一致性哈希

Consistent hashing is a special kind of hashing such that when a hash table is resized, only K/n keys need to be remapped on average, where K is the number of keys, and n is the number of slots. In contrast, in most traditional hash tables, a change in the number of array slots causes nearly all keys to be remapped because the mapping between the keys and the slots is defined by a modular operation.

上述是wiki中的解释。

翻译一下：一致性哈希是一种特别种类的哈希映射：当哈希表的大小改变时，平均只有K/n个键需要重新映射(其中K是键的个数，n是槽的个数)。对于传统的基于取模的哈希表而言，当槽的个数改变时，会导致几乎所有的键都要重新映射。

还有这里说一下：一致性哈希只是说使用通用的哈希函数的一个哈希映射用法，并不是说一种特别的哈希函数。哈希函数并没有什么特别的。

现提出几个问题？

1. 为何对于传统的基于取模的哈希表而言，当槽的个数改变时，会导致几乎所有的键都要重新映射？

2. 一致性哈希的原理是什么？

3. 为何一致性哈希可以做到当槽的个数改变，多数键不用重新映射？

4. 一致性哈希的应用场景是什么？

答1：

比如现在有3个object[o1, o2, o3]，3个槽[a1, a2, a3]。

传统的做法是，先对3个object哈希得到的hashcode % 3，这样3个object就会映射到0~2中。

然后3个槽就对应a1-0/a2-1/a3-2。

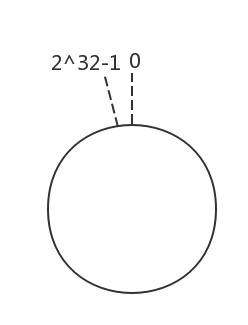
这样我们就得到了键与槽的映射关系(比如o1--a2/o2--a1/o3--a3)。

此时如果我们去掉一个槽a2。则我们就必须对所有的obejct哈希得到的hashcode进行 % 2操作，而这样做势必会导致很多键与槽的映射关系发生改变。

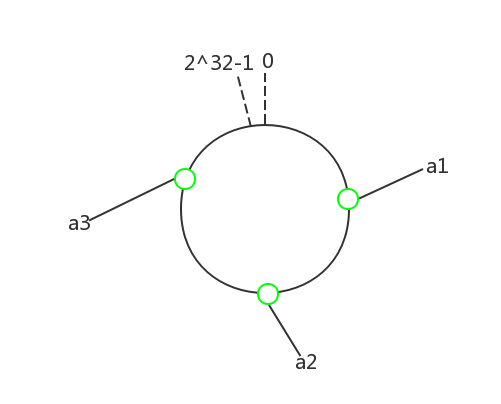
答2：

一致性哈希的原理是：

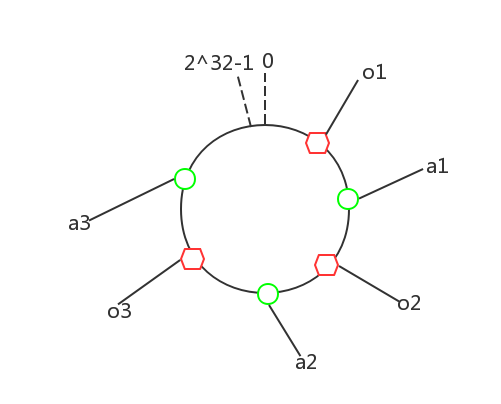
第一步：因为一般性的哈希算法的输出空间都是32位(也可以是64位)，我们先0~2^32-1这个空间想象成一个圆，如图：



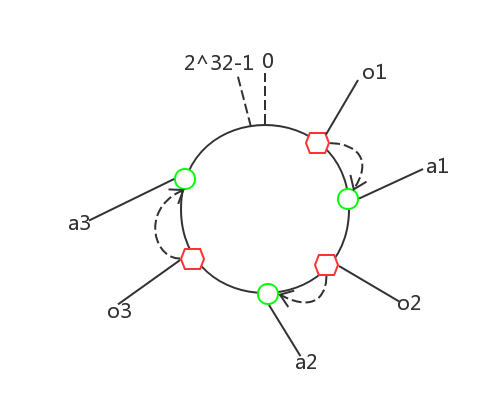
第二步：将槽通过哈希函数映射到这个圆形空间，如图：



第三步：将object通过同样的哈希函数映射到这个圆形空间，如图：

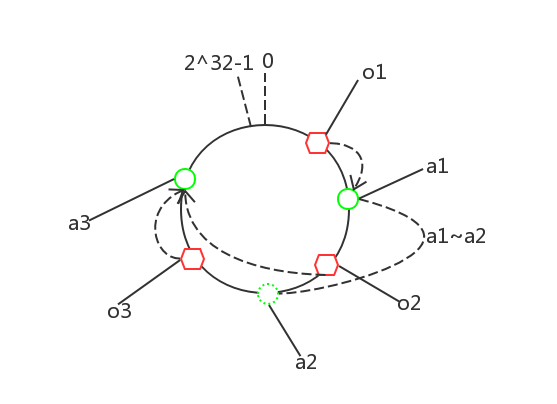


第四步：每个object顺时针走，遇到的第一个槽就作为自己的映射对象。



答3：

当某一个槽失效时，比如a2失效



则o2就会映射到a3，此时受影响的只有a1~a2之间的键。且这些键也能够找到另一个槽。

答4：

一般在cache系统中得到广泛应用。

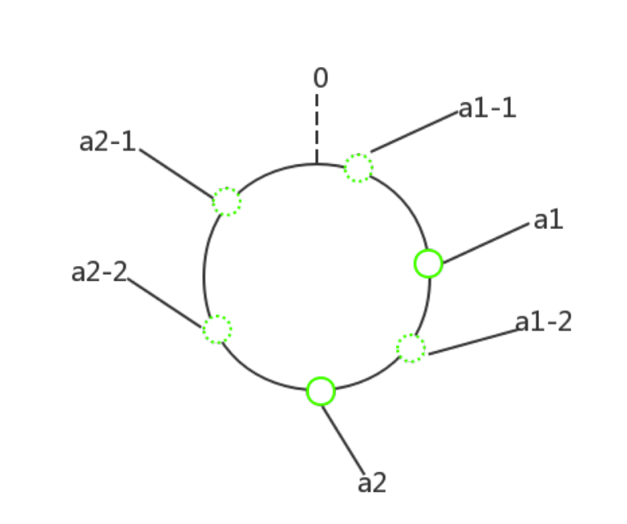
在一个cache集群中，当某台cache宕机，则传统的%N就会变成%N-1，然而这样做的话会导致很多object分发到cache的机器变了，这样的结果就是几乎完成找不到cache中的信息，这样的结果就是几乎所有的缓存失效。。。这个结果很严重。

而如果使用一致性哈希呢，如果某台cache宕机，则只有少量的映射关系会改变，导致这些缓存失效，而且这些改变是必须的，因为宕机了数据肯定丢失了。而其他大量的映射关系都没有变，所以很好地解决了这个问题。

宕机和新添机器的道理是一样的。

此时我们又重新引入一个问题：

我们把机器通过哈希函数得到的hashcode不平均怎么办？传统的%N最起码是平均的啊。我们的解决办法就是引入虚节点，虚节点就是真实节点的复制。如图：



如果只有a1和a2两个节点的话，看起来是很不平均的，现每个节点加了两个虚节点，这样a1开头的都映射到a1，a2开头的都映射到a2，这样就蛮平均的了。

哈希函数分为cryptographic hash function和non-cryptographic hash function

non-cryptographic hash function: murmur hash

cryptographic hash function: SHA-256/MD5