Hbase的table存储详解

Hbase是bigtable的开源山寨版本。是建立在HDFS之上，提供高可靠性、高性能、列存储、可伸缩、实时读写的数据库系统。

它介于nosql和RDBMS之间，仅能通过主键（row key）和主键的range来检索数据，仅支持单行事务（可通过hive支持来实现多表join等复杂操作）。主要用来存储非结构化和半结构化的松散数据。

与hadoop一样，Hbase目标主要依靠横向扩展，通过不断增加廉价的商用服务器，来增加计算和存储能力。

Hbase中的表一般有这样的特点：

1. **大**：

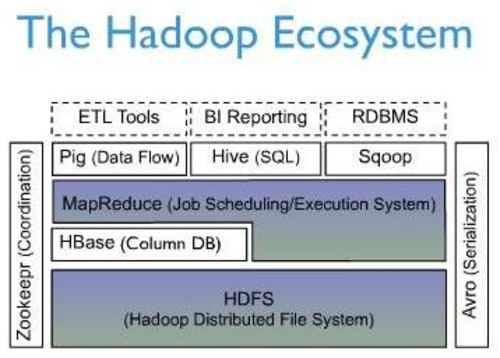
一个表可以有上亿行，上百万列

1. **面向列**：

面向列（族）的存储和权限控制，列（族）独立检索。

1. **稀疏**：

对于为空（null）的列，并不占用存储空间，因此，表可以设计的非常稀疏。



hbase在hadoop ecosystem中的位置

# 逻辑视图

Hbase以表的形式存储数据。表由行和列组成。列划分成若干个列族（columns family）

## Row Key

与nosql数据库们一样，row key是用来检索记录的主键。访问Hbase table中的行，只有三种方式：

1. 通过单个row key访问
2. 通过row key的range
3. 全表扫描

Row key行键可以是任意字符串（最大长度是64KB，实际应用中长度一般为10 - 100bytes），在Hbase内部，row key保存为字节数组。

存储时，数据按照Row key的字典序（byte order）排序存储。设计key时，要充分考虑排序存储这个特性，将经常一起读取的行存储放到一起。（位置相关性）

行的一次读写是原子操作（不论一次读写多少列）。这个涉及决策能够使用户很容易地理解程序在对同一个行进行并发更新操作时的行为。

【补充】

字典序对int排序的结果是1, 10, 100, 11, 12, 13, 14, 15, 16, 17, 18, 19, 2, 20, 21, …, 9, 91, 92, 93, 94, 95, 96, 97, 98, 99。要保持整型的自然序，行键必须用0做左填充。

## 列族

Hbase表中的每个列，都归属于某个列族。

列族是表的schema(概要)的一部分，而列不是。必须在使用表之前定义。列名都以列族作为前缀，例如：courses:history、courses:math，都属于courses这个列族。

访问控制、磁盘和内存的使用统计都是在列族层面进行的。实际应用中，列族上的控制权限能帮助我们管理不同类型的应用：我们允许一些应用可以添加新的基本数据、一些应用可以读取基本数据并创建继承的列族、一些应用则只允许浏览数据（甚至可以因为隐私的原因不能浏览所有数据）。

## 时间戳

Hbase中通过row和columns确定的唯一存储单元称为cell。每个cell都保存着同一份数据的多个版本。版本通过时间戳来索引。时间戳的类型是64位整型。时间戳可以由Hbase（在数据写入时自动）赋值，此时时间戳为精确到毫秒的当前系统时间。时间戳也可以由用户显式赋值。如果应用程序要避免数据版本冲突，就必须自己生成具有唯一性的时间戳。每个cell中，不同版本的数据按照时间倒序排序，即最新的数据排在最前面。

为了避免数据存在过多版本造成的管理负担（包括存储和索引），Hbase提供了两种数据版本回收方式。一是保存数据的最后n个版本，二是保存最近一段时间内的版本（比如最近七天）。用户可以针对每个列族进行设置。

## Cell

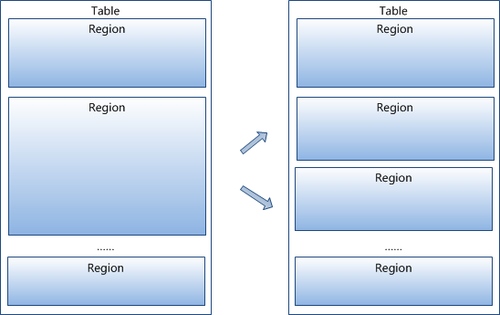
由{row key, column(= +), version}唯一确定的单元。cell中的数据是没有类型的，全部是字节码形式存储。

# 物理存储

1. Table中的所有行都按照row key的字典序排列。
2. Table在行的方向上分割为多个Hregion。（一个表一开始只有一个region，随着数据不断插入表，region不断增大，当增大到一个阈值的时候，Hregion就会等分成两个新的Hregion。）

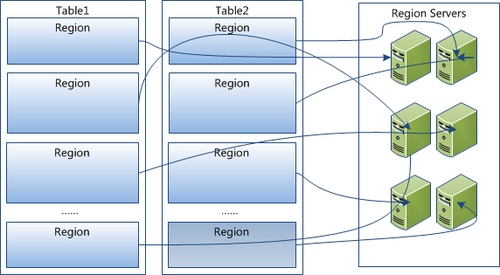


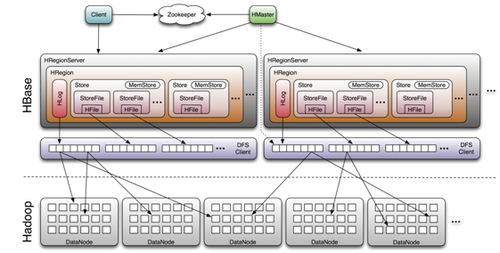
当table中的行不断增多，就会有越来越多的Hregion。



1. HRegion是Hbase中分布式存储和负载均衡的最小单元。

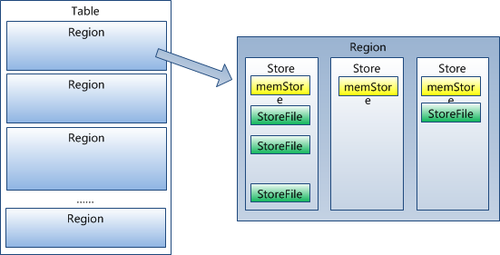
最小单元就表示不同的Hregion可以分布在不同的HRegion server上。但一个HRegion是不会拆分到多个server上的。



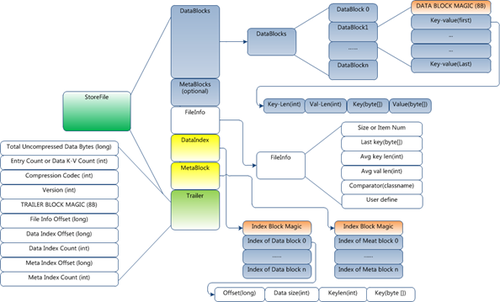


1. HRegion虽然是分布式存储的最小单元，但并不是存储的最小单元。

事实上，HRegion由一个或者多个Store组成，每个store保存一个columns family。每个Store又由一个memStore和0至多个StoreFile组成。StoreFile以HFile格式保存在HDFS上。



HFile的格式为：



HFile分为六个部分：

* Data Block段

保存表中的数据，这部分可以被压缩。

* Meta Block段（可选的）

保存用户自定义的kv对，可以被压缩。

* File Info段

HFile的元信息，不被压缩，用户也可以在这一部分添加自己的元信息。

* Data Block Index段

Data Block的索引。每条索引的key是被索引的block的第一条记录的key。

* Meta Block Index段（可选的）

Meta Block的索引。

* Trailer

这一段是定长的。保存了每一段的偏移量，读取一个HFile时，会首先读取Trailer，Trailer保存了每个段的起始位置（段的Magic Number用来做安全check），然后，DataBlock Index会被读取到内存中。这样，当检索某个key时，不需要扫描整个HFile，而只需从内存中找到key所在的block，通过一次磁盘IO将整个block读取到内存中，再找到需要的key。DataBlock Index采用LRU机制淘汰。

HFile的Data Block、Meta Block通常采用压缩方式存储，压缩之后可以大大减少网络IO和磁盘IO，随之而来的开销当然是需要花费cpu进行压缩和解压缩。

目前HFile的压缩支持两种方式：Gzip、Lzo。

1. HLog（WAL log）

WAL意为Write ahead log，类似mysql中的binlog，用来做灾难恢复使用，Hlog记录数据的所有变更，一旦数据修改，就可以从log中进行恢复。

每个Region Server维护一个Hlog，而不是每个Region一个。这样不同region（来自不同table）的日志会混在一起，这样做的目的是不断追加单个文件，相对于同时写多个文件而言，可以减少磁盘寻址次数，因此可以提高对table的写性能。带来的麻烦是，如果一台region server下线，为了恢复其上的region，需要将region server上的log进行拆分，然后分发到其它region server上进行恢复。

HLog文件就是一个普通的Hadoop Sequence File，Sequence File的Key是HLogKey对象，HLogKey中记录了写入数据的归属信息，除了table和region名字外，同时还包括sequence number和timestamp，timestamp是“写入时间”，sequence number的起始值为0，或者是最近一次存入文件系统中的sequence number。HLog Sequence File的Value是Hbase的KeyValue对象，即对应HFile中的KeyValue。

# 系统架构

## Client

包含访问Hbase的接口，client维护着一些cache来加快对Hbase的访问，比如region的位置信息。

## Zookeeper

1. 保证任何时候，集群中只有一个master
2. [存储] 所有Region的寻址入口
3. 实时监控Region Server的状态，将Region server的上线和下线信息实时通知给Master。
4. [存储] Hbase的schema（包括有哪些table，每个table有哪些column family）。

## Master

1. 为Region server分配region
2. 负责Region server的负载均衡
3. 发现失效的Region server并重新分配其上的region
4. GFS上的垃圾文件回收
5. 处理schema更新请求（master仅仅维护者table和region的元数据信息）

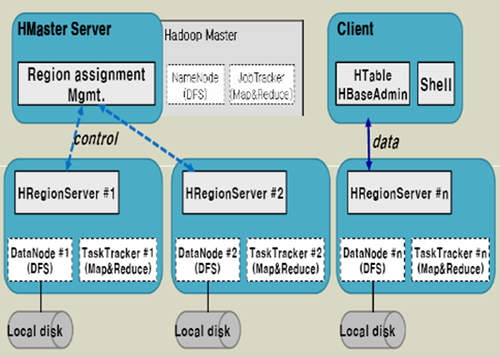
## Region Server

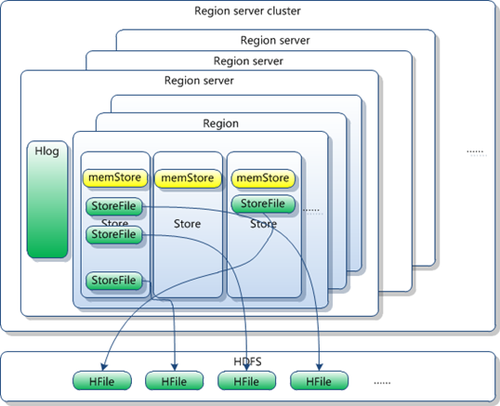
1. Region server维护Master分配给它的region，处理对这些region的IO请求。
2. Region server负责切分在运行过程中变得过大的region。

## 总结

可以看到，client访问Hbase上数据的过程并不需要master参与（寻址访问zookeeper和region server，数据读写访问region server）。

master仅仅维护着table和region的元数据信息，负载很低。





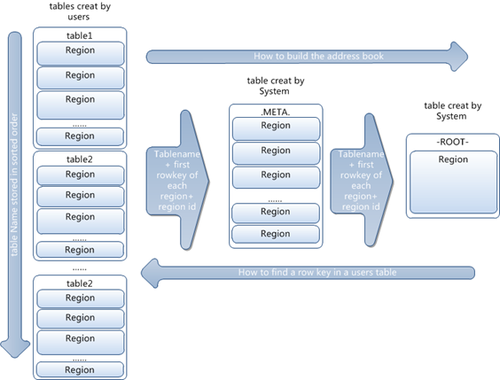
# 关键算法 / 流程

## Region定位

系统如何找到某个row key (或者某个row key range)所在的region？

bigtable使用三层类似B+树的结构来保存region的位置：

1. 第一层保存的是zookeeper里面的文件，它持有root region的位置。
2. 第二层，root region是META表的第一个region，其中保存了META表其它region的位置。通过root region，我们就可以访问META表的数据。
3. 第三层，META表，它是一个特殊的表，保存了Hbase中左右数据表的region位置信息。



说明：

1. root region永远不会被split，保证了只需要三次跳转，就能定位到任意region。
2. META表每行保存一个region的位置信息，row key采用表名+表的最后一行编码而成。
3. 为了加快访问，META表的全部region都保存在内存中。

假设，META表的一行在内存中大约占用1KB。并且每个region限制为128MB。那么，上面的三层结构可以保存的region数目为：(128MB/1KB) \* (128MB/1KB) == 234个region。

1. client会将查询过的位置信息缓存起来，缓存不会主动失效，因此如果client上的缓存全部失效，则需要进行6次网络来回，才能定位到正确的region (其中三次用来发现缓存失效，另外三次用来获取位置信息)。

## 读写过程

## Region分配

## Region server上线

## Region server下线

## Master上线

## Master下线

<https://blog.csdn.net/zh521zh/article/details/51918664>