第 38 卷 第 6 期
Vol.38 No.6

计 算 机 工 程
Computer Engineering

2012 年 3 月
March 2012

·软件技术与数据库· 文章编号：1000—3428(2012)06—0040—03 文献标识码：A 中图分类号：TP311

NoSQL 的分布式存储与扩展解决方法

姚 林[a]，张永库[b]

(辽宁工程技术大学 a. 研究生学院；b. 电子与信息工程学院，辽宁 葫芦岛 125105)

摘 要：传统的关系型数据库已无法满足海量数据的存储与访问需求。针对该问题，提出一种非关系型数据库(NoSQL)的分布式存储与扩展解决方法。分析并改进 NoSQL，讨论基于一致性哈希算法键值对的分布式存储，以及基于双 hash 环的数据库服务器节点的扩展方法，提出将 NoSQL 作为镜像引入数据库架构系统。实际应用结果表明，该方法可以避免资源浪费及服务器过载。

关键词：非关系型数据库；一致性哈希算法；键值对；镜像

Solution of NoSQL Distributed Storage and Extension

YAO Lin[a], ZHANG Yong-ku[b]

(a. Graduate School; b. College of Electronic and Information Engineering, Liaoning Technical University, Huludao 125105, China)

【Abstract】The traditional relational database can not meet the storage and access requirements. Aiming at this problem, this paper presents a solution of NoSQL distributed storage and extension. It analyzes and improves the NoSQL, and discusses the solution on NoSQL key/value distributed storage based on consistent hashing algorithm and extension based on double hash ring. NoSQL is led into the current structure as a mirror system. The practical application results show that the solution can avoid the waste of resources and server overload.

【Key words】NoSQL; consistent hashing algorithm; key-value pair; mirror

DOI: 10.3969/j.issn.1000-3428.2012.06.013

1 概述

Web2.0 网站的兴起对数据库提出了高并发读写、高效率存储和访问、高扩展和高可用性等需求，而传统的关系型数据库无法满足这些需求，于是非关系型数据库(NoSQL)成为了 Web2.0 开发的研究热点。

针对数据库高并发读写的需求[1]，Web2.0 网站要根据用户个性化信息实时生成动态页面、提供动态信息，基本上无法使用动态页面静态化技术，因此，数据库并发负载非常高，往往要达到每秒上万次读写请求。关系数据库可以应付上万次 SQL 查询，但对于上万次 SQL 写数据请求，硬盘 IO 就无法承受了。

类似校内、twitter 这样的 SNS 网站具有对海量数据的高效率存储和访问的需求[1]，每天产生海量的用户动态。以 twitter 为例，用户每天发送的信息量超过 1.3 亿条，对于关系数据库而言，在一张 1.3 亿条记录的表里进行 SQL 查询，效率极其低下。

在基于 Web2.0 的架构当中，数据库是最难进行横向扩展的[2]。当一个应用系统的用户量和访问量与日俱增时，数据库却无法像 Webserver 和 Appserver 那样简单地通过添加更多的硬件和服务节点来扩展性能和负载能力。对于很多需要提供 24 h 不间断服务的网站来说，对数据库系统进行升级和扩展是件很麻烦的事情，往往需要停机维护和数据迁移，通过添加服务器节点来实现扩展几乎是不可能的。

为解决关系型数据库的不足，本文在 NoSQL 的分布式存储和节点扩展方面做出改进，将 NoSQL 作为一个镜像引入现有的数据库架构系统。

2 NoSQL 存取方式

在非关系型数据库中，key 和 value 都可以是任意长度的字节序列，既可以是二进制，也可以是字符串[3]。与关系型数据库不同的是，在非关系数据库中没有数据类型和数据表的概念。当作为 hash 表数据库使用时，每个 key 必须是不同的，因此，无法存储 2 个 key 相同的值。NoSQL 提供了以下访问方法：提供 key、value 参数来存储，按 key 删除记录，按 key 来读取记录。此外，遍历 key 也被支持，顺序是任意的，不能被保证。当按 B+树[4]来存储时，拥有相同 key 的记录也能被存储。像 hash 表一样的读取、存储、删除函数也都有提供。记录按照用户提供的比较函数来存储，可以采用顺序或倒序的游标来读取每一条记录。依据这个原理，向前的字符串匹配搜索和整数区间搜索也实现了。另外，B+树的事务也是可用的。对于定长的数组，记录按照自然数来标记存储，不能存储 key 相同的 2 条或更多记录。另外，每条记录的长度受到限制，读取方法和 hash 表的一样。

3 NoSQL 分布存储与改进

非关系型数据库中分布的原则是由客户端的 API 来决定的，API 根据存储用的 key 以及已知的服务器列表，然后根据 key 的 hash 计算值将指定的 key 存储到对应的服务器列表上。通常使用的方法是根据 key 的 hash 值与服务器数取余数的方法来决定当前这个 key 的内容发往哪一个服务器的。这里会涉及到一个 hash 算法[4]的分布问题。哈希算法用一句话解释就是 2 个集合间的映射关系函数，集合 A 中的记录去查找集合 B 中的记录，在这里集合 A 代表 key 值，集合 B 中代表对应的记录。显然，在实际应用中我们集合 A 中所对应的记录应该更合理地分布在集合 B 中的各个位置中，这样才能尽量避免数据被分布发送到单一的服务器上。例如，Danga

基金项目：辽宁省教育厅基金资助项目(05L169)

作者简介：姚 林(1987－)，男，硕士研究生，主研方向：分布式数据库技术，数据挖掘；张永库，副教授

收稿日期：2011-06-30 **E-mail：**lntu_dianxin@163.com

的 Memcached 在原始版本中使用的是 crc32 的算法用 Java 的实现写出来，代码如下：

```
Private static int origCompathashingAlg(String key )
{
        int hash = 0;
        char[ ] cArr = key.toCharArray( );
        for ( int i = 0; i < cArr.length; ++i )
    {
            hash = (hash * 33) + cArr[i];
        }
        return hash;     }
```

此外，还有另一个改进版本的算法，代码如下：

```
Private static int newCompathashingAlg(String key ) {
        CRC32 checksum = new CRC32( );
        checksum.update( key.getBytes( ) );
        int crc = (int) checksum.getValue( );
        return (crc >> 16) & 0x7fff;    }
```

通过分布率的测试，当随机选择的 key 在服务器数量为 5 时，所有 key 在服务器群组上的分布概率如表 1 所示。

表 1　crc32 算法改进前后的 key 分布概率

Memcached 服务器号	key 分布概率/(%)	
	原 crc32 算法	改进的 crc32 算法
0	10	19
1	9	19
2	60	20
3	9	20
4	9	20

显然，使用原的 crc32 算法会导致第 3 个 Memcached 服务的负载更高，但使用新算法会让服务之间的负载更加均衡。由此可知，在应用中要做到尽量让映射更加均匀分布，可以使服务的负载更加合理均衡。

在服务实例本身发生变动时，导致服务列表变动从而会照成大量的 Cache 数据请求会丢失，几乎大部分数据会需要迁移到另外的服务实例上。这样在大型服务在线时，瞬间对后端数据库或硬盘造成的压力很可能导致整个服务的瘫痪。

本文采用了一致性哈希[5](consistent hashing)算法来解决问题，如图 1 所示，处理服务器的选择不再仅仅依赖 key 的 hash 本身，而是将服务实例(节点)的配置也进行 hash 运算。

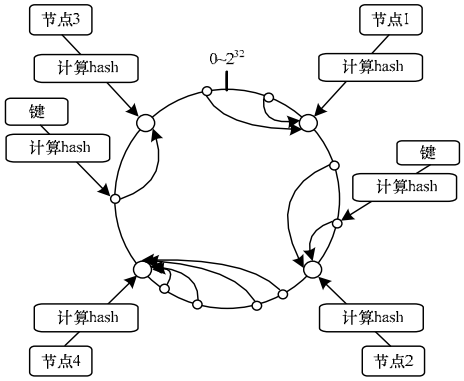

图 1　一致性哈希算法示意图

算法步骤如下：

(1)求出每个服务节点的 hash 值，并将其配置到一个 $0\sim2^{32}$ 的圆环(continuum)区间上。其次使用同样的方法求出你所需要存储的 key 的 hash 值，也将其配置到这个圆环上。

(2)从数据映射到的位置开始顺时针查找，将数据保存到找到的第 1 个服务节点上。如果超过 2^{32} 仍然找不到服务节点，就会保存到第 1 个 Memcached 服务节点上。

4　NoSQL 节点扩展

早先分布式 key-value 数据库备份采用的办法是复制数据到多个节点上，如图 2 所示，Key A 的值被复制到了系统的其他 3 个节点中。

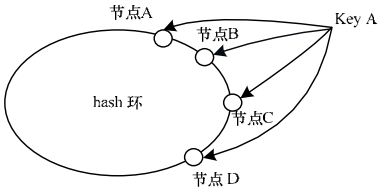

图 2　节点备份示意图

而现在比较流行的是直接使用 Master-Master 复制功能。在它的 hash 环上，单个节点其实是一对 Master-Master 的 2 个节点，称之为“双人组”。如图 3 所示，Key A 的哈希值存储于 A 和 A’ 2 个节点上，而 A 与 A’间的复制刚是通过 Master-Master 复制功能来完成的。为保证 A’的高使用率，它也可以被其他的数据中心所使用。

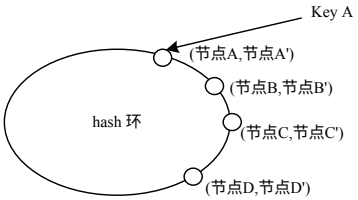

图 3　Master-Master 复制示意图

在新增数据节点时，如果没有路由服务找到正确的服务器，可能会损失数据。可采用流氓手段解决这个问题，再上一个环，保证不会发生意外。这 2 个 hash 环里的节点仍然是之前提到的“双人组”，一个环命名为 Look，记录了每一个 key 保存在哪个 Storage 节点上；另外一个环命名为 Storage，这是真正存放数据的地方。其结构如图 4、图 5 所示。

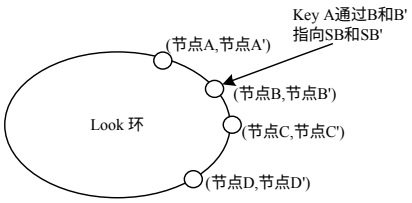

图 4　Look 环结构

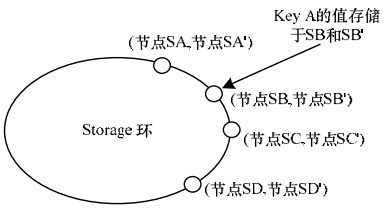

图 5　Storage 环结构

当在 Storage 环上增加一个节点后，如 Key A 原来通过 hash 算法后是在节点 B 上的。当增加节点 A 后，Key A 通过 hash 算法后在节点 A 上，在获取时就会找不到，所以要到 Look 环去先得到 Key A 的存储节点 B，然后在到存储节点 B 上去找到 Key A 对应的值。Look 环充当了记录的角色。

当 Look 环上增加了一个节点时，如 Key A 原来通过 hash 算法后是在 Look 环上的节点 B 上的。当在 Look 环上增加了节点 A 后，Key A 通过 hash 算法后也在节点 A 上。那

么，当在 Look 环上找 Key A 的存储节点时，A 上面是没有记录 Key A 对应的存储节点的，所以要找下一个节点 B，这时就找到了 Key A 的存储节点，然后到存储节点上取得 Key A 对应的值。在 hash 环上，其实是把 Key A 所有可能的节点都通过 hash 算法按照顺序找出来，然后从第 1 个开始找直到找到存储节点，或找了 3 个以后还没有找到就返回 NULL。当不在第 1 个上找到时，会把 Key A 和对应的存储节点记录到第一个节点，并将找到的那个节点上的这个 key 删除，这样下次找的时候在第 1 个节点上就能找到。

这样通过 2 个 hash 环解决了节点的增加，而且在 2 个 hash 环上都可以增加节点。同时，删除节点也只影响删除的节点上的 key，其他节点上都不影响。注意一定要避免同时增加 Look 和 Storage 节点，这很可能会丢失数据。

5 NoSQL 辅助镜像

本节将 NoSQL 引入现有的架构系统，以 NoSQL 的优势来弥补关系型数据库[6]的不足。以 MySQL 为例，把 NoSQL 引入到系统架构设计中，需要根据系统的业务场景来分析，什么样类型的数据适合存储在 NoSQL 数据库中，什么样类型的数据必须使用关系数据库存储。明确引入的 NoSQL 数据库带给系统的作用，它能解决什么问题。

在不改变原有的以 MySQL 作为存储的架构的情况下，使用 NoSQL 作为辅助镜像存储，用 NoSQL 的优势辅助提升性能。NoSQL 镜像示意图如图 6 所示。

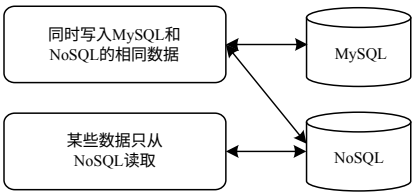

图 6 NoSQL 镜像示意图

这种架构在原有基于 MySQL 数据库的架构上增加了一层辅助的 NoSQL 存储，代码量不大，技术难度小，却在可扩展性和性能上起到了非常大的作用。只需要程序在写入 MySQL 数据库后，同时写入到 NoSQL 数据库，让 MySQL 和 NoSQL 拥有相同的镜像数据。在某些可以根据主键查询的地方使用高效的 NoSQL 数据库查询，这样能节省 MySQL 的查询，用 NoSQL 的高性能来抵挡这些查询。写入数据具体伪代码如下：

```
//data 为要存储的数据对象
data.title=”title”;
data.name=”name”;
data.time=”2011-04-21 10:10:01”;
data.from=”1”;
id=DB.Insert(data);//写入 MySQL 数据库
NoSQL.Add(id,data);//以写入 MySQL 产生的自增 id 为主键写入
//NoSQL 数据库
```

如果有数据一致性要求，可采用如下方式：

```
//data 为要存储的数据对象
    bool status=false;
    DB.startTransaction();//开始事务
    id=DB.Insert(data);//写入 MySQL 数据库
if(id>0)
{
status=NoSQL.Add(id,data);//以写入 MySQL 产生的自增 id 为主
//键写入 NoSQL 数据库
}
if(id>0 && status==true)
{
    DB.commit();//提交事务
}
else
{
DB.rollback();//不成功，进行回滚
}
```

通过以上代码，只需要在 DB 类或者 ORM 层做一个统一的封装，就可以实现重用，而其他的代码都不用做任何的修改。

另外，可以不通过程序代码，而是通过 MySQL 把数据同步到 NoSQL 中。如图 7 所示，这种模式是将 NoSQL 作为镜像的一种变体，是一种对写入透明但是具有更高技术难度一种模式。这种模式适用于现有的比较复杂的老系统，因为通过修改代码不易实现，还有可能会引起新的问题。同时也适用于需要把数据同步到多种类型的存储中。

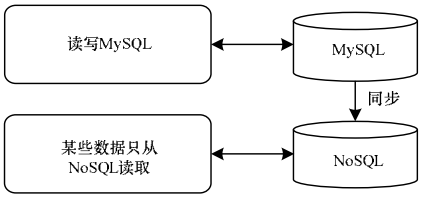

图 7 NoSQL 同步模式镜像

MySQL 到 NoSQL 同步的实现可以使用 MySQL UDF 函数、MySQL binlog 的解析来实现，也可以利用现有的开源项目来实现。例如：MySQL Memcached UDFs：从通过 UDF 操作 Memcached 协议；国内开源的 Mysql-Udf-Http：通过 UDF 操作 Http 协议。

有了这 2 个 MySQL UDF 函数库，就能通过 MySQL 透明的处理 Memcached 或 Http 协议，这样只要有兼容 Memcached 或 Http 协议的 NoSQL 数据库，就能通过 MySQL 去操作以进行同步数据。结合 Lib_MySqlUdf_Json，通过 UDF 和 MySQL 触发器功能的结合，就可以实现数据的自动同步。

6 结束语

本文研究了 NoSQL 的分布式存储与扩展解决方法，使用一致性哈希算法，降低服务及硬件环境变化带来的数据迁移代价和风险，采用双环策略解决了在增加新的数据节点时的数据丢失问题，在现有架构系统中引入 NoSQL 镜像，从而实现了关系型数据库与非关系型数据库的优势互补。

参考文献

[1] 黄贤立. NoSQL 非关系型数据库的发展及应用初探[J]. 福建电脑, 2010, (7): 30-45.

[2] 谢 毅. NoSQL. 非关系型数据库综述[J]. 先进技术研究通报, 2010, 4(8): 46-50.

[3] Weth C, Datta A. GutenTag: A Multi-term Caching Optimized Tag Query Processor for Key-value Based NoSQL Storage Systems[EB/OL]. [2011-06-10]. http://arxiv.org/pdf/1105.4452.

[4] 严蔚敏, 吴伟民. 数据结构[M]. 北京: 清华大学出版社, 2008.

[5] Cormen T H. 算法导论[M]. 潘金贵, 译. 北京: 机械工业出版社, 2006.

[6] 史晔翎, 黎建辉. 关系数据库模式到 XML Schema 的通用映射模型[J]. 计算机工程, 2009, 35(7): 35-38.

编辑 顾姣健