记Redis那坑人的HGETALL

发表于2013-01-21

世上本没有坑，摔的人多了，也便成了坑。

早就听人说过Redis的HGETALL是个坑，可我偏偏不信邪：不管什么坑，一定要自己踩上去跺两脚才肯罢休。说好听点这是不到黄河心不死，说难听点就是不见棺材不落泪。

开始程序运行的非常稳定，稳定到我想送所有说HGETALL是个坑的人一个字：呸！此时的我就像温水里的青蛙一样忘记了危险的存在，时间就这样一天一天的过去，突然有一天需求变了，我不得不把HASH数据的内容从十几个字段扩展到一百多个字段，同时使用了Pipelining一次性获取上百个HGETALL的结果。于是我掉坑里了：服务器宕机。

为什么会这样？Redis是单线程的！当它处理一个请求时其他的请求只能等着。通常请求都会很快处理完，但是当我们使用HGETALL的时候，必须遍历每个字段来获取数据，这期间消耗的CPU资源和字段数成正比，如果还用了PIPELINING，无疑更是雪上加霜。

如何解决这个问题？请容许我煞有其事的给出一个公式：

```
PERFORMANCE = CPUs / OPERATIONs
```

也就是说，此场景下为了提升性能，要么增加运算过程中的CPU数量；要么降低运算过程中的操作数量。具体来说，我大致想到了以下几种方法：

借助Memcached

Redis存储方式不做任何改变，额外的，我们借助Memcached实现一套缓存，里面存储原本需要在Redis里HGETALL的HASH，当然，由于Memcached里存储的都是字符串，所以当我们存储HASH的时候，实际上存储的是HASH序列化后的字符串，查询的时候再反序列化即可，通常Memcached客户端驱动可以透明实现序列化和反序列化的过程。此方案的优势在于因为Memcached支持多线程，所以可以让更多的CPU参与运算，同时由于不用再

遍历每一个字段，所以相应的操作会减少；当然劣势也不少，因为引入了一个新的缓存层，所以浪费了内存，增加了复杂性，另外，有时候即便我们只需要获取少数几个字段的数据，也不得不先查询完整的数据，然后再筛选，这无疑浪费了带宽。当然这种情况下我们可以直接查询Redis，但是无疑又提升了一些复杂性。

顺便说一句，Memcached支持Multiget，可以实现类似Pipelining的效果，但你要格外小心这里面有关Memcached的坑，也就是Mulitiget无底洞问题。

序列化字段冗余

Redis在存储HASH的时候，多保存一个名为「all」的字段，其内容是原HASH数据的序列化，实际查询的时候，只要HGET这个冗余字段后再反序列化即可。此方案的优势在于通过序列化字段冗余，我们把原本的HGETALL操作简化为HGET，也就是说，不再需要遍历HASH中的每一个字段，因此即便不能让多个CPU参与运算，但是却大幅降低了操作数量，所以性能的提升仍然是显著的；当然劣势也很明显，和所有的冗余方式一样，此方案浪费了大量的内存。

有人会问，这样虽然没有了遍历字段的过程，但是却增加了反序列化的过程，而反序列化的成本往往也是很高的，难道这样也能提升性能？问题的关键在于开始我们遍历字段的操作是在一个CPU上完成的，后来反序列化的操作，不管是什么语言，都可以通过多进程或多线程来保证是在多个CPU上完成的，所以性能总体上是提升的。

…

另外，很多人直觉是通过运行Redis多实例来解决问题。确实，这样可以增加运算过程中的CPU数量，有助于提升性能，但是需要注意的是，HGETALL和PIPELINING往往会让运算过程中的操作数量呈几何级爆炸式增长，相比之下，我们能增加的Redis多实例数量简直就是杯水车薪，所以本例中这种方法不能彻底解决问题。

…

坑，就是用来踩的。不用怕掉进去，当然前提是你能自己爬出来！