# hbase分布式存储机制（工作原理详解）

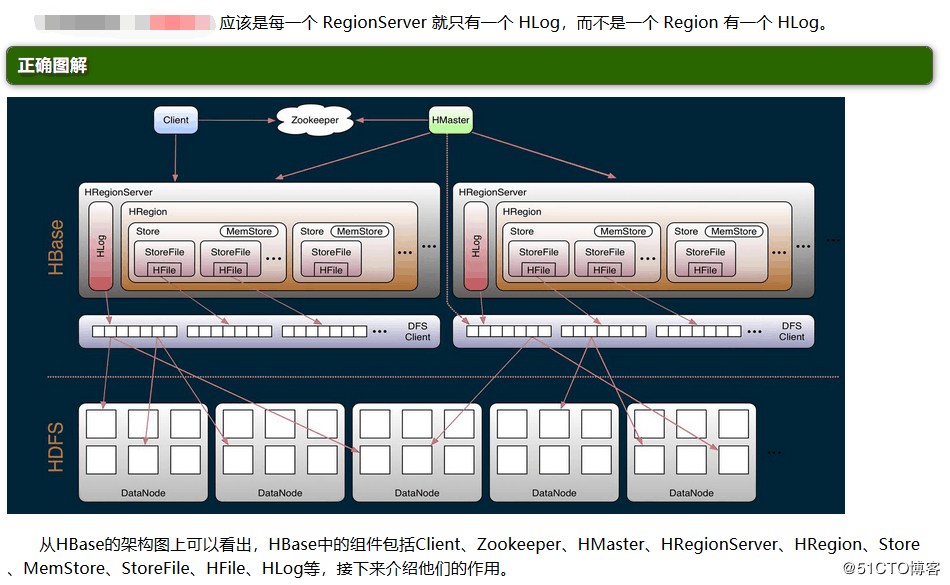
## 

参考链接

<https://www.cnblogs.com/qingyunzong/p/8692430.html>

**hbase分布式存储机制（工作原理详解）**

hbase系统架构图



**hbase集群中每个组件的作用**

**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regione 的位置信息。

1、HBase 有两张特殊表：

.META.：记录了用户所有表拆分出来的的 Region 映射信息，.META.可以有多个 Regoin

-ROOT-：记录了.META.表的 Region 信息，-ROOT-只有一个 Region，无论如何不会分裂

2、Client 访问用户数据前需要首先访问 ZooKeeper，找到-ROOT-表的 Region 所在的位置，然 后访问-ROOT-表，接着访问.META.表，最后才能找到用户数据的位置去访问，中间需要多次 网络操作，不过 client 端会做 cache 缓存。

//客户端直接与regionserver联系的

备注：0.96 版本以后将-ROOT-表去掉了。至于为什么，请详细见，hbase的寻址地址过程。

.META. -ROOT- 这2个表的信息都是通过zookeeper中的信息去找的，.META. -ROOT-这两个表具体是在regionserver的服务器上

**Hmaster** ：

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（模式） 更新请求 //可以通俗的理解为管理用户对表的增删改成操作

**regionserver**

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 //真正管理表实际数据的

3 regionsever 负责响应用户的IO请求，并且向hdfs中读写数据

//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regionserver，数据读写访问regione server），master 仅仅维护者table 和region 的元数据信息，所有负载很低。

提示：一般的regionserver和hdfs中的datanode并置，datanode存储有regionserver正在管理的数据，hbase的数据最终都是存储在hdfs上。

**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（模式）,包括有哪些table，每个table 有哪些column family

**HRegion**

table在行的方向上分隔为多个Region。Region是HBase中分布式存储和负载均衡的最小单元，即不同的region可以分别在不同的Region Server上，但同一个Region是不会拆分到多个server上。Region按大小分隔，每个表一般是只有一个region。随着数据不断插入表，region不断增大，当region的某个列族达到一个阈值时就会分成两个新的region。每个region由以下信息标识：< 表名,startRowkey,创建时间> 由目录表(-ROOT-和.META.)记录该region的endRowkey

**Memstore store 与storefile**

一个region 由多个store 组成，每个store 包含一个列族的所有数据 Store 包括位于把内存的memstore 和位于硬盘的storefile

写操作先写入memstore,当memstore 中的数据量达到某个阈值，regionserver 启动flashcache进程写入storefile,每次写入形成单独一个storefile。

当storefile 大小超过一定阈值后，会把当前的region 分割成两个，并由Hmaster 分配奥相应的region 服务器，实现负载均衡

客户端检索数据时，先在memstore 找，找不到再找storefile

**HLog(WAL log)**

WAL 意为Write ahead log([http://en.wikipedia.org/wiki/Write-ahead\_logging)，类似mysql](http://en.wikipedia.org/wiki/Write-ahead_logging)%EF%BC%8C%E7%B1%BB%E4%BC%BCmysql) 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regionserver 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。

**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。

Trailer 部分的格式:

HFile 分为六个部分：

1、Data Block 段–保存表中的数据，这部分可以被压缩

2、Meta Block 段(可选的)–保存用户自定义的kv 对，可以被压缩。

3、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。

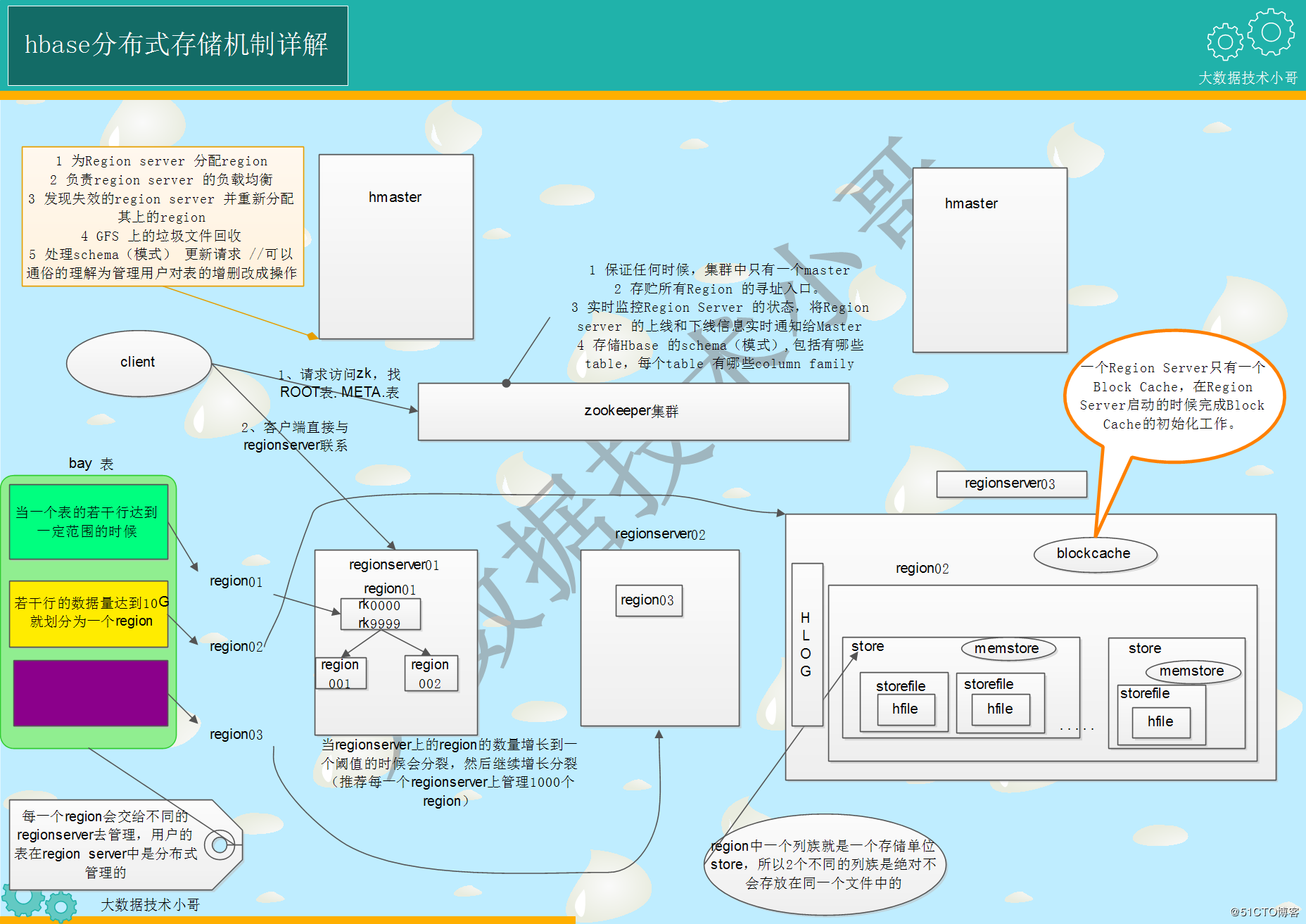
4、Data Block Index 段–Data Block 的索引。每条索引的key 是被索引的block 的第一条记录的key。

5、Meta Block Index 段(可选的)–Meta Block 的索引。

6、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。

**hbase分布式存储机制详解**

****

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region。

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，region 就会等分会两个新的region。当table 中的行不断增多，就会有越来越多的region。

4 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HRegion server 上。但一个region 是不会拆分到多个server 上的。

5、当regionserver上的region的数据量增长到一个阈值的时候会分裂，然后继续增长分裂。（推荐每一个regionserver管理1000个region）

再到细节中，一个region中的数据最终存储到hdfs上去的时候，会采用什么样子的机制呢？

6、Region 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，Region 由一个或者多个Store 组成，每个store 保存一个columns family（列族）。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。

//region中的一个列族就是一个物理的存储单位，所有2个不同的列族是绝对不可能存放在同一个文件中的。

7、每个Strore 又由一个memStore 和0 至多个StoreFile 组成。memstore是共享的，相当于整个store的内存缓存，如果客户端去读数据的时候，如果命中到哪里就会优先从memstore中拿或者写数据。如果memstore中没有我们要取的数据的话，接下来才会到storefile中去找。memstore可以理解为一个缓存的机制。如果是写数据的时候，会先向memstore中去写，然后过段时间才会把memstore刷成storefile（这其实就是region flush：写数据的时候先写到memstore中，过段时间再flush成storefile storefile会转化成hfile 最终存储到hdfs上）//StoreFile 以HFile 格式保存在HDFS 上。

8、当有多个storefile超过我们设定的阈值的时候，会出现compactition（压缩）操作。将允许操作将所有的storefile文件作为一个storefile重新写入（每次memstore刷新写入一个storefile）你可以指定更大数量延压缩，但压缩将运行更长时间，在压缩期间，更新将无法刷新到磁盘。长时间的压缩需要足够的内存，再以压缩的持续时间内记录所有更新，如太大，压缩期间，客户端会超时。

9、在hfile端也会出现压缩的情况：hbase会自动拾取一些较小的hfile，并将它们写入一些较大的hFile中，这个过程叫minor compacition ：minor compacatition通过重写操作，利用合并排序将较小的文件转化为较大数量却数量较少的文件中。

**提示：在有些资料中会单独的提到blockcache是读操作的缓存，他保存了内存中经常被读的一些信息。memstore是写操作的缓存，**

**我们把他们理解成一样的，作用为缓存就可以了。不必深究。**

*如果要深入理解 memstore和blockcache的具体区别，请参考此文中的问题1 问题2 的解答*

HBase原理－迟到的‘数据读取流程’部分细节 <http://hbasefly.com/2017/06/11/hbase-scan-2/>

HBase BlockCache系列 – 走进BlockCache <http://hbasefly.com/2016/04/08/hbase-blockcache-1/>

1、BlockCache是Region Server级别的，

2、一个Region Server只有一个Block Cache，在Region Server启动的时候完成Block Cache的初始化工作。

3、到目前为止，HBase先后实现了3种Block Cache方案，LRUBlockCache是最初的实现方案，也是默认的实现方案；HBase 0.92版本实现了第二种方案SlabCache，见HBASE-4027；HBase 0.96之后官方提供了另一种可选方案BucketCache，见HBASE-7404。

具体memstore 和blockcache的具体区别 请看后续的博客！

问题1：

常说HBase数据读取要读Memstore、HFile和Blockcache，为什么上面Scanner只有StoreFileScanner和MemstoreScanner两种？没有BlockcacheScanner?

答

1、HBase中数据仅仅独立地存在于Memstore和StoreFile中

2、Blockcache中的数据只是StoreFile中的部分数据（热点数据）即所有存在于Blockcache的数据必然存在于StoreFile中。

3、因此MemstoreScanner和StoreFileScanner就可以覆盖到所有数据。

4、首先查数据的时候，先会从blockcache中找，如果存在直接取出；如果没有再到storefile中查找。

问题2：

数据更新（写）操作先将数据写入Memstore，再落盘。落盘之后需不需要更新Blockcache中对应的kv？如果不更新，会不会读到脏数据？

答：

1、不需要更新blockcache中的数据

2、不会读到脏数据，因为数据写到memstore中落盘形成新的文件。

3、memstore形成的新的文件和blockcache里面的数据是相互独立的，以版本方式存在。

**Region 管理机制**

(1) 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 server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。

(2) region server 上线

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。

(3)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，(或者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。

的某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。

**HMaster 工作机制**

（1）Hmaster 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。

（2）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（模式），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。