ISSN 1009-3044
Computer Knowledge and Technology 电脑知识与技术
Vol.7, No.22, August 2011.

E-mail: jslt@cccc.net.cn
http://www.dnzs.net.cn
Tel:+86-551-5690963 5690964

分布式存储系统中一致性哈希算法的研究

杨彧剑,林波

(78098 部队,四川 成都 611237)

摘要:一致性哈希算法目前在分布式存储中应用广泛。该文对一致性哈希算法基本原理进行了分析,讨论了其在分布式存储领域的优势和不足,分析了优化策略,并对哈希函数的选择进行了分析和实验测试。

关键词:一致性哈希算法;分布式存储

中图分类号:TP393 **文献标识码**:A **文章编号**:1009-3044(2011)22-5295-02

Research of Consistent Hashing in Distribute Storage System

YANG Yu-jian, LIN Bo

(78098 Troop, Chengdu 611237, China)

Abstract: The consistent hashing algorithm is used in distribute storage system at present. This paper introduced the basic principle of algorithm , discuss its Advantage and disadvantage, analyzed the tactics of optimizing. At last analyzed and test hash function.

Key words: consistent hashing; distribute storage

数据均衡分布技术是分布式存储系统技术中的一个重要分支,按照一定的策略将数据尽可能均匀分布到所有的存储节点上去,使得系统具有良好的负载均衡性能和扩展性,深入研究可靠的高性能数据均衡技术对于分布式存储系统而言,具有重要意义。

1 简单哈希算法

哈希(hash)计算是常见的数据分布技术,其通过求模运算来计算哈希值,然后据此将数据映射到存储空间中。设有由 N 个存储节点组成的存储空间,采用简单哈希计算将一个数据对象 object 映射到存储空间上的公式为:hash(object)%N。由于只是采用了简单的求模运算,使得简单哈希计算存在很多不足:

1)增删节点时,更新效率低。当系统中存储节点数量发生增加或减少时,映射公式将发生变化为 hash(object)%(N±1),这将使得所有 object 的映射位置发生变化,整个系统数据对象的映射位置都需要重新进行计算,系统无法对外界访问进行正常响应,将导致系统处于崩溃状态。

2)平衡性差,未考虑节点性能差异。由于硬件性能的提升,新添加的节点具有更好的承载能力,如何对算法进行改进,使节点性能可以得到较好利用,也是亟待解决的一个问题。

3)单调性不足。衡量数据分布技术的一项重要指标是单调性,单调性[1]是指如果已经有一些内容通过哈希计算分派到了相应的缓冲中,当又有新的缓冲加入到系统中时,哈希的结果应能够保证原有已分配的内容可以被映射到新的缓冲中去,而不会被映射到旧的缓冲集合中的其他缓冲区。

由上述分析可知,简单地采用模运算来计算 object 的 hash 值的算法显得过于简单,存在节点冲突,且难以满足单调性要求。修改后的一致性哈希算法正是近年来提出的一个解决方案,在 Amazon 公司的 Dynamo 中得到了很好的体现。

2 一致性哈希算法的工作原理

一致性哈希算法[2-3]是当前较主流的分布式哈希表协议之一,它对简单哈希算法进行了修正,解决了热点(hot Pot)问题,它的原理分为两步,如图 1 所示:

首先,对存储节点的哈希值进行计算,其将存储空间抽象为一个环,将存储节点配置到环上。环上所有的节点都有一个值。

其次,对数据进行哈希计算,按顺时针方向将其映射到离其最近的节点上去。

当有节点出现故障离线时,按照算法的映射方法,受影响的仅仅为环上故障节点开始逆时针方向至下一个节点之间区间的数据对象,而这些对象本身就是映射到故障节点之上的。当有节点增加时,比如,在节点 A 和 B 之间重新添加一个节点 H,受影响的也仅仅是节点 H 逆时针遍历直到 B 之间的数据对象,将这些重新映射到 H 上即可,因此,当有节点出现变动时,不会使得整个存储空间上的数据都进行重新映射,解决了简单哈希算法增删节点,重新映射所有数据带来的效率低下的问题。

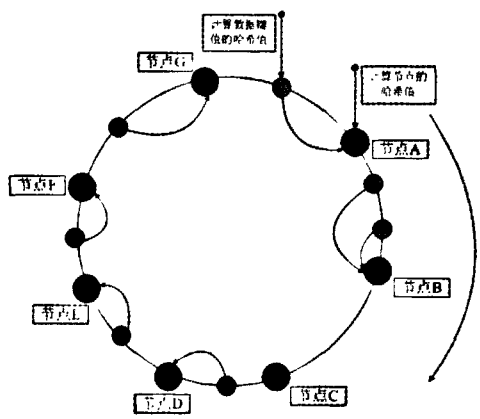

图 1 一致性哈希算法示意图

收稿日期:2011-05-27

作者简介:杨彧剑(1983-),男,重庆铜梁人,助理讲师,主要研究方向为分布式存储;林波(1982-),男,四川内江人,硕士,主要研究领域为分布式存储。

一致性哈希算法作为分布式存储领域的一个重要算法，它基本解决了以P2P为代表的存储环境中一个关键的问题——如何在动态的网络拓扑中对数据进行分发和选择路由。在算法所构成的存储拓扑中，每个存储节点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，这使得一致性哈希算法成为一个具有实用意义的DHT(Distributed Hash Table，分布式哈希表)算法。

但是一致性哈希算法尚有不足之处。第一，在查询过程中，查询消息要经过O(n)步(n代表系统内的节点总数)才能到达被查询的节点。不难想象，当系统规模非常大时，节点数量可能超过百万，这样的查询效率显然难以满足使用的需要。第二，当应用一致性哈希算法的分布式存储系统中添加或者删除新的物理节点时，要将下一个节点与之相关的数据迁移过来，查询命中率和存储效率下降，影响系统的整体性能。

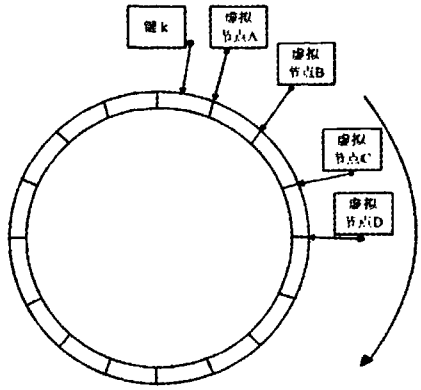

图2　虚拟节点划分示意图

3 一致性哈希算法的改进

一致性哈希算法具有随机性，当节点数量较少时节点在环上分布不够均匀，对于节点可能存在的性能差异也没有考虑，为解决这个问题，以Dynamo为代表，提出了基于虚拟节点的改进算法，其核心思路是引入虚拟节点，每个虚拟节点都有一个对应的物理节点，而每个物理节点可以对应若干个虚拟节点。

Dynamo将圆环划分成了M等分，若加入物理节点为N个，则每个物理节点拥有V=M/N个节点数。当有物理节点离线时，由于该节点对应的虚拟节点均匀地分布在环上，其附近的节点将会均匀地分担这个节点的原有负载，当有新节点加入时，同理，其他节点的负担也将均匀转移到其上。此外依据物理节点的实际性能为权值分配环上的虚拟节点数目给物理节点，也解决了存储节点性能差异的问题。

为解决数据迁移带来的效率降低问题，可以将新增节点完成的数据迁移分布在每一次的查询任务中去，相当于每一次查询都可以迁移一小部分数据，此外，还可以在系统闲暇时间进行数据迁移，这样可以有效地提高效率。

4 一致性哈希散列函数的选择

对于一致性哈希算法来说，哈希值的计算尤为重要，从一致性哈希算法提出至今，已经有多种应用于一致性哈希算法的哈希函数，比较有代表性的有CRC32_HASH、KETAMA _HASH、FNV1_32 HASH、NATIVE_HASH、MYSQL _HASH等。下面将在使用一致性哈希算法的情况下，通过增加物理节点，测试不同散列函数下命中率和数据分布所发生的变化，测试方法参考了资料[4]。

对一段英文进行单词统计，并将结果存储到10个节点上去，在存储统计结果之后，再增加两台存储节点，设单词为哈希计算中的key，次数为value，单词总数为3996，并统计此时的缓冲命中率和数据分布的情况，结果如表1所示。

表1　命中率及分布统计结果

	CRC32	KETAMA	FNV1_32	NATIVE	MYSQL
命中率	76.1%	82.8%	76.3%	98.77%	85.2%
节点1	521	403	263	3988	613
节点2	316	377	480	1	676
节点3	561	409	497	0	783
节点4	380	432	199	1	425
节点5	407	412	816	1	310
节点6	452	347	579	2	497
节点7	205	385	153	0	193
节点8	363	429	306	1	99
节点9	467	362	302	0	363
节点10	324	440	401	2	37
分布范围	200~600	300~450	100~850	0~4000	50~700

NATIVE命中率最高，但其分布极不均匀，主要集中在1个节点上；KETAMA是基于MD5的散列函数，其在保证较高命中率的同时，数据分布也最为均匀，事实上，它也是公认的广受推荐的一致性哈希算法的散列函数。

5 总结

在以云存储为代表的分布式存储系统日益发展的今天，作为数据均衡算法的一个代表，改进后的一致性哈希算法可以使得系统负载更加均衡，最大程度的避免资源浪费以及服务器过载，降低硬件变化带来的数据迁移代价和风险，如何进一步优化算法性能，值得进一步研究。

参考文献：

[1] 刘鹏.云计算[M].北京:电子工业出版社,2010:71-76.

[2] Robert Devine.Design and Implementation of DDH:A Distributed Dynamic Hashing Algorithm[C].Proceedings of 4th International Conference on Foundations of Data Organizations and Algorithms,1993.

[3] http://blog.sina.com.cn/s/blog_56fd58ab0100oe5c.html.