Computer Science and Application 计算机科学与应用, 2020, 10(1), 154-159
Published Online January 2020 in Hans. http://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2020.101017

Hans汉斯

Efficient Expansion Method for Distributed Database Based on Consistent Hashing Algorithm

Chao Han[1,2], Ruitao Zheng[2], Wei Yu[1,2*], Meng Xiong[2], Banji Guan[1]

[1]Cloud Computing Center of Chinese Academy of Science, Dongguan Guangdong
[2]G-Cloud Technology Incorporated Company, Dongguan Guangdong
Email: *yuwei2005@yeah.net, ruitao83@qq.com

Received: Jan. 2nd, 2020; accepted: Jan. 14th, 2020; published: Jan. 21st, 2020

Abstract

Under the big date background, distributed database became bigger during use. In the process of distributed database expansion, storage node's hash value would be recomputed, and the data object would be migration, so a large amount of data will be transferred. An efficient expansion method for distributed database was adopted, by reserved identification bit and storage node's code "High Bit Constant, Low Bit Set". Contrast experiment shows that this method can avoid re-computed hash value of existing storage nodes as well as reduce data migration, and improve the efficiency of the distributed database expansion.

Keywords

Consistent Hashing Algorithm, Distributed Database, Data Object, Database Expansion

基于一致性哈希算法的分布式数据库高效扩展方法

韩　超[1,2]，郑锐韬[2]，于　伟[1,2*]，熊　梦[2]，关班记[1]

[1]中国科学院云计算中心，广东　东莞
[2]国云科技股份有限公司，广东　东莞
Email: *yuwei2005@yeah.net, ruitao83@qq.com

收稿日期：2020年1月2日；录用日期：2020年1月14日；发布日期：2020年1月21日

*通讯作者。

文章引用: 韩超, 郑锐韬, 于伟, 熊梦, 关班记. 基于一致性哈希算法的分布式数据库高效扩展方法[J]. 计算机科学与应用, 2020, 10(1): 154-159. DOI: 10.12677/csa.2020.101017

摘　要

大数据背景下，分布式数据库在使用过程中经常需要扩容，在扩容过程中，各存储节点的哈希值需要重新计算，数据对象也需要大量迁移数据。本文通过预留子分区识别位、数据库扩容过程中物理存储节点编码“高位不变，低位置1”等技术手段实现数据库的高效扩展。对比实验标明：该方法避免了分布式数据库在扩容时的已有存储节点哈希值的重新计算工作，减少了数据对象的数据迁移量，提高了分布式数据库的扩展效率。

关键词

一致性哈希算法，分布式数据库，数据对象，数据库扩容

1. 引言

随着大数据广泛应用于各个行业领域，越来越多的应用系统采用分布式数据库来管存储管理业务数据[1] [2] [3] [4]。分布式数据库是指在物理上分散存储在的多个计算机，但逻辑上统一的数据库[5]。分布式数据库中的数据分散在多个存储节点上，所以可适当提高系统冗余备份，保障数据的可靠性。在分布式存储系统中，存储节点经常会发生动态的变化，比如节点的失效和增添，如何在节点状态变化时仍然能够对外提供良好的服务，保证数据的一致性，成为一个重要的问题。为此麻省理工学院的 Karger 等人提出了一致性哈希算法，修正了简单哈希算法带来的问题，建立了从物理存储节点到虚拟存储节点的映射关系——哈希表(DHT, Distributed Hash Table)，通过对存储对象关键因子的 Hash 计算，把数据对象均衡地分布于多个物理存储节点上。在节点发生变化时，发生迁移的数据对象最少，这样分布式数据库在节点变化频繁的场合也可以得到应用[6] [7] [8]。

大数据时代的来临，数据库的增长速度越来越快，在数据库扩展过程中，当原有的地址定义字节长度不能满足物理节点的编号需求时，如 256 个存储节点升级为 512 个存储节点，就要对地址定义进行“升位”。此时计算各个存储节点的哈希值也需要重新，这样就耗费了大量的计算资源和时间，同时存储在各个节点的存储对象也因 Hash 值的对应关系发生变化而需要大量迁移。

本文通过对基于一致性哈希算法的哈希空间进行改进，在对物理存储节点进行编码时，预留了若干的子空间标识位，现有物理存储节点的编码放在高 N 位，低 M 位预留为子空间标识位，预留标识位全部预设为 0。当物理存储节点的数量超过 N 位编码容量而需要扩容时，采用“高位不变，低位置 1”的方法，进行编码的扩展，这样，原有的物理节点编码不变，其哈希值也不变，在哈希空间的位置也不变，存在原物理节点上的数据也不需要迁移。减小了数据库扩展过程中哈希值的计算量，降低了数据迁移量。

2. 预留分区的哈希空间构造

一致性哈希算法将整个哈希空间组织成一个虚拟的圆环，我们称这个圆环为 DHT (Distributed Hash Table)环，分布式数据库的各个物理存储节点按一致性哈希算法映射到 DHT 环中，成为虚拟存储节点。在实现过程中，首先根据对物理存储节点进行编码 C，对编码采用 CRC 等算法对编码进行哈希运算得到

节点的哈希值 H，将 H 映射到 DHT 环的相应位置上；接着对数据对象的关键值(key value)进行哈希运算得到数据对象的哈希值 h，也映射到哈希空间，确定 h 在 DHT 环的位置；然后，从 h 所在的位置沿环顺时针“行走”，第一台遇到的虚拟存储节点就是其应该对应的存储节点；最后根据虚拟存储节点与物理存储节点的对应关系，将存储对象的数据存入物理存储节点。

如果物理节点的编码表已经填满，需要“升位”，这个时候就需要对物理存储节点进行重新编码，此时所有物理节点的哈希值 H 都要重新计算，原有的数据对象——存储节点的对应关系将会被打乱，大量的数据都需要迁移。

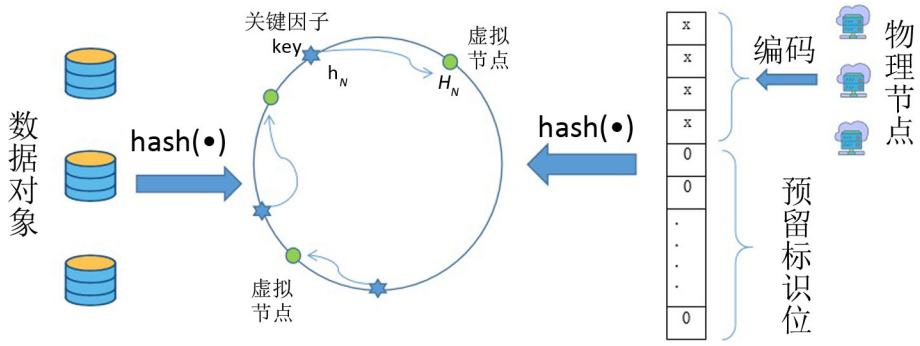

Figure 1. A hash space with reserved partition's identification bits
图 1. 预留分区标识位的哈希空间构造示意图

按照图 1 所示的方法在构造哈西空间时对物理存储节点的编码方法进行改造，现有物理存储节点的编码放在高 N 位，低 M 位预留为子空间标识位，预留标识位全部预设为 0。数据库需要扩展新的物理节点时，从预留标识位最高位开始置 1。原有节点的编码不变，设原有编码为 C，升位后所有的物理节点就有了一个新的编码 C_N，将这个新的编码映射到哈希空间，得到哈希值 H_N。原有编码：

$$C = x_{M+N-1}\cdot 2^{M+N-1} + \cdots + x_M \cdot 2^M + 0\cdot 2^{M-1} + \cdots + 0\cdot 2^0, \quad x_i \in \{0,1\} \tag{1}$$

新编码：

$$C_N = x_{M+N-1}\cdot 2^{M+N-1} + \cdots + x_{M-1}\cdot 2^{M-1} + 0\cdot 2^{M-2} + \cdots + 0\cdot 2^0, \quad x_i \in \{0,1\} \tag{2}$$

很显然，从 C 到 C_N 物理节点的编码容量从 2^N 扩展到 2^{N+1} 个，而原有各节点的编码值并没有发生变化。对其进行哈希运算，得到扩容后的数据库存储节点在 DHT 环上的映射值：

$$H_N = \text{hash}(C_N) \tag{3}$$

显然，原有的存储节点的哈希值并未发生变化，所以存在原有存储节点上的数据对象并不需要迁移。

3. 基于 CRC32 算法的哈希值生成

CRC32 算法实际上是一种循环校验算法，原理是在一个 P 位二进制数据序列之后附加一个 32-P 位二进制检验码序列，从而构成一个总长为 32 位的二进制序列；附加检验码序列与原数据序列的内容之间存在着某种特定的关系。如果在传输过程中，数据序列中的某些位发生错误，这种特定关系就会被破坏，依据数据发送方和接收方原有的约定，接收方就可以发现这个错误。

典型的 CRC32 校验多项式为：

$$\begin{aligned}\text{CRC32} &= 2^{32} + 2^{26} + 2^{23} + 2^{22} + 2^{16} + 2^{12} + 2^{11} + 2^{10} + 2^8 + 2^7 + 2^5 + 2^4 + 2^2 + 2^1 + 1 \\ &= 0\text{x}04\text{C}11\text{DB}7\end{aligned}$$

设原数据序列为 Messa_0，求取哈希值 hash_value 的运算 $\text{hash}(\cdot)$ 步骤如下：

1) 取校验多项式的阶次，CRC32 算法的最高次项为 2^{32}，所以其阶次为 32；

2) 将原数据序列 Messa_0 升位，得到 $\text{Messa}_1 = \text{Messa}_0 \ll 32$；

3) 对 Messa_1 进行模 2 除法，除数为 CRC32，余数即 CRC32 的校验码：

$$\text{CRC}_{\text{Striing}} = \text{mod}\left(\text{Messa}_1 / \text{CRC32}\right) \tag{4}$$

4) 合成哈希值： $\text{hash\_value} = \text{Messa}_1 + \text{CRC}_{\text{String}}$。

当校验多项式的阶次发生变化时，通用的 CRC 算法构成框图如图 2 所示。

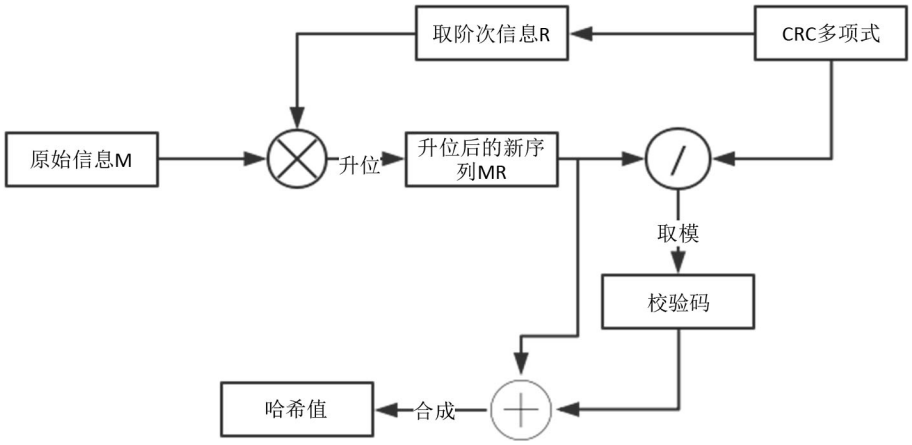

Figure 2. Realization block diagram of classical CRC algorithm
图 2. 典型 CRC 算法实现框图

4. 分布式数据库扩展方法比较

为验证本文提出的分布式数据库扩展方法，根据典型的分布式数据库扩展场景对比两种不预留子空间标识位的扩展方法的哈希值运算量和存储节点在哈希空间的分布情况。设某分布式数据库原有 256 个物理存储节点，此时用 8 位二进制编码即可表征所有存储节点的编号。随着数据量的增大，需要将数据库规模扩充为 512 个节点，此时需用 9 位二进制才能表征所有存储节点的编码(实际上由于计算机的存储方式必须以字节为单位进行扩展，也就是 8 位二进制为单位进行扩展，这里为将分析过程简化，只考虑 1 位二进制扩展)。

第一种情况，在 8 位原编码基础上，在高位增加 1 位，这样新的编码为变为 9 位，如图 3 所示。此时所有存储节点的编码在数值上没有变化，附加的校验序列依旧是原来的 32 位。如原有存储节点编码 0X5F，CRC32 校验码为：0X29D6A3E8；扩展后存储节点编码 0X05F，CRC32 校验码为：0X29D6A3E8。扩展前后哈希值均为：0X5F29D6A3E8，没有发生变化。所以这种方法并不需要对原有的存储节点的哈希值重新计算，也不用迁移原有存储节点上的数据。但是由于哈希空间发生了变化——比原来扩大了，所以这种方法会使原有存储节点在哈希空间的位置分布发生变化，如图 3 所示，现有的节点在哈希空间的位置由原来的近似均匀分布，变为集中在数值低的那一段。这样后面有新增数据对象需要存储时，都大量集中到第一个存储节点上，使存储节点的负载不均衡。

第二种情况在 8 位原编码基础上，在低位增加 1 位，这样新的编码为变为 9 位，如图 4 所示。此时所有的存储节点的编码数值都发生了变化，如 0X5F 变为 0XBE，扩展后 CRC32 校验码为：0XFEDB7106，扩展后哈希值：0XBEFEDB7106，与原哈希值相比有较大的变化。所有已有存储节点的哈希值都需要重新计算，数据对象与存储节点的对应关系也会发生变化，大量的数据需要迁移。但是这样做也有一定的优势，就是存储节点在哈希空间的分布依旧是均匀的，这样就避免了新增数据对象后，存储节点的负载不均衡问题。

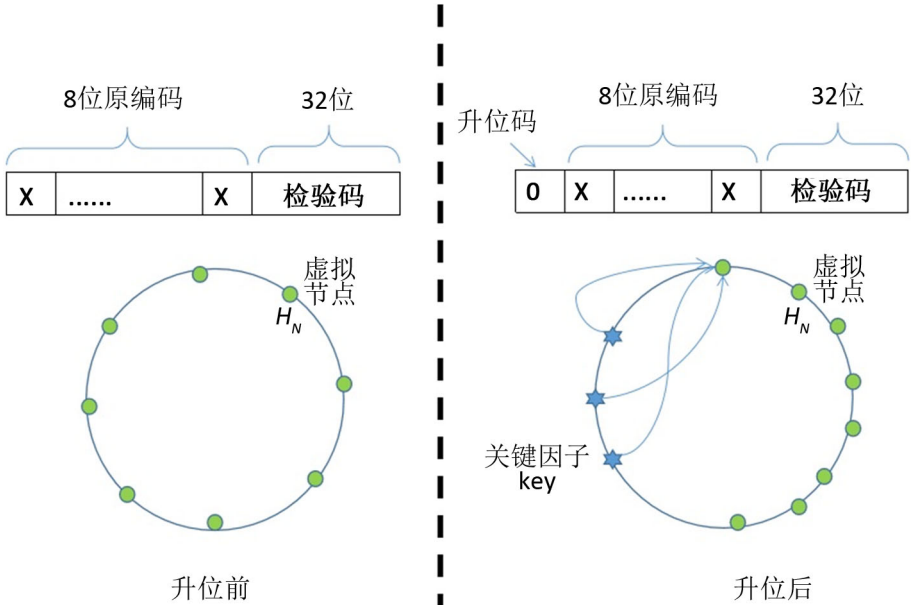

Figure 3. Position change of storage node' hash value in hash space when expand digits at high position

图 3. 在编码高位升位，存储节点哈希值在哈希空间的位置变化

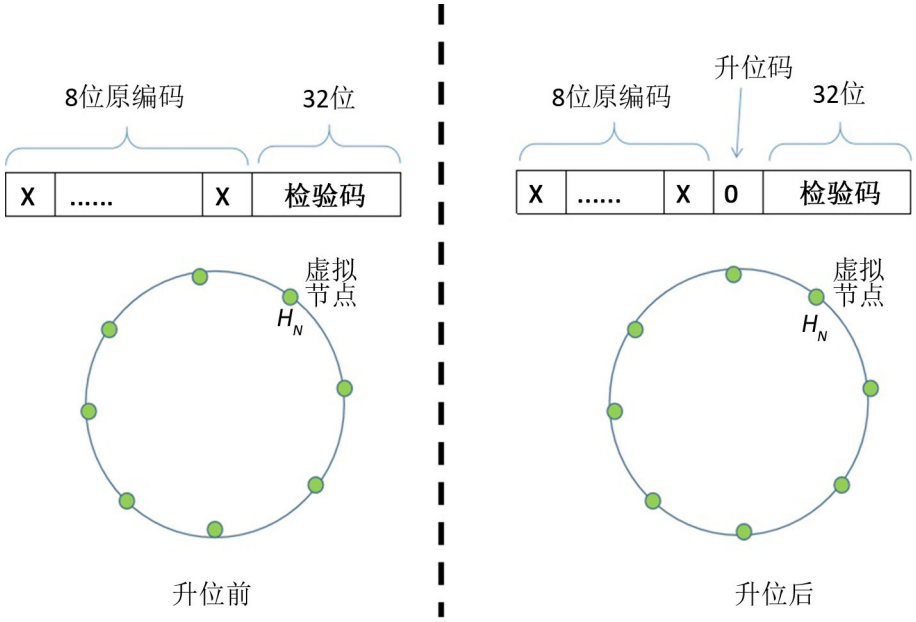

Figure 4. Position change of storage node' hash value in hash space when expand digits at low position

图 4. 在编码低位升位，存储节点哈希值在哈希空间的位置变化

第三种情况，按本文提出的方法，预留一个字节的子空间标识位。如 0X5F，预留 8 位的子空间标识位，变为 0X5F00，当数据库需要扩展时，新增存储节点在低八位予以标识，如 0X5F01。此时原有存储节点的哈希值 0X5F007CFA5364 保持不变，哈希空间没有发生变化，所以原有存储节点在哈希空间上的分布也不会发生变化。这样就不会带来重新计算哈希值问题，也不会在新增数据对象时带来存储节点的负载不均衡问题，具有明显的优势。

5. 结束语

针对大数据背景下分布式数据库频繁的扩展需求，基于一致性哈希算法提出了一种分布式数据库高效扩展方法。通过预留子空间标识位、数据库在扩容过程中，物理节点编码“高位不变，低位置 1”的方法，避免了对已有数据库 Hash 值进行重新计算，从而大大减小了扩展过程中的工作量，也降低了数据对象的迁移操作，为分布式数据库的高效管理提供了理论支持。

致　谢

感谢 2018 国家重点研发计划项目(项目编号 2018YFB1004604，项目名称“天空地一体化公共安全事件智能感知与理解系统”)的资助；感谢住房城乡建设部 2016 年科学技术项目(项目编号 2016-K3-008，项目名称“智慧城市实验云－城市级大数据开放及统一运营平台建设及产业化应用”)资助。

基金项目

2018 国家重点研发计划项目资助(项目编号 2018YFB1004604)；住房城乡建设部 2016 年科学技术项目资助(项目编号 2016-K3-008)。

参考文献

[1] 刘文洁, 李戬勃, 李战怀, 等. 一种面向金融应用的海量分布式关系数据库[J]. 华中科技大学学报(自然科学版), 2019, 47(2): 121-126.

[2] 门威, 邹香玲. 浅谈 MapReduce 与关系型数据库技术的融合[J]. 河北软件职业技术学院学报, 2017, 19(3): 10-13.

[3] 杨健, 刘方舟. 大数据背景下党建统计数据的存储电子化问题研究[J]. 云南民族大学学报(哲学社会科学版), 2019, 36(2): 26-30.

[4] 单维锋, 滕云田, 刘海军, 等. 大数据环境下地震观测数据存储方案研究[J]. 中国地震, 2019, 35(3): 558-564.

[5] 肖凌, 刘继红, 姚建初. 分布式数据库系统的研究与应用[J]. 计算机工程, 2001, 27(1): 33-35.

[6] Karger, D., Lehman, E., Leighton, T., *et al*. (1997) Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing*, El Paso, Texas, 4-6 May 1997, 654-663.

[7] 王康, 李东静, 陈海光. 分布式存储系统中改进的一致性哈希算法[J]. 计算机技术与发展, 2016, 26(7): 24-29.

[8] 陈凤, 蒙祖强. 基于哈希算法的异构多模态数据检索研究[J]. 计算机科学, 2019, 46(10): 49-54.