存储一定的键值对，这个“一定”可以通过默认或自定义的哈希函数来决定，当一个 Redis 收到请求后，会首先查看此键值对是否该由自己来处理，是则继续往下执行；否则会产生一个类似于 http 3XX 的重定向，要求客户端去请求集群中的另一个 Redis。

Redis 每一个实例都会通过遵守一定的协议来维护这个集群的可用性，稳定性。有兴趣可前往官网了解 Redis 集群的实现细则。