Redis热点Key发现及常见解决方案

技术小能手 (/users/o3u4mvu6rfxm2) 2018-11-23 14:16:09 浏览1393

redis (/tags/type_blog-tagid_32/) 架构 (/tags/type_blog-tagid_36/) 性能 (/tags/type_blog-tagid_455/) 模块 (/tags/type_blog-tagid_572/) 主机 (/tags/type_blog-tagid_964/) 解决方案 (/tags/type_blog-tagid_1255/) Server (/tags/type_blog-tagid_1347/) 多线程 (/tags/type_blog-tagid_1382/) slb (/tags/type_blog-tagid_1589/) 存储 (/tags/type_blog-tagid_2618/)

热点Key问题产生的原因大致有以下两种:

1、用户消费的数据远大于生产的数据（热卖商品、热点新闻、热点评论、明星直播）。

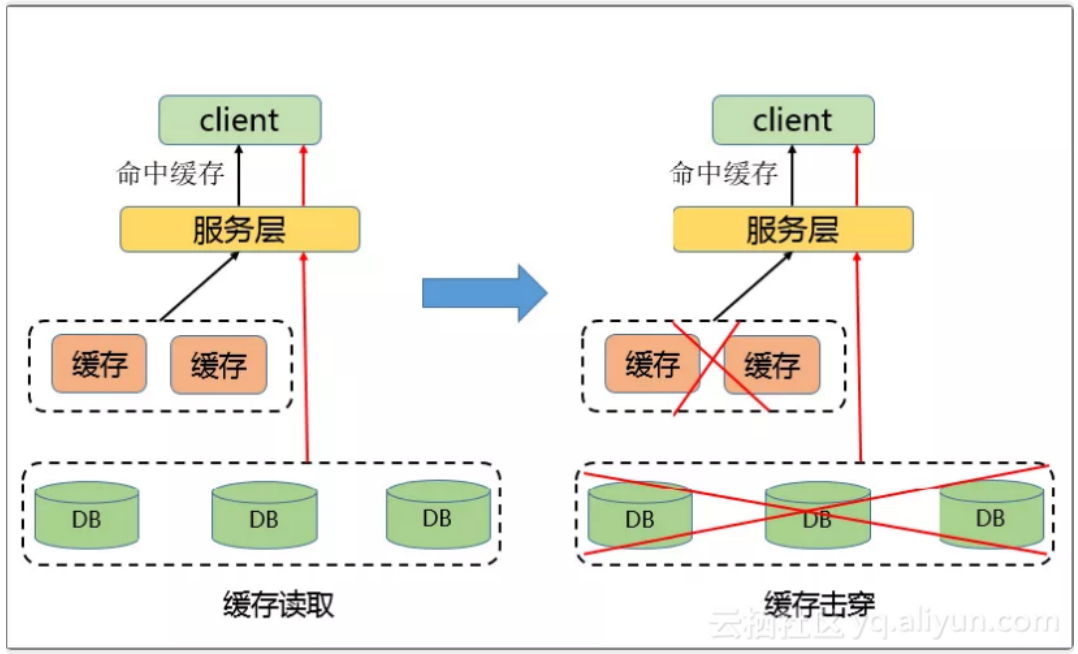
在日常工作生活中一些突发的的事件，例如：双十一期间某些热门商品的降价促销，当这其中的某一件商品被数万次点击浏览或者购买时，会形成一个较大的需求量，这种情况下就会造成热点问题。

同理，被大量刊发、浏览的热点新闻、热点评论、明星直播等，这些典型的读多写少的场景也会产生热点问题。

2、请求分片集中，超过单 Server 的性能极限。

在服务端读数据进行访问时，往往会对数据进行分片切分，此过程中会在某一主机 Server 上对相应的 Key 进行访问，当访问超过 Server 极限时，就会导致热点 Key 问题的产生。

热点Key问题的危害



1、流量集中，达到物理网卡上限。

2、请求过多，缓存分片服务被打垮。

3、DB 击穿，引起业务雪崩。

如前文讲到的，当某一热点 Key 的请求在某一主机上超过该主机网卡上限时，由于流量的过度集中，会导致服务器中其它服务无法进行。

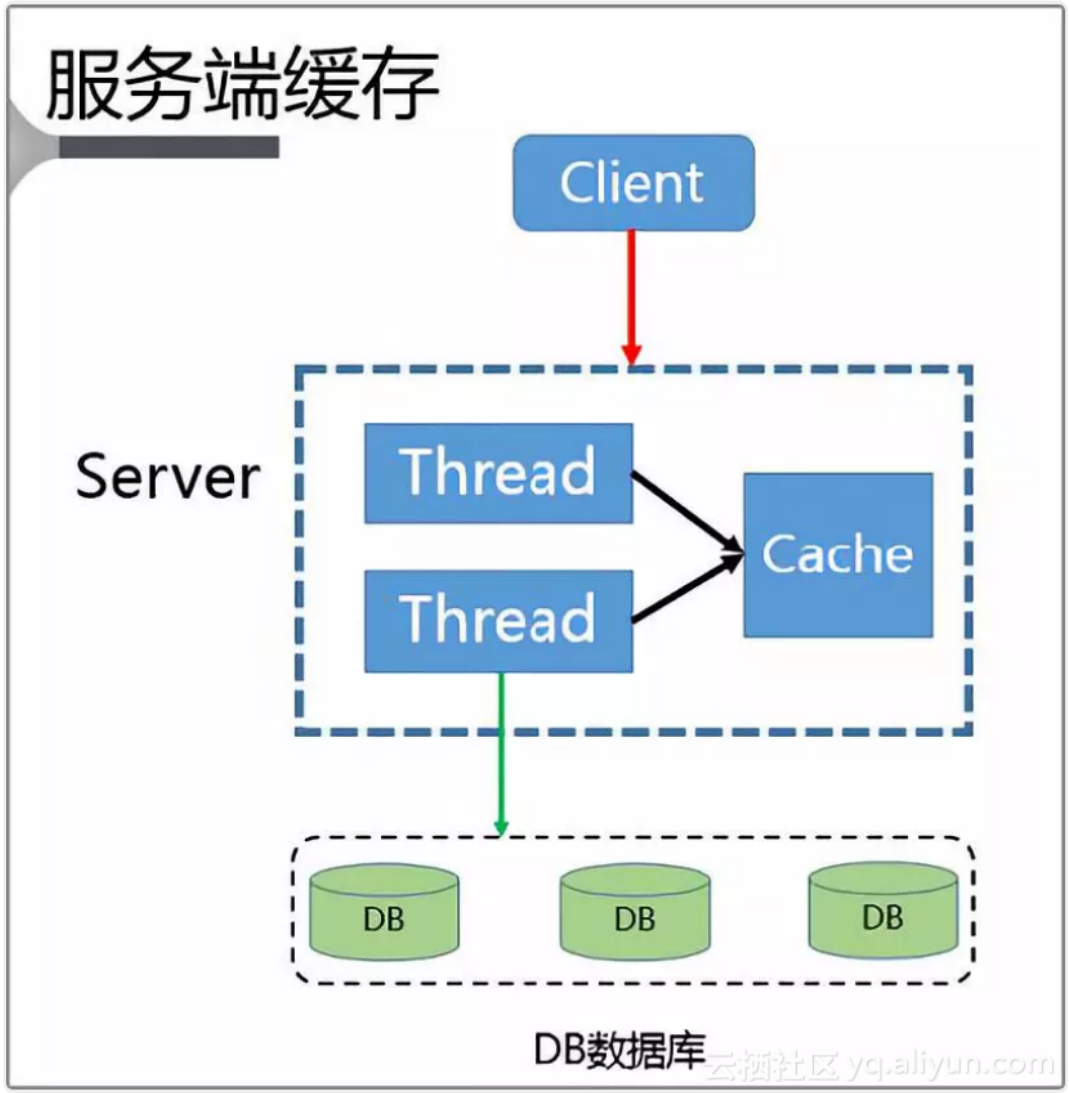
如果热点过于集中，热点 Key 的缓存过多，超过目前的缓存容量时，就会导致缓存分片服务被打垮现象的产生。

当缓存服务崩溃后，此时再有请求产生，会缓存到后台 DB 上，由于DB 本身性能较弱，在面临大请求时很容易发生请求穿透现象，会进一步导致雪崩现象，严重影响设备的性能。

解决方案

通常的解决方案主要集中在对客户端和 Server 端进行相应的改造。

1、服务端缓存方案



首先 Client 会将请求发送至 Server 上，而 Server 又是一个多线程的服务，本地就具有一个基于 Cache LRU 策略的缓存空间。

当 Server 本身就拥堵时，Server 不会将请求进一步发送给 DB 而是直接返回，只有当 Server 本身畅通时才会将 Client 请求发送至 DB，并且将该数据重新写入到缓存中。

此时就完成了缓存的访问跟重建。

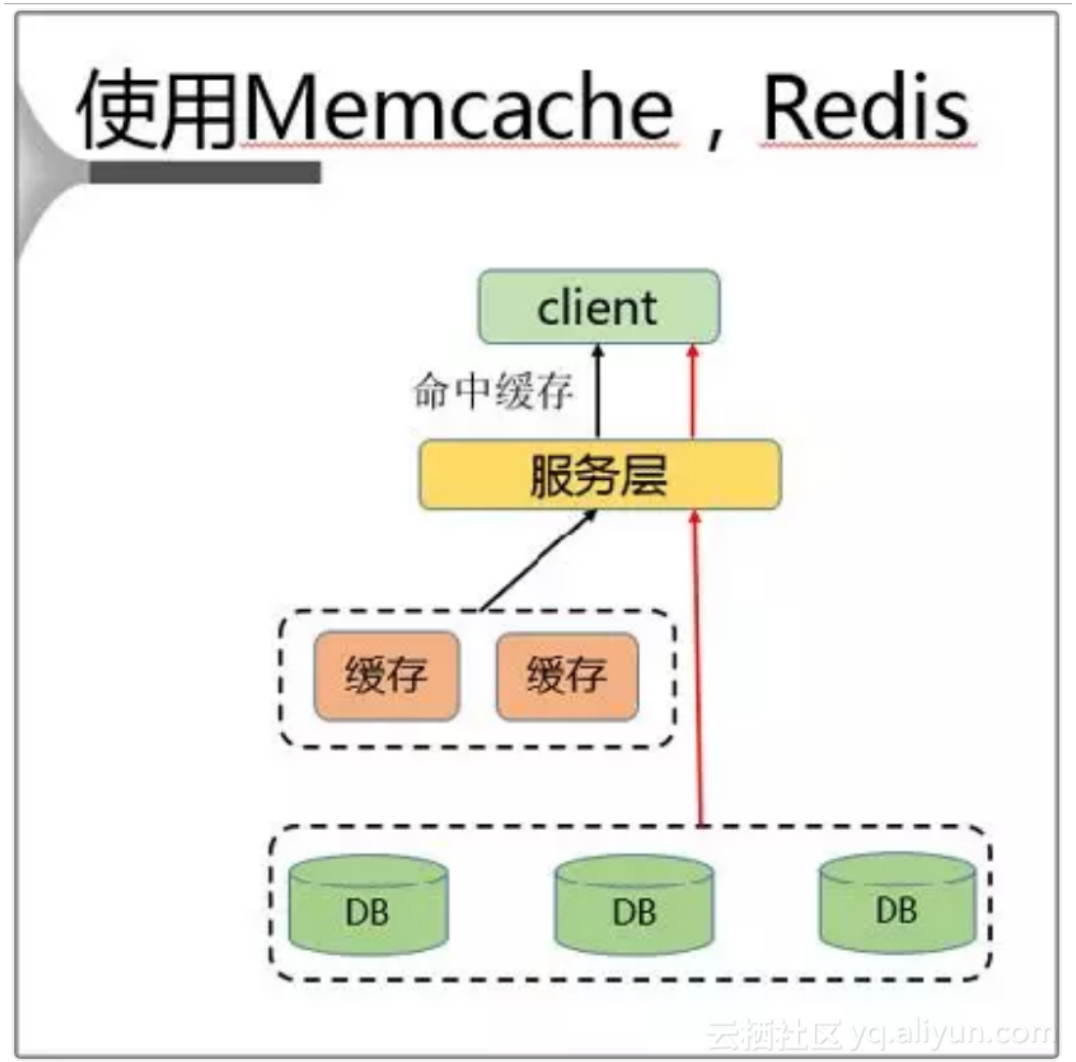
但该方案也存在以下问题：

1、缓存失效，多线程构建缓存问题

2、缓存丢失，缓存构建问题

3、脏读问题

2、使用 Memcache、Redis 方案



该方案通过在客户端单独部署缓存的方式来解决热点 Key 问题。

使用过程中 Client 首先访问服务层，再对同一主机上的缓存层进行访问。

该种解决方案具有就近访问、速度快、没有带宽限制的优点，但是同时也存在以下问题。

1、内存资源浪费

2、脏读问题

3、使用本地缓存方案

使用本地缓存则存在以下问题：

1、需要提前获知热点

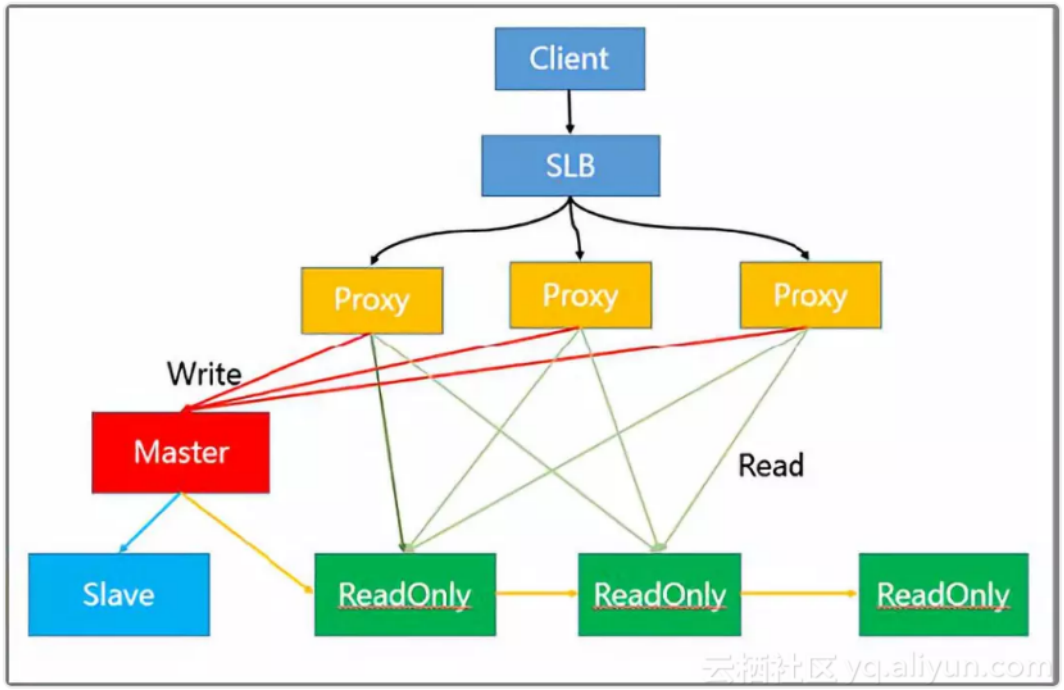
2、缓存容量有限

3、不一致性时间增长

4、热点 Key 遗漏

传统的热点解决方案都存在各种各样的问题，那么究竟该如何解决热点问题呢？

4、读写分离方案解决热读



架构中各节点的作用如下：

1、SLB 层做负载均衡

2、Proxy 层做读写分离自动路由

3、Master 负责写请求

4、ReadOnly 节点负责读请求

5、Slave 节点和 Master 节点做高可用

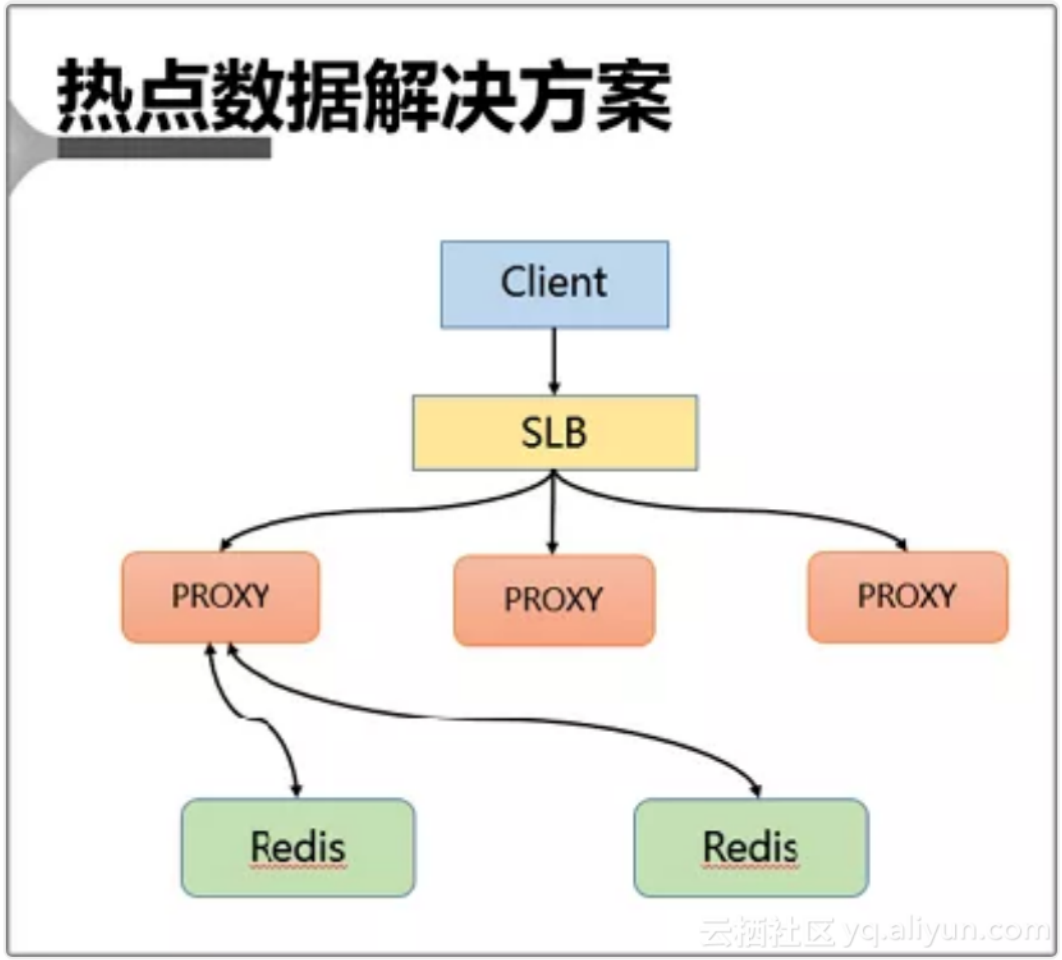
实际过程中 Client 将请求传到 SLB，SLB 又将其分发至多个 Proxy 内，通过 Proxy 对请求的识别，将其进行分类发送。

例如，将同为 Write 的请求发送到 Master 模块内，而将 Read 的请求发送至 ReadOnly 模块。

而模块中的只读节点可以进一步扩充，从而有效解决热点读的问题。

读写分离同时具有可以灵活扩容读热点能力、可以存储大量热点Key、对客户端友好等优点。

5、热点数据解决方案



该方案通过主动发现热点并对其进行存储来解决热点 Key 的问题。

首先 Client 也会访问 SLB，并且通过 SLB 将各种请求分发至 Proxy 中，Proxy 会按照基于路由的方式将请求转发至后端的 Redis 中。

在热点 key 的解决上是采用在服务端增加缓存的方式进行。

具体来说就是在 Proxy 上增加本地缓存，本地缓存采用 LRU 算法来缓存热点数据，后端 db 节点增加热点数据计算模块来返回热点数据。

Proxy 架构的主要有以下优点：

1、Proxy 本地缓存热点，读能力可水平扩展

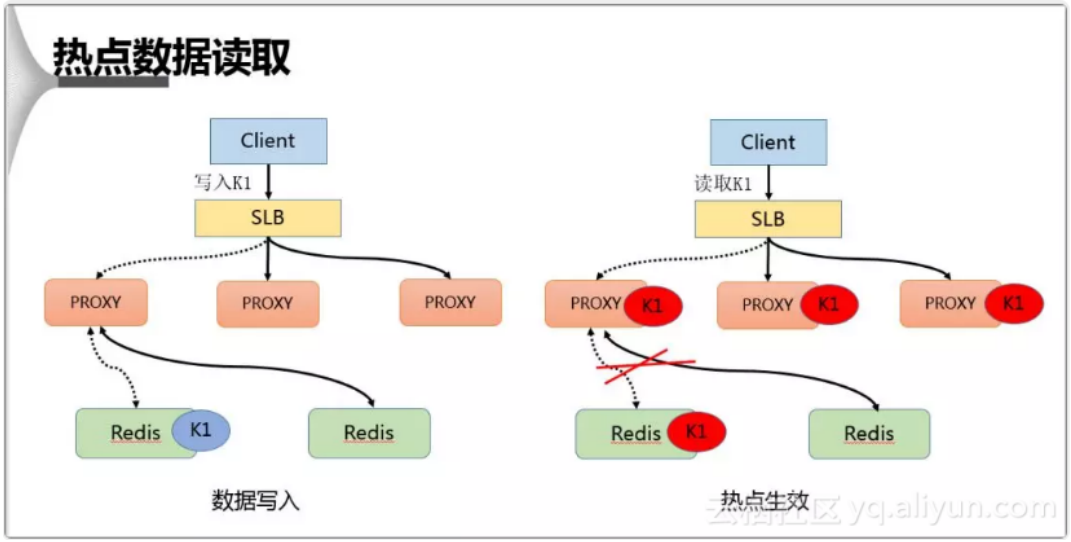
2、DB 节点定时计算热点数据集合

3、DB 反馈 Proxy 热点数据

4、对客户端完全透明，不需做任何兼容

热点 key 处理

热点数据的读取

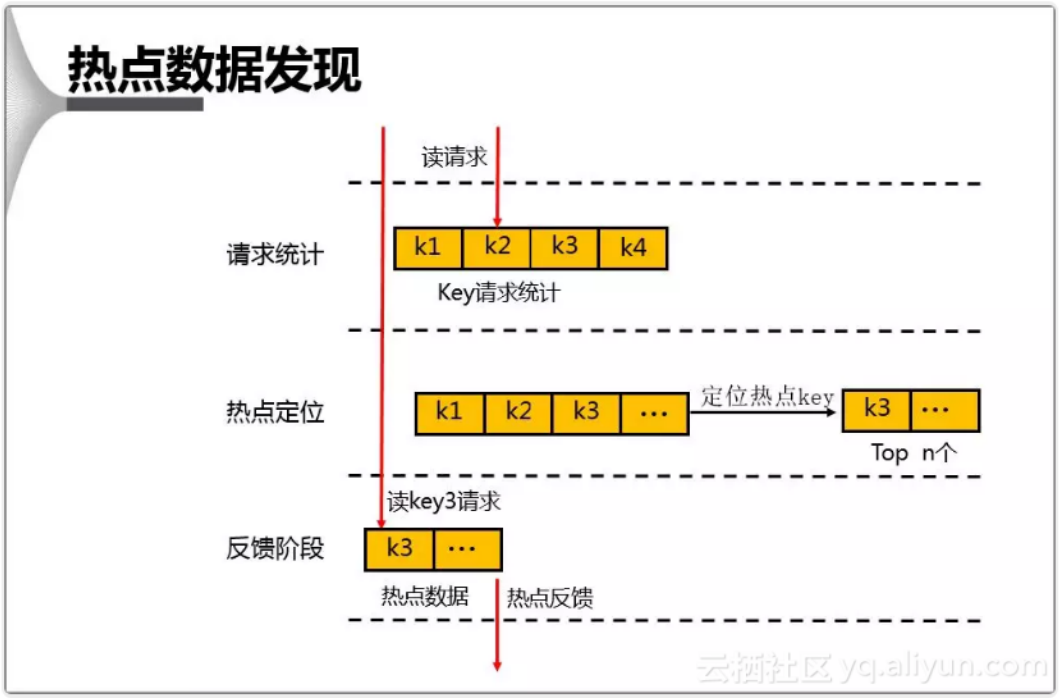


在热点 Key 的处理上主要分为写入跟读取两种形式，在数据写入过程当 SLB 收到数据 K1 并将其通过某一个 Proxy 写入一个 Redis，完成数据的写入。

假若经过后端热点模块计算发现 K1 成为热点 key 后， Proxy 会将该热点进行缓存，当下次客户端再进行访问 K1 时，可以不经 Redis。

最后由于 proxy 是可以水平扩充的，因此可以任意增强热点数据的访问能力。

热点数据的发现



对于 db 上热点数据的发现，首先会在一个周期内对 Key 进行请求统计，在达到请求量级后会对热点 Key 进行热点定位，并将所有的热点 Key 放入一个小的 LRU 链表内，在通过 Proxy 请求进行访问时，若 Redis 发现待访点是一个热点，就会进入一个反馈阶段，同时对该数据进行标记。

DB 计算热点时，主要运用的方法和优势有：

1、基于统计阀值的热点统计

2、基于统计周期的热点统计

3、基于版本号实现的无需重置初值统计方法

4、DB 计算同时具有对性能影响极其微小、内存占用极其微小等优点

方案对比

通过上述对比分析可以看出，在解决热点 Key 上较传统方法相比都有较大的提高，无论是基于读写分离方案还是热点数据解决方案，在实际处理环境中都可以做灵活的水平能力扩充、都对客户端透明、都有一定的数据不一致性。

此外读写分离模式可以存储更大量的热点数据，而基于 Proxy 的模式有成本上的优势。

原文发布时间为：2018-11-22

本文来自云栖社区合作伙伴“Java架构沉思录 (https://yq.aliyun.com/go/articleRenderRedirect?url=https%3A%2F%2Fmp.weixin.qq.com%2Fs%3F__biz%3DMzAxNjM2MTk0Ng%3D%3D%26amp%3Bmid%3D2247485739%26amp%3Bidx%3D1%26amp%3Bsn%3D6a92bb73df7e0be2d0122434223c2fcf%26amp%3Bchksm%3D9bf4b99eac8330886d56bf824927129a3fba3e6433af3f70d5f8b3ed94e4ec7b9333e5937b6b%26amp%3Bscene%3D0%26amp%3Bxtrack%3D1%23rd)”，了解相关信息可以关注“Java架构沉思录 (https://yq.aliyun.com/go/articleRenderRedirect?url=https%3A%2F%2Fmp.weixin.qq.com%2Fs%3F__biz%3DMzAxNjM2MTk0Ng%3D%3D%26amp%3Bmid%3D2247485739%26amp%3Bidx%3D1%26amp%3Bsn%3D6a92bb73df7e0be2d0122434223c2fcf%26amp%3Bchksm%3D9bf4b99eac8330886d56bf824927129a3fba3e6433af3f70d5f8b3ed94e4ec7b9333e5937b6b%26amp%3Bscene%3D0%26amp%3Bxtrack%3D1%23rd)”。