分布式数据库HBase

内容

- HBase的历史和特点
- HBase和其他Hadoop成员的关系
- HBase的数据模型
- HBase的访问接口
- HBase的运行组成
- HBase Region
- HBase的读写
- 其他HBase功能

为何需要构建分布式数据库

- 从数据类型来说，在网络上存在大量的结构化与半结构化数据，如果仅仅使用分布式文件系统的方式就会丢失大量结构信息
  - URL数据，针对每一个网页有：网页内容，元数据，链接信息，链接描述数据，PageRank的值
  - 对于每个用户的数据来说，有用户自己特定的设定，最近的查询历史等
  - 对于网络应用程序来说，例如购物网站的商品信息以及用户信息；卫星数据的一些标志数据等
- 数据规模十分庞大
  - 对于网页来说，显然整个互联网的数据无法估量
  - 用户数目也极其庞大，造成用户数据也十分庞大
  - 对于网络应用程序来说，只要服务于整个互联网的，造成的数据规模也十分庞大

传统的数据库管理系统是否能够完成工作

- 对于绝大多数商业数据库来说，云条件下的数据都过于庞大
- 关系模型对数据的操作使数据的存贮变得复杂。传统数据库的强一致性的要求，往往在处理互联网规模的数据的时候显得无法胜任
- 即使不考虑数据规模，对于任何应用来说，使用如此大规模的商业数据库都会造成花费极多（想象一下昂贵的商用数据库）
- 可以对传统的数据库进行改良，但是改良的关系数据库（副本、分区等）难于安装与维护
- 因此，Google启动了BigTable项目，用以处理Google内部的大规模的结构化以及半结构化数据

关系数据库的理论局限性

- RDBMS选择了ACID
  - 超强的一致性（transaction）牺牲了水平可扩展性
- Scale up, not out
  - 并行数据库 的扩展性
    - 经验定律：当集群节点数每增加4~16台，每个节点的效率下降一半
  - 无法扩展超过~40节点
    - 最新的TPC-C世界纪录：27 servers
    - 最新的TPC-H世界纪录：32 servers
- “One size does not fit all”
  - 在所有数据库的主要应用领域，新的架构轻易得有10x倍性能提升（数据仓库，流处理,科学计算，非结构化数据处理，OLTP在线事务处理)*

** ACM SIGACT News, Volume 33 Issue 2 (2002): “Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services”*
** ICDE 2005: “One Size Fits All”: An Idea Whose Time Has Come and Gone*
** VLDB 2007: The End of an Architectural Era (It’s Time for a Complete Rewrite)*

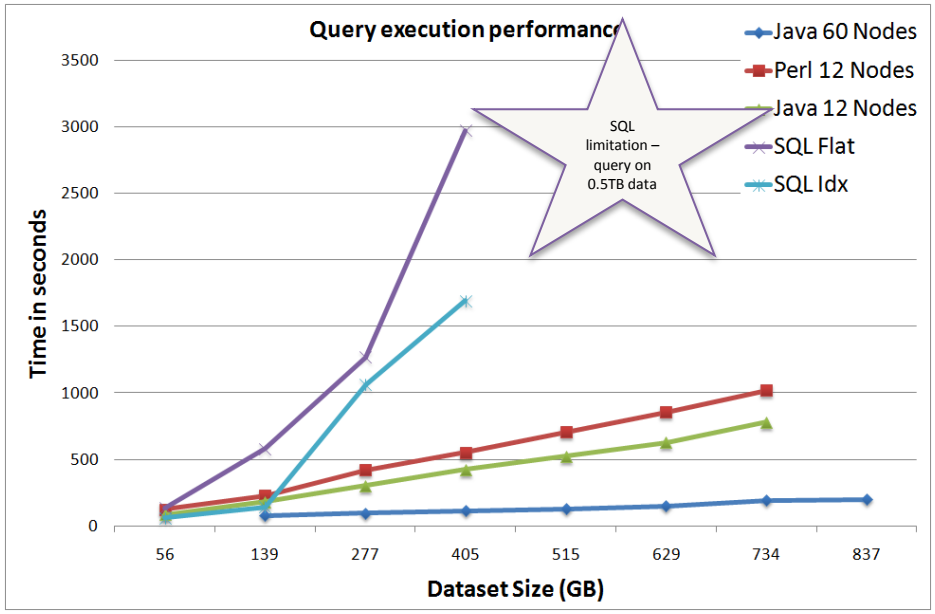
CAP

- One size does not fit all*
- CAP
  - 一致性(Consistency)
    - 客户端知道一系列的操作都会同时发生(生效)
  - 可用性(Availability)
    - 每个操作都必须以可预期的响应结束.
  - 分区容错性(Partition tolerance)
    - 即使出现单个组件无法可用,操作依然可以完成.

RDBMS的实现局限性

- RDBMS**实现和操作上的局限性 – 不适合新的应用**
  - *大表* – 在一张表中存储500GB的数据?
  - *灵活动态可变的表结构* – 为大表修改表结构(Alter Table)?
  - *无停机时间的在线大表分区和动态扩容* – …

关系数据库 vs. Hadoop/Hive



Intel silicon design environment usage analysis, 40M records/day

HBase的历史与简介

- 2006年Google发表了BigTable论文
- 2006年底由PowerSet 的Chad Walters和Jim Kellerman 发起了HBase项目，依据BigTable的论文重构关系数据库
- 2007年2月建立了HBase的原型版本
- 2007年10月建立了第一个可用的HBase版本
- 2008年成为Apache Hadoop的一个子项目

- HBase是Google Bigtable的开源实现。Bigtable利用GFS作为其文件存储系统；HBase使用HDFS作为其文件存储系统。Google Bigtable利用Chubby作为一个可靠服务的根，Chubby提供了BigTable中的根元数据表的指针以及用以监控所有数据分表服务；在HBase中，对应的系统服务为ZooKeeper

HBase的特性

Hadoop database and NoSQL database

- 基本的数据库操作CRUD
- 强一致性
- 无SQL语言支持
- 稀疏的多维映射表
  - 列存储
  - 只用row key来定位行
  - 每行可以有不同的列
  - 数据有多个版本（在不同的时间点的快照信息）
- 非常高的数据读写速度，为写特别优化
  - 高效的随机读取
  - 对于数据的某一个子集能够进行有效地扫描

HBase特性纵览(2)

- 分布式的多层次映射表结构（key-value形式，value有多个）
  - 固定一个数据模型（固定数据模型能得到高性能，同时满足应用需求）
  - 无数据类型
- 具有容错特性，能够将数据持久化的非易失性存储中
  - 使用HDFS做底层存储，可利用Hadoop的压缩Codec等减少空间占用
- 自动水平扩展
  - 只需要加入新的结点即可提高存储容量和吞吐量
  - 服务器能够被动态加入或者删除（用以维护和升级）
  - 服务器自动调整负载平衡

事务的ACID

- 原子性(Atomicity)
  - 事务要么全部完成，要么全部不完成
- 一致性(Consistency)
  - 在事务执行过程中以及前后，数据库的完整性约束没有被破坏
- 隔离性(Isolation)
  - 事务的执行是互不干扰的，不能看到其他事务运行时中间状态的数据
- 持久性(Durability)
  - 成功完成的事务所对数据库所作的更改便持久的保存在数据库之中，并不会被回滚
- 可见性(Visibility)
  - 事务对数据的更改对任何后续的读操作可见

HBase的原子性保证

HBase仅保证对行操作的原子性

- 任何行级的操作是原子的
  - 一条记录的Put操作要么完全成功，要么完全失败。
    - 操作返回成功(success)表示操作完成
    - 操作返回失败(failure)表示操作全部失败
    - 超时操作可能是成功也可能失败，但不可能部分成功
  - 即使跨column family的行操作也是原子的
- 支持一次性修改多行的API并不保证跨行的原子性操作
  - 一般情况下，API会在结果中分别返回执行成功、失败以及超时的行操作列表
- checkAndPut()方法是原子性的
  - 相当于典型的compareAndSet (CAS)语义
- 对行的修改顺序被严格定义，不可能出现修改交织
  - 比如，一个用户发出修改“a=1, b=1, c=1”，而另一个用户发出修改“a=2, b=2, c=2”，这行的值要么为“a=1, b=1, c=1”，要么为“a=2, b=2, c=2”，而不可能是 "a=1, b=2, c=1".
  - 如果是跨行的批量修改不保证所有行之间不出现交织

HBase的一致性以及可见性

- 任意API返回的任意多的行都曾经完整的在表历史上的某个时间点上出现过
  - 即使跨column family也成立
  - 如果试图取一行完整的记录，该纪录同时依次在进行1, 2, 3, 4, 5更改操作，则返回的必定是在1和5之间某个时间里出现过的一条完整的记录
- 行的状态是按时间单调递增的(日志)
- Scan**不**能返回一个恒定的表视图，及Scan**不**是快照隔离的 (*snapshot isolation*)

内容

- HBase的历史和特点
- HBase和其他Hadoop成员的关系
- HBase的数据模型
- HBase的访问接口
- HBase的运行组成
- HBase Region
- HBase的读写
- 其他HBase功能

HBase运行环境

- HBase与集群中其它系统之间的关系：
  - HDFS：分布式文件系统，用以存储数据，将数据持久化
  - Zookeeper：分布式的协调服务器，用以提供高可靠的锁服务，提供可靠的小文件的读写；HBase使用Zookeeper服务来进行节点管理以及表数据的定位
  - MapReduce：分布式环境下的一个编程框架；可以把HBase作为数据源和目的

ZooKeeper简介

- ZooKeeper是一个为分布式应用程序进行协调的服务，这样的话，每一个分布式的应用程序如果需要进行协调的话就可以直接使用ZooKeeper所提供的服务
- ZooKeeper提供了一系列分布式系统的基本服务或者可以基于ZooKeeper完成分布式系统的基本服务：同步、配置管理、分组和命名
- ZooKeeper提供了一个易于编程的环境，实现了一个简化的文件系统，提供类似的目录树结构
- ZooKeeper使用Java编写，支持了Java以及C语言绑定
- 分布式的协调服务coordination非常容易出错，出错之后也很难恢复，例如死锁状态，或者出现资源竞争状态，通过ZooKeeper可以以良好的编程接口将程序员从自己构造协调服务的负担中解放出来

ZNode

- ZooKeeper文件树中的节点称作znode。
- znode会维护一个包含数据修改和ACL修改版本号的Stat结构体，这个结构体还包含时间戳字段。
- 版本号和时间戳让ZooKeeper可以校验缓存，协调更新。每次修改znode数据的时候，版本号会增加。
- 客户端获取数据的同时，也会取得数据的版本号。
- 执行更新或者删除操作时，客户端必须提供版本号。
- 如果提供的版本号与数据的实际版本不匹配，则更新操作失败。

临时节点

- ZooKeeper有临时节点的概念
- 临时节点在创建它的会话活动期间存在
- 会话终止的时候，临时节点被删除，所以临时节点不能有子节点

Znode上的观察器

- 客户端可以在znode上设置观察器。对znode的修改将触发观察器，然后移除观察器。观察器被触发时，ZooKeeper向客户端发送一个通知。
- 观察器用来让应用程序可以得到异步的通知

Zookeeper数据存取

- 存储在名字空间中每个znode节点里的数据是原子地读取和写入的。读取操作获取节点的所有数据，写入操作替换所有数据。
- 节点的访问控制列表(ACL)控制可以进行操作的用户。
- ZooKeeper不用以真正存储数据，而是保存协调数据。
  - 协调数据例如配置、状态信息等，通常比较小，以千字节来衡量。
- znode数据应该小于1MB，实际数据一般远小于1MB。
- 较大的数据不适合使用ZooKeeper存储，如果需要大数据存储，通常方式是存储到块存储系统，如NFS或者HDFS中，然后在ZooKeeper中保存到存储位置的指针。

内容

- HBase的历史和特点
- HBase和其他Hadoop成员的关系
- HBase的数据模型
- HBase的访问接口
- HBase的运行组成
- HBase Region
- HBase的读写
- 其他HBase功能

HBase中的数据模型

行主键

列族

row Key	column-family1		column-family2			column-family3
	column1	column2	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

时间戳

HBase表不要求是结构化的。本质上，HBase表可认为是一个巨大的稀疏矩阵。

行主键(Rowkey)

- 行主键是用来检索记录的主键
- 行主键是最大长度64KB的数组，实际应用中长度一般为 10-100bytes
- 访问HBase table中的行，有三种方式：
  - 通过单个row key访问
  - 通过row key的范围（range）来访问
  - 全表扫描
- 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。(空间局部性)
  - 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)

列族

- 列族的设计与传统数据库中的列不一致
- HBase表中的每个列，都归属与某个列族。列族是表的schema的一部分，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。

时间戳timestamp

- HBase中通过row和column确定一个存贮单元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(64位整型)来索引
  - 时间戳可以由HBase(在数据写入时自动用当前系统时间)赋值
  - 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
  - 保存数据的最后n个版本
  - 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TTL （Time to Live）

数据单元的映射关系

- 每行每列族中的列数量是可以不同的，数量可以很大
- 简单来说，可以认为每行每列族中保存的是一个Map

- 由*{row key, column family, column name, timestamp}*可以唯一的确定一个单元。单元中的数据是没有类型的，全部是字节码形式存贮
- 实际存储时，也是这个key-value结构
- 表设计时要避免由此引起更多的负担

内容

- HBase的历史和特点
- HBase和其他Hadoop成员的关系
- HBase的数据模型
- HBase的访问接口
- HBase的运行组成
- HBase Region
- HBase的读写
- 其他HBase功能

HBase的访问接口

- Java API
  - 最常规和高效的访问方式
- HBase Shell
  - HBase的命令行工具，最简单的接口，适合HBase管理使用
- Thrift Gateway
  - 利用Thrift序列化技术，支持C++，PHP，Python等多种语言，适合其他异构系统在线访问HBase表数据
- REST Gateway
  - 支持REST 风格的Http API访问HBase，和thrift类似
- MapReduce/Pig/Hive
  - 使用TableInputFormat和TableOutputFormat来支持HBase作为MapReduce的输入和输出

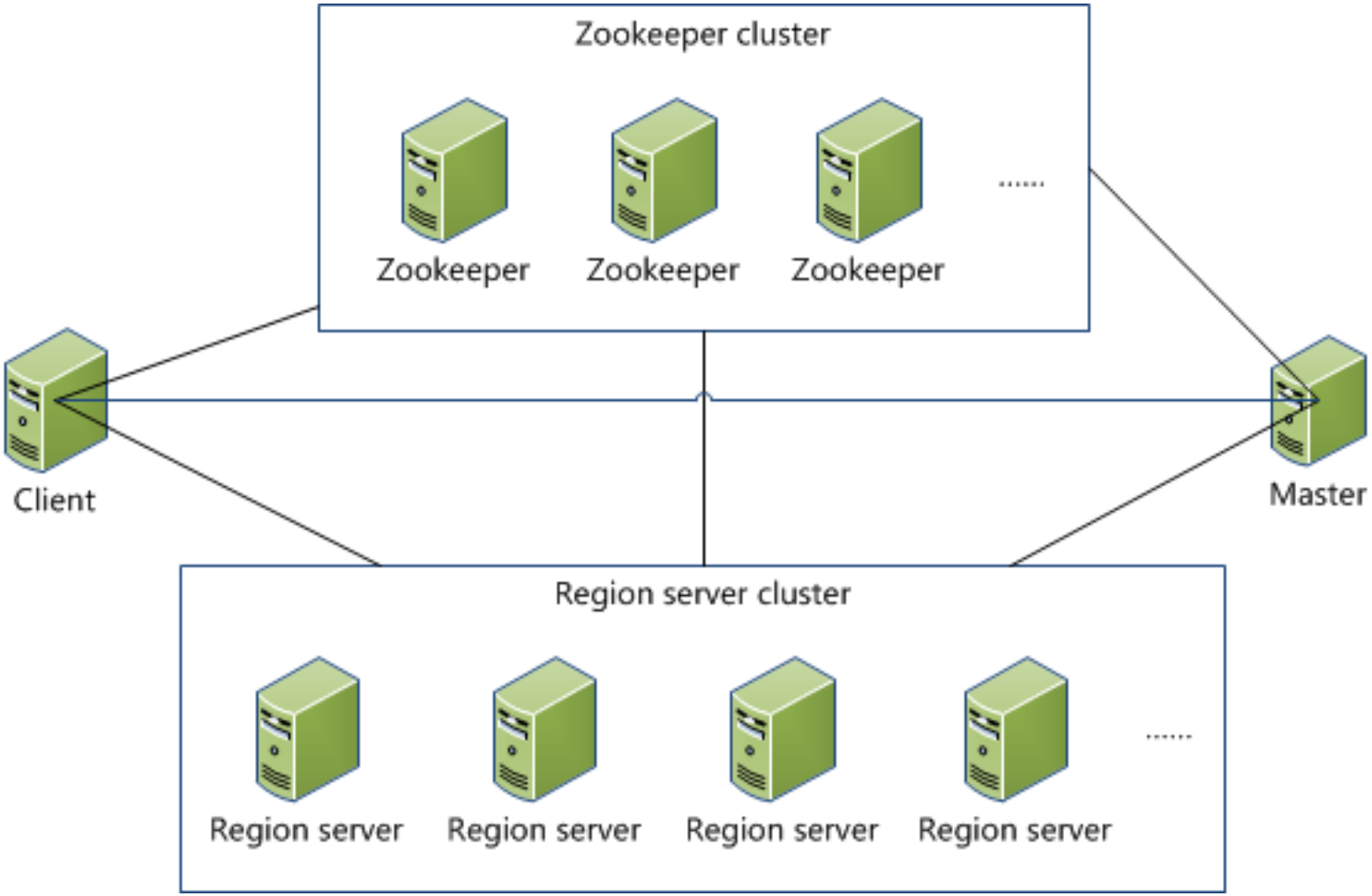
HBase的客户端基本操作

- 写入，没有transaction，但对单行的更新是原子的
  - Put
  - Delete
  - Append
  - Increment
- 读取
  - Get
  - Scan

内容

- HBase的历史和特点
- HBase和其他Hadoop成员的关系
- HBase的数据模型
- HBase的访问接口
- HBase的运行组成
- HBase Region
- HBase的读写
- 其他HBase功能

HBase的运行组成


Zookeeper cluster
Zookeeper
Zookeeper
Zookeeper
.....
Client
Master
Region server cluster
Region server
Region server
Region server
Region server
.....

HBase的组成模块(1)

- **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**
  - 保证任何时候，集群中只有一个master
  - 存贮目录表的寻址入口。
  - 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
  - 存储Hbase的schema,包括有哪些表，每个表有哪些列族

HBase的组成模块(2)

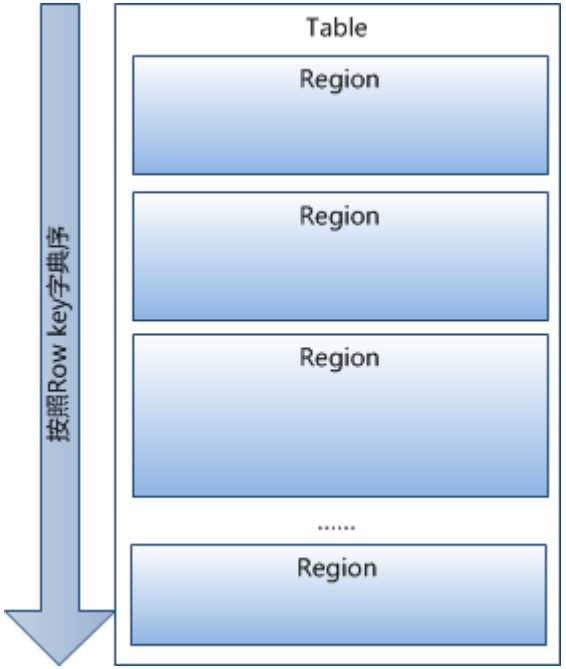
- **Master**
  - 为Region server分配region
  - 负责region server的负载均衡
  - 发现失效的region server并重新分配其上的region
  - HDFS上的垃圾文件回收
  - 处理schema更新请求
- **Region Server**
  - Region server维护Master分配给它的region，处理对这些region的IO请求
  - Region server负责切分在运行过程中变得过大的region
- 注意：client访问hbase上数据的过程并不需要master参与（寻址访问zookeeper和region 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。

内容

- HBase的历史和特点
- HBase和其他Hadoop成员的关系
- HBase的数据模型
- HBase的访问接口
- HBase的运行组成
- HBase Region
- HBase的读写
- 其他HBase功能

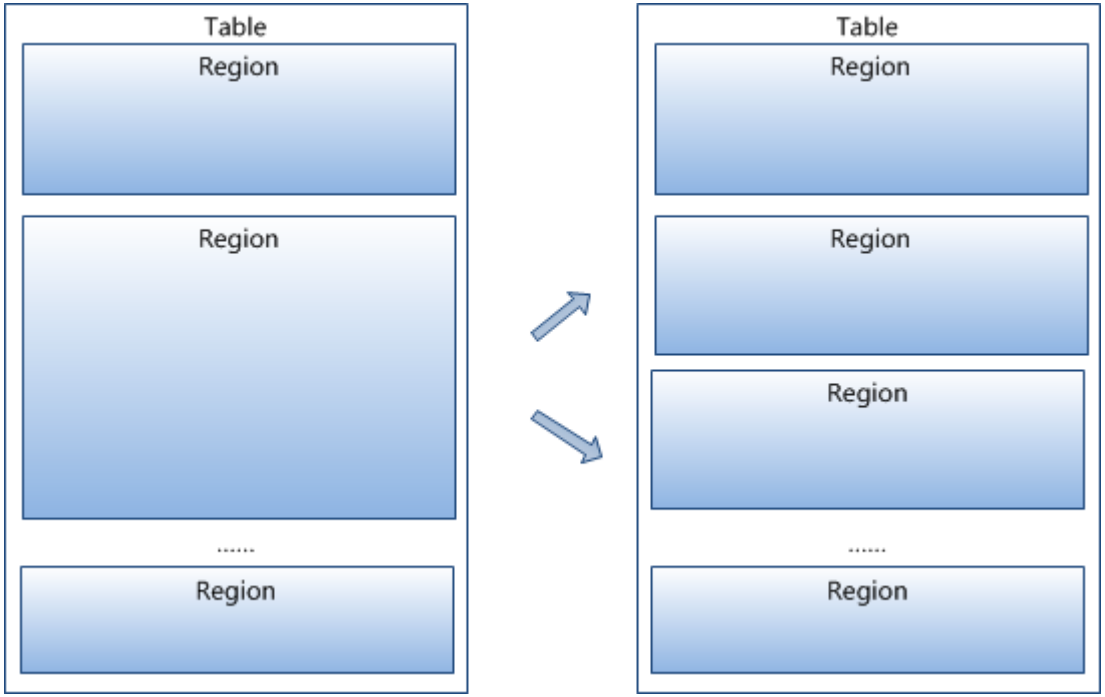
Region in HBase

- Table 在行的方向上分割为多个Region



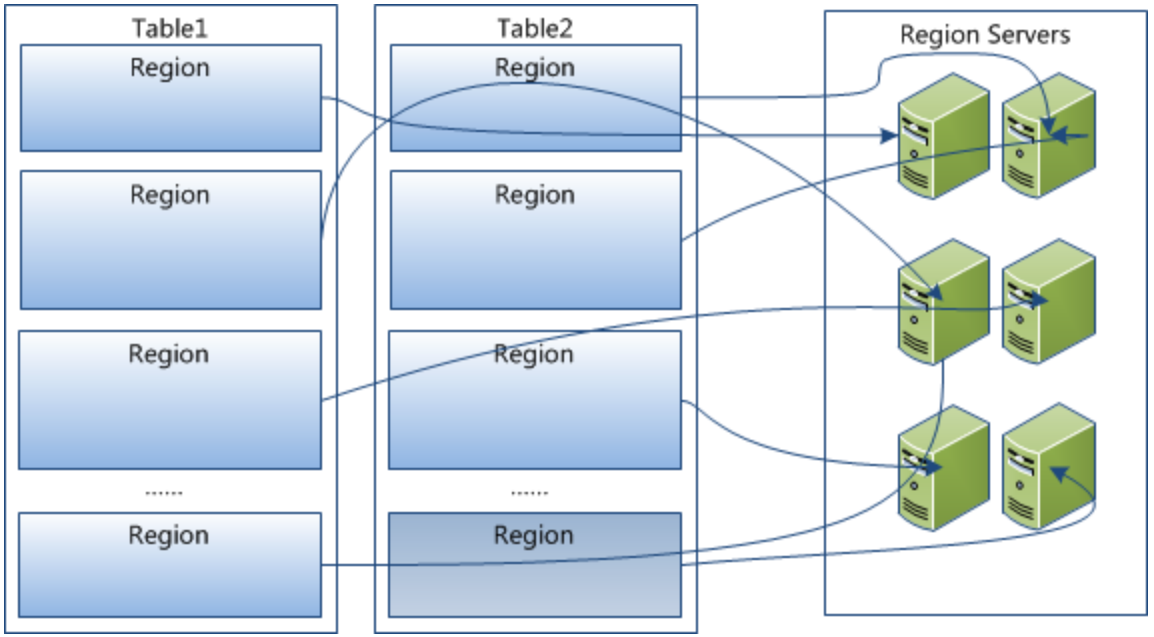
Region的分割

- Region按大小分割，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，原来的Region就会等分成两个新的Region。当表中的行不断增多，Region的数目也会逐渐增多



Region到数据分表的分配

- Region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Region可以分布在不同的Region server上。但一个Region是不会拆分到多个server上



Region元数据

- Region是Region Server管理和调度的最小单位
- 每个Region由以下信息标识：
  - <表名，startRowKey，创建时间>
  - 由目录表(-ROOT-和.META.)可知该Region的endRowKey
- 每个列族单独存储，当列族大小 > 某阈值时，自动分裂成两个Region

如何找到某行属于哪个region呢？

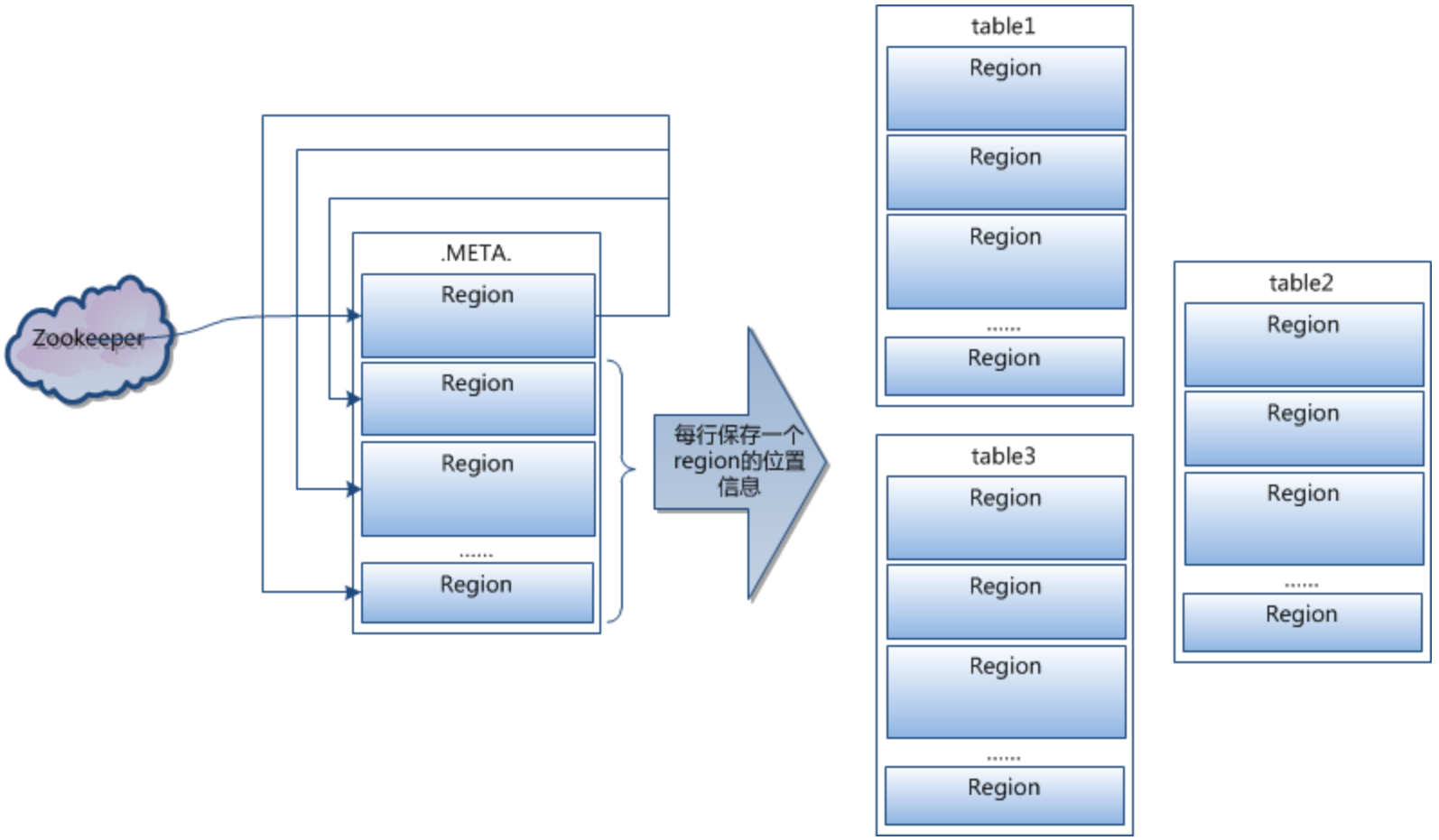
- -ROOT- & .META.

HBase Region的定位(1)

- Region被分配给哪个Region Server是完全动态的，所以需要机制来定位Region具体在哪个region server
- HBase使用三层结构来定位region

1. 通过zookeeper里的文件得到-ROOT- 表的位置
   - -ROOT-表只有一个region
2. 通过-ROOT-表查找.META.表中相应region的位置
   - 其实-ROOT-表是.META.表的第一个region
   - .META.表中的每个region在-ROOT-表中都是一行记录
3. 通过.META.找到所要的用户表region的位置
   - 用户表中的每个region在.META.表中都是一行记录

HBase Region的定位(2)

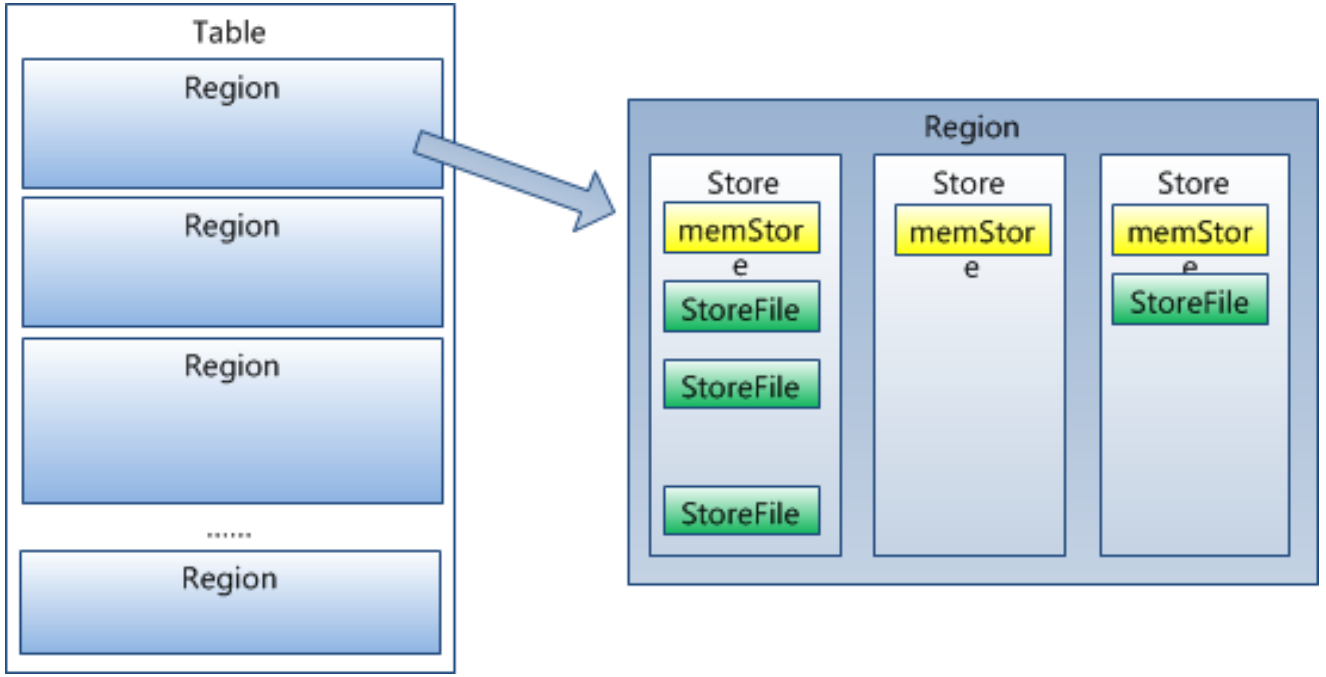


HBase Region的定位（3）

- -ROOT-表永远不会被分割为多个region，保证了只需要三次跳转，就能定位到任意region
- .META.表每行保存一个region的位置信息
- 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
- 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)。

Region的内部

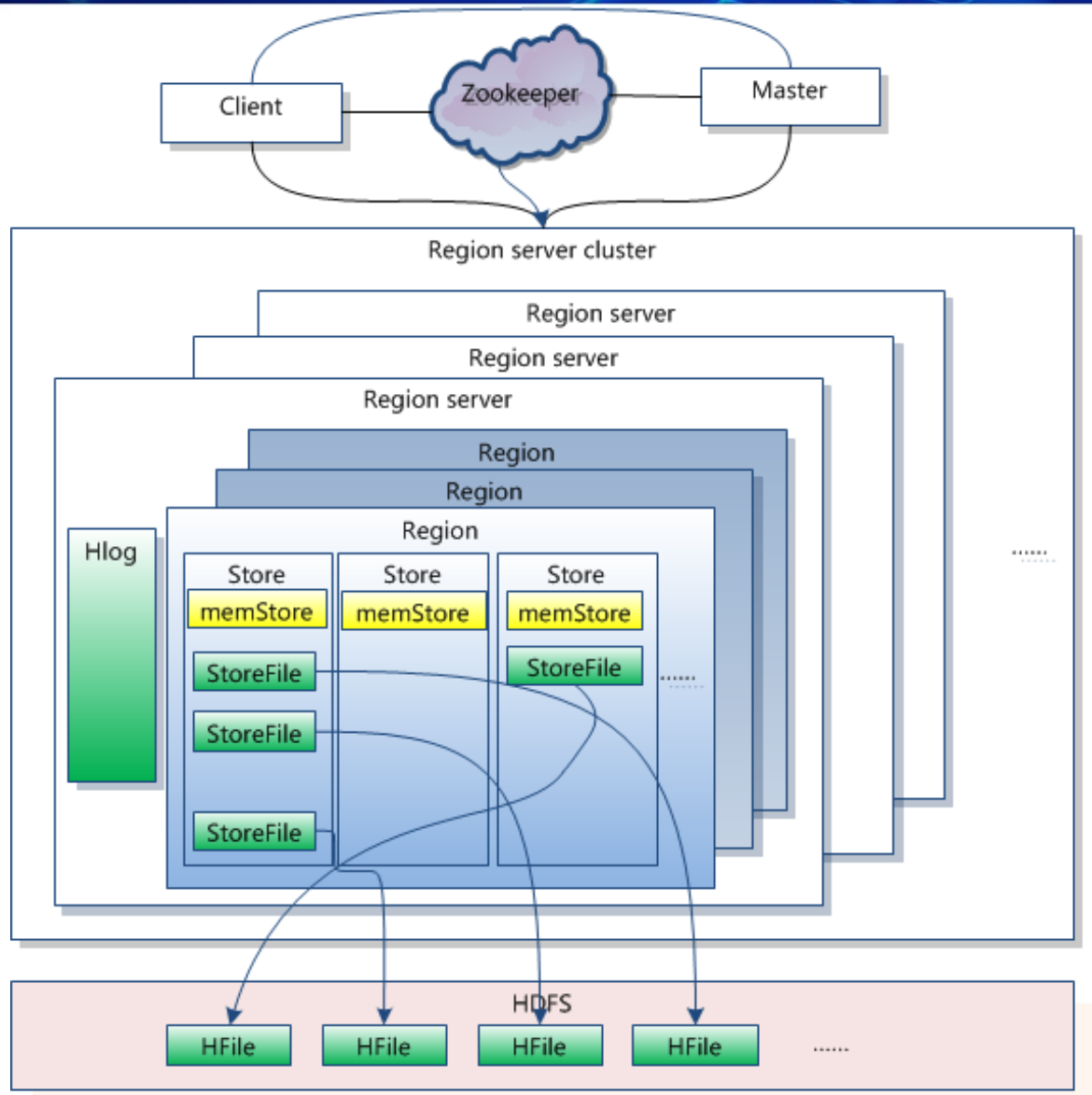
- Region由一个或者多个Store组成，每个列族一个Store。
- 每个Store又由一个memStore和0至多个StoreFile组成。
- StoreFile以HFile格式保存在HDFS上。



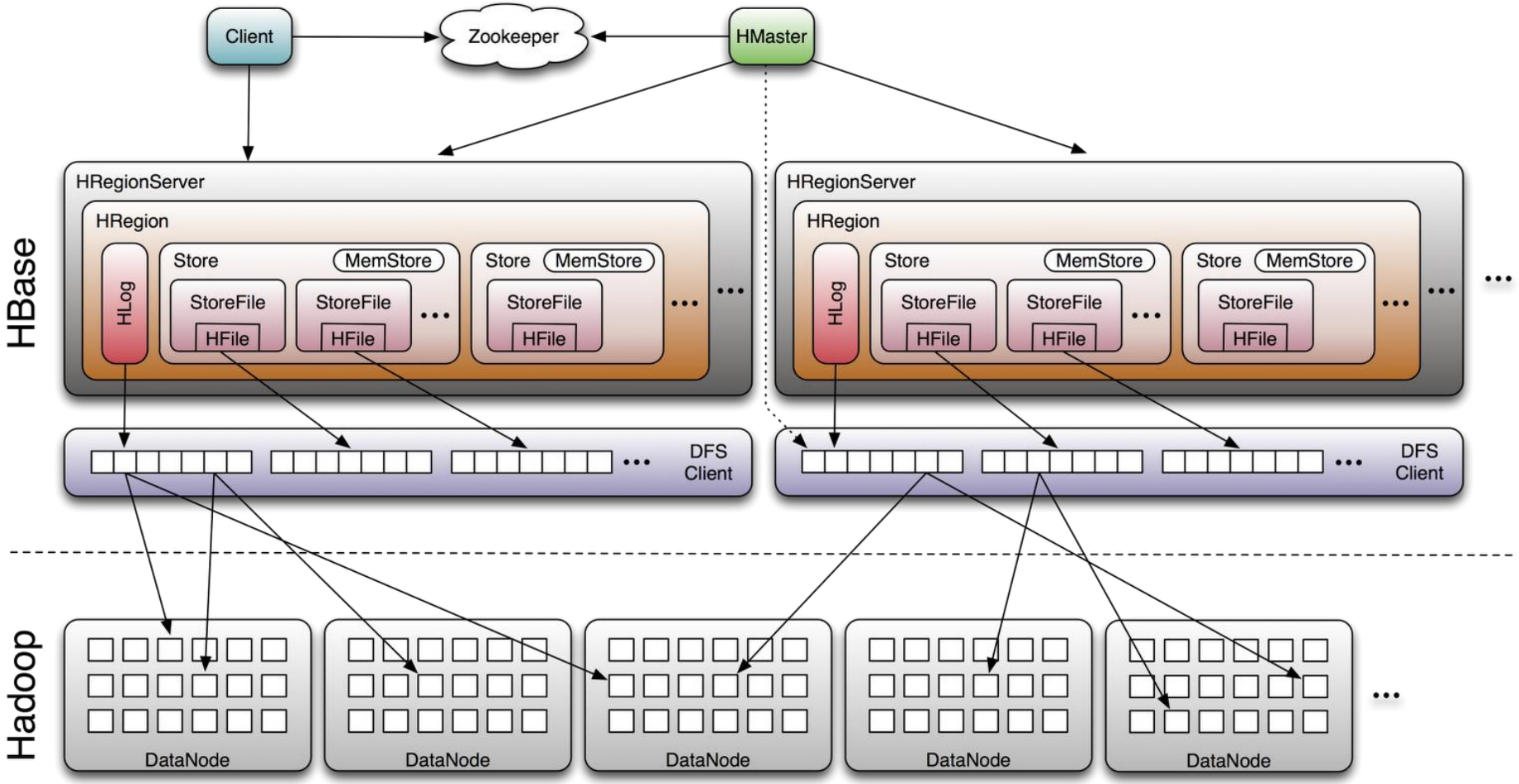
HBase存储格式

- HBase中的所有数据文件都存储在Hadoop HDFS文件系统上，主要包括两种文件类型：
- 1.HFile， HBase中KeyValue数据的存储格式，HFile是Hadoop的二进制格式文件，实际上StoreFile就是对HFile做了轻量级包装，即StoreFile底层就是HFile
- 2.HLogFile，HBase中WAL（Write Ahead Log） 的存储格式，物理上是Hadoop的Sequence File

HBase的数据存储


Client
Zookeeper
Master
Region server cluster
Region server
Region server
Region server
Region
Region
Region
Hlog
Store
memStore
StoreFile
StoreFile
StoreFile
Store
memStore
Store
memStore
StoreFile
HDFS
HFile
HFile
HFile
HFile

HBase体系结构


Client
Zookeeper
HMaster
HBase
HRegionServer
HRegion
HLog
Store
MemStore
StoreFile
HFile
DFS Client
Hadoop
DataNode

Region的分配

- 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，master从当前活着的region server中选取一个，向其发送一个装载请求，把region分配给这个region server。
- region server得到请求后，就开始加载这个region，并对外提供服务。

Region Server的上线

- master使用zookeeper来跟踪region server状态
- 当region server启动时，会在zookeeper上的/hbase/rs目录下建立代表自己的文件
- 由于master订阅了/hbase/rs目录的变更消息，当该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的下线

- 当region server下线时，它和zookeeper的会话断开，zookeeper自动释放代表这台server的文件
- 因此master会得到zookeeper的通知某台region server下线了，master会

1. 让其他region server帮助处理这台region server的WAL log保证数据完整性
    1. 先按region将WAL分割成多个log
    2. 再将不同的region+log交由不同的region server去replay log，从而将数据恢复到最新
2. 将这台region server的region分配给其它还活着的同志

Master上线

1. 从zookeeper上获取唯一一个代表master的锁（/hbase/master），用来阻止其它master活跃
2. 扫描zookeeper上的/hbase/rs目录，获得当前可用的region server列表
3. master与每个region server通信，获得当前已分配的region和region server的对应关系
4. 确保-ROOT-和.META.表已分配
5. 扫描.META.表，计算当前还未分配的region，将他们放入待分配region列表，执行分配

Master下线

- 由于master只维护表和region的元数据，而不参与region的读写，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。
- 因为master保存的信息全是冗余信息（都可以从系统其它地方收集到或者计算出来），因此，HBase的HA很容易，启动多个master就可以了

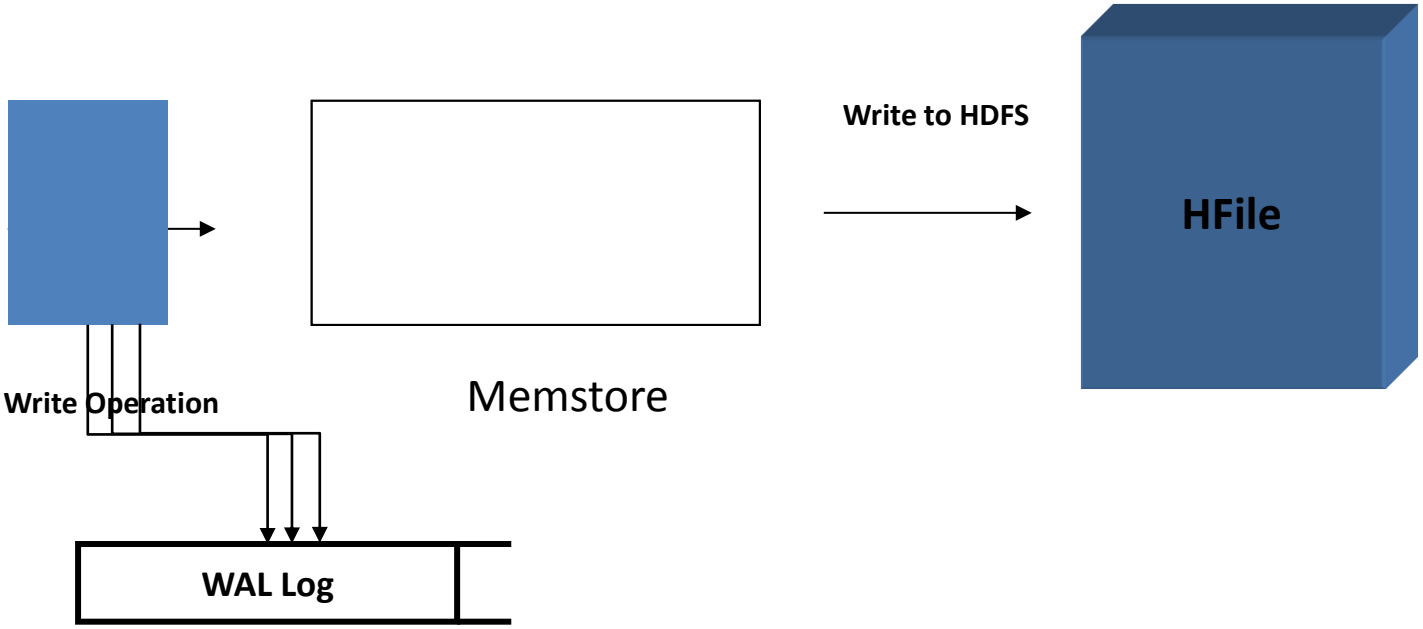
内容

- HBase的历史和特点
- HBase和其他Hadoop成员的关系
- HBase的数据模型
- HBase的访问接口
- HBase的运行组成
- HBase Region
- HBase的读写
- 其他HBase功能

数据写入过程

- 写入过程
  1. Client先根据row key找到对应的region和region server
  2. client向region server提交写请求
  3. region server找到目标region
  4. region检查数据是否与schema一致
  5. 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
  6. 将更新写入WAL log
  7. 将更新写入Memstore
  8. 判断Memstore的是否需要flush为Store文件

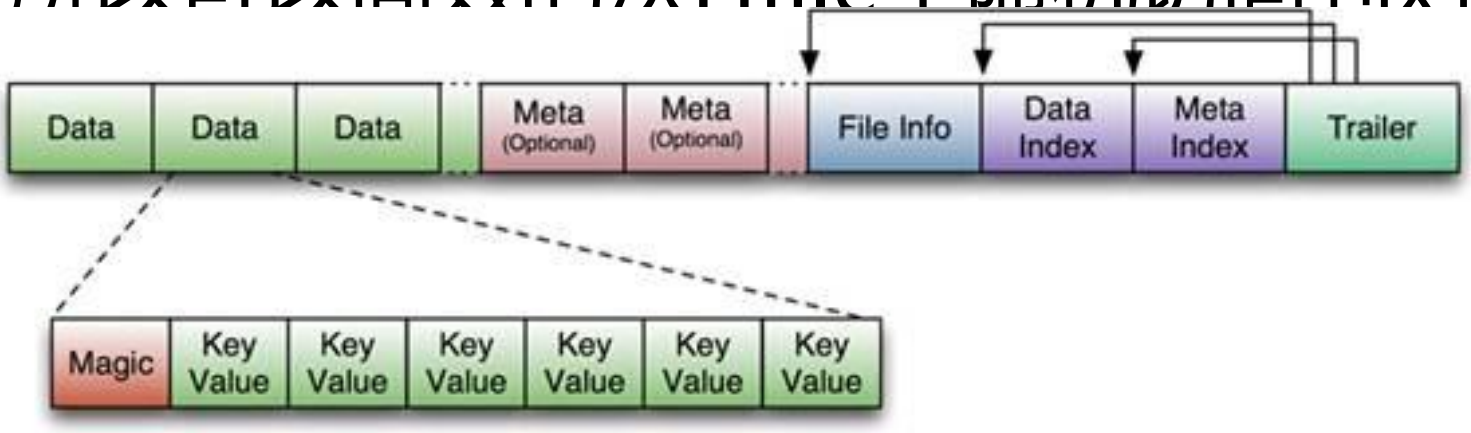
数据的写入



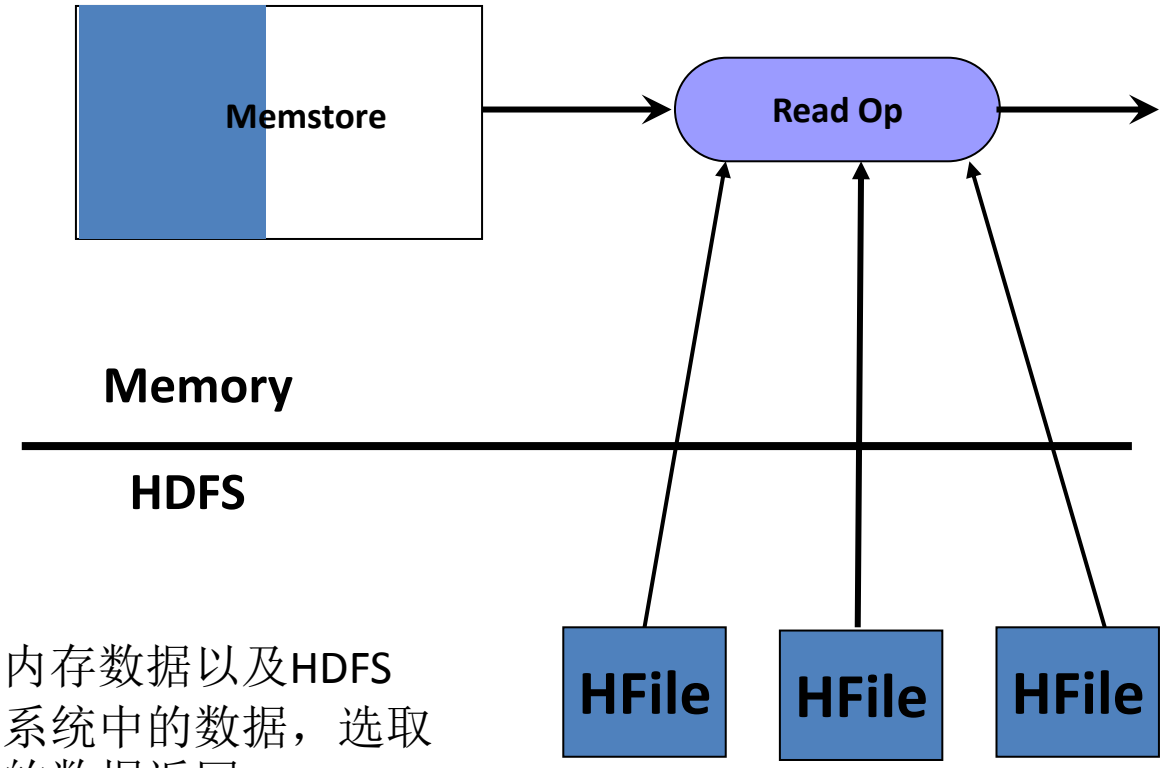
先写日志，然后将数据放入到内存表中，
到达一定数目后写入到磁盘，格式为HFile

HFile

- Hfile是只读的，不可修改
- Hfile内存储的key-value对是按照key排过序的
- Hfile内的数据块有索引
- 所以可以高效的从Hfile中随机访问到对应



数据的读取



扫描内存数据以及HDFS文件系统中的数据，选取正确的数据返回

记录数据的改写以及删除

- 记录数据的改写以及删除与数据写入的过程是完全一样的：
- Update
  - 写入：仅需要将最新的数据按正常写入操作写入
  - 读取：按正常读取，但将旧数据从结果集中删除

Delete

  - 写入：将数据删除事件作为一个正常的操作写入
  - 读取：按正常读取，但将要删除的数据从结果集中删除

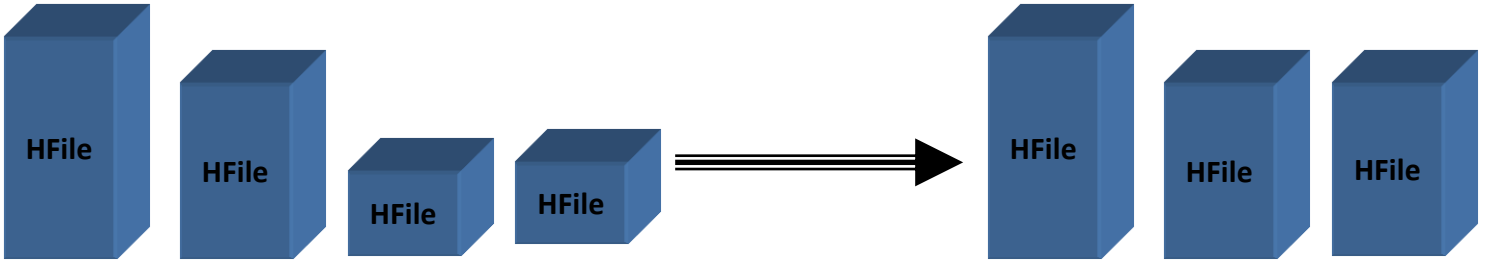
数据整理（Compaction）

进行数据整理以：

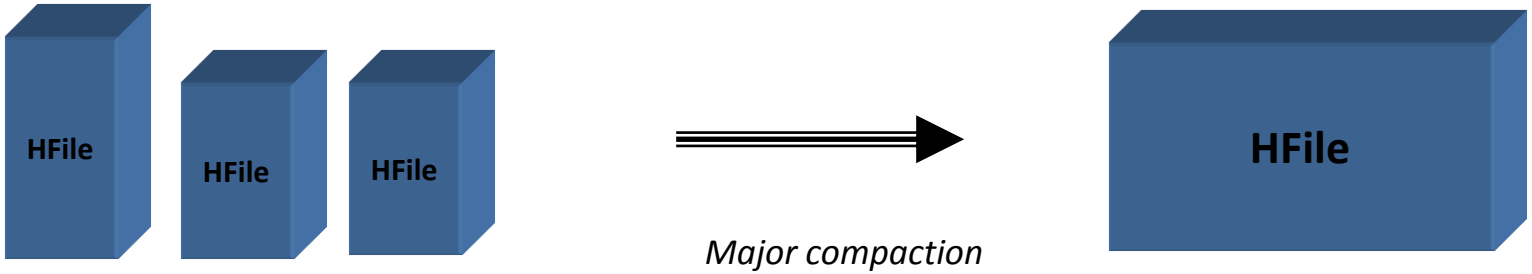
- 提高读取性能。读取要将所有Hfile的结果合并，所以文件数对读取性能影响很大
- 消减update和delete动作产生的无用数据

- Minor Compaction，小规模数据整理
  - 将最新的若干Hfile合并，减少文件数
- Major Compaction，大规模的数据整理
  - 将region某个列族的所有Hfile合并成一个Hfile
  - Delete动作只在major compaction时发生

数据整理



Minor compaction



Major compaction

Bloom Filter和Block Cache

- Bloom Filter:用来高效检测一个元素是不是集合成员。返回“在集合内（可能错误）”和“绝对不在集合内”两种情况
- 每个Hfile可设置Bloom Filter
  - 可设置为row key 或者row key + column name
- 每个Hfile的bloom filter和index会在内存中
  - 当读取时，如果bloom filter说要读的行不在Hfile中出现就不读取对应文件，从而减少IO
  - 仅对Get有效，对scan无效
- Block cache: 用于缓存Hfile中的数据块，也用于缓存bloom filter和index

HBase写入性能讨论

写入时的性能瓶颈：

- 客户端
  - 使用Write buffer减少RPC
  - 避免频繁创建HTable对象
  - 如果可以，关闭WAL
- Region负载不均衡：要让写均匀分布到所有的region server上
  - 如果写入的row key是基本单调的（例如时序数据），那么基本上会都落在同一个region上，所以只有一个region server活跃，总体性能会很差。（但这种情况下通常读性能最好）
- 过多的compaction和compaction不及时
  - 尽量避免：如增加compaction thread数，防止高峰期做compaction等策略
- 过多的split
  - 预分配region，有时因上层逻辑便利需禁止split

HBase读性能讨论

- 尽量使用Get和基于range的Scan
  - 如果读取的数据按row key连续，访问非常快，毫秒级返回
  - 如果读取非连续的数据，就要scan全表，非常慢
- 使用RowFilter减少不匹配行，提高性能
  - 如使用FuzzyRowFilter跳过不匹配区间
- 使用Filter限定结果集
- 二级索引(Secondary Index)
  - HBase无二级索引功能
  - 可选用其他项目的增强功能或在客户端插入时用户自行建立索引

Bulk Load

- 批量将数据导入到HBase中
- 步骤：
  1. 用MapReduce将数据根据当前的region情况用TotalOrderPartitioner按region直接生成对应的Hfile
  2. 将这些Hfile加载到HBase中去

- 在大数据量时，性能要比直接Put好
  - 一定要预分配region
- 有命令行接口，但很多时候还是要自己写点代码

HBase的性能优化

- 预分配region
- 启用压缩已减少HDFS数据量，可提高读性能
- Region Server进程配置大内存（>16G）
- 每个Region Server拥有的region数量<300
- 优化表结构设计，防止少数几个region成为瓶颈
  - 一个简单的经验公式：每台region server纯写入时高负载应能达到>1万条记录/秒（每记录200字节）

内容

- HBase的历史和特点
- HBase和其他Hadoop成员的关系
- HBase的数据模型
- HBase的访问接口
- HBase的运行组成
- HBase Region
- HBase的读写
- 其他HBase功能

Coprocessor

HBase 0.92引入了coprocessor框架，包含：

- Observer
  - 类似于数据库的trigger。当某些事件发生时触发，如pre-Get, post-Get, pre-Flush等
- Endpoint
  - 一种分布式计算框架，类似MPP架构。用法相当于数据库的存储过程/函数。可并行的在每个region server上以region为单位进行并行处理，然后返回到客户端对结果进行汇总处理。

优点：性能好，可扩展性强

- 本地化计算，减少网络传输

缺点：稳定性+安全性

- Region Handler内执行
- 不适合复杂逻辑或IO相关应用

参考资源

- 项目主页: hbase.apache.org
- HBase参考指南 http://abloz.com/hbase/book.html

谢谢！