一致性哈希算法学习与实现

刘宇庭

中山大学数据科学与计算机学院

1. **一致性哈希算法的学习**
2. **基本场景：**假设有N个cache服务器，对服务器集群的管理，路由算法至关重要，就和负载均衡算法一样，路由算法决定着究竟该访问集群中的哪台服务器。
3. **余数哈希：**计算 object 的 hash 值，然后均匀的映射到到 N 个 cache 服务器；

hash(object) % N

但是当需要增加或者减少cache服务器的时候，这个算法的问题就暴露出来了，将导致几乎所有的cache服务器失效。

1. **一致性哈希：**一致性哈希是一种hash算法，通过一个叫做一致性hash环的数据结构实现key到cache服务器的hash映射。

**以下分步骤阐述一致性hash算法原理：**

* 1. **环形hash空间。**构造一个长度为232的整数环（这个环被称为一致性Hash环）
  2. **把cache服务器映射到hash空间。**根据节点名称的Hash值（其分布为[0, 232-1]）将缓存服务器节点放置在这个Hash环上
  3. **把对象映射到hash空间和cache服务器。**根据需要缓存的数据的Key值计算得到其Hash值（其分布也为[0, 232-1]），然后在Hash环上顺时针查找距离这个Key值的Hash值最近的服务器节点，完成Key到服务器的映射查找。
  4. **当cache服务器数量发生改变。**当新增一个cache服务器的时候，受影响的将仅是那些沿着新增cache服务器顺时针遍历直到下一个cache服务器之间的对象，其他的不会有影响；当减少一个cache服务器的时候，受影响的将仅是那些沿着新增cache服务器逆时针遍历直到下一个cache服务器之间的对象，其他的不会有影响；
  5. **虚拟节点。**“虚拟节点”（ virtual node ）是实际节点在 hash 空间的复制品（ replica ），一个实际节点对应了若干个“虚拟节点”，这个对应个数也成为“复制个数”，“虚拟节点”在 hash 空间中以 hash 值排列。

本文中一致性哈希算法的哈希值hash值的计算通过Murmurhash算法计算得到， MurmurHash 是一种非加密型哈希函数，适用于一般的哈希检索操作。 由Austin Appleby在2008年发明， 并出现了多个变种，都已经发布到了公有领域(public domain)。与其它流行的哈希函数相比，对于规律性较强的key，MurmurHash的随机分布特征表现更良好。

1. **一致性哈希算法的实现**

定义虚拟服务节点：

1. /\* 虚拟节点的结构定义 \*/
2. **struct** virtualNode {
3. string corrNodeName;
4. string virNodeName;
5. string hashCode;
7. virtualNode(string \_corrNodeName, string \_virNodeName) {
8. corrNodeName = \_corrNodeName;
9. virNodeName = \_virNodeName;
10. }
12. string toString() {
13. string str = "The virtual node: " + virNodeName + " is a virtual node of the real node: " + corrNodeName;
14. **return** str;
15. }
16. };

定义cache服务器节点：

1. /\* 真实节点的结构定义 \*/
2. **struct** hashNode {
3. string name;                        // 节点标记
4. string hashCode;                    // 节点的哈希值
5. **int** numOfVirtualhashNode;           // 虚拟节点的个数
6. **int** routeService;                   // 记录负载的路由
8. // 初始化
9. hashNode(string \_name, **int** \_numOfVirtualhashNode) {
10. name = \_name;
11. numOfVirtualhashNode = \_numOfVirtualhashNode + 1;
12. routeService = 0;
13. }
14. };

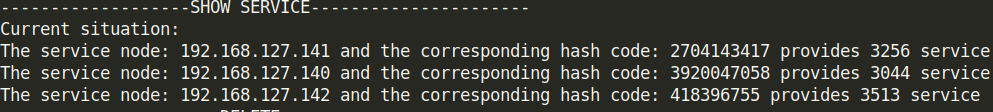
一致性哈希的方法实现：

实现思路：为了实现快速查找，使用c++的stl中map来对节点进行存储和管理，其相对于一个hash环。其中map的key值为节点的hash值，map默认按照key值进行升序排序，value则存储定义好的节点的指针。至于hash值的生成则使用murmurhash算法计算得到，主要该算法计算得到的hash值随机分布性比较好，使得各个节点更加的均匀分布在hash环中。

主要方法：

1. // 添加一个节点
2. **bool** addNode(hashNode\* node);
4. // 删除一个节点
5. **bool** delNode(hashNode\* node);
7. // 通过哈希码查找一个服务节点
8. hashNode\* find(string hashKey);
10. // 计算得到相应一个哈希值
11. string getHashCode(string str);
13. // 判断当前节点是否为0
14. **bool** empty();
16. // 得到应当路由到的结点
17. string getService(string node);
19. // 获得真实节点的个数
20. **int** getServiceNodeNum();
22. // 获得虚拟节点的个数
23. **int** getVirtualNodeNum();
25. // 显示当前所有服务器路由情况
26. **void** showService();
27. **测试文件的简单说明**

测试文件（main.cpp）主要测试了一致性哈希算法的平衡性，只设置了3个cache服务器，每个cache服务器有30个虚拟节点，10000个随机产生的请求对象模拟对cache服务器的访问。输出结果如下：



由输出结果看出，每个cache服务器节点处理的对象数目相近，负载均衡，假如希望某个cache服务器处理更多的对象，可以增大其虚拟节点的个数，则该cache服务器将处理更多的对象。

1. **总结**

本次学习主要学习了一致性哈希算法，也了解了一些别的分布式算法和hadoop。就整个一致性哈希的实现过程而言，主要难度在于对一致性哈希算法特性的理解，以及选择适合的计算哈希值的方法，需要将计算得到哈希值随机分布并且均匀，其余的实现起来还比较简单，主要实现的一致性哈希没有应用到实际，只是将大概的思路实现了而已。一致性哈希解决了在p2p环境中最为关键的问题---如何在动态的网络拓扑中分布存储和路由，每个[节点](https://baike.baidu.com/item/%E8%8A%82%E7%82%B9)仅需维护少量相邻节点的信息，并且在节点加入/退出系统时，仅有相关的少量节点参与到拓扑的维护中，一致性哈希是十分重要的分布式算法。