# 第一章 简介

HBase以列式存储的格式在磁盘上存储数据。但它与传统的列式数据库又有不同，传统的列式数据库比较适合实时存储数据的场景，HBase比较适合键值对的取，或者有序的数据存取。

关系型数据库对大数据的解决方案：

1. 增加并行读取的从服务器，将读写分离
2. 如果方案1也不能完全解决问题，则增加缓存，如memcacahe(但是有数据一致性问题)

数量大的时候，索引量也大到了足以让数据库的性能直线下降。

1. 做数据分区

1.3 非关系型数据库系统 NoSQL

数据一致性可能会影响操作延时

RDBMS非常适合事务性操作，但不见长与超大规模大的数据分析处理，因为超大规模的查询需要尽心大范围的数据记录扫描或全表扫描

分区通常不是一个切合实际大的解决方案，因为它需要客户端采用非常复杂的方式和较高的代价来维护分区信息。

一致性模型：数据库必须保证每一次的操作都是从一个一致的状态到下一个一致的状态。

1.3.3 数据库的范式化和反范式化

解决数据规模的问题：反范式化、复制和智能的主键。

Hush允许用户将长网址映射为短网址

数据库存储数据之前可以将数据压缩

查询时使用join的开销很大

1.4.1 背景

GFS:the google file system

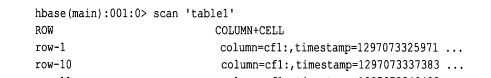
GFS缺陷:缺乏实时随机存储数据的能力，而且适合存储少许非常非常大的文件，但是不适合存储成千上万的小文件。

文件越多，CPU压力越大

1.4.2 表、行、列和单元格

最基本的单位是列，一列或多列形成一行，并由唯一的行健来确定存储。一个表中有若干行，每列可能有多个版本，在每个单元格中存储了不同的值。每个单元格可以保留若干个版本的数据这一点。

所有的行按照行健字典序进行排序存储。在字典序中，是按照二进制逐字节从做到有一次对比每个行健。如row-1... 小于row-2...，因此无论后面是什么，将始终按照这个顺序排列



行健总是唯一的，并且只出现一次，可以是任意字节数组，但它并不一定是人直接可读的。

一行由若干列组成，若干列又构成一个列族。有助于给他们设置某些特性，而且利于操作。

一个列族的所有列存储在同一个底层的存储文件中，这个存储文件叫做HFile

列族需要在表创建时就定义好，而且数量不应太多，列族名必须由可打印字符组成。

引用列的格式：family:qualifier,qualifier可以是任意的字节数组。列的数量没有限制，数据类型长度也没有限定。

可以将关系型数据库看做二维表，hbase看做三维表

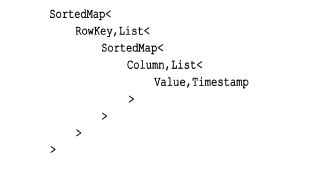
每一列的值或单元格的值都具有时间戳，默认由系统指定，也可以用户自定义。

HBase中的数据由行健、列键和时间戳索引。

存取模式：



也可以表示如下:



，还可以表示如下：



行的存取操作时原子的。保证了数据的一致性。

1.4.3 自动分区

HBase中扩展和负载均衡的基本单元成为region。Region本质上是行健排序的连续存取的区间，如果region太大，系统就会把他们动态拆分，相反的就把多个region合并以减少存储文件的数量。

HBase中的region就如同数据库分区的范围划分，可以被分配到若干台服务器上以均衡负载，因此提供了较强的扩展性。

每一个region只能由一台region服务器加载。每一台region可以同时加载多个region。

Region拆分和服务相当于其他系统提供的自动分区。

1.4.4 存储API

系统支持单行事务

HBase系统没有提供查询数据的特定域语言，类似SQL,数据的存取不是以声明的方式进行的，而是通过客户端API以纯粹的命令完成的，HBase的命令主要是Java代码。

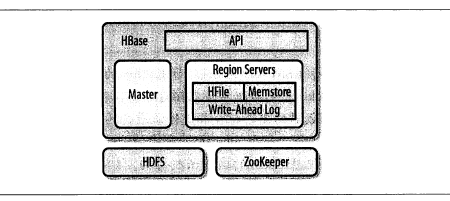
可行的办法是做个删除标记-HBase里面做的是墓碑标记，在检索过程中，这些删除标记掩盖了实际值，客户端读不到真实的值。

返回的数据来自两个部分：一是内存memstore中还未存储到文件系统的，另一种则来自文件系统。

数据合并

HBase中有三个主要组件：客户端库、一台主服务器和多台region服务器。

主服务器利用Apache Zookeeper来协调系统。



主服务器不存储数据，只负责管理和协调。

客户端直接与region服务器通信，处理所有数据相关的操作。

1. 安装

创建表的时候就必须指定列族。