转 HBase详解（很全面）

2018年06月08日 16:12:32 卢子墨 阅读数：1842

【转自：http://jiajun.iteye.com/blog/899632】

一、 简介

history

started by chad walters and jim

2006.11 G release paper on BigTable

2007.2 inital HBase prototype created as Hadoop contrib

2007.10 First useable Hbase

2008.1 Hadoop become Apache top-level project and Hbase becomes subproject

2008.10 Hbase 0.18,0.19 released

hbase是bigtable的开源山寨版本。是建立的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。

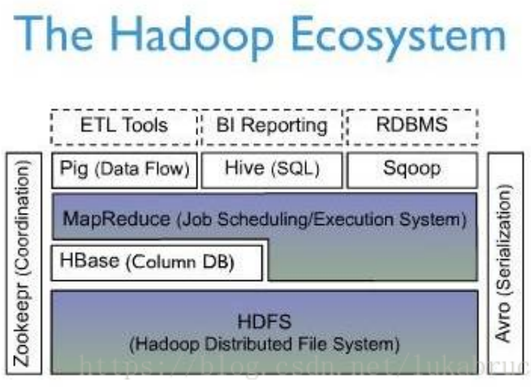
HBase中的表一般有这样的特点：

1 大：一个表可以有上亿行，上百万列

2 面向列:面向列(族)的存储和权限控制，列(族)独立检索。

3 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。

下面一幅图是Hbase在Hadoop Ecosystem中的位置。



二、 逻辑视图

HBase以表的形式存储数据。表有行和列组成。列划分为若干个列族(row family)

Row Key	column-family1	column-family2	column-family3			
column1	column1	column1	column2	column3	column1	
key1	t1:abc t2:gdxdf		t4:dfads t3:hello t2:world			
key2	t3:abc t1:gdxdf		t4:dfads t3:hello		t2:dfdsfa t3:dfdf	
key3		t2:dfadfasd t1:dfdasddsf				t2:dfxxdfasd t1:taobao.com

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 (Row key)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)

注意：

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。要保持整形的自然序，行键必须用0作左填充。

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

列族

hbase表中的每个列，都归属与某个列族。列族是表的chema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（甚至可能因 为隐私的原因不能浏览所有数据）。

时间戳

HBase中通过row和columns确定的为一个存贮单元称为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，此时时间戳是精确到毫秒的当前系统时间。时间戳也可以由客户显式赋值。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的管理 (包括存贮和索引)负担，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。

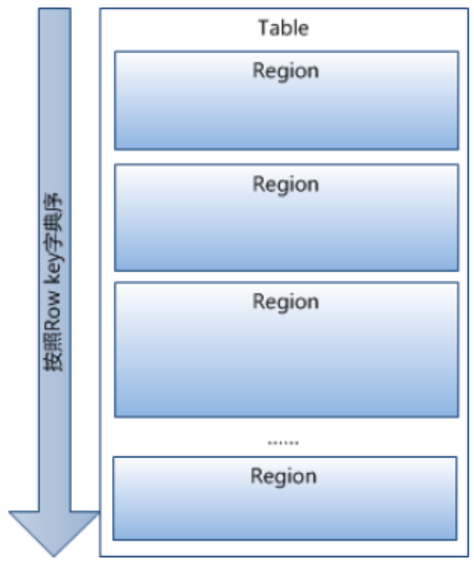
Cell

由{*row key, column(* =<family> + <label>*), 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贮。

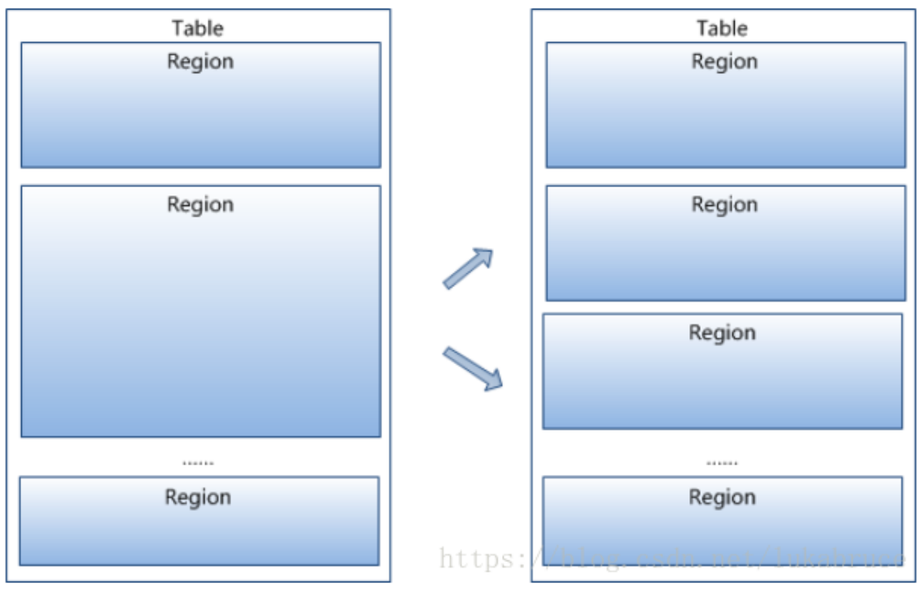
三、 物理存储

1 已经提到过，Table中的所有行都按照row key的字典序排列。

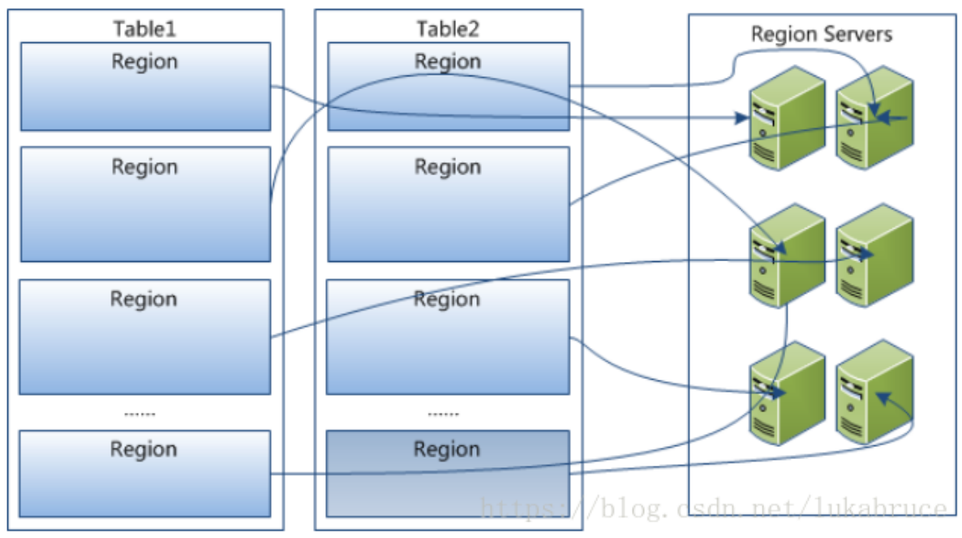
2 Table 在行的方向上分割为多个Hregion。



3 region按大小分割的，每个表一开始只有一个region，随着数据不断插入表，region不断增大，当增大到一个阀值的时候，Hregion就会等分会两个新的Hregion。当table中的行不断增多，就会有越来越多的Hregion。



4 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。

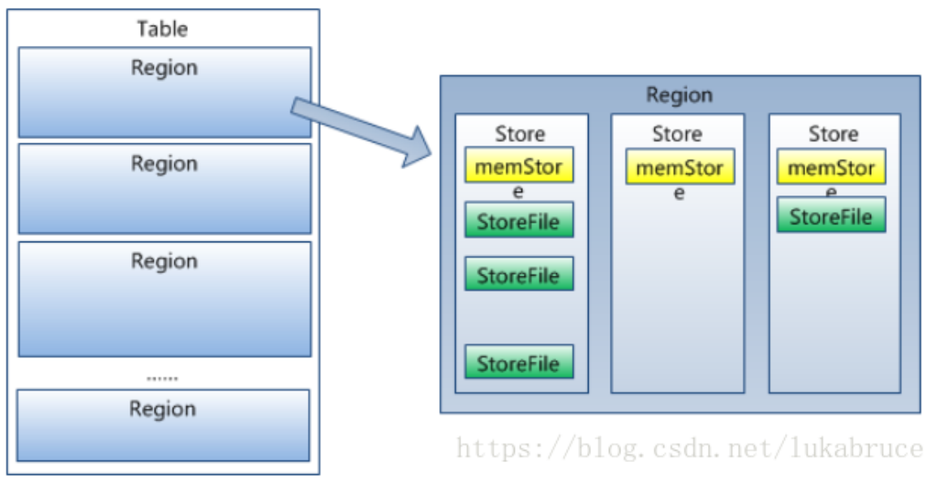


5 HRegion虽然是分布式存储的最小单元，但并不是存储的最小单元。

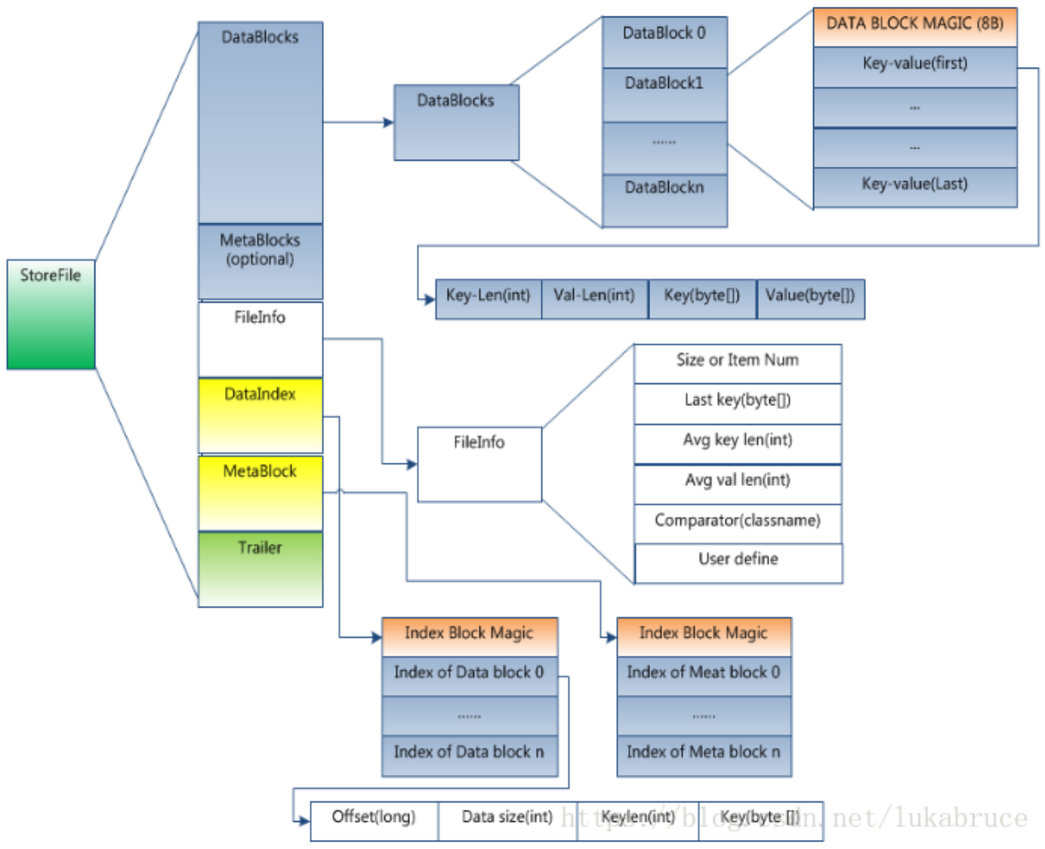
事实上，HRegion由一个或者多个Store组成，每个store保存一个columns family。

每个Strore又由一个memStore和0至多个StoreFile组成。如图：

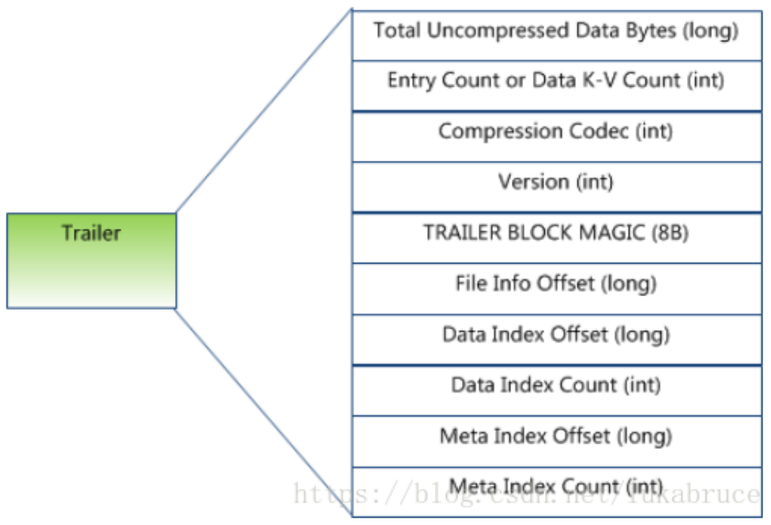
StoreFile以HFile格式保存在HDFS上。



HFile的格式为：



Trailer部分的格式：



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。

目标Hfile的压缩支持两种方式：Gzip，Lzo。

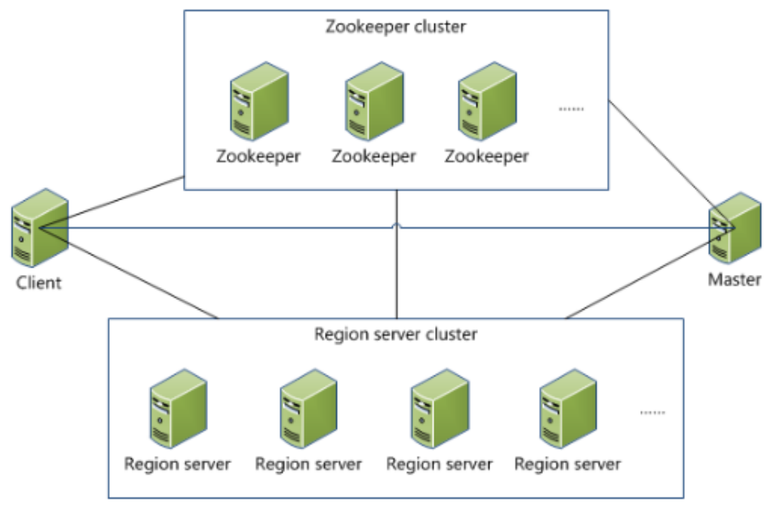
HLog(WAL log)

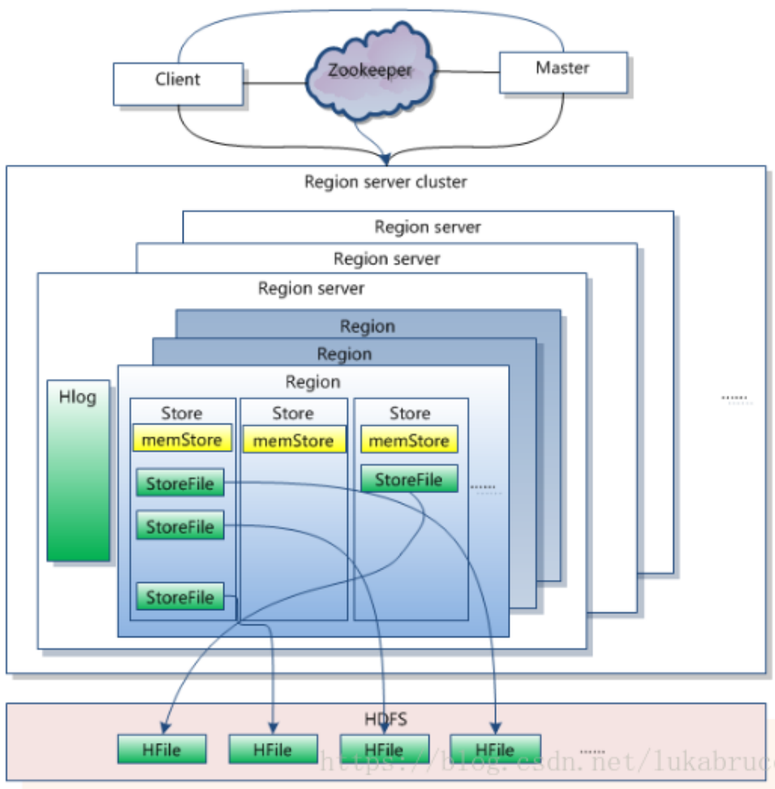
WAL 意为Write ahead log(http://en.wikipedia.org/wiki/Write-ahead_logging)，类似mysql中的binlog,用来 做灾难恢复只用，Hlog记录数据的所有变更,一旦数据修改，就可以从log中进行恢复。

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sequence number。HLog Sequece File的Value是HBase的KeyValue对象，即对应HFile中的KeyValue，可参见上文描述。

四、 系统架构





Client

1 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regione的位置信息。

Zookeeper

1 保证任何时候，集群中只有一个master

2 存贮所有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

2 Region server负责切分在运行过程中变得过大的region

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regione server），master仅仅维护者table和region的元数据信息，负载很低。

五、关键算法 / 流程

region定位

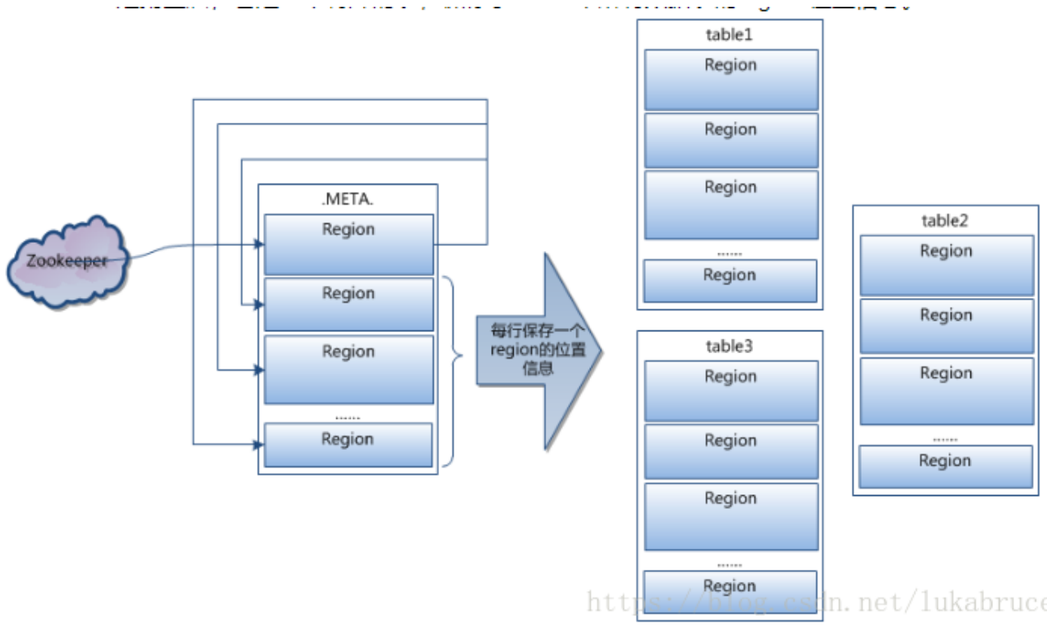
系统如何找到某个row key (或者某个 row key range)所在的region

bigtable 使用三层类似B+树的结构来保存region位置。

第一层是保存zookeeper里面的文件，它持有root region的位置。

第二层root region是.META.表的第一个region其中保存了.META.z表其它region的位置。通过root region，我们就可以访问.META.表的数据。

.META.是第三层，它是一个特殊的表，保存了hbase中所有数据表的region 位置信息。



说明：

1 root region永远不会被split，保证了最需要三次跳转，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+表的最后一样编码而成。

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) = = 2(34)个region

4 client会将查询过的位置信息保存缓存起来，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

上文提到，hbase使用MemStore和StoreFile存储对表的更新。

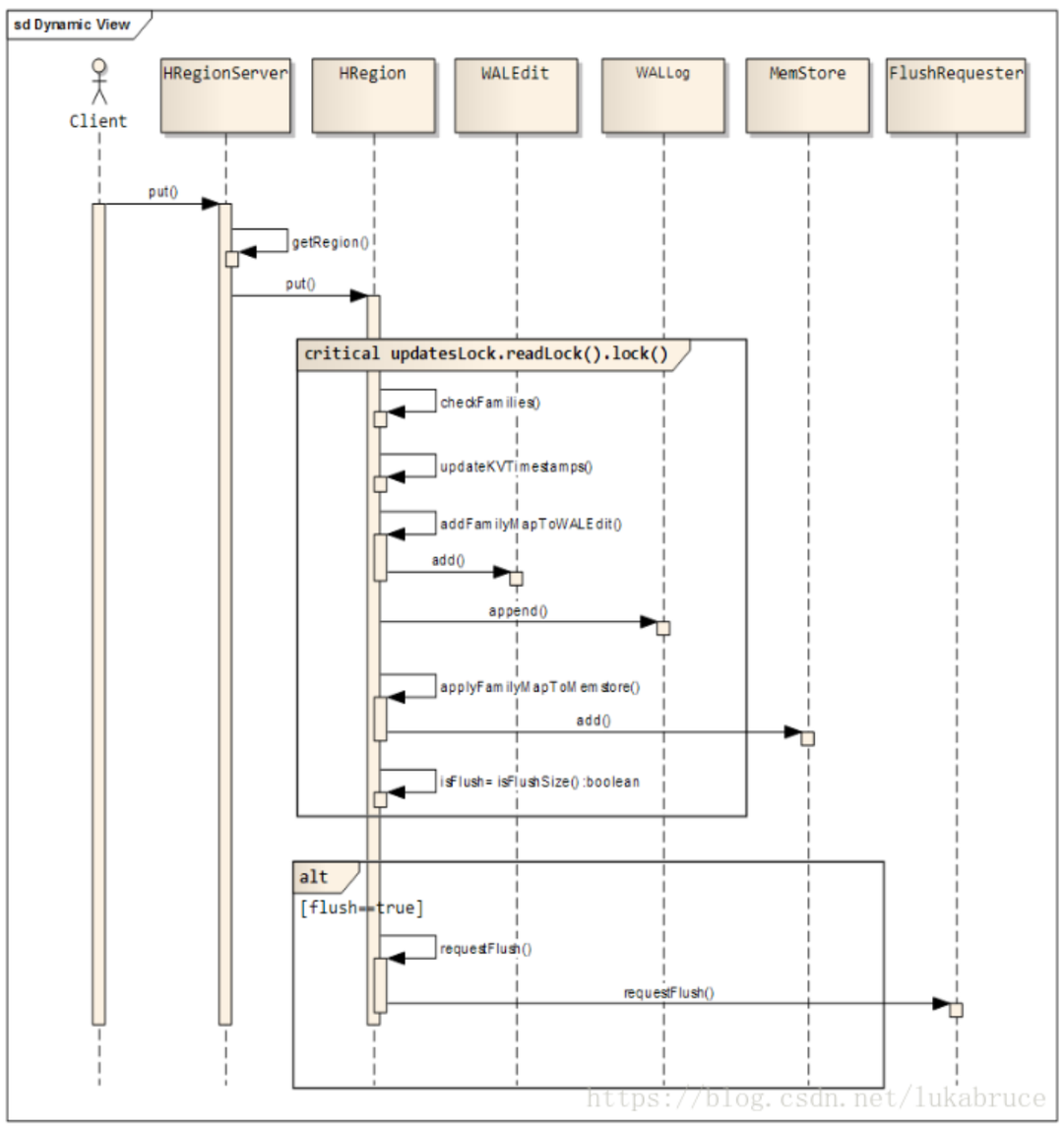
数据在更新时首先写入Log(WAL log)和内存(MemStore)中，MemStore中的数据是排序的，当MemStore累计到一定阈值时，就会创建一个新的MemStore，并 且将老的MemStore添加到flush队列，由单独的线程flush到磁盘上，成为一个StoreFile。于此同时，系统会在zookeeper中 记录一个redo point，表示这个时刻之前的变更已经持久化了。(minor compact)

当系统出现意外时，可能导致内存(MemStore)中的数据丢失，此时使用Log(WAL log)来恢复checkpoint之后的数据。

前面提到过StoreFile是只读的，一旦创建后就不可以再修改。因此Hbase的更 新其实是不断追加的操作。当一个Store中的StoreFile达到一定的阈值后，就会进行一次合并(major compact),将对同一个key的修改合并到一起，形成一个大的StoreFile，当StoreFile的大小达到一定阈值后，又会对 StoreFile进行split，等分为两个StoreFile。

由于对表的更新是不断追加的，处理读请求时，需要访问Store中全部的 StoreFile和MemStore，将他们的按照row key进行合并，由于StoreFile和MemStore都是经过排序的，并且StoreFile带有内存中索引，合并的过程还是比较快。

写请求处理过程



1 client向region server提交写请求

2 region server找到目标region

3 region检查数据是否与schema一致

4 如果客户端没有指定版本，则获取当前系统时间作为数据版本

5 将更新写入WAL log

6 将更新写入Memstore

7 判断Memstore的是否需要flush为Store文件。

region分配

任何时刻，一个region只能分配给一个region server。master记录了当前有哪些可用的region server。以及当前哪些region分配给了哪些region server，哪些region还没有分配。当存在未分配的region，并且有一个region 上有可用空间时，master就给这个region server发送一个装载请求，把region分配给这个region server。region server得到请求后，就开始对此region提供服务。

master使用zookeeper来跟踪region server状态。当某个region server启动时，会首先在zookeeper上的server目录下建立代表自己的文件，并获得该文件的独占锁。由于master订阅了server 目录上的变更消息，当server目录下的文件出现新增或删除操作时，master可以得到来自zookeeper的实时通知。因此一旦region server上线，master能马上得到消息。

region server下线

当region server下线时，它和zookeeper的会话断开，zookeeper而自动释放代表这台server的文件上的独占锁。而master不断轮询 server目录下文件的锁状态。如果master发现某个region server丢失了它自己的独占锁，(master连续几次和region server通信都无法成功),master就是尝试去获取代表这个region server的读写锁，一旦获取成功，就可以确定：

1 region server和zookeeper之间的网络断开了。

2 region server挂了。

的其中一种情况发生了，无论哪种情况，region server都无法继续为它的region提供服务了，此时master会删除server目录下代表这台region server的文件，并将这台region server的region分配给其它还活着的同志。

如果网络短暂出现问题导致region server丢失了它的锁，那么region server重新连接到zookeeper之后，只要代表它的文件还在，它就会不断尝试获取这个文件上的锁，一旦获取到了，就可以继续提供服务。

master上线

master启动进行以下步骤:

1 从zookeeper上获取唯一一个代码master的锁，用来阻止其它master成为master。

2 扫描zookeeper上的server目录，获得当前可用的region server列表。

3 和2中的每个region server通信，获得当前已分配的region和region server的对应关系。

4 扫描.META.region的集合，计算得到当前还未分配的region，将他们放入待分配region列表。

master下线

由于master只维护表和region的元数据，而不参与表数据IO的过 程，master下线仅导致所有元数据的修改被冻结(无法创建删除表，无法修改表的schema，无法进行region的负载均衡，无法处理region 上下线，无法进行region的合并，唯一例外的是region的split可以正常进行，因为只有region server参与)，表的数据读写还可以正常进行。因此master下线短时间内对整个hbase集群没有影响。从上线过程可以看到，master保存的 信息全是可以冗余信息（都可以从系统其它地方收集到或者计算出来），因此，一般hbase集群中总是有一个master在提供服务，还有一个以上 的’master’在等待时机抢占它的位置。

六、访问接口

- HBase Shell
- Java clietn API
- HBase non-java access
- - languages talking to the JVM
  - - Jython interface to HBase
    - Groovy DSL for HBase
    - Scala interface to HBase
  - languages with a custom protocol
  - - REST gateway specification for HBase
    - 充分利用HTTP协议：GET POST PUT DELETE

§

- - - text/plain
    - text/xml
    - application/json
    - application/x-protobuf
  - Thrift gateway specification for HBase
  - - java
    - cpp
    - rb
    - py
    - perl
    - php
- HBase Map Reduce
- Hive/Pig

七、结语：

全文对 Hbase做了 简单的介绍，有错误之处，敬请指正。未来将结合 Hbase 在淘宝数据平台的应用场景，在更多细节上进行深入。

参考文档

Bigtable: A Distributed Storage System for Structured Data

HFile: A Block-Indexed File Format to Store Sorted Key-Value Pairs for a thorough introduction Hbase Architecture 101

Hbase source code