【扩展资料】如何解决业务系统的热点问题

如何解决业务系统的热点问题

我们在做各种业务研发的时候经常会碰到热点问题影响系统稳定性和性能瓶颈，例如支付系统中的热点账户进出款，电商系统中的热点商品参与秒杀，金融系统中的热点理财产品抢购等，那今天就让我带大家来一起看下我们如何解决热点问题。

首先我们要搞清楚的是热点问题必须包含两个字，**一个是热一个是点**，点表示我们在系统的业务路径上有一个地方存在性能的瓶颈，比如数据库，文件系统，网络，甚至于内存等，这个点一般有io，锁等问题构成。热表示其被访问的频率很高，就是说一个被访问频率很高的io或锁自然而然就编成我们系统业务路径上的性能瓶颈。

其次我们需要弄清楚我们的热点问题是属于读热点问题还是写热点问题，两种热点问题的处理方案完全不一样，比如我们对一个热门的秒杀商品详页的访问就属于是读热点问题，对一个秒杀商品的库存抢操作就是一个写热点问题。虽然分离了读热点问题和写热点问题，但是往往在读热点问题也需要处理写热点问题的解决方案，比如我对一个热门的秒杀商品详情的读热点问题使用了缓存解决方案，但因为商家对商品做了更新价格的东西立马需要对写热点而造成的缓存脏数据做清理的操作，因此就变成了一个读写混合的热点问题。

然后我们开始先讨论比较容易解决的读热点问题的解决方案。一般我们做系统之出使用数据库，直接对用户的请求做sql的select操作，那对于此类热点问题我们首先想到的是需要优化数据库的读操作，我们对应的查询是否走了索引，走的是否是唯一索引甚至于主健效果最佳，优化了sql性能后我们可以借助于mysql innodb的buffer做一些文章，在数据库层面就提供足够的缓冲区，加速对应的性能，实验证明，只要走的是主健或唯一索引，且innodb缓冲区足够大的情况下，mysql抗上亿的数据也是没有任何问题的，真正出问题的不是点而是热，由于访问频次太高，mysql的cpu扛不住了，这个时候我们考虑到的是将对应的读热点放到例如redis的缓存中用于卸载压力，由于redis4版本以后就可以支持cluster的集群模式，其借助分片集群的效果理论上可以扩展到1000个左右的节点，如此一来我们可以依靠缓存去解决读热点问题，一旦商家变更了读热点的数据，我们可以在业务应用使用提交后异步清除缓存的方式将redis的数据清除，这样在下一次的请求中可以依靠数据库的回源更新redis数据。

那既然我们讨论的是读热点问题，就和redis的水平扩展能力无关，因为是个热点数据，则必定会被分片路由到一个redis节点上，当热度大到连redis节点都无法承受的时候，我们可以考虑将原本的一个热点做三分拷贝，比如我们的热点key叫"miaosha_item_1"，我们可以考虑随机的生成三个key分别叫"miaosha_item_1_key_1",“miaosha_item_1_key_2”,“miaosha_item_1_key_3”，对应的value都是这个商品value本身，这样当用户请求过来后我们可以随机的生成1-3的数字以决定这次请求我们访问哪个key，这样人为的将一个热点的tps降到了原来的三分之一，以空间换时间，另外我们还可以考虑在应用服务器上做本地的cache内存，由于应用服务器本身容量有限，内存中不能放太多数据，也不能存很长时间，我们推荐使用google研发的guava cache包，提供给我们很好的lru cache队列的能力，一般本地的缓存不要设置太长时间，一是出于内存容量考虑，二是出于清理本地缓存不好清理redis，需要我们的每台应用服务感知到数据的变更，一般可以用广播型的mq消息解决，推荐rocketmq 的广播型消息，使得订阅对应商品信息更的所有应用服务器都有机会清理本地缓存。

接下来我们来解决更有挑战的写热点问题，为什么写热点问题处理起来比读热点更难的，因为读操作可以并发，我可以做到无锁操作的解决，但是写操作不行啊，从来没有说写操作可以真正意义上并发的，都需要加锁以防并发的方式写入数据库或者文件存储中，那我们怎么优化呢？

首先我们还是一样先解决点的问题再解决热的问题，针对点我们一般写操作会选用数据库之类的文件存储设备，mysql对数据存储有比较好的优化其基于写事务日志，也就是redo，undo log，然后等系统空闲的时候将数据刷入磁盘的，由于事务日志的存在，即使系统挂了再启动的时候也可以根据redo log恢复数据，那为啥写log比写数据快那么多的，因为写日志是一个顺序追加写的方式，磁盘的磁头不需要随机的移动寻找写入点，只要顺序的写下去即可，配合ssd固态硬盘，整个写入性能可以做到很高。但是磁盘操作终究是磁盘操作，我们试着可以将写入的目标点移到缓存中，比如将秒杀的库存移到redis中，这么一来，点的瓶颈的天花板瞬间就提升到了很多倍，但是一旦将数据落到没有办法保证磁盘落地能力的缓存中就需要靠一些机制去保证可靠性，不至于在缓存丢失的情况下造成超卖等灾难，我们可以依靠rocketmq异步事务型的消息保持redis和数据库之间的数据同步，解决缓存异常情况下我们可以依靠数据库恢复对应的数据。

那异步化是解决问题的最终方案吗？显然不是，异步化只是将对应的写热点问题延迟到后面去解决，不至于卡住前端的用户体验，但是一旦这个点上了起来，后端服务器和磁盘的压力还在，那我们还有什么方法去解决呢？我们都知道写入操作之所以在热点问题的情况下那么难解决，是因为写同一份数据的操作不能并发，必须得要通过竞争锁的机制去竞争以获得线程的写入权限，我们突然可以想到，锁这个东西本身就是一个耗性能的来源，想两个人要抢同一个食堂阿姨拿出来的饭，你争我抢，我抢到了吃晚了再给后面的其他人在争，在竞争的过程中所有人，所有线程的资源都被白白耗掉了，最终还是只有一个人在那个时刻可以吃到饭。那针对这种情况我们是否可以有更好的解决方案呢？还是考虑抢饭吃这个场景，在现实生活最高效的方式是什么，就是排队，大家都不要竞争，按照先到先得的方式将所有对热点的写入访问操作队列化，使用单线程的方式去队列中取得下个写入操作，然后写完后再取下一个，这样可以避免掉写锁竞争的无谓cpu和内存消耗，也可以使用单线程的方式解决，没有cpu调度切换的开销，就是我们常说的在无锁的情况下，单线程排队比多线程更高效，我们把这种解决写热点的方式叫做"缓冲入账"～

以上讲了那么多其实还有一些比较重要的点，无论是读热点还是写热点问题的解决方案，如果流量超过我的系统能力的上限，不好意思，拒绝服务将内部的等待队列先消化完再来服务您，保证我们可以高效的做完这单再接下单。

下一节：事务型消息（上）(13:54)

下一节