# HBASE学习

1. 是什么

HBASE是一个hadoop应用，是专为分布式数据准备的数据库

1. 为什么

需要满足大量数据对RDBMS性能的要求（但是HBASE并不是关系数据库）

1. 怎么办

# 概念学习

HBase是一个分布式的、持久的、强一致性的存储系统，具有近似最优的写性能（能使I/O利用率达到饱和）和出色的读性能，它充分利用了磁盘空间，支持特定列族切换可选压缩算法。

1. 简要的总结：最基本的单位是列，一列或多列形成一行，并由唯一的行键来确定存储。反过来，一个表中有若干行，其中每列可能有多个版本，在每个单元格中存储了不同的值。（行序按照行键字典序进行排序）
2. HBASE是以列簇为组织形式的，只要预先给出了一个列簇，就可以动态的增加该列簇的列；
3. HBASE的实现：正如hadoop的分布式方法一样，HBASE也有自己的主从机结构以及其管理方式；其中，主机叫作master，而从机叫作regionserver
4. 目标：将HBASE的可伸缩性变得简单，以不同的方法访问HBASE，了解HBASE架构的细节，包括存储格式、预写日志、后台进程等；在HBase中集成MapReduce框架；了解如何调节集群、设计模式、拷贝表、导入批量数据和删除节点等。并且在读源码的同时掌握JAVA编码技巧。
5. 协处理器：在服务端地址空间执行来自客户端的代码。

## 结构

1、自动分区，region本质上是以行键排序的连续存储区间，太大了系统会自动拆分，太小的时候则会自己合并。一个region服务器可以加载多个region，但是一个region只能被一个region服务器加载

2、更新数据流程，每次更新时，都会先将数据记录在提交日志中，然后再写入内存的memstore中，内存保存的数据超过一定值的时候会刷写到磁盘，这时会用新的memstore获取更新数据而将满的memstore刷写入磁盘。

3、删除数据是先将数据标记为删除的，此时将对用户不可见，在执行一次major合并的时候会忽略删除的数据，从而达到更新的目的。（major合并是带压缩的，而minor合并不）

4、Hbase中有三个主要组件：客户端库、一台主服务器、多台region服务器。可以动态的增加或移除region服务器，以适应不断变化的负载要求。主服务器利用ZooKeeper管理

# 深入代码

## 一、客户API基础知识

Hbase保持了行的原子性，因此，多个客户端同时对行进行操作可能会出现锁的问题，推荐的方法是采用批处理的办法。

只创建一个HTable的实例，或者使用HTablePool类，因为创建有代价。

同时，应该为每个线程创建一个实例，并且一般在程序开始时创建。

### 1、CRUD（create、read、update、delete操作）

行键必须指定，

它最好代表了实际的意义，如订单号，用户等；

它是一个btye[]；

它的内容可以是简单的数字，也可以是一个UUID（Universally Unique Identifer）

提供了一些转化方法供别的类型转化为btye[];

命令行：

start-hbase.sh

create 'testtable','colfam1'

list

list 'testtable'

put 'testtable','myrow-1','colfam1:q1','value-1'

scan 'testtable'

delete 'testtable','myrow-1','colfam1:q1'

disable 'testtable'

drop 'testtable'

stop-hbase.sh

观察一个table：

Describe ‘testtable’

修改一个table:

Alter ‘testtable’,{NAME=>’colfam1’,VERSIONS=>4}

Put：

Put操作是对一行进行的操作，可以单独设置WAL是否开启；

Get：

Get仍是针对一行的操作，产生的Get实例传入table后得到result，

Result可以有几种取值方法，来针对需求获得相应的对象。

Delete:

删除是分删除最新版本或所有版本，默认是删除最新的版本。

基本的行增加工作：

Configuration conf = HBaseConfiguration.create();

HTable table = new HTable (conf,”testtable”);

Put put = new put(Bytes.toBytes(“row1”));

put.add(Bytes.toBytes(“colfam1”),Bytes.toBytes(“qual1”),Bytes.toBytes(“val1”));

缓冲区的概念：

1、用户的缓冲机制：put实例在一开始的时候被保留在内存中，当发起commit命令的时候才通过一次RPC批量的发送到服务器

2、设置写缓冲机制：

table.setAutoFlush(false);

3、强制把数据写到服务端：

void flushCommits() throws IOException

4、配置缓冲区大小

long getWriteBufferSize();

void setWriteBufferSize(long writeBufferSize) throws IOException

5、为了避免为每个HTable实例都设定缓冲区的大小，在配置文件中添加一个较大的预设值：

<property>

<name>hbase.client.write.buffer</name>

<value>20971520</value>

</property>

这是把缓冲区设置为20M的配置

6、在强制调用刷缓冲区方法、缓冲区满、设置缓冲区大小且目前缓冲区出限制以及HTable类被close的时候都会刷写缓冲区。

列表的使用：

使用put列表可以批量的刷写数据，使用方法就是先定义一个列表，然后在调用列表的add方法向其中增加元素，在直接将列表作为put的参数：

List<Put> puts = new ArrayList<Put>();

Put put1 = new Put...

puts.add(put1);

Put put2 = …

puts.add(put2);

table.put(puts);

原子操作：

table.checkAndPut可以保证写操作的原子性；

Put put1 = …

put1.add...

boolean res1 = table.checkAndPut(Bytes.toBytes(“row1”),

Bytes.toBytes(“colfam1”),Bytes.toBytes(“qual1”),null,put1);

检查列族，列，行是否存在，且值为空，如果都满足，则put

get的用法：

利用get从HBase 中获取数据：

Configuration conf = HbaseConfiguration.create();

HTable table = new HTable (conf, “testtable”);

Get get = new Get(Bytes.toBytes(“row1”));

get.addColumn(Bytes.toBytes(“colfam1”),Bytes.toBytes(“qual1”));

Result result = table.get(get);

byte[] val = result.getValue(Bytes.toBytes(“colfam1”),

Bytes.toBytes(“qual1”));

System.out.println(“Value: “ + Bytes.toString(val));

result类的用法：

用户使用get()方法获取的数据存储在一个result实例中

getValue、value、getRow...等等方法以各种方式返回了相应的数据

delete的用法，和get、put等相类似

batch的使用，batch的目的就是批量处理，可以把put、get、delete等操作合在一个batch上。

lock的使用

lock用来对行进行锁定

scan方法

可以以规定行等方式扫描表

扫描器缓存：为了一次RPC获取更多行的数据而存在；

有两种存在的方式，一种是存在于表，那么这个表的所有扫描器实例都会生效，

另一种是存在与扫描器，那么这个扫描器实例会被影响。

缓存：缓存决定了行数，每次传送缓存个数的行数；

批量：批量决定传送的列数；

所以RPC次数 = 行数\*列数/min（批量，行数）/缓存再加一；

在HBase中，所谓的一列就是指一行中的一个单元格，而非所有行的那一列，因为每个单元格还需要保存很多个版本，所以一列仍然有很多数据。

## 高级API:

### 过滤器：

有多种过滤器，都继承了Filter接口或FilterBase抽象类

和过滤器一起使用的还有比较器：CompareFilter，比较只能正面指定需要过滤的值，而不是取反。

匹配、正则、逻辑相关的比较器由于只返回真假，所以只能设置为EQUE与否，而不能设置大小比较。

以行过滤器为例；

Filter filter1 = new RowFilter(CompareFilter.CompareOp.LESS\_OR\_EQUAL,

new BinaryComparator(Bytes.toBytes(“row-22”)));

scan.setFilter(filter1);

Resultscanner scanner1 = table.getScanner(scan);

for (Result res : scanner1) {

System.out.println(res);

}

scanner1.close();

完成的就是过滤查找的工作，可以根据示例，进行使用。

含有各种功能，各种形态的过滤器，同时也可以对过滤器进行组合，来达到期望的作用。

使用自定义过滤器的方法如下：

书写自定义过滤器；

将hbase-env.sh中的HBASE\_CLASSPATH= 设置为编译出的jar的地址

开始使用。

### 计数器

如果使用传统的修改列的方法进行计数处理，涉及到锁的持有释放，大量的操作会引发崩溃。因此实现了计数器的功能，可以原子的进行对列的加减。

计数器可以一次更新多个，但必须是同一行内的列，否则需要多个实例，也就仍会出现多次RPC调用。

计数器可以不初始化，默认为零，也可以初始化为一个数，但是要编解码：

解码：Bytes.tolong();

编码：Bytes.toBytes(long);

使用计数器的时候，可以使用table的静态方法进行单计数器的处理，可以创建一个increment实例，行进多行的计数。

### 协处理器

使用过滤功能已经可以使传回客户端的数据大大减少了，如果使用协处理器，让一些对数据的处理的代码也在服务器完成，则可以进一步减少远程数据传输，从而节省带宽，增加效率。

协处理器是客户可以把一部分计算放到数据存放端。

协处理器允许用户在region服务器上运行自己的代码，通过一些触发形式如put时触发，get时触发等，以达到目的。可以应用的场景诸如RDBMS的average、sum以及sql等相关聚合工作。

协处理器应当只与被提供的环境交互，这样可以防止数据被破坏，协处理器应当通过getTable获取表对象，因为这个方法中包含了一些安全措施。

有三个类：

1、coprocessor

2、coprocessorEnviroment

3、CoprocessorHost 维护了所有的协处理器实例和环境。

协处理器的加载：

1、通过配置文件加载，这种加载会影响所有的region、master、wal（只加载对应的协处理器）见P198

2、通过表描述符加载，这种加载只针对特定的表，所以只影响该表所在的region，必须使有HTableDescriptor.setValue()方法定义它们，键以COPROCESSOR开头，值符合如下格式：

<path-to-jar>|<classname>|<priority>

RegionObserver类

一个特定的region级别操作发生时，他的钩子函数会触发

钩子函数大部分的形态是pre…（）和post…()代表一个事件刚开始之前和刚结束之前该干些什么，例如，可以在region的open前调用preOpen以解除这一次的open操作。

RegionCoprocessorEnviroment类

实现RegionObserver类的协处理器环境的实例是基于RegionCoprocessorEnvironment类的。

ObserverContext类

RegionObserver类提供的所有回调函数都需要一个特殊的上下文作为共同的参数。

MasterCoprocessorEnvironment也有相同的用法

HTablePool：为了不使每次RPC请求都创建一个实例，且避免此实例复用的时候线程不安全的问题，而产生的。

使用过程：

创建表实例池；

创建实例；

推入实例池中；

从实例池中取用；

使用完后放回池中；

释放实例；

销毁表实例池；

### 连接管理

每个HTable实例都需要建立和远程主机的连接。这些连接在内部使用HConnection类表示，其被HConnectionManager类管理，客户端API可以通过创建Configuration实例来使用这些类，而不必直接和它打交道。

在HTable关闭的时候会自动关闭连接；

也可以显示的打开显示的关闭，这样一来连接就是唯一的。

## 管理功能

模式定义：

### 表

表的构造函数：HTableDescriptor();

由于Hadoop RPC框架的原因，所以远程方法中的参数都实现Writable接口，这样才能序列化对象并进行远程传输。Writable接口有两个必需实现的方法：

Void write()

Void readFields()

框架通过调用这两个方法把对象序列化成输出流；

如果用户开发并扩展了HBase的基础实现还应满足一些条件，P224

利用getter和setter方法可以对表进行设置；

### 列族

HColumnDescriptor();为列族类，列类名必须是可见字符；

列族最好包括具名的列而不是无名列，因为无名列在之后不能简单重命名；

生存期TTL：

Hbase不仅支持保存的版本数，也支持设置版本保存的时间。

在内存中标志：

这个标志承诺尽量把列族加载到内存中以加快访问速度；

布隆过滤器：

布隆过滤器可以加速特定模式下的访问速度，但是会增加内存和存储的负担，默认情况下是关闭的。

复制范围：

提供了跨集群复制的能力；

### 集群管理

Static void checkHbaseAvailable (Configuration conf)

ClusterStatus getClusterStatus()

验证客户端是否可以与给定文件配置中的远程HBase集群进行通信；

返回集群信息，包含集群的详细信息；

Void closeRegion ()

让在region服务器中上线的特定region下线

Flush()

刷写未写入region的memstore内容

Void compact() / void majorcompact()

合并操作

Void split()

分离region或表的操作

Void assign() / void unassign()

控制region的上下线操作

Void move()

控制某个region在哪台服务器上线

Boolean balanceSwitch() / Boolean balancer()

控制负载均衡算法是否开启以及运行负载均衡再分配；

Void shutdown() / void stopMaster() / void stopRegionServer()

关闭集群，关闭Master进程，关闭某台region服务；

## 可用客户端

REST、Thrift和avro

这些客户端允许多编程环境去操作数据库。

## 框架

HBase的主要特点之一就是紧密地与hadoop和Mapreduce框架集成；

### 细节如下：

P302 mapreduce过程图解；

1、首先要接触到的类是inputformat,做两件事：

一、对数据进行分割；

二、返回一个recordreader实例，这个实例定义了键值对象的类，并提供了next方法来遍历输入的数据。

这里HBase提供了专有的实现，叫TableInputFormatBase；是一个抽象类；实现了大部分功能。

2、其次是map类；

在下一个阶段，从recordreader中读到的第条信息都要由map进行处理；

HBase提供了一个TableMapper类，将键的类型强转为ImmutablebytesWritable，同时将值的类型转化为Result类型。

3、再次是reduce类

在这个阶段，map输出的键值对经过shuffle和sort后，传给reduce进行处理；

4、最后由OutputFormat类

这一阶段将结果数据输出到持久化储存；

### 支撑类的细节：

1、MapReduce的支撑类与TableMapReduceUtil()协作，使作业运行在HBase上，它有静态方法可以配置作业，并使作业能够以HBase作为源或目标；

### 使用Mapreduce和HBase

库依赖：要么在每个节点静态地配置所需库，要么直接提供；

1、静态方法：

复制库

将库目录写入hadoop-env.sh

重启tracker

2、动态方法：

可以在作业运行的时候更新库；

做法就是把工作jar以及依赖的jar一起打包到一个jar文件中

可以以两种方法做到：1、是maven 2、是hadoop自己的libjars工具

3、实例

P310，从文件中读取数据，并写到HBase表中的方法。

## 架构

### 数据的查找和传输：

P329：块的概览

查询流程如下：

1. 先通过zookeeper查找含有-ROOT-的region服务器，通过这个服务器查询到含有.META.表中对应的region服务器名，其中包含请求的行键信息；
2. 一旦查询到数据实际位置，即region，HBase就会缓存这次查询，并联系HRegionServer，之后的查询可以直接从缓存取址，而非再走一次流程；

Region和HRegionServer

HRegionServer负责打开region，并创建对应的HRegion实例，HRegion打开后会为每个表的HColumnFamily创建一个store实例每个store包含一个或多个StoreFile实例，它是实际数据存储文件HFile的简单封装；每个Store对应一个MenStore一个HRegionServer分享一个HLog实例；

### HLOG

未持久化的数据存在于WAL（write-ahead-log）上，在服务器崩溃的时候可以回滚到崩溃前的状态；

WAL是顺序存储的，在一个regionserver上的所有region的log被写在一起，当须要回滚的时候，必须对log进行拆分，然后把拆分的文件分配到各个对应的region，则可以进行恢复工作。

### Region

概念：

Region本质上是以键排序的连续存储的空间；

Bigtable论文指出，每个服务器的Region的数量推荐为10-1000，大小则因硬盘而异，如2T的硬盘，1000个Region，则每个Region大小为2G

Region的拆分：当Region的大小查过预设值的时候会发生拆分，拆分大小为原来的一半；

拆分Region的时候.META.表对过程进行记录；最后生成新的Region的信息；

拆分后的Region还是原来的存储，直到合并把存储异步的写成独立的文件。

Region拆分时会对新的Region重新初始化，这个时候正是.META.表更新的时候，然后父Region会把数据复制到新Region的.tmp文件夹，复制完成后.tmp会替代Region的数据文件，最终父Region会被清理掉，而新Region会在初始化的时候执行一次合并；

Region的查找

通过两张特殊的表-ROOT-和.META.

1、Zookeeper中查找root region的信息，

2、root Region不会被拆分，从中查找meta region的信息

3、.META.表中查找对应的region位置

### Storefile

数据存储在存储文件store file中，称为HFile，HFile存储的是经过排序的键值映射结构。

文件由连续的块组成，块的索引位于文件尾部，当一个storefile载入到内存中时，索引被优先加载，每个块的大小默认是64KB，可以通过配置改变。

### Memstore

通过Memstore的滚动，可以使得memstore往文件中写的过程不需要阻塞系统的读写。

### 合并

Major合并一个Region中的列族中若干个HFile，重写为一个HFile

# 设置

1、ubuntu上设置文件句柄：P71

# 要点总结

1. 以前因为技术的原因无法分析每个数据点，而现在，不分析所有数据点的公司将失去竞争力。
2. 大数据为什么大：数十亿行\*数百万列\*数千个版本=TB级或PB级的存储。

## 小知识

hbase目前只能依赖特定版本hadoop，主要原因之一是hbase和hadoop的远程调用过程Remote Procedure Call是版本协议化的

需要注意的hbase运行系统配置:

1、需要同步时间

2、增大文件描述符限制

问题点：

1、既然不可用不存在的列族，那么如何创建新的列族？

答：只能在创建table的时候规定列族

相关依赖：

org.apache.hadoop.util.PlatformName hadoop-auth-\*.jar(hadoop)

zookeeper zookeeper-\*.jar (hbase)

htrace htrace-core\*.jar (hbase)

io.netty.channel.EventLoopGroup netty-all-\*.jar (hbase)

# 资料总结

1、论文：BigTable：A Distributed Storage System for Structured Data