设计研究与应用

基于虚节点的一致性哈希算法的优化

巴子言，吴军，马严

(北京邮电大学网络技术研究院 北京 100876)

摘 要: 一致性哈希算法在分布式存储集群中得到了比较广泛的应用，但是在较大数据存储压力下会面临数据分布方面的负载均衡的问题，和数据迁移方面高效性的问题，策略的缺失会造成存储系统整体的可用性偏低。针对以上问题，本文提出了具体的虚节点反馈机制，如何合理的初始化设定虚节点数量，根据存储环境实现虚节点数量的自动分配和对存储节点的负载反馈，保证分布式存储中的负载均衡，以及在存储环境发生变化时，有效的提高数据迁移的效率，增强系统的自适应能力。最后，通过实验测试，得出数据分布结果和节点变化对数据迁移的影响，验证了该机制的有效性。

关键词: 云存储；一致性哈希；虚节点；负载均衡

中图分类号: TP311.3 **文献标识码:** A **DOI:** 10.3969/j.issn.1003-6970.2014.12.006

本文著录格式: 巴子言，吴军，马严. 基于虚节点的一致性哈希算法的优化[J]. 软件，2014，35(12)：26–29

The Optimization for Consistent Hash Based on Virtual Node

BA Zi-yan, WU Jun, MA Yan

(*Institute of Network Technology*, *Beijing University of Posts and Telecommunications*, *Beijing* 100876, *China*)

【Abstract】: Consistent hash algorithm has been extensively applied in distribute storage cluster. However, the distribute storage system would encounter the load balance and data migration problems when faced with high volume of data storage situation. The lack of reasonable storage strategy will bring down the availability of the whole storage system. Therefore, for the problems above, a self-management strategy for the virtual node has been proposed in this article. The load balance and the efficiency of data migration are ensured by considering the allocation of the virtual nodes and the load feedback from virtual node. And the result of the strategy is validated by the experimental data.

【Key words】: Cloud Storage; Consistent Hash; Virtual Node; Load Balance

0 引言

分布式的存储系统是将分散的，容量较为有限的，单一性能不高的存储资源集合起来，经过统一的管理之后，形成一个整体上具有较高存储能力的资源池[6]。随着互联网的广泛普及和高速发展，分布式存储也得到了更加普遍的应用，越来越多的信息被分布式存储所接纳。业界 Google 推出的 BigTable，Amazon 提出的 dynamo[2] 等都是各式各样的分布式存储系统解决方案。

但是，分布式存储系统在迎来了发展机遇的同时，也面临着很多的挑战。首先，由于在分布式存储系统内部中，各个存储服务器是相对分散的，当面对大量数据存储请求时，如何保证整个系统的负载均衡和避免数据倾斜的发生，成为了一个新的研究方向[7]；其次，信息的产生是不断加速的，数据量在以前所未有的速度增长，现代的存储系统需要动态地调整存储规模，支持存储节点随时的加入和退出，这种存储条件的不确定性，对整个系统资源的动态划分提出了挑战。学术界为解决这些问题展开了诸多的研究，一致性哈希算法是其中之一。一致性哈希算法最初是在 1997 年由麻省理工学院的 Karger 等提出，目的是解决分布式缓存的问题。然后这一算法在大量的分布式存储系统中都有应用，包括 Amazon 的 Dynamo 和 OpenStack 中的 Swift 存储模块。

本文在现有的一致性哈希算法的研究基础上，通过对一致性哈希算法中虚拟节点在存储资源负载均衡和数据迁移中重要性的分析，提出了虚节点的生成和分配方式，同时使虚节点具有根据系统状态调整负载的能力。这样可以有效的针对存储服务器的动态变化来做出相应的判断，来维护数据存储的平衡性和分散性。当需要数据迁移时，可以实现最快速的迁移数据的提取，避免重复的数据属性计算，提高迁移效率。

作者简介: 巴子言(1989–), 男, 河北省保定人, 硕士, 主要研究方向: 软件工程

1 虚节点在一致性哈希中的应用

1.1 基本一致性哈希算法设计思路

一致性哈希算法解决的是在存储集群中，信息与存储服务器节点之间映射关系的问题。在一致性哈希算法之中，设定有一个环状的地址空间，环上的每个点是一个从 0 到 2^{32}-1 之间的整数值。算法实现分为以下几步骤[1,5]：

- 将存储服务器节点的某一个特征值(比如 IP 地址)，经过哈希后，映射到环上的某一点；
- 将资源(key 值)经过相同的哈希函数之后，得到的整数值映射到环结构的某一点；
- 从资源映射的位置开始，顺时针查找，所遇到的第一个存储节点，即为这个 key 值对应的存储服务器；

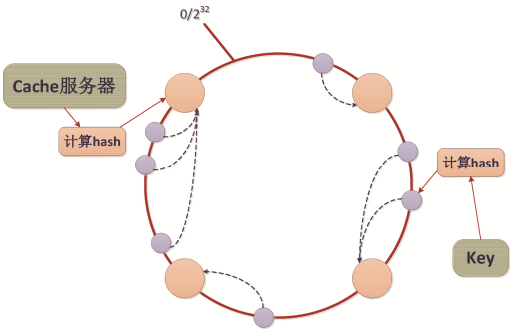

图 1 初始的一致性哈希算法
Fig. 1 Basic Consistent Hash

当存储服务器较少的时候，由于存储节点哈希过后，在环上分配的随机性，导致存储节点所承担的负载并不均匀。这时，引入虚节点的概念来弥补这种不均匀的分布。所谓虚节点，即是在环状结构上，用若干虚拟节点，在逻辑上，将环分成若干等份，每个虚节点各自对应着真实的存储服务器[2-4]。

这样便将普通的一致性哈希算法对 key 的映射关系分成了两步，

- 计算出 key 和虚拟节点之间的映射关系；
- 根据虚拟节点和存储节点的映射来找到 key 值对应的真实的存储节点；

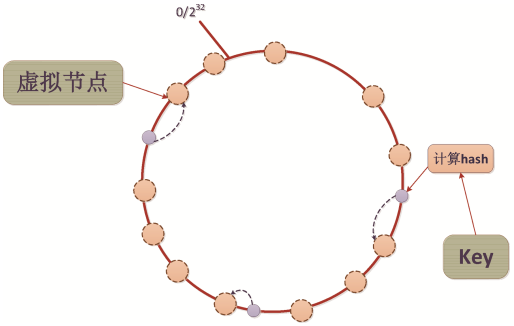

图 2 带虚节点的一致性哈希算法
Fig. 2 Consistent Hash with Virtual Node

1.2 单纯依靠虚节点的问题

虚拟节点在保证一致性哈希算法的负载均衡上提供了帮助，避免了单一依靠存储节点的哈希值而导致的在环状结构上分配不均的问题。但是，虚节点在以下方面还需要完善：

1.2.1 虚拟节点数量及分配

每个分布式存储系统的存储空间，机器性能，网络条件都是不同的，那么虚拟节点的数量也应当是不同的，如何根据系统的实际情况来决定虚拟节点的数量？虚拟节点的数量过少，则无法起到平衡负载均衡的作用；虚拟节点的数量过多，则意味着需要维护大量的虚拟节点和真实节点之间的映射关系，造成较大的查询开销。

1.2.2 虚拟节点数据迁移的开销

当有存储节点加入或者退出时，会造成信息在虚节点之间分布状态的改变。当有新节点加入时，需要将现有存储服务器的负载分担给新节点；而当有节点退出时，需要将此节点上的信息负载尽可能的平均分摊到剩余的存储服务器上。因为分布式存储系统中的数据量一般都很大，所以在这两个环节中，都要尽量保证数据可以高效地迁移，包括快速识别定位需要迁移的数据，以及将数据迁移的压力最大范围的分摊。

但现有的虚节点方案中，由于缺少对节点状态的反馈机制，如果存储节点的变化牵扯到了虚节点的分布，则被影响的虚节点需要重新哈希其上关联的 key 值，来确定哪些 key 值需要移动，而哪些不需要，造成比较大的计算开销。

2 虚节点的自我调整机制

2.1 虚节点配置

系统有 S 个存储服务器，需要把这 S 个真实的存储服务器虚拟成若干个虚拟节点。设置一个虚拟倍数 N(N 通常较大)，即平均每一台存储节点对应着 N 个虚拟节点。这样，系统总共的虚拟节点个数是 S*N，为了使 key 值迅速映射到虚拟节点的位置，进行如下操作：

- 将哈希环等分为 S*N 个区间；
- 每个区间以一个虚节点为结束地址，此虚节点负责这个区间内的 key 值映射；

- 当 key 值取模后，只需要对区间数量再进行一次取模运算，即可找到对应的虚节点；

将虚节点和区间的映射关系存储在内存之中，方便快速的读取，生成后将随着虚节点的稳定而不变。这种策略简化了得到 key 值后，在环上顺时针的查找最近的虚节点的查找过程。只需要根据 key 值哈希取模后，再对区间的数量进行取模，就能得到它所在的区间，快速定位负责这个区间的虚节点位置。

由于 S 台服务器的存储容量，处理性能是不同的，所以在负载上也是有差异的。由于所有虚拟节点是平均分配在环状结构上，对应的虚拟节点数量越多，意味着在环状上负责的区域越多，可以承担的负载越大。所以容量大，性能好的存储服务器应当有更多的虚拟节点与之映射。在虚拟节点的分配方面，引入权重。一台存储服务器的权重 W 是根据它的存储容量和性能而设定的，挑选一个存储基准值 S_0，设为 W_0，而后每台服务器和基准值比较，得到 $W_n=W_0*S_n/S_0$。之后，将 S*N 个虚拟节点按照权重的列表$\{W_1,W_2,W_3,\ldots,W_n\}$分配给存储服务器$\{S_1,S_2,S_3,\ldots,S_n\}$。

之后是取存储服务器虚拟节点个数列表$\{N_1,N_2,N_3,\ldots,N_n\}$中的近似最大公约数 m，将环分解为 m 个弧度，每个弧度内随机的分配虚节点$\{N_1/m,N_2/m,N_3/m,\ldots,N_n/m\}$。所有的虚节点都分配到特定的存储服务器，并且相对均匀的分布在哈希环上，实现了根据机器性能差异而实现的负载均衡。

Cache	Weight	Virtual node
Node-1	2	Virtual-node-0
		Virtual-node-2
		Virtual-node-4
		Virtual-node-5
Node-2	1	Virtual-node-1
		Virtual-node-3

⇨

Cache	Weight	Virtual node
Node-1	2	Virtual-node-0
		Virtual-node-5
Node-2	1	Virtual-node-1
Node-3	3	Virtual-node-2
		Virtual-node-4
		Virtual-node-3

图 3 虚拟节点数据迁移
Fig. 3 Data migration for virtual nodes

2.2 基于虚节点的数据迁移

当有新的存储服务器加入或退出进来时，如果改变节点的分布，将会对现有数据的重新分布造成较大的影响[1,2]。所以一致性哈希结构的变化主要体现在虚拟节点和服务器的对应关系上。当存储服务器变化时，需要根据它的权重值 W，来获取或释放虚拟节点。

这样一来，当存储服务器发生变化时，由于虚拟节点的整体数量没有变化，对系统整体的改变是最小化的，新加入的存储节点只需要从其他各个节点将虚拟节点的负载整体迁移即可，避免了在数据迁移时对信息的二次哈希，同时保证了系统的负载均衡。

由于为每台存储服务器加入了权重，当系统感知到系统出现数据倾斜的状况时，可以根据存储服务器的权重来调整虚节点的分配状态。设置系统数据倾斜的阈值为 10%，当系统中节点负载最高值 L_{high}，和负载最低的节点 L_{low} 相比，L_{high}-L_{low}>10%，超出阈值时，系统判定出现数据倾斜，此时将高负载存储节点对应虚节点按照权重分摊给低负载的节点，使整体负载差值低于 10%，实现负载的动态调整。

3 实验验证

以下为 5 台存储服务器存在，平均虚拟节点倍数设为 200，测试策略在实际操作中的可行性的实验结果[8]:

表 1 测试参数
Tab. 1 Test Parameters

测试文件数量(个)	存储服务器(台)	虚拟节点数量(个)	负载差值阈值	环状地址空间	哈希函数
10000	5	1000	10%	2^10	MD5

通过以上的数据可以看到，负载并不是完美的均衡，但是整体的负载差值控制在了 10%(6.88%)之内，符合本策略的预期。造成现有负载没有完全均衡的原因，是测试文件数量和虚拟节点的数量较少，有较大的随机性

表 2 测试结果
Tab. 2 Test Result

服务器	负载文件数量(个)	负载比例
Cache1	1865	18.65%
Cache2	2374	23.74%
Cache3	2058	20.58%
Cache4	1917	19.17%
Cache5	1686	16.86%

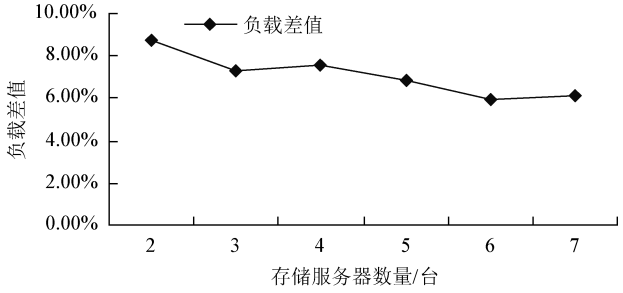

图 4 负载差值测试
Fig. 4 Test on Load Deviation

误差。可以推测到，随着测试文件数量和虚拟节点的数量增多，测试结果将越接近理想的均值。

接下来测试当哈希环中存储服务器变化时，整体的负载均衡变化趋势。测试结果如下：

负载差值定义为存储服务器之间，所负载的文件数量的最大差值与平均负载值的比率。理想状态下，随着真实存储节点数量的变化，系统整体的负载率差值应当位置在一个基本恒定的水平上。真实的情况中，在测试文件数量并没有达到足够大时，由于哈希散列的随机性，所以负载的差值会出现波动。不过总体上看，随着真实存储节点数量的变化，系统整体的负载变化是较为平缓的，不会发生明显的上升或下降，说明节点的加入和退出不会对负载造成过大的影响。

4 结论

本文给出了一套基于虚节点的一致性哈希优化方法，针对单纯依靠虚节点时，一致性哈希算法在实现思路和性能方面面临的问题，提出了具体可行的虚节点的生成和分配策略，对一致性哈希算法在负载均衡方面和数据迁移方面做出了一定改进。最后通过实验结果验证了这一个优化策略的可行性。

但是，虚拟节点的数量和数据的数量之间的平衡会对系统的负载产生影响，因此，如何针对数据量的变化而调整虚拟节点的数量，实现二者的动态最优化，将是下一步的研究方向。

参考文献

[1] D. Darger, E. Lehman, T. Leighton, M. Levine, D. Lewin and R. Panigrahy. Consistent Hashing and Random Trees:Distributed Caching Protocols for Relieving Hot Spots On the World Wide Web. ACM Symposium on Theory of Computing, 1997. 1997: 654–663.

[2] Dynamo: Amazon’s Highly Available Key-value Store, SOSP 2007

[3] Karger, D.; Sherman, A.; Berkheimer, A.; Bogstad, B.; Dhanidina, R.; Iwamoto, K.; Kim, B.; Matkins, L.; Yerushalmi, Y. (1999). "Web Caching with Consistent Hashing". Computer Networks 31 (11): 1203–1213. doi:10.1016/S1389-1286(99)00055-9.

[4] 周敬利, 周正达. 改进的云存储系统数据分布策略[J]. 计算机应用, 2012, 32(2): 309–312.
ZHOU J L, ZHOU Z D. Improved data distribution strategy for cloud storage system [J]. Journal of Computer Applications, 2012, 32(2): 309–312.

[5] 杨彧剑, 林波, 分布式存储系统中一致性哈希算法的研究[J]. 电脑知识与技术, 2011, Vol.7(22).
YANG H J, LIN B. Research of Consistent Hashing in Distribute Storage System [J]. Computer Knowledge and Technology, 2011, Vol.7 (22).

[6] 姚墨涵, 谢红薇. 一致性哈希算法在分布式系统中的应用[J]. 电脑开发与应用, 2012, 25(7).
YAO M H, XIE H W. Application of Consistent Hashing Algorithm to the Distributed System [J]. Computer Development & Applications, 2012, 25(7).

[7] 张栋梁, 谭永杰. 云计算中负载均衡优化模型及算法研究[J]. 软件, 2013, 34(8): 52–55.
ZHANG D L, TAN Y J. Research on Optimization Model and Algorithm for Load Balancing in Cloud Computing [J]. Computer engineering & Software, 2013, 34(8): 52–55.

[8] 李霄, 王常洲, 田雅. 计算机应用系统性能测试技术及应用研究[J]. 软件, 2013, 34(4): 69–73.
LI X, WANG C Z, TIAN Y. Computer application system performance testing technology and applied research [J]. Computer engineering & Software, 2013, 34(4): 69–73.