【第七期分享主题】

HBase概述、数据模型及表设计优化

【分享嘉宾】

斗鱼数据平台部大数据处理组成员刘宇超

【分享内容】

大家好，我是刘宇超，是斗鱼大数据处理组工程师，主要负责斗鱼搜索服务。2017年4月份加入斗鱼，之前负责爱奇艺搜索组线上服务

今天跟大家分享的主题是《HBase概述、数据模型及表设计优化》

## 一、Hbase概述

HBase 是Google Bigtable 的开源版本，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。它属于列式（Column-Oriented）NoSQL数据库，目前仅支持单行事务，**没有成熟的二级索引**方案，主要用来存储非结构化和半结构化的松散数据。

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；

4.数据多版本：每个单元中的数据可以有多个版本，默认情况下版本号自动分配，是单元插入时的时间戳；

**面向行的数据库：是按照行存储，维护大量的索引和物化视图无论是在时间还是空间成本方面都很高（mysql），mysql表很大的时候，添加列、索引等操作都很耗时**

后面会解决Hbase如何面向列族存储

## 二、基本概念

**Row Key**：格式为byte array，是表中每条记录的“主键”，方便快速查找。Table中所有行都按照row key的字典序排列，因此row key的设计非常重要。

**Column Family**：列族，拥有一个名称(string)，包含一个或者多个相关列。每个column family存储在HDFS上的一个单独文件中。

**Column**：属于某一个column family，列名定义为family:qualifier，其中qualifier可以是任意的字符串。每个column family可以有上百万个columns。

**Version Number**：类型为Long，默认值是系统时间戳，可由用户自定义。

**Value(Cell)**：都是Byte array，没有类型。每个cell中的数据可以有多个版本，默认是3个。每个值对应的索引为<key, column family, qualifier, timestamp>

## 三、数据模型

**1）逻辑视图**

下面是根据BigTable 论文稍加修改的例子。 有一个名为webtable的表，包含两个列族：contents和anchor。在这个例子里面，anchor有两个列anchor:cssnsi.com和anchor:my.look.ca，contents仅有一个列contents:html。

表 webtable

|  |  |  |  |
| --- | --- | --- | --- |
| **Row Key** | **Time Stamp** | **ColumnFamily contents** | **ColumnFamily anchor** |
| "com.cnn.www" | t9 |  | anchor:cnnsi.com = "CNN" |
| "com.cnn.www" | t8 |  | anchor:my.look.ca = "CNN.com" |
| "com.cnn.www" | t6 | contents:html = "<html>..." |  |
| "com.cnn.www" | t5 | contents:html = "<html>..." |  |
| "com.cnn.www" | t3 | contents:html = "<html>..." |  |

**2）物理视图**

在逻辑视图里，表被看成是一个稀疏的行的集合。但在物理上，它是按**列族**存储的。

ColumnFamily anchor

|  |  |  |
| --- | --- | --- |
| **Row Key** | **Time Stamp** | **Column Family anchor** |
| "com.cnn.www" | t9 | anchor:cnnsi.com = "CNN" |
| "com.cnn.www" | t8 | anchor:my.look.ca = "CNN.com" |

ColumnFamily contents

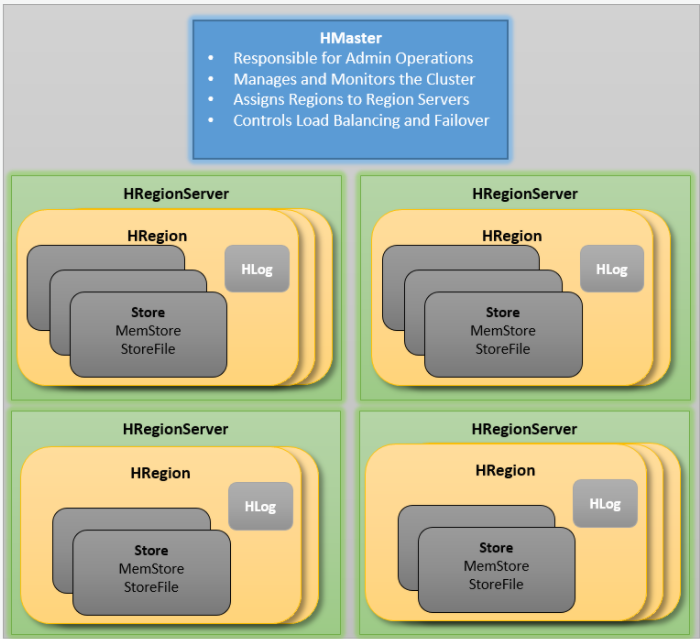
|  |  |  |
| --- | --- | --- |
| **Row Key** | **Time Stamp** | **ColumnFamily contents** |
| "com.cnn.www" | t6 | contents:html = "<html>..." |
| "com.cnn.www" | t5 | contents:html = "<html>..." |
| "com.cnn.www" | t3 | contents:html = "<html>..." |

值得注意的是，在上面的逻辑视图中空cell在物理上是不存储的。因此若一个请求为要获取t8时间的contents:html，返回的结果就是空。但是如果不指明时间，将会返回最新时间的行。例如，请求为获取row key为"com.cnn.www"，没有指明时间戳的话，返回的结果是t6的contents:html、t9的anchor:cnnsi.com和t8的anchor:my.look.ca。

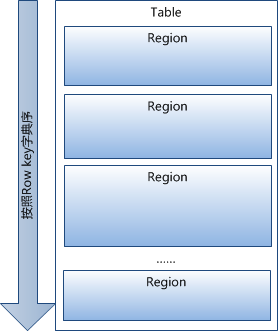
## 四、物理模型

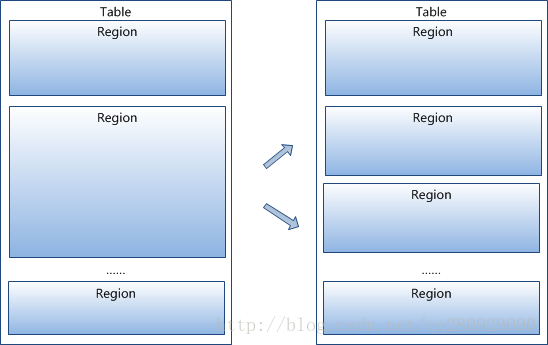
Region：Table按row key区间分割为多个region，即每个region包含一定范围的row key。随着数据量的增加，region会进行自动的split。

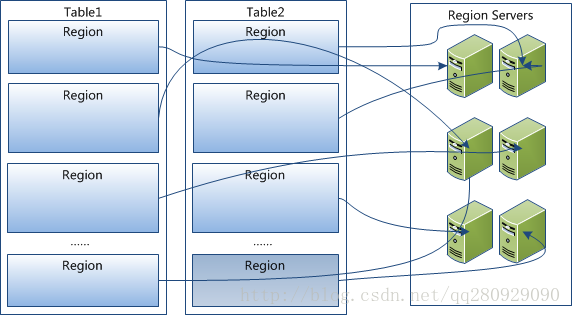
Region Server：每个region由region server来管理。一个region server可以管理一到多个regions。



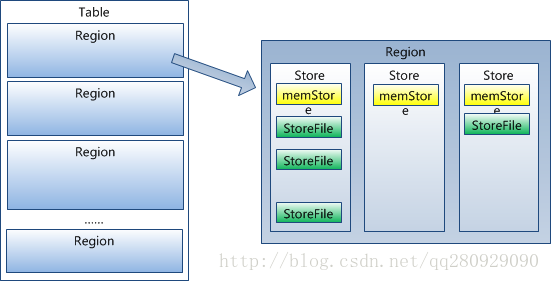
1. HRegion :HBase中表在行的方向上分割为多个HRegion。



1. HRegion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，HRegion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。
2. 
3. 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Strore又由一个memStore和0至多个StoreFile组成。



## ****五、Hbase工作过程（不讲）****

### 5.1、Zookeeper：

Zookeeper Quorum存储-ROOT-表地址、HMaster地址

HRegionServer把自己以Ephedral方式注册到Zookeeper中，HMaster随时感知各个HRegionServer的健康状况，Zookeeper能避免HMaster单点问题

### 5.2、HMaster：

HMaster没有单点问题，HBase中可以启动多个HMaster，通过Zookeeper的Master Election机制保证总有一个Master在运行

主要负责Table和Region的管理工作：

1 管理用户对表的增删改查操作

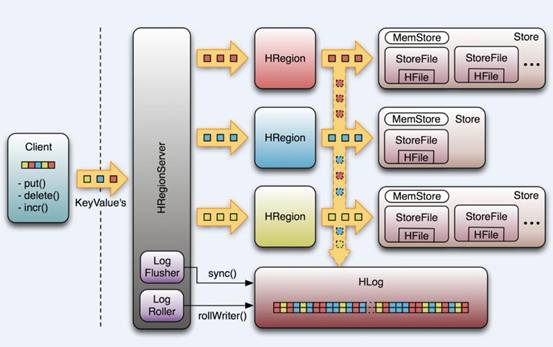
2 管理HRegionServer的负载均衡，调整Region分布

3 Region Split后，负责新Region的分布

4 在HRegionServer停机后，负责失效HRegionServer上Region迁移

### ****5.3、HRegionServer：****

HBase中最核心的模块，主要负责响应用户I/O请求，向HDFS文件系统中读写数据



**HRegionServer管理一些列HRegion对象；**

**每个HRegion对应Table中一个Region，HRegion由多个HStore组成；**

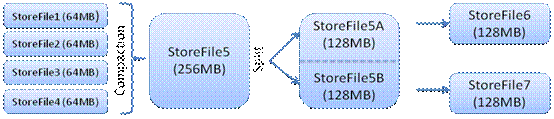
**每个HStore对应Table中一个Column Family的存储；**

**Column Family就是一个集中的存储单元，故将具有相同IO特性的Column放在一个Column Family会更高效**

### ****5.4、HStore：****

HBase存储的核心。由MemStore和StoreFile组成。

MemStore是Sorted Memory Buffer。用户写入数据的流程：



Client写入 -> 存入MemStore，一直到MemStore满 -> Flush成一个StoreFile，直至增长到一定阈值 -> 出发Compact合并操作 -> 多个StoreFile合并成一个StoreFile，同时进行版本合并和数据删除 -> 当StoreFiles Compact后，逐步形成越来越大的StoreFile -> 单个StoreFile大小超过一定阈值后，触发Split操作，把当前Region Split成2个Region，Region会下线，新Split出的2个孩子Region会被HMaster分配到相应的HRegionServer上，使得原先1个Region的压力得以分流到2个Region上

由此过程可知，HBase只是增加数据，有所得更新和删除操作，都是在Compact阶段做的，所以，用户写操作只需要进入到内存即可立即返回，从而保证I/O高性能。

### **5.5、HLog**

引入HLog原因：

在分布式系统环境中，无法避免系统出错或者宕机，一旦HRegionServer以外退出，MemStore中的内存数据就会丢失，引入HLog就是防止这种情况

工作机制：

每个HRegionServer中都会有一个HLog对象，HLog是一个实现Write Ahead Log的类，每次用户操作写入Memstore的同时，也会写一份数据到HLog文件，HLog文件定期会滚动出新，并删除旧的文件(已持久化到StoreFile中的数据)。当HRegionServer意外终止后，HMaster会通过Zookeeper感知，HMaster首先处理遗留的HLog文件，将不同region的log数据拆分，分别放到相应region目录下，然后再将失效的region重新分配，领取到这些region的HRegionServer在Load Region的过程中，会发现有历史HLog需要处理，因此会Replay HLog中的数据到MemStore中，然后flush到StoreFiles，完成数据恢复。

## ****六、HBase存储格式****

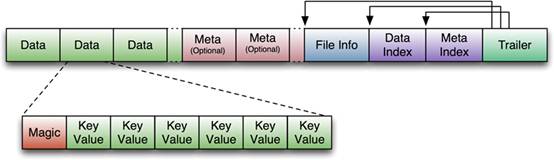
HBase中的所有数据文件都存储在Hadoop HDFS文件系统上，格式主要有两种：

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： HLog File，HBase中WAL（Write Ahead Log） 的存储格式，物理上是Hadoop的Sequence File

### ****Hfile格式****

这对理解hbase的特性很重要



图片解释：

HFile文件不定长，长度固定的块只有两个：Trailer和FileInfo

Trailer中指针指向其他数据块的起始点

File Info中记录了文件的一些Meta信息，例如：AVG\_KEY\_LEN, AVG\_VALUE\_LEN, LAST\_KEY, COMPARATOR, MAX\_SEQ\_ID\_KEY等

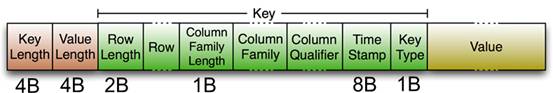
Data Index和Meta Index块记录了每个Data块和Meta块的起始点

Data Block是HBase I/O的基本单元，为了提高效率，HRegionServer中有基于LRU的Block Cache机制

每个Data块的大小可以在创建一个Table的时候通过参数指定，大号的Block有利于顺序Scan，小号Block利于随机查询

每个Data块除了开头的Magic以外就是一个个KeyValue对拼接而成, Magic内容就是一些随机数字，目的是防止数据损坏

HFile里面的每个KeyValue对就是一个简单的byte数组。这个byte数组里面包含了很多项，并且有固定的结构。



KeyLength和ValueLength：两个固定的长度，分别代表Key和Value的长度

Key部分：Row Length是固定长度的数值，表示RowKey的长度，Row 就是RowKey

Column Family Length是固定长度的数值，表示Family的长度

接着就是Column Family，再接着是Qualifier，然后是两个固定长度的数值，表示Time Stamp和Key Type（Put/Delete）

Value部分没有这么复杂的结构，就是纯粹的二进制数据

## 七、表设计优化：

### 5.1、列簇的设计

列簇尽量少。因为每个列簇是存在一个独立的HFile里的，flush和compaction操作都是针对一个Region进行的，当一个列簇的数据很多需要flush的时候，其它列簇即使数据很少也需要flush，这样就产生的大量不必要的io操作。

在多列簇的情况下，注意各列簇数据的数量级要一致。如果两个列簇的数量级相差太大，会使数量级少的列簇的数据扫描效率低下。

将经常查询和不经常查询的数据放到不同的列簇。

因为列簇和列的名字会存在HBase的每个Cell中，所以他们的名字应该尽可能的短。比如，用f:q代替mycolumnfamily:mycolumnqualifier

### 5.2、rowkey的设计

避免使用递增的数字或时间做为rowkey。

如果rowkey是整型，用二进制的方式比用string来存储更节约空间

合理的控制rowkey的长度，尽可能短,因为rowkey的数据也会存在每个Cell中。

如果需要将表预分裂为多个region是，最好自定义分裂的规则。

## 八、和其他数据库对比：

**Hbase VS MQ**：由于列族可以设计为IN\_MEMORY模式，这样，列族数据在内存中，扫描和处理非常快速，可使用这种方式作为消息通知，Hbase可以在很大成度上代替mq

**Hbase VS redis**：都是KV存储，hbase速度慢，但数据持久性好，而且能支持巨量KV数据

**Hbase VS mysql :** Hbase无二级缓存，故不像mysql，可以对多列设置索引，加快查询速度

Hbase的特点概括：面向列、CP非CA（会有一些阶段不可用）、KV存储，做技术选型一般错不了

## 九、Best Practices

### 9.1存储

1. hbase默认不使用压缩进行存储，一般情况下，hbase是以大表的方式存在，如果不进行压缩的话，势必会造成空间的浪费。而且由于hbase是对随机访问进行优化的，所以需要采用压缩解压效率较高的算法。压缩比大的算法但速度慢的算法，比如gzip，不太适合。推荐使用LZO和SNAPY压缩，以损失部分空间换取较高的响应速度。
2. hbase块大小指定。默认值为64M，这个值和hadoop的块大小不太一样，hadoop的块大小是在文件存储时生效，hbase的块大小是指memstore一次性写入filestore的大小或者由filestore一次性读入memstore的大小。对于数据读写比较频繁的表，需要适当调大hbase blocksize，可以减少大量的磁盘IO操作，同时提高读写效率。<-此项没有经过实际的测试，是否正确需要探讨
3. bloomfilter：指定布隆过滤器理论上可以提高读写速度，但会损失一定的存储空间用来保存索引。

### 9.2 参数配置

1. scan.setCaching接受的配置是**行数，不是Byte**，建议根据数据大小进行估算，总额是10MB；若一行是10KB，cache设置为1000；若一行是2MB，cache设置为5。
2. htable.setBufferSize直接接受Byte，建议2MB~5MB，**一般不允许超过10MB。**若设置为5MB，则是htable.setBufferSize(5\*1024\*1024)。
3. 新建HBase表的时候，最好能够预先划分好Region，将负载分散到集群的多台机器中，再写入数据。否则开始写入数据时压力集中在一台机器上，频繁地Split，容易引发超时等错误。
4. hbase-site.xml中配置项调整。由于Client配置会覆盖Server端配置，请检查自己的配置项是否为最新的值。

|  |
| --- |
| <property>    <name>hbase.client.pause</name>    <value>100</value>  </property>    <property>    <name>hbase.client.retries.number</name>    <value>31</value>  </property> |

1. 若一个Value可能会很大，配置hbase.client.keyvalue.maxsize

集群默认配置是20971520，即20MB。若估计表的一个Value可能会超过这一大小，请在客户端调整配置文件hbase-site.xml手动修改该配置项：

<property>

  <name>hbase.client.keyvalue.maxsize</name>

  <value>20971520</value>

</property>

否则可能会遇到如下异常：java.lang.IllegalArgumentException: KeyValue size too large

# QA

**1）正确设置scan.setCaching()，htable.setBufferSize()**

重要：HBase很多Bug是由于这两个数值没有被正确设置导致的，请千万要注意！

scan.setCaching接受的配置是**行数，不是Byte**，建议根据数据大小进行估算，总额是10MB；

若一行是10KB，cache设置为1000；若一行是2MB，cache设置为5。

htable.setBufferSize直接接受Byte，建议2MB~5MB，**一般不允许超过10MB。**

若设置为5MB，则是htable.setBufferSize(5\*1024\*1024)

**2）HBase Job运行速度慢（99%停滞，卡住，超时）**

原因1：数据不均匀。有一些Region过大，导致Mapper的输入很大。

解决方案：手动对该Region执行Split；

原因2：Scan时没有设置Cache

解决方案：Scan cache接收的配置是行数，建议值需要根据数据大小估算，总额是10M左右（危险）

       如果一行是10K， scan cache可以设置为1000；如果一行是2M， cache设为5

解决方案：参照1）正确设置scan.setCaching()。

**3）Scan触发OutOfMemoryError导致任务失败**

原因：scan.setCaching()参数过大

解决方案：参照1）正确设置scan.setCaching()。

**4）Scan过程中丢失数据**

原因：HBase后端有一处安全配置hbase.client.scanner.max.result.size，是为了防止scan cache设置过大导致的集群OOM。这个max.result.size 是10M。

          当客户端的scan cache 设置过大，scanner会在扫描过程中被默默地停止，因此会造成少数据。

**警告：该丢失数据的过程不会触发任何警报**

解决方案：参照1）正确设置scan.setCaching()。

**5）执行大批量数据写入时遇到：Call to hadoop-nodexxxxxxxx failed on socket timeout exception: java.net.SocketTimeoutException: 60000 millis timeout**

原因：a. 数据量太大，b. RPC Timeout值设置的较小

解决方案：减少批量操作Buffer的大小，或者增加RPC Timeout的时间

参考配置：

hbase.client.write.buffer = 1024\*1024\*1；//1MB

hbase.rpc.timeout = 600000 //10分钟

**6）HBase Put操作后FlushTable失败**

原因：setWriteBufferSize设置过大，导致大量put在同一时间写入table，致使table不断触发compact和split。

解决方案：将writeBuffer设置在10MB以下，建议2MB~5MB

**7）写表时遇到 Call queue is full, is ipc.server.max.callqueue.size too small?**

原因：RegionServer最大handle的RPC请求数量是100，writeBuffer如果设置为10m, 集群的callQueue size是1G，刚好会超出。

解决方案：将writeBuffer设置在10MB以下，建议2MB~5MB

**8）java.net.UnknownHostException 或者**

**ERROR hbase.HServerAddress: Could not resolve the DNS name of xxxxx**

原因：目前Hadoop集群未使用DNS服务，每次添加 / 修改机器，都需要在集群所有的机器（包括客户端机器）手动添加HOST/IP 映射

解决方案：手动添加相应结点的HOST/IP映射

**9）zookeeper连接时连接本地：zookeeper.ClientCnxn: Opening socket connection to server localhost.localdomain/127.0.0.1:2181.**

**Will not attempt to authenticate using SASL (Unable to locate a login configuration)**

原因1：用户运行环境classpath变更；

解决方案：检查classpath配置

原因2：用户运行提交job的操作是封装在一个shell脚本中的，而export classpath的操作是在脚本外部执行的

解决方案：把export HADOOP\_CLASSPATH的操作放到程序环境内部（程序执行脚本中，或者程序执行脚本能调用到的配置文件中）

**10）java.net.SocketTimeoutException 或者 RetriesExhaustedWithDetailsException**

原因：初始创建的表只有一个region，在插入数据的过程中不断的被split，split的过程中region会频繁下线，导致写入失败。

解决方案：在创建表之后，写入数据之前，先估测表的大小，预先分好region，再执行插入数据的代码。

**11）HBase创建表时设置COMPRESSION => 'LZO'导致失败**

原因：由于LICENSE的原因，LZO未与HBase一起发布

解决方案：不使用压缩或使用其他压缩方式

**12）HBase访问卡死，也不抛出任何Exception**

原因：hbase访问会多次重试，卡死是因为它在不停地重试

 解决方案：修改重试次数，快速结束重试，发现错误的原因。具体设置方法

 a. 代码中设置conf.set("hbase.client.retries.number", "1");

b. 修改配置文件/etc/hbase/conf/hbase-site.xml中的hbase.client.retries.number属性 //不建议，可能会影响入口机其他任务的执行。