Apache HBase™ 参考指南

HBase 官方文档中文版

Revision History	
Revision 0.97.0-SNAPSHOT	2013-04-07T14:59
中文版翻译整理 周海汉	

译者：HBase新版 0.97 文档和0.90版相比，变化较大，文档补充更新了很多内容，章节调整较大。本翻译文档的部分工作基于颜开工作。英文原文地址在此处。旧版0.90版由颜开翻译文档在此处。0.97版翻译最后更新请到此处(http://abloz.com/hbase/book.html)浏览。反馈和参与请到此处 (https://github.com/ablozhou/hbasedoc_cn/)或访问我的blog(http://abloz.com)，或给我发email。

可以通过浏览器直接存储为pdf或本地文件。

贡献者：

周海汉邮箱：ablozhou@gmail.com, QQ:7268188 网址：http://abloz.com/
颜开邮箱: yankaycom@gmail.com, 网址：http://www.yankay.com/

摘要

这是 Apache HBase (TM)的官方文档。 HBase是一个分布式，版本化，面向列的数据库，构建在 Apache Hadoop和 Apache ZooKeeper之上。

目录

表索引

序

这本书是 HBase 的官方指南。 版本为 *0.95-SNAPSHOT* 。可以在HBase官网上找到它。也可以在 javadoc, JIRA 和 wiki 找到更多的资料。

此书正在编辑中。 可以向 HBase 官方提供补丁JIRA.

这个版本系译者水平限制，没有理解清楚或不需要翻译的地方保留英文原文。

最前面的话

若这是你第一次踏入分布式计算的精彩世界，你会感到这是一个有趣的年代。分布式计算是很难的，做一个分布式系统需要很多软硬件和网络的技能。你的集群可以会因为各式各样的错误发生故障。比如HBase本身的Bug,错误的配置(包括操作系统)，硬件的故障(网卡和磁盘甚至内存) 如果你一直在写单机程序的话，你需要重新开始学习。这里就是一个好的起点: 分布式计算的谬论.

Chapter 1. 入门

Table of Contents

1.1. 介绍

Section 1.2, “快速开始”会介绍如何运行一个单机版的HBase.他运行在本地磁盘上。 Section 2, “配置” 会介绍如何运行一个分布式的HBase。他运行在HDFS上

1.2. 快速开始

本指南介绍了在单机安装HBase的方法。会引导你通过**shell**创建一个表，插入一行，然后删除它，最后停止HBase。只要10分钟就可以完成以下的操作。

1.2.1. 下载解压最新版本

选择一个 Apache 下载镜像，下载 *HBase Releases*. 点击 `stable`目录，然后下载后缀为 `.tar.gz` 的文件; 例如 `hbase-0.95-SNAPSHOT.tar.gz`.

解压缩，然后进入到那个要解压的目录.

```
$ tar xfz hbase-0.95-SNAPSHOT.tar.gz
$ cd hbase-0.95-SNAPSHOT
```

现在你已经可以启动HBase了。但是你可能需要先编辑 `conf/hbase-site.xml` 去配置`hbase.rootdir`，来选择HBase将数据写到哪个目录 .

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>hbase.rootdir</name>
    <value>file:///DIRECTORY/hbase</value>
  </property>
</configuration>
```

将 `DIRECTORY` 替换成你期望写文件的目录. 默认 `hbase.rootdir` 是指向 `/tmp/hbase-${user.name}` ，也就说你会在重启后丢失数据(重启的时候操作系统会清理`/tmp`目录)

1.2.2. 启动 HBase

现在启动HBase:

```
$ ./bin/start-hbase.sh
starting Master, logging to logs/hbase-user-master-example.org.out
```

现在你运行的是单机模式的Hbaes。所有的服务都运行在一个JVM上，包括HBase和Zookeeper。HBase的日志放在`logs`目录,当你启动出问题的时候，可以检查这个日志。

是否安装了 java ?

你需要确认安装了Oracle的1.6 版本的java.如果你在命令行键入java有反应说明你安装了Java。如果没有装，你需要先安装，然后编辑`conf/hbase-env.sh`，将其中的`JAVA_HOME`指向到你Java的安装目录。

1.2.3. Shell 练习

用**shell**连接你的HBase

```
$ ./bin/hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version: 0.90.0, r1001068, Fri Sep 24 13:55:42 PDT 2010

hbase(main):001:0>
```

输入 **help** 然后 **<RETURN>** 可以看到一列shell命令。这里的帮助很详细，要注意的是表名，行和列需要加引号。

创建一个名为 `test` 的表，这个表只有一个 列族 为 `cf`。可以列出所有的表来检查创建情况，然后插入些值。

```
hbase(main):003:0> create 'test', 'cf'
0 row(s) in 1.2200 seconds
hbase(main):003:0> list 'table'
test
1 row(s) in 0.0550 seconds
hbase(main):004:0> put 'test', 'row1', 'cf:a', 'value1'
0 row(s) in 0.0560 seconds
hbase(main):005:0> put 'test', 'row2', 'cf:b', 'value2'
0 row(s) in 0.0370 seconds
hbase(main):006:0> put 'test', 'row3', 'cf:c', 'value3'
0 row(s) in 0.0450 seconds
```

以上我们分别插入了3行。第一个行key为`row1`, 列为 `cf:a`， 值是 `value1`。HBase中的列是由 列族前缀和列的名字组成的，以冒号间隔。例如这一行的列名就是`a`.

检查插入情况.

Scan这个表，操作如下

```
hbase(main):007:0> scan 'test'
ROW        COLUMN+CELL
row1       column=cf:a, timestamp=1288380727188, value=value1
row2       column=cf:b, timestamp=1288380738440, value=value2
row3       column=cf:c, timestamp=1288380747365, value=value3
3 row(s) in 0.0590 seconds
```

Get一行，操作如下

```
hbase(main):008:0> get 'test', 'row1'
COLUMN      CELL
cf:a        timestamp=1288380727188, value=value1
1 row(s) in 0.0400 seconds
```

disable 再 drop 这张表，可以清除你刚刚的操作

```
hbase(main):012:0> disable 'test'
0 row(s) in 1.0930 seconds
hbase(main):013:0> drop 'test'
0 row(s) in 0.0770 seconds
```

关闭shell

```
hbase(main):014:0> exit
```

1.2.4. 停止 HBase

运行停止脚本来停止HBase.

```
$ ./bin/stop-hbase.sh
stopping hbase...............
```

1.2.5. 下一步该做什么

以上步骤仅仅适用于实验和测试。接下来你可以看 Section 2., “配置” ，我们会介绍不同的HBase运行模式，运行分布式HBase中需要的软件 和如何配置。

2. 配置

本章是慢速开始配置指导。

HBase有如下需要，请仔细阅读本章节以确保所有的需要都被满足。如果需求没有能满足，就有可能遇到莫名其妙的错误甚至丢失数据。

HBase使用和Hadoop一样配置系统。要配置部署，编辑conf/hbase-env.sh文件中的环境变量——该配置文件主要启动脚本用于获取已启动的集群——然后增加配置到XML文件，如同覆盖HBase缺省配置，告诉HBase用什么文件系统， 全部ZooKeeper位置 [1] 。

在分布模式下运行时，在编辑HBase配置文件之后，确认将conf目录复制到集群中的每个节点。HBase不会自动同步。使用**rsync**.

[1] 小心编辑XML。确认关闭所有元素。采用 **xmllint** 或类似工具确认文档编辑后是良好格式化的。

2.1. 基础条件

This section lists required services and some required system configuration.

2.1.1 java

和Hadoop一样，HBase需要Oracle版本的Java6.除了那个有问题的u18版本其他的都可以用，最好用最新的。

2.1. 操作系统

2.1.2.1. ssh

必须安装**ssh**， **sshd** 也必须运行，这样Hadoop的脚本才可以远程操控其他的Hadoop和HBase进程。ssh之间必须都打通，不用密码都可以登录，详细方法可以Google一下 ("ssh passwordless login").

2.1.2.2. DNS

HBase使用本地 hostname 才获得IP地址. 正反向的DNS都是可以的.

如果你的机器有多个接口，HBase会使用hostname指向的主接口.

如果还不够，你可以设置 `hbase.regionserver.dns.interface` 来指定主接口。当然你的整个集群的配置文件都必须一致，每个主机都使用相同的网络接口

还有一种方法是设置 `hbase.regionserver.dns.nameserver`来指定nameserver，不使用系统带的.

2.1.2.3. Loopback IP

HBase expects the loopback IP address to be 127.0.0.1. Ubuntu and some other distributions, for example, will default to 127.0.1.1 and this will cause problems for you.

`/etc/hosts` should look something like this:

```
            127.0.0.1 localhost
            127.0.0.1 ubuntu.ubuntu-domain ubuntu
```

2.1.2.4. NTP

集群的时钟要保证基本的一致。稍有不一致是可以容忍的，但是很大的不一致会造成奇怪的行为。 运行 NTP 或者其他什么东西来同步你的时间.

如果你查询的时候或者是遇到奇怪的故障，可以检查一下系统时间是否正确!

2.1.2.5. `ulimit` 和 `nproc`

HBase是数据库，会在同一时间使用很多的文件句柄。大多数linux系统使用的默认值1024是不能满足的，会导致FAQ: Why do I see "java.io.IOException...(Too many open files)" in my logs?异常。还可能会发生这样的异常

```
      2010-04-06 03:04:37,542 INFO org.apache.hadoop.hdfs.DFSClient: Exception increateBlockOutputStream java.io.EOFException
      2010-04-06 03:04:37,542 INFO org.apache.hadoop.hdfs.DFSClient: Abandoning block blk_-6935524980745310745_1391901
```

所以你需要修改你的最大文件句柄限制。可以设置到10k。大致的数学运算如下：每列族至少有1个存储文件(StoreFile) 可能达到5-6个如果区域有压力。将每列族的存储文件平均数目和每区域服务器的平均区域数目相乘。例如：假设一个模式有3个列族，每个列族有3个存储文件，每个区域服务器有100个区域，JVM 将打开3 * 3 * 100 = 900 个文件描述符(不包含打开的jar文件，配置文件等)

你还需要修改 hbase 用户的 `nproc`，在压力下，如果过低会造成 `OutOfMemoryError`异常[3] [4]。

需要澄清的，这两个设置是针对操作系统的，不是HBase本身的。有一个常见的错误是HBase运行的用户，和设置最大值的用户不是一个用户。在HBase启动的时候，第一行日志会现在ulimit信息，确保其正确。[5]

2.1.2.5.1. 在Ubuntu上设置`ulimit`

如果你使用的是Ubuntu,你可以这样设置:

在文件 `/etc/security/limits.conf` 添加一行，如:

```
hadoop  -       nofile  32768
```

可以把 `hadoop` 替换成你运行HBase和Hadoop的用户。如果你用两个用户，你就需要配两个。还有配nproc hard 和 soft limits. 如:

```
hadoop soft/hard nproc 32000
```

.

在 `/etc/pam.d/common-session` 加上这一行:

```
session required  pam_limits.so
```

否则在 `/etc/security/limits.conf`上的配置不会生效.

还有注销再登录，这些配置才能生效!

2.1.2.6. Windows

HBase没有怎么在Windows下测试过。所以不推荐在Windows下运行.

如果你实在是想运行，需要安装Cygwin 并虚拟一个unix环境.详情请看 Windows 安装指导 . 或者 搜索邮件列表找找最近的关于windows的注意点

2.1.3. hadoop

选择 Hadoop 版本对HBase部署很关键。下表显示不同HBase支持的Hadoop版本信息。基于HBase版本，应该选择合适的Hadoop版本。我们没有绑定 Hadoop 发行版选择。可以从Apache使用 Hadoop 发行版，或了解一下Hadoop发行商产品： http://wiki.apache.org/hadoop/Distributions%20and%20Commercial%20Support

Table 2.1. Hadoop version support matrix

	HBase-0.92.x	HBase-0.94.x	HBase-0.96
Hadoop-0.20.205	S	X	X
Hadoop-0.22.x	S	X	X
Hadoop-1.0.x	S	S	S
Hadoop-1.1.x	NT	S	S
Hadoop-0.23.x	X	S	NT
Hadoop-2.x	X	S	S

S = supported and tested,支持
X = not supported,不支持
NT = not tested enough.可以运行但测试不充分

由于 HBase 依赖 Hadoop，它配套发布了一个Hadoop jar 文件在它的 lib 下。该套装jar仅用于独立模式。在分布式模式下，Hadoop版本必须和HBase下的版本一致。用你运行的分布式Hadoop版本jar文件替换HBase lib目录下的Hadoop jar文件，以避免版本不匹配问题。确认替换了集群中所有HBase下的jar文件。Hadoop版本不匹配问题有不同表现，但看起来都像挂掉了。

2.1.3.1. Apache HBase 0.92 and 0.94

HBase 0.92 and 0.94 versions can work with Hadoop versions, 0.20.205, 0.22.x, 1.0.x, and 1.1.x. HBase-0.94 can additionally work with Hadoop-0.23.x and 2.x, but you may have to recompile the code using the specific maven profile (see top level pom.xml)

2.1.3.2. Apache HBase 0.96

Apache HBase 0.96.0 requires Apache Hadoop 1.x at a minimum, and it can run equally well on hadoop-2.0. As of Apache HBase 0.96.x, Apache Hadoop 1.0.x at least is required. We will no longer run properly on older Hadoops such as 0.20.205 or branch-0.20-append. Do not move to Apache HBase 0.96.x if you cannot upgrade your Hadoop[6].

2.1.3.3. Hadoop versions 0.20.x - 1.x

HBase will lose data unless it is running on an HDFS that has a durable sync implementation. DO NOT use Hadoop 0.20.2, Hadoop 0.20.203.0, and Hadoop 0.20.204.0 which DO NOT have this attribute. Currently only Hadoop versions 0.20.205.x or any release in excess of this version -- this includes hadoop-1.0.0 -- have a working, durable sync[7]. Sync has to be explicitly enabled by setting dfs.support.append equal to true on both the client side -- in hbase-site.xml -- and on the serverside in hdfs-site.xml (The sync facility HBase needs is a subset of the append code path).

```
  <property>      <name>dfs.support.append</name>      <value>true</value>    </property>
```

You will have to restart your cluster after making this edit. Ignore the chicken-little comment you'll find in the hdfs-default.xml in the description for thedfs.support.append configuration.

2.1.3.4. Hadoop 安全性

HBase运行在Hadoop 0.20.x上，就可以使用其中的安全特性 -- 只要你用这两个版本0.20S 和CDH3B3，然后把hadoop.jar替换掉就可以了.

2.1.3.5. dfs.datanode.max.xcievers

一个 Hadoop HDFS Datanode 有一个同时处理文件的上限. 这个参数叫 xcievers (Hadoop的作者把这个单词拼错了). 在你加载之前，先确认下你有没有配置这个文件conf/hdfs-site.xml里面的xceivers参数，至少要有4096:

```
      <property>
        <name>dfs.datanode.max.xcievers</name>
        <value>4096</value>
      </property>
```

对于HDFS修改配置要记得重启.

如果没有这一项配置，你可能会遇到奇怪的失败。你会在Datanode的日志中看到xcievers exceeded，但是运行起来会报 missing blocks错误。例如: `10/12/08 20:10:31 INFO hdfs.DFSClient: Could not obtain block blk_XXXXXXXXXXXXXXXXXXXXXX_YYYYYYYY from any node: java.io.IOException: No live nodes contain current block. Will get new block locations from namenode and retry...` [5]

See also Section 13.3.4, “Case Study #4 (xcievers Config)”

2.2. HBase运行模式:单机和分布式

HBase有两个运行模式: Section 2.4.1, “单机模式” 和 Section 2.4.2, “分布式模式”. 默认是单机模式，如果要分布式模式你需要编辑 conf 文件夹中的配置文件.

不管是什么模式，你都需要编辑 conf/hbase-env.sh来告知HBase **java**的安装路径.在这个文件里你还可以设置HBase的运行环境，诸如 heapsize和其他 JVM有关的选项, 还有Log文件地址，等等. 设置 JAVA_HOME指向 **java**安装的路径.

2.21. 单机模式

这是默认的模式，在 Section 1.2, “快速开始” 一章中介绍的就是这个模式. 在单机模式中，HBase使用本地文件系统，而不是HDFS ，所有的服务和zooKeeper都运作在一个JVM中。zookeep监听一个端口，这样客户端就可以连接HBase了。

2.2.2. 分布式模式

分布式模式分两种。*伪分布式模式*是把进程运行在一台机器上，但不是一个JVM.而*完全分布式模式*就是把整个服务被分布在各个节点上了 [6].

分布式模式需要使用 *Hadoop Distributed File System* (HDFS).可以参见 HDFS需求和指导来获得关于安装HDFS的指导。在操作HBase之前，你要确认HDFS可以正常运作。

在我们安装之后，你需要确认你的*伪分布式模式*或者 *完全分布式模式*的配置是否正确。这两个模式可以使用同一个验证脚本 Section 2.2.3, “运行和确认你的安装”。

2.2.2.1. 伪分布式模式

伪分布式模式是一个相对简单的分布式模式。这个模式是用来测试的。不能把这个模式用于生产环节，也不能用于测试性能。

你确认HDFS安装成功之后，就可以先编辑 conf/hbase-site.xml。在这个文件你可以加入自己的配置，这个配置会覆盖 Section 2.6.1.1, “HBase 默认配置” and Section 2.2.2.2.3, “HDFS客户端配置”. 运行HBase需要设置hbase.rootdir 属性.该属性是指HBase在HDFS中使用的目录的位置。例如，要想 /hbase 目录，让namenode 监听locahost的9000端口，只有一份数据拷贝(HDFS默认是3份拷贝)。可以在 hbase-site.xml 写上如下内容

```
<configuration>
  ...
  <property>
    <name>hbase.rootdir</name>
    <value>hdfs://localhost:9000/hbase</value>
    <description>The directory shared by RegionServers.
    </description>
  </property>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
    <description>The replication count for HLog & HFile storage. Should not be greater than HDFS datanode count.
    </description>
  </property>
  ...
</configuration>
```

Note

让HBase自己创建 hbase.rootdir

Note

上面我们绑定到 localhost. 也就是说除了本机，其他机器连不上HBase。所以你需要设置成别的，才能使用它。

现在可以跳到 Section 2.2.3, “运行和确认你的安装” 来运行和确认你的伪分布式模式安装了。 [7]

2.2.2.1.1. 伪分布模式配置文件

下面是伪分布模式设置的配置文件示例。

hdfs-site.xml

```
<configuration>    ...    <property>      <name>dfs.name.dir</name>      <value>/Users/local/user.name/hdfs-data-name</value>
```

hbase-site.xml

```
<configuration>    ...    <property>      <name>hbase.rootdir</name>      <value>hdfs://localhost:8020/hbase</value>    </propert
```

2.2.2.1.2. 伪分布模式附加

2.2.2.1.2.1. 启动

启动初始 HBase 集群...

```
% bin/start-hbase.sh
```

在同一服务器启动额外备份主服务器

```
% bin/local-master-backup.sh start 1
```

... '1' 表示使用端口 60001 & 60011, 该备份主服务器及其log文件放在logs/hbase-${USER}-1-master-${HOSTNAME}.log.

启动多个备份主服务器...

```
% bin/local-master-backup.sh start 2 3
```

可以启动到 9 个备份服务器 (总数10 个).

启动更多 regionservers...

```
% bin/local-regionservers.sh start 1
```

'1' 表示使用端口 60201 & 60301 ，log文件在 logs/hbase-${USER}-1-regionserver-${HOSTNAME}.log.

在刚运行的regionserver上增加 4 个额外 regionservers ...

```
% bin/local-regionservers.sh start 2 3 4 5
```

支持到 99 个额外regionservers (总100个).

2.2.2.1.2.2. 停止

假设想停止备份主服务器 # 1, 运行...

```
% cat /tmp/hbase-${USER}-1-master.pid |xargs kill -9
```

注意 bin/local-master-backup.sh 停止 1 会尝试停止主服务器相关集群。

停止单独 regionserver, 运行...

```
% bin/local-regionservers.sh stop 1
```

2.2.2.2. 完全分布式模式

要想运行完全分布式模式，你要进行如下配置，先在 hbase-site.xml, 加一个属性 hbase.cluster.distributed 设置为 true 然后把 hbase.rootdir 设置为HDFS的NameNode的位置。 例如，你的namenode运行在namenode.example.org，端口是9000 你期望的目录是 /hbase,使用如下的配置

```
<configuration>
  ...
  <property>
    <name>hbase.rootdir</name>
    <value>hdfs://namenode.example.org:9000/hbase</value>
    <description>The directory shared by RegionServers.
    </description>
  </property>
  <property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
    <description>The mode the cluster will be in. Possible values are
      false: standalone and pseudo-distributed setups with managed Zookeeper
      true: fully-distributed with unmanaged Zookeeper Quorum (see hbase-env.sh)
    </description>
  </property>
  ...
</configuration>
```

2.2.2.2.1. regionservers

完全分布式模式的还需要修改conf/regionservers. 在 Section 2.7.1.2, “regionservers” 列出了你希望运行的全部 HRegionServer，一行写一个host (就像Hadoop里面的 slaves 一样). 列在这里的server会随着集群的启动而启动，集群的停止而停止.

2.2.2.2.2. ZooKeeper 和 HBase

2.2.2.2.3. HDFS客户端配置

如果你希望Hadoop集群上做*HDFS 客户端配置* ，例如你的HDFS客户端的配置和服务端的不一样。按照如下的方法配置，HBase就能看到你的配置信息:

- 在hbase-env.sh里将HBASE_CLASSPATH环境变量加上HADOOP_CONF_DIR 。
- 在${HBASE_HOME}/conf下面加一个 hdfs-site.xml (或者 hadoop-site.xml) ，最好是软连接
- 如果你的HDFS客户端的配置不多的话，你可以把这些加到 hbase-site.xml上面.

例如HDFS的配置 dfs.replication.你希望复制5份，而不是默认的3份。如果你不照上面的做的话，HBase只会复制3份。

2.2.3. 运行和确认你的安装

首先确认你的HDFS是运行着的。你可以运行HADOOP_HOME中的 bin/start-hdfs.sh 来启动HDFS.你可以通过**put**命令来测试放一个文件，然后有**get**命令来读这个文件。通常情况下HBase是不会运行mapreduce的。所以比不需要检查这些。

*如果*你自己管理ZooKeeper集群，你需要确认它是运行着的。如果是HBase托管，ZoopKeeper会随HBase启动。

用如下命令启动HBase:

```
bin/start-hbase.sh
```

这个脚本在HBASE_HOME目录里面。

你现在已经启动HBase了。HBase把log记在 logs 子目录里面. 当HBase启动出问题的时候，可以看看Log.

HBase也有一个界面，上面会列出重要的属性。默认是在Master的60010端口上H (HBase RegionServers 会默认绑定 60020端口，在端口60030上有一个展示信息的界面).如果Master运行在 master.example.org，端口是默认的话，你可以用浏览器在 http://master.example.org:60010看到主界面. .

一旦HBase启动，参见Section 1.2.3, “Shell 练习”可以看到如何建表，插入数据，scan你的表，还有disable这个表，最后把它删掉。

可以在HBase Shell停止HBase

```
$ ./bin/stop-hbase.sh
stopping hbase...............
```

停止操作需要一些时间，你的集群越大，停的时间可能会越长。如果你正在运行一个分布式的操作，要确认在HBase彻底停止之前，Hadoop不能停.

2.3. 配置文件

HBase的配置系统和Hadoop一样。在`conf/hbase-env.sh`配置系统的部署信息和环境变量。 -- 这个配置会被启动shell使用 -- 然后在XML文件里配置信息，覆盖默认的配置。告知HBase使用什么目录地址，ZooKeeper的位置等等信息。 [10] .

当你使用分布式模式的时间，当你编辑完一个文件之后，记得要把这个文件复制到整个集群的`conf` 目录下。HBase不会帮你做这些，你得用 **rsync**.

2.3.1. `hbase-site.xml` 和 `hbase-default.xml`

正如Hadoop放置HDFS的配置文件`hdfs-site.xml`，HBase的配置文件是 `conf/hbase-site.xml`. 你可以在 Section 2.3.1.1, "HBase 默认配置"找到配置的属性列表。你也可以看有代码里面的`hbase-default.xml`文件，他在`src/main/resources`目录下。

不是所有的配置都在 `hbase-default.xml`出现.只要改了代码，配置就有可能改变，所以唯一了解这些被改过的配置的办法是读源代码本身。

要注意的是，要重启集群才能是配置生效。

2.3.1.1. HBase 默认配置

HBase 默认配置

该文档是用hbase默认配置文件生成的，文件源是 `hbase-default.xml`

`hbase.rootdir`

这个目录是region server的共享目录，用来持久化HBase。URL需要是'完全正确'的，还要包含文件系统的scheme。例如，要表示hdfs中的'/hbase'目录，namenode 运行在namenode.example.org的9090端口。则需要设置为hdfs://namenode.example.org:9000/hbase。默认情况下HBase是写到/tmp的。不改这个配置，数据会在重启的时候丢失。

默认: `file:///tmp/hbase-${user.name}/hbase`

`hbase.master.port`

HBase的Master的端口.

默认: `60000`

`hbase.cluster.distributed`

HBase的运行模式。false是单机模式，true是分布式模式。若为false,HBase和Zookeeper会运行在同一个JVM里面。

默认: `false`

`hbase.tmp.dir`

本地文件系统的临时文件夹。可以修改到一个更为持久的目录上。(/tmp会在重启时清楚)

默认:${java.io.tmpdir}/hbase-${user.name}

hbase.local.dir

作为本地存储，位于本地文件系统的路径。

默认: ${hbase.tmp.dir}/local/

`hbase.master.info.port`

HBase Master web 界面端口. 设置为-1 意味着你不想让他运行。

默认: `60010`

`hbase.master.info.bindAddress`

HBase Master web 界面绑定的端口

默认: `0.0.0.0`

`hbase.client.write.buffer`

HTable客户端的写缓冲的默认大小。这个值越大，需要消耗的内存越大。因为缓冲在客户端和服务端都有实例，所以需要消耗客户端和服务端两个地方的内存。得到的好处是，可以减少RPC的次数。可以这样估算服务器端被占用的内存: hbase.client.write.buffer * hbase.regionserver.handler.count

默认: `2097152`

`hbase.regionserver.port`

HBase RegionServer绑定的端口

默认: `60020`

`hbase.regionserver.info.port`

HBase RegionServer web 界面绑定的端口 设置为 -1 意味这你不想与运行 RegionServer 界面.

默认: `60030`

`hbase.regionserver.info.port.auto`

Master或RegionServer是否要动态搜一个可以用的端口来绑定界面。当hbase.regionserver.info.port已经被占用的时候，可以搜一个空闲的端口绑定。这个功能在测试的时候很有用。默认关闭。

默认: `false`

`hbase.regionserver.info.bindAddress`

HBase RegionServer web 界面的IP地址

默认: `0.0.0.0`

`hbase.regionserver.class`

RegionServer 使用的接口。客户端打开代理来连接region server的时候会使用到。

默认: `org.apache.hadoop.hbase.ipc.HRegionInterface`

`hbase.client.pause`

通常的客户端暂停时间。最多的用法是客户端在重试前的等待时间。比如失败的get操作和region查询操作等都很可能用到。

默认: `1000`

`hbase.client.retries.number`

最大重试次数。所有需重试操作的最大值。例如从root region服务器获取root region，Get单元值，行Update操作等等。这是最大重试错误的值。 Default: 10.

默认: `10`

hbase.bulkload.retries.number

最大重试次数。 原子批加载尝试的迭代最大次数。 0 永不放弃。默认: 0.

默认: 0

`hbase.client.scanner.caching`

当调用Scanner的next方法，而值又不在缓存里的时候，从服务端一次获取的行数。越大的值意味着Scanner会快一些，但是会占用更多的内存。当缓冲被占满的时候，next方法调用会越来越慢。慢到一定程度，可能会导致超时。例如超过了hbase.regionserver.lease.period。

默认: 100

`hbase.client.keyvalue.maxsize`

一个KeyValue实例的最大size.这个是用来设置存储文件中的单个entry的大小上界。因为一个KeyValue是不能分割的，所以可以避免因为数据过大导致region不可分割。明智的做法是把它设为可以被最大region size整除的数。如果设置为0或者更小，就会禁用这个检查。默认10MB。

默认: `10485760`

`hbase.regionserver.lease.period`

客户端租用HRegion server 期限，即超时阀值。单位是毫秒。默认情况下，客户端必须在这个时间内发一条信息，否则视为死掉。

默认: `60000`

`hbase.regionserver.handler.count`

RegionServers受理的RPC Server实例数量。对于Master来说，这个属性是Master受理的handler数量

默认: `10`

`hbase.regionserver.msginterval`

RegionServer 发消息给 Master 时间间隔，单位是毫秒

默认: `3000`

`hbase.regionserver.optionallogflushinterval`

将Hlog同步到HDFS的间隔。如果Hlog没有积累到一定的数量，到了时间，也会触发同步。默认是1秒，单位毫秒。

默认: `1000`

`hbase.regionserver.regionSplitLimit`

region的数量到了这个值后就不会在分裂了。这不是一个region数量的硬性限制。但是起到了一定指导性的作用，到了这个值就该停止分裂了。默认是MAX_INT.就是说不阻止分裂。

默认: `2147483647`

`hbase.regionserver.logroll.period`

提交commit log的间隔，不管有没有写足够的值。

默认: `3600000`

`hbase.regionserver.hlog.reader.impl`

HLog file reader 的实现.

默认: `org.apache.hadoop.hbase.regionserver.wal.SequenceFileLogReader`

`hbase.regionserver.hlog.writer.impl`

HLog file writer 的实现.

默认: `org.apache.hadoop.hbase.regionserver.wal.SequenceFileLogWriter`

`hbase.regionserver.nbreservationblocks`

储备的内存block的数量(译者注:就像石油储备一样)。当发生out of memory 异常的时候，我们可以用这些内存在RegionServer停止之前做清理操作。

默认: `4`

`hbase.zookeeper.dns.interface`

当使用DNS的时候，Zookeeper用来上报的IP地址的网络接口名字。

默认: `default`

`hbase.zookeeper.dns.nameserver`

当使用DNS的时候，Zookeepr使用的DNS的域名或者IP 地址，Zookeeper用它来确定和master用来进行通讯的域名.

默认: `default`

`hbase.regionserver.dns.interface`

当使用DNS的时候，RegionServer用来上报的IP地址的网络接口名字。

默认: `default`

`hbase.regionserver.dns.nameserver`

当使用DNS的时候，RegionServer使用的DNS的域名或者IP 地址，RegionServer用它来确定和master用来进行通讯的域名.

默认: `default`

`hbase.master.dns.interface`

当使用DNS的时候，Master用来上报的IP地址的网络接口名字。

默认: `default`

`hbase.master.dns.nameserver`

当使用DNS的时候，RegionServer使用的DNS的域名或者IP 地址，Master用它来确定用来进行通讯的域名.

默认: `default`

`hbase.balancer.period`

Master执行region balancer的间隔。

默认: `300000`

`hbase.regions.slop`

当任一区域服务器有average + (average * slop)个分区，将会执行重新均衡。默认 20% slop .

默认:0.2

`hbase.master.logcleaner.ttl`

Hlog存在于.oldlogdir 文件夹的最长时间, 超过了就会被 Master 的线程清理掉.

默认: `600000`

`hbase.master.logcleaner.plugins`

LogsCleaner服务会执行的一组LogCleanerDelegat。值用逗号间隔的文本表示。这些WAL/HLog cleaners会按顺序调用。可以把先调用的放在前面。你可以实现自己的LogCleanerDelegat，加到Classpath下，然后在这里写下类的全称。一般都是加在默认值的前面。

默认: `org.apache.hadoop.hbase.master.TimeToLiveLogCleaner`

`hbase.regionserver.global.memstore.upperLimit`

单个region server的全部memtores的最大值。超过这个值，一个新的update操作会被挂起，强制执行flush操作。

默认: `0.4`

`hbase.regionserver.global.memstore.lowerLimit`

当强制执行flush操作的时候，当低于这个值的时候，flush会停止。默认是堆大小的 35% . 如果这个值和 hbase.regionserver.global.memstore.upperLimit 相同就意味着当update操作因为内存限制被挂起时，会尽量少的执行flush(译者注:一旦执行flush，值就会比下限要低，不再执行)

默认: `0.35`

`hbase.server.thread.wakefrequency`

service工作的sleep间隔，单位毫秒。 可以作为service线程的sleep间隔，比如log roller.

默认: `10000`

hbase.server.versionfile.writeattempts

退出前尝试写版本文件的次数。每次尝试由 hbase.server.thread.wakefrequency 毫秒数间隔。

默认: 3

`hbase.hregion.memstore.flush.size`

当memstore的大小超过这个值的时候，会flush到磁盘。这个值被一个线程每隔hbase.server.thread.wakefrequency检查一下。

默认:134217728

`hbase.hregion.preclose.flush.size`

当一个region中的memstore的大小大于这个值的时候，我们又触发了close.会先运行“pre-flush”操作，清理这个需要关闭的memstore，然后将这个region下线。当一个region下线了，我们无法再进行任何写操作。如果一个memstore很大的时候，flush操作会消耗很多时间。"pre-flush"操作意味着在region下线之前，会先把memstore清空。这样在最终执行close操作的时候，flush操作会很快。

默认: `5242880`

`hbase.hregion.memstore.block.multiplier`

如果memstore有hbase.hregion.memstore.block.multiplier倍数的hbase.hregion.flush.size的大小，就会阻塞update操作。这是为了预防在update高峰期会导致的失控。如果不设上界，flush的时候会花很长的时间来合并或者分割，最坏的情况就是引发out of memory异常。(译者注:内存操作的速度和磁盘不匹配，需要等一等。原文似乎有误)

默认: `2`

`hbase.hregion.memstore.mslab.enabled`

体验特性：启用memStore分配本地缓冲区。这个特性是为了防止在大量写负载的时候堆的碎片过多。这可以减少GC操作的频率。(GC有可能会Stop the world)(译者注：实现的原理相当于预分配内存，而不是每一个值都要从堆里分配)

默认: `true`

`hbase.hregion.max.filesize`

最大HStoreFile大小。若某个列族的HStoreFile增长达到这个值，这个Hegion会被切割成两个。 默认: 10G.

默认:10737418240

`hbase.hstore.compactionThreshold`

当一个HStore含有多于这个值的HStoreFiles(每一个memstore flush产生一个HStoreFile)的时候，会执行一个合并操作，把这HStoreFiles写成一个。这个值越大，需要合并的时间就越长。

默认: `3`

`hbase.hstore.blockingStoreFiles`

当一个HStore含有多于这个值的HStoreFiles(每一个memstore flush产生一个HStoreFile)的时候，会执行一个合并操作，update会阻塞直到合并完成，直到超过了hbase.hstore.blockingWaitTime的值

默认: `7`

`hbase.hstore.blockingWaitTime`

hbase.hstore.blockingStoreFiles所限制的StoreFile数量会导致update阻塞，这个时间是来限制阻塞时间的。当超过了这个时间，HRegion会停止阻塞update操作，不过合并还有没有完成。默认为90s.

默认: `90000`

`hbase.hstore.compaction.max`

每个“小”合并的HStoreFiles最大数量。

默认: `10`

`hbase.hregion.majorcompaction`

一个Region中的所有HStoreFile的major compactions的时间间隔。默认是1天。 设置为0就是禁用这个功能。

默认: `86400000`

hbase.storescanner.parallel.seek.enable

允许 StoreFileScanner 并行搜索 StoreScanner, 一个在特定条件下降低延迟的特性。

默认: false

hbase.storescanner.parallel.seek.threads

并行搜索特性打开后，默认线程池大小。

默认: 10

`hbase.mapreduce.hfileoutputformat.blocksize`

MapReduce中HFileOutputFormat可以写 storefiles/hfiles. 这个值是hfile的blocksize的最小值。通常在HBase写Hfile的时候，bloocksize是由table schema(HColumnDescriptor)决定的，但是在mapreduce写的时候，我们无法获取schema中blocksize。这个值越小，你的索引就越大，你随机访问需要获取的数据就越小。如果你的cell都很小，而且你需要更快的随机访问，可以把这个值调低。

默认: `65536`

`hfile.block.cache.size`

分配给HFile/StoreFile的block cache占最大堆(-Xmx setting)的比例。默认0.25意思是分配25%，设置为0就是禁用，但不推荐。

默认:0.25

`hbase.hash.type`

哈希函数使用的哈希算法。可以选择两个值:: murmur (MurmurHash) 和 jenkins (JenkinsHash). 这个哈希是给 bloom filters用的.

默认: `murmur`

hfile.block.index.cacheonwrite

This allows to put non-root multi-level index blocks into the block cache at the time the index is being written.

Default: false

hfile.index.block.max.size

When the size of a leaf-level, intermediate-level, or root-level index block in a multi-level block index grows to this size, the block is written out and a new block is started.

Default: 131072

hfile.format.version

The HFile format version to use for new files. Set this to 1 to test backwards-compatibility. The default value of this option should be consistent with FixedFileTrailer.MAX_VERSION.

Default: 2

io.storefile.bloom.block.size

The size in bytes of a single block ("chunk") of a compound Bloom filter. This size is approximate, because Bloom blocks can only be inserted at data block boundaries, and the number of keys per data block varies.

Default: 131072

hfile.block.bloom.cacheonwrite

Enables cache-on-write for inline blocks of a compound Bloom filter.

Default: false

hbase.rs.cacheblocksonwrite

Whether an HFile block should be added to the block cache when the block is finished.

Default: false

hbase.rpc.server.engine

Implementation of org.apache.hadoop.hbase.ipc.RpcServerEngine to be used for server RPC call marshalling.

Default: org.apache.hadoop.hbase.ipc.ProtobufRpcServerEngine

hbase.ipc.client.tcpnodelay

Set no delay on rpc socket connections. See http://docs.oracle.com/javase/1.5.0/docs/api/java/net/Socket.html#getTcpNoDelay()

Default: true

`hbase.master.keytab.file`

HMaster server验证登录使用的kerberos keytab 文件路径。(译者注：HBase使用Kerberos实现安全)

默认:

`hbase.master.kerberos.principal`

例如. "hbase/_HOST@EXAMPLE.COM". HMaster运行需要使用 kerberos principal name. principal name 可以在: user/hostname@DOMAIN 中获取. 如果 "_HOST" 被用做hostname portion，需要使用实际运行的hostname来替代它。

默认:

`hbase.regionserver.keytab.file`

HRegionServer验证登录使用的kerberos keytab 文件路径。

默认:

`hbase.regionserver.kerberos.principal`

例如. "hbase/_HOST@EXAMPLE.COM". HRegionServer运行需要使用 kerberos principal name. principal name 可以在: user/hostname@DOMAIN 中获取. 如果 "_HOST" 被用做hostname portion，需要使用实际运行的hostname来替代它。在这个文件中必须要有一个entry来描述 hbase.regionserver.keytab.file

默认:

hadoop.policy.file

The policy configuration file used by RPC servers to make authorization decisions on client requests. Only used when HBase security is enabled.

Default: hbase-policy.xml

hbase.superuser

List of users or groups (comma-separated), who are allowed full privileges, regardless of stored ACLs, across the cluster. Only used when HBase security is enabled.

Default:

hbase.auth.key.update.interval

The update interval for master key for authentication tokens in servers in milliseconds. Only used when HBase security is enabled.

Default: 86400000

hbase.auth.token.max.lifetime

The maximum lifetime in milliseconds after which an authentication token expires. Only used when HBase security is enabled.

Default: 604800000

`zookeeper.session.timeout`

ZooKeeper 会话超时.HBase把这个值传递改zk集群，向他推荐一个会话的最大超时时间。详见 http://hadoop.apache.org/zookeeper/docs/current/zookeeperProgrammers.html#ch_zkSessions "The client sends a requested timeout, the server responds with the timeout that it can give the client. "。 单位是毫秒

默认: `180000`

`zookeeper.znode.parent`

ZooKeeper中的HBase的根ZNode。所有的HBase的ZooKeeper会用这个目录配置相对路径。默认情况下，所有的HBase的ZooKeeper文件路径是用相对路径，所以他们会都去这个目录下面。

默认: `/hbase`

`zookeeper.znode.rootserver`

ZNode 保存的 根region的路径. 这个值是由Master来写，client和regionserver 来读的。如果设为一个相对地址，父目录就是 ${zookeeper.znode.parent}.默认情形下，意味着根region的路径存储在/hbase/root-region-server.

默认: `root-region-server`

zookeeper.znode.acl.parent

Root ZNode for access control lists.

Default: acl

hbase.coprocessor.region.classes

A comma-separated list of Coprocessors that are loaded by default on all tables. For any override coprocessor method, these classes will be called in order. After implementing your own Coprocessor, just put it in HBase's classpath and add the fully qualified class name here. A coprocessor can also be loaded on demand by setting HTableDescriptor.

Default:

hbase.coprocessor.master.classes

A comma-separated list of org.apache.hadoop.hbase.coprocessor.MasterObserver coprocessors that are loaded by default on the active HMaster process. For any implemented coprocessor methods, the listed classes will be called in order. After implementing your own MasterObserver, just put it in HBase's classpath and add the fully qualified class name here.

Default:

`hbase.zookeeper.quorum`

Zookeeper集群的地址列表，用逗号分割。例如："host1.mydomain.com,host2.mydomain.com,host3.mydomain.com".默认是localhost,是给伪分布式用的。要修改才能在完全分布式的情况下使用。如果在hbase-env.sh设置了HBASE_MANAGES_ZK，这些ZooKeeper节点就会和HBase一起启动。

默认: `localhost`

`hbase.zookeeper.peerport`

ZooKeeper节点使用的端口。详细参见：http://hadoop.apache.org/zookeeper/docs/r3.1.1/zookeeperStarted.html#sc_RunningReplicatedZooKeeper

默认: `2888`

`hbase.zookeeper.leaderport`

ZooKeeper用来选择Leader的端口，详细参见：http://hadoop.apache.org/zookeeper/docs/r3.1.1/zookeeperStarted.html#sc_RunningReplicatedZooKeeper

默认: `3888`

hbase.zookeeper.useMulti

Instructs HBase to make use of ZooKeeper's multi-update functionality. This allows certain ZooKeeper operations to complete more quickly and prevents some issues with rare Replication failure scenarios (see the release note of HBASE-2611 for an example). IMPORTANT: only set this to true if all ZooKeeper servers in the cluster are on version 3.4+ and will not be downgraded. ZooKeeper versions before 3.4 do not support multi-update and will not fail gracefully if multi-update is invoked (see ZOOKEEPER-1495).

Default: false

`hbase.zookeeper.property.initLimit`

ZooKeeper的zoo.conf中的配置。 初始化synchronization阶段的ticks数量限制

默认: `10`

`hbase.zookeeper.property.syncLimit`

ZooKeeper的zoo.conf中的配置。 发送一个请求到获得承认之间的ticks的数量限制

默认: `5`

`hbase.zookeeper.property.dataDir`

ZooKeeper的zoo.conf中的配置。 快照的存储位置

默认: `${hbase.tmp.dir}/zookeeper`

`hbase.zookeeper.property.clientPort`

ZooKeeper的zoo.conf中的配置。 客户端连接的端口

默认: `2181`

`hbase.zookeeper.property.maxClientCnxns`

ZooKeeper的zoo.conf中的配置。 ZooKeeper集群中的单个节点接受的单个Client(以IP区分)的请求的并发数。这个值可以调高一点，防止在单机和伪分布式模式中出问题。

默认: `300`

`hbase.rest.port`

HBase REST server的端口

默认: `8080`

`hbase.rest.readonly`

定义REST server的运行模式。可以设置成如下的值： false: 所有的HTTP请求都是被允许的 - GET/PUT/POST/DELETE. true:只有GET请求是被允许的

默认: `false`

hbase.defaults.for.version.skip

Set to true to skip the 'hbase.defaults.for.version' check. Setting this to true can be useful in contexts other than the other side of a maven generation; i.e. running in an ide. You'll want to set this boolean to true to avoid seeing the RuntimException complaint: "hbase-default.xml file seems to be for and old version of HBase (\${hbase.version}), this version is X.X.X-SNAPSHOT"

Default: false

hbase.coprocessor.abortonerror

Set to true to cause the hosting server (master or regionserver) to abort if a coprocessor throws a Throwable object that is not IOException or a subclass of IOException. Setting it to true might be useful in development environments where one wants to terminate the server as soon as possible to simplify coprocessor failure analysis.

Default: false

hbase.online.schema.update.enable

Set true to enable online schema changes. This is an experimental feature. There are known issues modifying table schemas at the same time a region split is happening so your table needs to be quiescent or else you have to be running with splits disabled.

Default: false

hbase.table.lock.enable

Set to true to enable locking the table in zookeeper for schema change operations. Table locking from master prevents concurrent schema modifications to corrupt table state.

Default: true

dfs.support.append

Does HDFS allow appends to files? This is an hdfs config. set in here so the hdfs client will do append support. You must ensure that this config. is true serverside too when running hbase (You will have to restart your cluster after setting it).

Default: true

hbase.thrift.minWorkerThreads

The "core size" of the thread pool. New threads are created on every connection until this many threads are created.

Default: 16

hbase.thrift.maxWorkerThreads

The maximum size of the thread pool. When the pending request queue overflows, new threads are created until their number reaches this number. After that, the server starts dropping connections.

Default: 1000

hbase.thrift.maxQueuedRequests

The maximum number of pending Thrift connections waiting in the queue. If there are no idle threads in the pool, the server queues requests. Only when the queue overflows, new threads are added, up to hbase.thrift.maxQueuedRequests threads.

Default: 1000

hbase.offheapcache.percentage

The amount of off heap space to be allocated towards the experimental off heap cache. If you desire the cache to be disabled, simply set this value to 0.

Default: 0

hbase.data.umask.enable

Enable, if true, that file permissions should be assigned to the files written by the regionserver

Default: false

hbase.data.umask

File permissions that should be used to write data files when hbase.data.umask.enable is true

Default: 000

hbase.metrics.showTableName

Whether to include the prefix "tbl.tablename" in per-column family metrics. If true, for each metric M, per-cf metrics will be reported for tbl.T.cf.CF.M, if false, per-cf metrics will be aggregated by column-family across tables, and reported for cf.CF.M. In both cases, the aggregated metric M across tables and cfs will be reported.

Default: true

hbase.metrics.exposeOperationTimes

Whether to report metrics about time taken performing an operation on the region server. Get, Put, Delete, Increment, and Append can all have their times exposed through Hadoop metrics per CF and per region.

Default: true

hbase.master.hfilecleaner.plugins

A comma-separated list of HFileCleanerDelegate invoked by the HFileCleaner service. These HFiles cleaners are called in order, so put the cleaner that prunes the most files in front. To implement your own HFileCleanerDelegate, just put it in HBase's classpath and add the fully qualified class name here. Always add the above default log cleaners in the list as they will be overwritten in hbase-site.xml.

Default: org.apache.hadoop.hbase.master.cleaner.TimeToLiveHFileCleaner

hbase.regionserver.catalog.timeout

Timeout value for the Catalog Janitor from the regionserver to META.

Default: 600000

hbase.master.catalog.timeout

Timeout value for the Catalog Janitor from the master to META.

Default: 600000

hbase.config.read.zookeeper.config

Set to true to allow HBaseConfiguration to read the zoo.cfg file for ZooKeeper properties. Switching this to true is not recommended, since the functionality of reading ZK properties from a zoo.cfg file has been deprecated.

Default: false

hbase.snapshot.enabled

Set to true to allow snapshots to be taken / restored / cloned.

Default: true

hbase.rest.threads.max

The maximum number of threads of the REST server thread pool. Threads in the pool are reused to process REST requests. This controls the maximum number of requests processed concurrently. It may help to control the memory used by the REST server to avoid OOM issues. If the thread pool is full, incoming requests will be queued up and wait for some free threads. The default is 100.

Default: 100

hbase.rest.threads.min

The minimum number of threads of the REST server thread pool. The thread pool always has at least these number of threads so the REST server is ready to serve incoming requests. The default is 2.

Default: 2

2.3.2. `hbase-env.sh`

在这个文件里面设置HBase环境变量。比如可以配置JVM启动的堆大小或者GC的参数。你还可在这里配置HBase的参数，如Log位置，niceness(译者注:优先级)，ssh参数还有pid文件的位置等等。打开文件`conf/hbase-env.sh`细读其中的内容。每个选项都是有详尽的注释的。你可以在此添加自己的环境变量。

这个文件的改动系统HBase重启才能生效。

2.3.3. `log4j.properties`

编辑这个文件可以改变HBase的日志的级别，轮滚策略等等。

这个文件的改动系统HBase重启才能生效。 日志级别的更改会影响到HBase UI

2.3.4. 连接HBase集群的客户端配置和依赖

因为HBase的Master有可能转移，所有客户端需要访问ZooKeeper来获得现在的位置。ZooKeeper会保存这些值。因此客户端必须知道Zookeeper集群的地址，否则做不了任何事情。通常这个地址存在 `hbase-site.xml` 里面，客户端可以从CLASSPATH取出这个文件.

如果你是使用一个IDE来运行HBase客户端，你需要将`conf/`放入你的 classpath,这样 `hbase-site.xml`就可以找到了，(或者把hbase-site.xml放到 `src/test/resources`，这样测试的时候可以使用).

HBase客户端最小化的依赖是 hbase, hadoop, log4j, commons-logging, commons-lang, 和 ZooKeeper ，这些jars 需要能在 CLASSPATH 中找到。

下面是一个基本的客户端 `hbase-site.xml` 例子：

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>hbase.zookeeper.quorum</name>
    <value>example1,example2,example3</value>
    <description>The directory shared by region servers.
    </description>
  </property>
```

```
</configuration>
```

2.3.4.1. Java客户端配置

Java是如何读到hbase-site.xml 的内容的

Java客户端使用的配置信息是被映射在一个HBaseConfiguration 实例中. HBaseConfiguration有一个工厂方法, HBaseConfiguration.create();,运行这个方法的时候，他会去CLASSPATH,下找hbase-site.xml，读他发现的第一个配置文件的内容。 (这个方法还会去找hbase-default.xml ; hbase.X.X.X.jar里面也会有一个an hbase-default.xml). 不使用任何hbase-site.xml文件直接通过Java代码注入配置信息也是可以的。例如，你可以用编程的方式设置ZooKeeper信息，只要这样做:

```
Configuration config = HBaseConfiguration.create();
config.set("hbase.zookeeper.quorum", "localhost");  // Here we are running zookeeper locally
```

如果有多ZooKeeper实例，你可以使用逗号列表。(就像在hbase-site.xml 文件中做得一样). 这个 Configuration 实例会被传递到 HTable, 之类的实例里面去.

2.4. 配置示例

2.4.1. 简单的分布式HBase安装

这里是一个10节点的HBase的简单示例，这里的配置都是基本的，节点名为 example0, example1... 一直到 example9 . HBase Master 和 HDFS namenode 运作在同一个节点 example0上. RegionServers 运行在节点 example1-example9. 一个 3-节点 ZooKeeper 集群运行在example1, example2, 和 example3，端口保持默认. ZooKeeper 的数据保存在目录 /export/zookeeper. 下面我们展示主要的配置文件-- hbase-site.xml, regionservers, 和 hbase-env.sh -- 这些文件可以在 conf目录找到.

2.4.1.1. hbase-site.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>hbase.zookeeper.quorum</name>
    <value>example1,example2,example3</value>
    <description>The directory shared by RegionServers.
    </description>
  </property>
  <property>
    <name>hbase.zookeeper.property.dataDir</name>
    <value>/export/zookeeper</value>
    <description>Property from ZooKeeper's config zoo.cfg.
    The directory where the snapshot is stored.
    </description>
  </property>
  <property>
    <name>hbase.rootdir</name>
    <value>hdfs://example0:9000/hbase</value>
    <description>The directory shared by RegionServers.
    </description>
  </property>
  <property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
    <description>The mode the cluster will be in. Possible values are
      false: standalone and pseudo-distributed setups with managed Zookeeper
      true: fully-distributed with unmanaged Zookeeper Quorum (see hbase-env.sh)
    </description>
  </property>
</configuration>
```

2.4.1.2. regionservers

这个文件把RegionServer的节点列了下来。在这个例子里面我们让所有的节点都运行RegionServer,除了第一个节点 example1，它要运行 HBase Master 和 HDFS namenode

```
    example1
    example3
    example4
    example5
    example6
    example7
    example8
    example9
```

2.4.1.3. hbase-env.sh

下面我们用**diff** 命令来展示 hbase-env.sh 文件相比默认变化的部分. 我们把HBase的堆内存设置为4G而不是默认的1G.

```

$ git diff hbase-env.sh
diff --git a/conf/hbase-env.sh b/conf/hbase-env.sh
index e70ebc6..96f8c27 100644
--- a/conf/hbase-env.sh
+++ b/conf/hbase-env.sh
@@ -31,7 +31,7 @@ export JAVA_HOME=/usr/lib//jvm/java-6-sun/
 # export HBASE_CLASSPATH=

 # The maximum amount of heap to use, in MB. Default is 1000.
-# export HBASE_HEAPSIZE=1000
+export HBASE_HEAPSIZE=4096

 # Extra Java runtime options.
 # Below are what we set by default.  May only work with SUN JVM.
```

你可以使用 **rsync** 来同步 `conf` 文件夹到你的整个集群.

2.5. 重要的配置

下面我们会列举*重要* 的配置. 这个章节讲述必须的配置和那些值得一看的配置。(译者注:淘宝的博客也有本章节的内容，HBase性能调优，很详尽)。

2.5.1. 必须的配置

参考 Section 2.2, “操作系统” 和 Section 2.3, “Hadoop” 节.

2.5.2. 推荐配置

2.5.2.1. zookeeper.session.timeout

这个默认值是3分钟。这意味着一旦一个server宕掉了，Master至少需要3分钟才能察觉到宕机，开始恢复。你可能希望将这个超时调短，这样Master就能更快的察觉到了。在你调这个值之前，你需要确认你的JVM的GC参数，否则一个长时间的GC操作就可能导致超时。（当一个RegionServer在运行一个长时间的GC的时候，你可能想要重启并恢复它）.

要想改变这个配置，可以编辑 `hbase-site.xml`, 将配置部署到全部集群，然后重启。

我们之所以把这个值调的很高，是因为我们不想一天到晚在论坛里回答新手的问题。“为什么我在执行一个大规模数据导入的时候Region Server死掉啦”，通常这样的问题是因为长时间的GC操作引起的，他们的JVM没有调优。我们是这样想的，如果一个人对HBase不很熟悉，不能期望他知道所有，打击他的自信心。等到他逐渐熟悉了，他就可以自己调这个参数了。

2.5.2.2. ZooKeeper 实例个数

参考 Section 2.5, “ZooKeeper”.

2.5.2.3. hbase.regionserver.handler.count

这个设置决定了处理用户请求的线程数量。默认是10，这个值设的比较小，主要是为了预防用户用一个比较大的写缓冲，然后还有很多客户端并发，这样region servers会垮掉。有经验的做法是，当请求内容很大(上MB，如大puts, 使用缓存的scans)的时候，把这个值放低。请求内容较小的时候(gets, 小puts, ICVs, deletes)，把这个值放大。

当客户端的请求内容很小的时候，把这个值设置的和最大客户端数量一样是很安全的。一个典型的例子就是一个给网站服务的集群，put操作一般不会缓冲,绝大多数的操作是get操作。

把这个值放大的危险之处在于，把所有的Put操作缓冲意味着对内存有很大的压力，甚至会导致OutOfMemory.一个运行在内存不足的机器的RegionServer会频繁的触发GC操作，渐渐就能感受到停顿。(因为所有请求内容所占用的内存不管GC执行几遍也是不能回收的)。一段时间后，集群也会受到影响，因为所有的指向这个region的请求都会变慢。这样就会拖累集群，加剧了这个问题。

你可能会对handler太多或太少有感觉，可以通过 Section 12.2.2.1, “启用 RPC级 日志” ，在单个RegionServer启动log并查看log末尾 (请求队列消耗内存)。

2.5.2.4. 大内存机器的配置

HBase有一个合理的保守的配置，这样可以运作在所有的机器上。如果你有台大内存的集群-HBase有8G或者更大的heap,接下来的配置可能会帮助你. TODO.

2.5.2.5. 压缩

应该考虑启用ColumnFamily 压缩。有好几个选项，通过降低存储文件大小以降低IO，降低消耗且大多情况下提高性能。

参考 Appendix C, *HBase压缩* 获取更多信息.

2.5.2.6. 较大 Regions

更大的Region可以使你集群上的Region的总数量较少。 一般来言，更少的Region可以使你的集群运行更加流畅。(你可以自己随时手工将大Region切割，这样单个热点Region就会被分布在集群的更多节点上)。

较少的Region较好。一般每个RegionServer在20到小几百之间。 调整Region大小以适合该数字。

0.90.x 版本, 默认情况下单个Region是256MB。Region 大小的上界是 4Gb. 0.92.x 版本, 由于 HFile v2 已经将Region大小支持得大很多， (如, 20Gb).

可能需要实验，基于硬件和应用需要进行配置。

可以调整`hbase-site.xml`中的 `hbase.hregion.max.filesize`属性. RegionSize 也可以基于每个表设置： HTableDescriptor.

2.5.2.7. 管理 Splitting

除了让HBase自动切割你的Region,你也可以手动切割。 [12] 随着数据量的增大，splite会被持续执行。如果你需要知道你现在有几个region,比如长时间的debug或者做调优，你需要手动切割。通过跟踪日志来了解region级的问题是很难的，因为他在不停的切割和重命名。data offlineing bug和未知量的region会让你没有办法。如果一个 `HLog` 或者 `StoreFile`由于一个奇怪的bug，HBase没有执行它。等到一天之后，你才发现这个问题，你可以确保现在的regions和那个时候的一样，这样你就可以restore或者replay这些数据。你还可以调优你的合并算法。如果数据是均匀的，随着数据增长，很容易导致split / compaction疯狂的运行。因为所有的region都是差不多大的。用手的切割，你就可以交错执行定时的合并和切割操作，降低IO负载。

为什么我关闭自动split呢？ 因为自动的splite是配置文件中的 `hbase.hregion.max.filesize`决定的. 你把它设置成`Long.MAX_VALUE`是不

推荐的做法，要是你忘记了手工切割怎么办.推荐的做法是设置成100GB，一旦到达这样的值，至少需要一个小时执行 major compactions。

那什么是最佳的在pre-splite regions的数量呢。这个决定于你的应用程序了。你可以先从低的开始，比如每个server10个pre-splite regions.然后花时间观察数据增长。有太少的region至少比出错好，你可以之后再rolling split.一个更复杂的答案是这个值是取决于你的region中的最大的storefile。随着数据的增大，这个也会跟着增大。 你可以当这个文件足够大的时候，用一个定时的操作使用`Store`的合并选择算法(compact selection algorithm)来仅合并这一个HStore。如果你不这样做，这个算法会启动一个 major compactions，很多region会受到影响，你的集群会疯狂的运行。需要注意的是，这样的疯狂合并操作是数据增长造成的，而不是手动分割操作决定的。

如果你 pre-split 导致 regions 很小,你可以通过配置`HConstants.MAJOR_COMPACTION_PERIOD`把你的major compaction参数调大

如果你的数据变得太大，可以使用`org.apache.hadoop.hbase.util.RegionSplitter` 脚本来执行针对全部集群的一个网络IO安全的rolling split操作。

2.5.2.8. 管理 Compactions

通常管理技术是手动管理主紧缩(major compactions)， 而不是让HBase 来做。 缺省HConstants.MAJOR_COMPACTION_PERIOD 是一天。主紧缩可能强行进行，在你并不太希望发生的时候——特别是在一个繁忙系统。关闭自动主紧缩，设置该值为0.

重点强调，主紧缩对存储文件(StoreFile)清理是绝对必要的。唯一变量是发生的时间。可以通过HBase shell进行管理，或通过 HBaseAdmin.

更多信息关于紧缩和紧缩文件选择过程，参考 Section 9.7.5.5, “紧缩”

2.5.2.9. 预测执行 (Speculative Execution)

MapReduce任务的预测执行缺省是打开的，HBase集群一般建议在系统级关闭预测执行，除非在某种特殊情况下需要打开，此时可以每任务配置。设置mapred.map.tasks.speculative.execution 和 mapred.reduce.tasks.speculative.execution 为 false.

2.5.3. 其他配置

2.5.3.1. 负载均衡

负载均衡器(LoadBalancer)是在主服务器上运行的定期操作，以重新分布集群区域。通过hbase.balancer.period 设置，缺省值300000 (5 分钟).

参考 Section 9.5.4.1, “负载均衡” 获取关于负载均衡器(LoadBalancer)的更多信息。

2.5.3.2. 禁止块缓存(Blockcache)

不要关闭块缓存 (通过hbase.block.cache.size 为 0 来设置)。当前如果关闭块缓存会很不好，因为区域服务器会花很多时间不停加载hfile指数。如果工作集如此配置块缓存没有好处，最少应保证hfile指数保存在块缓存内的大小(可以通过查询区域服务器的UI，得到大致的数值。可以看到网页的上方有块指数值统计).

2.5.3.3. Nagle算法 或小包问题

如果操作HBase时看到大量40ms左右的偶然延时，尝试Nagles配置。如，参考用户邮件列表线索, Inconsistent scan performance with caching set to 1 ，该议题在其中启用tcpNoDelay (译者注，本英文原文notcpdelay有误)提高了扫描速度。你也可以查看该文档的尾部图表： HBASE-7008 Set scanner caching to a better default (xie liang)，我们的Lars Hofhansl 尝试了各种不同的数据大小，Nagle打开或关闭的测量结果。

[1] Be careful editing XML. Make sure you close all elements. Run your file through **xmllint** or similar to ensure well-formedness of your document after an edit session.

[2] The hadoop-dns-checker tool can be used to verify DNS is working correctly on the cluster. The project README file provides detailed instructions on usage.

[3] 参考 Jack Levin's major hdfs issues note up on the user list.

[4] The requirement that a database requires upping of system limits is not peculiar to HBase. 参考 for example the section *Setting Shell Limits for the Oracle User* inShort Guide to install Oracle 10 on Linux.

[5] A useful read setting config on you hadoop cluster is Aaron Kimballs' Configuration Parameters: What can you just ignore?

[6] <title>On Hadoop Versions</title>

[6] The Cloudera blog post An update on Apache Hadoop 1.0 by Charles Zedlweski has a nice exposition on how all the Hadoop versions relate. Its worth checking out if you are having trouble making sense of the Hadoop version morass.

[7] Until recently only the branch-0.20-append branch had a working sync but no official release was ever made from this branch. You had to build it yourself. Michael Noll wrote a detailed blog, Building an Hadoop 0.20.x version for HBase 0.90.2, on how to build an Hadoop from branch-0.20-append. Recommended.

[8] Praveen Kumar has written a complimentary article, Building Hadoop and HBase for HBase Maven application development.

[9] dfs.support.append

[10] 参考 Hadoop HDFS: Deceived by Xciever for an informative rant on xceivering.

[11] The pseudo-distributed vs fully-distributed nomenclature comes from Hadoop.

[12] 参考 Section 2.4.2.1.2, “Pseudo-distributed Extras” for notes on how to start extra Masters and RegionServers when running pseudo-distributed.

[13] 对 ZooKeeper 全部配置，参考ZooKeeper 的`zoo.cfg`. HBase 没有包含 `zoo.cfg` ，所以需要浏览合适的独立ZooKeeper下载版本的 `conf` 目录找到。

[14] What follows is taken from the javadoc at the head of the org.apache.hadoop.hbase.util.RegionSplitter tool added to HBase post-0.90.0 release.

Chapter 3. 升级

不能跳过主要版本升级。如果想从0.20.x 升级到 0.92.x，必须从0.20.x 升级到 0.90.x ，再从0.90.x 升级到 0.92.x.

参见 Section 2, “配置”, 需要特别注意有关Hadoop 版本的信息.

3.1. 从 0.94.x 升级到 0.96.x

The Singularity

You will have to stop your old 0.94 cluster completely to upgrade. If you are replicating between clusters, both clusters will have to go down to upgrade. Make sure it is a clean shutdown so there are no WAL files laying around (TODO: Can 0.96 read 0.94 WAL files?). Make sure zookeeper is cleared of state. All clients must be upgraded to 0.96 too.

The API has changed in a few areas; in particular how you use coprocessors (TODO: MapReduce too?)

3.2. 从 0.92.x 升级到 0.94.x

0.92 和 0.94 接口兼容，可平滑升级。

3.3. 从 0.90.x 到 0.92.x 升级

升级指引

You will find that 0.92.0 runs a little differently to 0.90.x releases. Here are a few things to watch out for upgrading from 0.90.x to 0.92.0.

If you've not patience, here are the important things to know upgrading.

1. Once you upgrade, you can’t go back.
2. MSLAB is on by default. Watch that heap usage if you have a lot of regions.
3. Distributed splitting is on by defaul. It should make region server failover faster.
4. There’s a separate tarball for security.
5. If -XX:MaxDirectMemorySize is set in your hbase-env.sh, it’s going to enable the experimental off-heap cache (You may not want this).

3.3.1. 不可回退!

To move to 0.92.0, all you need to do is shutdown your cluster, replace your hbase 0.90.x with hbase 0.92.0 binaries (be sure you clear out all 0.90.x instances) and restart (You cannot do a rolling restart from 0.90.x to 0.92.x -- you must restart). On startup, the .META. table content is rewritten removing the table schema from the info:regioninfo column. Also, any flushes done post first startup will write out data in the new 0.92.0 file format, HFile V2. This means you cannot go back to 0.90.x once you’ve started HBase 0.92.0 over your HBase data directory.

3.3.2. MSLAB 缺省启用

In 0.92.0, the hbase.hregion.memstore.mslab.enabled flag is set to true (参考 Section 11.3.1.1, “Long GC pauses”). In 0.90.x it was false. When it is enabled, memstores will step allocate memory in MSLAB 2MB chunks even if the memstore has zero or just a few small elements. This is fine usually but if you had lots of regions per regionserver in a 0.90.x cluster (and MSLAB was off), you may find yourself OOME'ing on upgrade because the thousands of regions * number of column families * 2MB MSLAB (at a minimum) puts your heap over the top. Set hbase.hregion.memstore.mslab.enabled to false or set the MSLAB size down from 2MB by setting hbase.hregion.memstore.mslab.chunksize to something less.

3.3.3. 分布式分割缺省启用

Previous, WAL logs on crash were split by the Master alone. In 0.92.0, log splitting is done by the cluster (参考 “HBASE-1364 [performance] Distributed splitting of regionserver commit logs”). This should cut down significantly on the amount of time it takes splitting logs and getting regions back online again.

3.3.4. 内存计算改变

In 0.92.0, Appendix E, *HFile format version 2* indices and bloom filters take up residence in the same LRU used caching blocks that come from the filesystem. In 0.90.x, the HFile v1 indices lived outside of the LRU so they took up space even if the index was on a ‘cold’ file, one that wasn’t being actively used. With the indices now in the LRU, you may find you have less space for block caching. Adjust your block cache accordingly. 参考 the Section 9.6.4, “Block Cache” for more detail. The block size default size has been changed in 0.92.0 from 0.2 (20 percent of heap) to 0.25.

3.3.5. 可用 Hadoop 版本

Run 0.92.0 on Hadoop 1.0.x (or CDH3u3 when it ships). The performance benefits are worth making the move. Otherwise, our Hadoop prescription is as it has been; you need an Hadoop that supports a working sync. 参考 Section 2.3, “Hadoop”.

If running on Hadoop 1.0.x (or CDH3u3), enable local read. 参考 Practical Caching presentation for ruminations on the performance benefits ‘going local’ (and for how to enable local reads).

3.3.6. HBase 0.92.0 带 ZooKeeper 3.4.2

If you can, upgrade your zookeeper. If you can’t, 3.4.2 clients should work against 3.3.X ensembles (HBase makes use of 3.4.2 API).

3.3.7. 在线切换缺省关闭

In 0.92.0, we’ve added an experimental online schema alter facility (参考 hbase.online.schema.update.enable). Its off by default. Enable it at your own risk. Online alter and splitting tables do not play well together so be sure your cluster quiescent using this feature (for now).

3.3.8. WebUI

The webui has had a few additions made in 0.92.0. It now shows a list of the regions currently transitioning, recent compactions/flushes, and a process list of running processes (usually empty if all is well and requests are being handled promptly). Other additions including requests by region, a debugging servlet dump, etc.

3.3.9. 安全 tarball

我们发布两个tarball： 安全和非安全 HBase. 如何设置安全HBase的文档正在制定中。

3.3.10. 试验离堆(off-heap)缓存

(译者注：on-heap和off-heap是Terracotta 公司提出的概念。on-heap指java对象在GC内存储管理，效率较高，但GC只能管理2G内存，有时成为性能瓶颈。off-heap又叫BigMemory ，是JVM的GC机制的替代，在GC外存储，100倍速于DiskStore，cache量目前(2012年底)达到350GB)

A new cache was contributed to 0.92.0 to act as a solution between using the “on-heap” cache which is the current LRU cache the region servers have and the operating system cache which is out of our control. To enable, set “-XX:MaxDirectMemorySize” in hbase-env.sh to the value for maximum direct memory size and specify hbase.offheapcache.percentage in hbase-site.xml with the percentage that you want to dedicate to off-heap cache. This should only be set for servers and not for clients. Use at your own risk. See this blog post for additional information on this new experimental feature: http://www.cloudera.com/blog/2012/01/caching-in-hbase-slabcache/

3.3.11. HBase 复制的变动

0.92.0 adds two new features: multi-slave and multi-master replication. The way to enable this is the same as adding a new peer, so in order to have multi-master you would just run add_peer for each cluster that acts as a master to the other slave clusters. Collisions are handled at the timestamp level which may or may not be what you want, this needs to be evaluated on a per use case basis. Replication is still experimental in 0.92 and is disabled by default, run it at your own risk.

3.3.12. 对OOME ， RegionServer 现在退出

If an OOME, we now have the JVM kill -9 the regionserver process so it goes down fast. Previous, a RegionServer might stick around after incurring an OOME limping along in some wounded state. To disable this facility, and recommend you leave it in place, you’d need to edit the bin/hbase file. Look for the addition of the -XX:OnOutOfMemoryError="kill -9 %p" arguments (参考 [HBASE-4769] - ‘Abort RegionServer Immediately on OOME’)

3.3.13. HFile V2 和 “更大, 更少” 趋势

0.92.0 stores data in a new format, Appendix E, *HFile format version 2*. As HBase runs, it will move all your data from HFile v1 to HFile v2 format. This auto-migration will run in the background as flushes and compactions run. HFile V2 allows HBase run with larger regions/files. In fact, we encourage that all HBasers going forward tend toward Facebook axiom #1, run with larger, fewer regions. If you have lots of regions now -- more than 100s per host -- you should look into setting your region size up after you move to 0.92.0 (In 0.92.0, default size is not 1G, up from 256M), and then running online merge tool (参考 “HBASE-1621 merge tool should work on online cluster, but disabled table”).

3.4. 从HBase 0.20.x or 0.89.x 升级到 HBase 0.90.x

0.90.x 版本的HBase可以在 HBase 0.20.x 或者 HBase 0.89.x的数据上启动. 不需要转换数据文件， HBase 0.89.x 和 0.90.x 的region目录名是不一样的 -- 老版本用md5 hash 而不是jenkins hash 来命名region-- 这就意味着，一旦启动，再也不能回退到 HBase 0.20.x.

在升级的时候，一定要将`hbase-default.xml` 从你的 `conf`目录删掉。 0.20.x 版本的配置对于 0.90.x HBase不是最佳的. `hbase-default.xml` 现在已经被打包在 HBase jar 里面了. 如果你想看看这个文件内容，你可以在src目录下 `src/main/resources/hbase-default.xml` 或者在 Section 2.31.1, “HBase 默认配置”看到.

最后，如果从0.20.x升级，需要在shell里检查 `.META.` schema . 过去，我们推荐用户使用16KB的 `MEMSTORE_FLUSHSIZE`. 在shell中运行 `hbase> scan '-ROOT-'`. 会显示当前的`.META.` schema. 检查 `MEMSTORE_FLUSHSIZE` 的大小. 看看是不是 16KB (16384)? 如果是的话，你需要修改它(默认的值是 64MB (67108864)) 运行脚本 `bin/set_meta_memstore_size.rb`. 这个脚本会修改 `.META.` schema. 如果不运行的话，集群会比较慢[15] .

[15] 参考 HBASE-3499 Users upgrading to 0.90.0 need to have their .META. table updated with the right MEMSTORE_SIZE

Chapter 4. HBase Shell

Table of Contents

HBase Shell is 在(J)Ruby的IRB的基础上加上了HBase的命令。任何你可以在IRB里做的事情都可在在HBase Shell中做。

你可以这样来运行HBase Shell:

```
$ ./bin/hbase shell
```

输入 **help** 就会返回Shell的命令列表和选项。可以看看在Help文档尾部的关于如何输入变量和选项。尤其要注意的是表名，行，列名必须要加引号。

参见 Section 1.2.3, “Shell 练习”可以看到Shell的基本使用例子。

4.1. 使用脚本

如果要使用脚本，可以看HBase的`bin` 目录.在里面找到后缀为 `*.rb`的脚本.要想运行这个脚本，要这样

```
$ ./bin/hbase org.jruby.Main PATH_TO_SCRIPT
```

就可以了

4.2. Shell 技巧

4.2.1. `irbrc`

可以在你自己的Home目录下创建一个`.irbrc`文件. 在这个文件里加入自定义的命令。有一个有用的命令就是记录命令历史，这样你就可以把你的命令保存起来。

```
                        $ more .irbrc
                        require 'irb/ext/save-history'
                        IRB.conf[:SAVE_HISTORY] = 100
                        IRB.conf[:HISTORY_FILE] = "#{ENV['HOME']}/.irb-save-history"
```

可以参见 ruby 关于 `.irbrc` 的文档来学习更多的关于IRB的配置方法。

4.2.2. LOG 时间转换

可以将日期'08/08/16 20:56:29'从hbase log 转换成一个 timestamp, 操作如下:

```
                    hbase(main):021:0> import java.text.SimpleDateFormat
                    hbase(main):022:0> import java.text.ParsePosition
                    hbase(main):023:0> SimpleDateFormat.new("yy/MM/dd HH:mm:ss").parse("08/08/16 20:56:29", ParsePosition.new(0)
```

也可以逆过来操作。

```
                    hbase(main):021:0> import java.util.Date
                    hbase(main):022:0> Date.new(1218920189000).toString() => "Sat Aug 16 20:56:29 UTC 2008"
```

要想把日期格式和HBase log格式完全相同，可以参见文档 SimpleDateFormat.

4.2.3. 调试

4.2.3.1. Shell 切换成debug 模式

你可以将shell切换成debug模式。这样可以看到更多的信息。 -- 例如可以看到命令异常的stack trace:

```
hbase> debug <RETURN>
```

4.2.3.2. DEBUG log level

想要在shell中看到 DEBUG 级别的 logging ，可以在启动的时候加上 **-d** 参数.

```
$ ./bin/hbase shell -d
```

Chapter 5. 数据模型

Table of Contents

简单来说，应用程序是以表的方式在HBase存储数据的。表是由行和列构成的，所有的列是从属于某一个列族的。行和列的交叉点称之为cell,cell是版本化的。cell的内容是不可分割的字节数组。

表的行键也是一段字节数组，所以任何东西都可以保存进去，不论是字符串或者数字。HBase的表是按key排序的，排序方式之针对字节的。所有的表都必须要有主键-key.

5.1. 概念视图

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`, `anchor:my.look.ca`)，`contents`仅有一列(`contents:html`)

列名

一个列名是由它的列族前缀和*修饰符(qualifier)*连接而成。例如列*contents:html*是列族 `contents`加冒号(`:`)加 *修饰符* `html`组成的。

Table 5.1. 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>..."	

5.2. 物理视图

尽管在概念视图里，表可以被看成是一个稀疏的行的集合。但在物理上，它的是区分列族 存储的。新的columns可以不经过声明直接加入一个列族.

Table 5.2. 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"

Table 5.3. 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概念视图中空白cell在物理上是不存储的，因为根本没有必要存储。因此若一个请求为要获取`t8`时间的`contents:html`，他的结果就是空。相似的，若请求为获取`t9`时间的`anchor:my.look.ca`，结果也是空。但是，如果不指明时间，将会返回最新时间的行，每个最新的都会返回。例如，如果请求为获取行键为"com.cnn.www"，没有指明时间戳的话，活动的结果是`t6`下的contents:html，`t9`下的`anchor:cnnsi.com`和`t8`下`anchor:my.look.ca`。

For more information about the internals of how HBase stores data, see Section 9.7, “Regions”.

5.3. 表

表是在schema声明的时候定义的。

5.4. 行

行键是不可分割的字节数组。行是按字典排序由低到高存储在表中的。一个空的数组是用来标识表空间的起始或者结尾。

5.5. 列族

在HBase是*列族*一些列的集合。一个列族所有列成员是有着相同的前缀。比如，列*courses:history* 和 *courses:math*都是 列族 *courses* 的成员.冒号(:)是列族的分隔符，用来区分前缀和列名。column 前缀必须是可打印的字符，剩下的部分(称为qualify),可以又任意字节数组组成。列族必须在表建立的时候声明。column就不需要了，随时可以新建。

在物理上，一个的列族成员在文件系统上都是存储在一起。因为存储优化都是针对列族级别的，这就意味着，一个colimn family的所有成员的是用相同的方式访问的。

5.6. Cells

A *{row, column, version}* 元组就是一个HBase中的一个 `cell`。Cell的内容是不可分割的字节数组。

5.7. 数据模型操作

四个主要的数据模型操作是 Get, Put, Scan, 和 Delete. 通过 HTable 实例进行操作.

5.7.1. Get

Get 返回特定行的属性。 Gets 通过 HTable.get 执行。

5.7.2. Put

Put 要么向表增加新行 (如果key是新的) 或更新行 (如果key已经存在)。 Puts 通过 HTable.put (writeBuffer) 或 HTable.batch (non-writeBuffer)执行。

5.7.3. Scans

Scan 允许多行特定属性迭代。

下面是一个在 HTable 表实例上的示例。 假设表有几行键值为 "row1", "row2", "row3", 还有一些行有键值 "abc1", "abc2", 和 "abc3". 下面的示例展示startRow 和 stopRow 可以应用到一个Scan 实例，以返回"row"打头的行。

```
HTable htable = ...      // instantiate HTable

Scan scan = new Scan();
scan.addColumn(Bytes.toBytes("cf"),Bytes.toBytes("attr"));
scan.setStartRow( Bytes.toBytes("row"));                   // start key is inclusive
scan.setStopRow( Bytes.toBytes("row" +  (char)0));  // stop key is exclusive
ResultScanner rs = htable.getScanner(scan);
try {
  for (Result r = rs.next(); r != null; r = rs.next()) {
  // process result...
} finally {
  rs.close();  // always close the ResultScanner!
}
```

5.7.4. Delete

Delete 从表中删除一行. 删除通过HTable.delete 执行。

HBase 没有修改数据的合适方法。所以通过创建名为墓碑(*tombstones*)的新标志进行处理。这些墓碑和死去的值，在主紧缩时清除。

参考 Section 5.8.1.5, “Delete” 获取删除列版本的更多信息。参考Section 9.7.5.5, “Compaction” 获取更多有关紧缩的信息。

5.8. 版本

一个 *{row, column, version}* 元组是HBase中的一个单元(`cell`).但是有可能会有很多的单元的行和列是相同的，可以使用版本来区分不同的单元.

rows和column key是用字节数组表示的，version则是用一个长整型表示。这个long的值使用 `java.util.Date.getTime()` 或者 `System.currentTimeMillis()`产生的。这就意味着他的含义是“当前时间和1970-01-01 UTC的时间差，单位毫秒。”

在HBase中，版本是按倒序排列的，因此当读取这个文件的时候，最先找到的是最近的版本。

有些人不是很理解HBase单元(`cell`)的意思。一个常见的问题是:

- 如果有多个包含版本写操作同时发起，HBase会保存全部还是会保持最新的一个？ [16]
- 可以发起包含版本的写操作，但是他们的版本顺序和操作顺序相反吗?[17]

下面我们介绍下在HBase中版本是如何工作的。[18].

5.8.1. HBase的操作(包含版本操作)

在这一章我们来仔细看看在HBase的各个主要操作中版本起到了什么作用。

5.8.1.1. Get/Scan

Gets实在Scan的基础上实现的。可以详细参见下面的讨论 Get 同样可以用 Scan来描述.

默认情况下，如果你没有指定版本，当你使用`Get`操作的时候，会返回最近版本的Cell(该Cell可能是最新写入的，但不能保证)。默认的操作可以这样修改:

- 如果想要返回返回两个以上的把版本,参见Get.setMaxVersions()
- 如果想要返回的版本不只是最近的，参见 Get.setTimeRange()

  要向查询的最新版本要小于或等于给定的这个值，这就意味着给定的'最近'的值可以是某一个时间点。可以使用0到你想要的时间来设置，还要把max versions设置为1.

5.8.1.2. 默认 Get 例子

下面的Get操作会只获得最新的一个版本。

```
        Get get = new Get(Bytes.toBytes("row1"));
        Result r = htable.get(get);
        byte[] b = r.getValue(Bytes.toBytes("cf"), Bytes.toBytes("attr"));  // returns current version of value
```

5.8.1.3. 含有的版本的Get例子

下面的Get操作会获得最近的3个版本。

```
        Get get = new Get(Bytes.toBytes("row1"));
        get.setMaxVersions(3);  // will return last 3 versions of row
        Result r = htable.get(get);
        byte[] b = r.getValue(Bytes.toBytes("cf"), Bytes.toBytes("attr"));  // returns current version of value
        List<KeyValue> kv = r.getColumn(Bytes.toBytes("cf"), Bytes.toBytes("attr"));  // returns all versions of this column
```

5.8.1.4. Put

一个Put操作会给一个`cell`,创建一个版本，默认使用当前时间戳，当然你也可以自己设置时间戳。这就意味着你可以把时间设置在过去或者未来，或者随意使用一个Long值。

要想覆盖一个现有的值，就意味着你的row,column和版本必须完全相等。

5.8.1.4.1. 不指明版本的例子

下面的Put操作不指明版本，所以HBase会用当前时间作为版本。

```
          Put put = new Put(Bytes.toBytes(row));
          put.add(Bytes.toBytes("cf"), Bytes.toBytes("attr1"), Bytes.toBytes( data));
          htable.put(put);
```

5.8.1.4.2. 指明版本的例子

下面的Put操作，指明了版本。

```
          Put put = new Put( Bytes.toBytes(row ));
          long explicitTimeInMs = 555;  // just an example
          put.add(Bytes.toBytes("cf"), Bytes.toBytes("attr1"), explicitTimeInMs, Bytes.toBytes(data));
          htable.put(put);
```

5.8.1.5. Delete

有三种不同类型的内部删除标记 [19]:

- Delete: 删除列的指定版本.
- Delete column: 删除列的所有版本.
- Delete family: 删除特定列族所有列

当删除一行，HBase将内部对每个列族创建墓碑(非每个单独列)。

删除操作的实现是创建一个*墓碑标记*。例如，我们想要删除一个版本，或者默认是`currentTimeMillis`。就意味着"删除比这个版本更早的所有版本".HBase不会去改那些数据，数据不会立即从文件中删除。他使用删除标记来屏蔽掉这些值。[20]若你知道的版本比数据中的版本晚，就意味着这一行中的所有数据都会被删除。

参考 Section 9.7.5.4, "KeyValue" 获取内部 KeyValue 格式更多信息。

5.8.2. 现有的限制

关于版本还有一些bug(或者称之为未实现的功能)，计划在下个版本实现。

5.8.2.1. 删除标记误标新Put 的数据

删除标记操作可能会标记其后put的数据。[21]记住，当写下一个墓碑标记后，只有下一个主紧缩操作发起之后，墓碑才会清除。假设你删除所有<= 时间T的数据。但之后，你又执行了一个Put操作，时间戳<= T。就算这个Put发生在删除操作之后，他的数据也打上了墓碑标记。这个Put并不会失败，但你做Get操作时，会注意到Put没有产生影响。只有一个主紧缩执行后，一切才会恢复正常。如果你的Put操作一直使用升序的版本，这个问题不会有影响。但是即使你不关心时间，也可能出现该情况。只需删除和插入迅速相互跟随，就有机会在同一毫秒中遇到。

5.8.2.2. 主紧缩改变查询的结果

"设想一下，你一个cell有三个版本t1,t2和t3。你的maximun-version设置是2.当你请求获取全部版本的时候，只会返回两个，t2和t3。如果你将t2和t3删除，就会返回t1。但是如果在删除之前，发生了major compaction操作，那么什么值都不好返回了。[22]"

5.9. 排序

所有数据模型操作 HBase 返回排序的数据。先是行，再是列族，然后是列修饰(column qualifier), 最后是时间戳(反向排序,所以最新的在前).

5.10. 列的元数据

对列族，没有内部的KeyValue之外的元数据保存。这样，HBase不仅在一行中支持很多列，而且支持行之间不同的列。 由你自己负责跟踪列名。

唯一获取列族的完整列名的方法是处理所有行。HBase内部保存数据更多信息，请参考 Section 9.7.5.4, “KeyValue”.

5.11. 联合查询(Join)

HBase是否支持联合是一个网上常问问题。简单来说 : 不支持。至少不想传统RDBMS那样支持(如 SQL中带 equi-joins 或 outer-joins). 正如本章描述的，读数据模型是 Get 和 Scan.

但并不表示等价联合不能在应用程序中支持，只是必须自己做。 两种方法，要么指示要写到HBase的数据，要么查询表并在应用或MapReduce代码中做联合(如 RDBMS所展示,有几种步骤来实现，依赖于表的大小。如 nested loops vs. hash-joins). 哪个更好？ 依赖于你准备做什么，所以没有一个单一的回答适合所有方面。

5.12. ACID

参考 ACID Semantics. Lars Hofhansl 也在 ACID in HBase 上写了说明.

[16] 目前，只有最新的那个是可以获取到的。.

[17] 可以

[18] 参考 HBASE-2406 for discussion of HBase versions. Bending time in HBase makes for a good read on the version, or time, dimension in HBase. It has more detail on versioning than is provided here. As of this writing, the limiitation *Overwriting values at existing timestamps* mentioned in the article no longer holds in HBase. This section is basically a synopsis of this article by Bruno Dumon.

[19] 参考 Lars Hofhansl's blog for discussion of his attempt adding another, Scanning in HBase: Prefix Delete Marker

[20] 当HBase执行一次major compaction,标记删除的数据会被实际的删除，删除标记也会被删除。

[21] HBASE-2256

[22] 参考垃圾收集： Bending time in HBase

Chapter 6. HBase 的 Schema 设计

Table of Contents

一份关于各种NSQL数据库的优点和缺点的通用介绍，就是 Ian Varley的博士论文， No Relation: The Mixed Blessings of Non-Relational Databases。 推荐。也可阅读 Section 9.7.5.4, “KeyValue” ， 了解HBase如何内部保存数据。

6.1. 模式(Schema) 创建

可以使用Chapter 4, *HBase Shell* 或Java API的HBaseAdmin来创建和编辑HBase的模式。

表必须禁用以修改列族，如：

```
Configuration config = HBaseConfiguration.create();
HBaseAdmin admin = new HBaseAdmin(conf);
String table = "myTable";

admin.disableTable(table);

HColumnDescriptor cf1 = ...;
admin.addColumn(table, cf1);      // adding new ColumnFamily
HColumnDescriptor cf2 = ...;
admin.modifyColumn(table, cf2);    // modifying existing ColumnFamily

admin.enableTable(table);
```

参考 Section 2.3.4, “Client configuration and dependencies connecting to an HBase cluster” ， 获取更多配置客户端连接的信息。

注意: 0.92.x 支持在线修改模式, 但 0.90.x 需要禁用表。

6.1.1. 模式更新

当表或列族改变时(如 region size, block size), 当下次存在主紧缩及存储文件重写时起作用。

参考 Section 9.7.5, “Store” 获取存储文件的更多信息。

6.2. 列族的数量

现在HBase并不能很好的处理两个或者三个以上的列族，所以尽量让你的列族数量少一些。目前，flush和compaction操作是针对一个Region。所以当一个列族操作大量数据的时候会引发一个flush。那些不相关的列族也有进行flush操作，尽管他们没有操作多少数据。Compaction操作现在是根据一个列族下的全部文件的数量触发的，而不是根据文件大小触发的。当很多的列族在flush和compaction时,会造成很多没用的I/O负载(要想解决这个问题，需要将flush和compaction操作只针对一个列族) 。 更多紧缩信息, 参考 Section 9.7.5.5, “Compaction”.

尽量在你的应用中使用一个列族。只有你的所有查询操作只访问一个列族的时候，可以引入第二个和第三个列族.例如，你有两个列族,但你查询的时候总是访问其中的一个，从来不会两个一起访问。

6.2.1. 列族的基数

一个表存在多列族，注意基数(如, 行数). 如果列族A有100万行，列族B有10亿行，列族A可能被分散到很多很多区(及区服务器)。这导致扫描列族A低效。

6.3. 行键(RowKey)设计

6.3.1. 单调递增行键/时序数据

在Tom White的Hadoop: The Definitive Guide一书中，有一个章节描述了一个值得注意的问题：在一个集群中，一个导入数据的进程一动不动，所有的client都在等待一个region(就是一个节点)，过了一会后，变成了下一个region...如果使用了单调递增或者时序的key就会造成这样的问题。详情可以参见IKai画的漫画monotonically increasing values are bad。使用了顺序的key会将本没有顺序的数据变得有顺序，把负载压在一台机器上。所以要尽量避免时间戳或者(e.g. 1, 2, 3)这样的key。

如果你需要导入时间顺序的文件(如log)到HBase中，可以学习OpenTSDB的做法。他有一个页面来描述他的schema.OpenTSDB的Key的格式是[metric_type][event_timestamp]，乍一看，似乎违背了不将timestamp做key的建议，但是他并没有将timestamp作为key的一个关键位置，有成百上千的metric_type就足够将压力分散到各个region了。

6.3.2. 尽量最小化行和列的大小(为何我的存储文件指示很大？)

在HBase中，值是作为一个单元(Cell)保存在系统的中的，要定位一个单元，需要行，列名和时间戳。通常情况下，如果你的行和列的名字要是太大(甚至比value的大小还要大)的话，你可能会遇到一些有趣的情况。例如Marc Limotte 在 HBASE-3551(推荐!)尾部提到的现象。在HBase的存储文件Section 9.7.5.2, “StoreFile (HFile)”中，有一个索引用来方便值的随机访问，但是访问一个单元的坐标要是太大的话，会占用很大的内存，这个索引会被用尽。所以要想解决，可以设置一个更大的块大小，当然也可以使用更小的列名 。压缩也能得到更大指数。参考话题 a question storefileIndexSize 用户邮件列表.

大部分时候，小的低效不会影响很大。不幸的是，这里会是个问题。无论是列族，属性和行键都会在数据中重复上亿次。参考 Section 9.7.5.4, “KeyValue” 获取更多信息，关于HBase 内部保存数据，了解为什么这很重要。

6.3.2.1. 列族

尽量使列族名小，最好一个字符。(如 "d" 表示 data/default).

参考 Section 9.7.5.4, “KeyValue” 获取更多信息，关于HBase 内部保存数据，了解为什么这很重要。

6.3.2.2. 属性

详细属性名 (如, "myVeryImportantAttribute") 易读，最好还是用短属性名 (e.g., "via") 保存到HBase.

参考 Section 9.7.5.4, “KeyValue” 获取更多信息，关于HBase 内部保存数据，了解为什么这很重要。

6.3.2.3. 行键长度

让行键短到可读即可，这样对获取数据有用(e.g., Get vs. Scan)。 短键对访问数据无用，并不比长键对get/scan更好。设计行键需要权衡。

6.3.2.4. 字节模式

long 类型有 8 字节. 8字节内可以保存无符号数字到18,446,744,073,709,551,615. 如果用字符串保存--假设一个字节一个字符--，需要将近3倍的字节数。

不信? 下面是示例代码，可以自己运行一下。

```
// long
//
long l = 1234567890L;
byte[] lb = Bytes.toBytes(l);
System.out.println("long bytes length: " + lb.length);   // returns 8

String s = "" + l;
byte[] sb = Bytes.toBytes(s);
System.out.println("long as string length: " + sb.length);    // returns 10

// hash
//
MessageDigest md = MessageDigest.getInstance("MD5");
byte[] digest = md.digest(Bytes.toBytes(s));
System.out.println("md5 digest bytes length: " + digest.length);    // returns 16
```

```
String sDigest = new String(digest);
byte[] sbDigest = Bytes.toBytes(sDigest);
System.out.println("md5 digest as string length: " + sbDigest.length);    // returns 26(译者注：实测值为22)
```

6.3.3. 倒序时间戳

一个数据库处理的通常问题是找到最近版本的值。采用倒序时间戳作为键的一部分可以对此特定情况有很大帮助。也在Tom White的Hadoop书籍的HBase 章节能找到: The Definitive Guide (O'Reilly), 该技术包含追加(Long.MAX_VALUE - timestamp) 到key的后面，如 [key][reverse_timestamp].

表内[key]的最近的值可以用[key]进行 Scan 找到并获取第一个记录。由于 HBase 行键是排序的，该键排在任何比它老的行键的前面，所以必然是第一个。

该技术可以用于代替Section 6.4, “ 版本的数量 ” ，其目的是保存所有版本到“永远”(或一段很长时间) 。同时，采用同样的Scan技术，可以很快获取其他版本。

6.3.4. 行键和列族

行键在列族范围内。所以同样的行键可以在同一个表的每个列族中存在而不会冲突。

6.3.5. 行键永远不变

行键不能改变。唯一可以“改变”的方式是删除然后再插入。这是一个网上常问问题，所以要注意开始就要让行键正确(且/或在插入很多数据之前)。

6.4. 版本数量

6.4.1. 最大版本数

行的版本的数量是HColumnDescriptor设置的，每个列族可以单独设置，默认是3。这个设置是很重要的，在Chapter 5, *数据模型*有描述，因为HBase是*不会去覆盖一个值的*，他只会在后面在追加写，用时间戳来区分、过早的版本会在执行主紧缩的时候删除。这个版本的值可以根据具体的应用增加减少。

不推荐将版本最大值设到一个很高的水平 (如, 成百或更多)，除非老数据对你很重要。因为这会导致存储文件变得极大。

6.4.2. 最小版本数

和行的最大版本数一样，最小版本数也是通过HColumnDescriptor 在每个列族中设置的。最小版本数缺省值是0，表示该特性禁用。 最小版本数参数和存活时间一起使用，允许配置如“保存最后T秒有价值数据，最多N个版本，但最少约M个版本”(M是最小版本数，M<N)。 该参数仅在存活时间对列族启用，且必须小于行版本数。

6.5. 支持数据类型

HBase 通过 Put 和 Result支持 "bytes-in/bytes-out" 接口，所以任何可被转为字节数组的东西可以作为值存入。输入可以是字符串，数字，复杂对象，甚至图像，只要他们能转为字节。

存在值的实际长度限制 (如 保存 10-50MB 对象到 HBase 可能对查询来说太长); 搜索邮件列表获取本话题的对话。 HBase的所有行都遵循 Chapter 5, *数据模型*, 包括版本化。 设计时需考虑到这些，以及列族的块大小。

6.5.1. 计数器

一种支持的数据类型，值得一提的是“计数器”(如, 具有原子递增能力的数值)。参考 HTable的 Increment .

同步计数器在区域服务器中完成，不是客户端。

6.6. 联合

如果有多个表，不要在模式设计中忘了 Section 5.11, “Joins” 的潜在因素。

6.7. 存活时间 (TTL)

列族可以设置TTL秒数，HBase 在超时后将自动删除数据。影响 *全部* 行的全部版本 - 甚至当前版本。HBase里面TTL 时间时区是UTC.

参考 HColumnDescriptor 获取更多信息。

6.8. 保留删除的单元

列族允许是否保留单元。这就是说 Get 或 Scan 操作仍可以获取删除的单元。由于这些操作指定时间范围，结束在删除单元发生效果之前。这甚至允许在删除进行时进行即时查询。

删除的单元仍然受TTL控制，并永远不会超过“最大版本数”被删除的单元。新 "raw" scan 选项返回所有已删除的行和删除标志。

参考 HColumnDescriptor 获取更多信息

6.9. 第二索引和改变路径查询

本节标题也可以为"如果表的行键像*这样*，但我又想像*那样*查询该表." A common example on the dist-list is where a row-key is of the format "user-timestamp" but there are are reporting requirements on activity across users for certain time ranges. Thus, selecting by user is easy because it is in the lead position of the key, but time is not.

There is no single answer on the best way to handle this because it depends on...

- Number of users
- Data size and data arrival rate
- Flexibility of reporting requirements (e.g., completely ad-hoc date selection vs. pre-configured ranges)
- Desired execution speed of query (e.g., 90 seconds may be reasonable to some for an ad-hoc report, whereas it may be too long for others)

... and solutions are also influenced by the size of the cluster and how much processing power you have to throw at the solution. Common techniques are in sub-sections below. This is a comprehensive, but not exhaustive, list of approaches.

It should not be a surprise that secondary indexes require additional cluster space and processing. This is precisely what happens in an RDBMS because the act of creating an alternate index requires both space and processing cycles to update. RBDMS products are more advanced in this regard to handle alternative index management out of the box. However, HBase scales better at larger data volumes, so this is a feature trade-off.

Pay attention to Chapter 11, *Performance Tuning* when implementing any of these approaches.

Additionally, see the David Butler response in this dist-list thread HBase, mail # user - Stargate+hbase

6.9.1. 过滤查询

根据具体应用，可能和 Section 9.4, “Client Request Filters” 用法相当。在这种情况下，没有第二索引被创建。然而，不要像这样从应用 (如单线程客户端)中对大表尝试全表扫描。

6.9.2. 定期更新第二索引

第二索引可以在另一个表中创建，并通过MapReduce任务定期更新。任务可以在当天执行，但依赖于加载策略，可能会同主表失去同步。

参考 Section 7.2.2, “HBase MapReduce Read/Write Example” 获取更多信息.

6.9.3. 双写第二索引

另一个策略是在将数据写到集群的同时创建第二索引(如：写到数据表，同时写到索引表)。如果该方法在数据表存在之后采用，则需要利用MapReduce任务来生成已有数据的第二索引。 (参考 Section 6.9.2, “ Periodic-Update Secondary Index ”).

6.9.4. 汇总表(Summary Tables)

对时间跨度长 (e.g., 年报) 和数据量巨大，汇总表是通用路径。可通过MapReduce任务生成到另一个表。

参考 Section 7.2.4, “HBase MapReduce Summary to HBase Example” 获取更多信息。

6.9.5. 协处理第二索引

协处理动作像 RDBMS 触发器。 这在 0.92中添加. 更多参考 Section 9.6.3, “Coprocessors”

6.10. 限制

HBase currently supports 'constraints' in traditional (SQL) database parlance. The advised usage for Constraints is in enforcing business rules for attributes in the table (eg. make sure values are in the range 1-10). Constraints could also be used to enforce referential integrity, but this is strongly discouraged as it will dramatically decrease the write throughput of the tables where integrity checking is enabled. Extensive documentation on using Constraints can be found at: Constraint since version 0.94.

6.11. 模式(schema)设计用例

This effectively is the OpenTSDB approach. What OpenTSDB does is re-write data and pack rows into columns for certain time-periods. For a detailed explanation, see:http://opentsdb.net/schema.html, and Lessons Learned from OpenTSDB from HBaseCon2012.

But this is how the general concept works: data is ingested, for example, in this manner…

```
[hostname][log-event][timestamp1]  [hostname][log-event][timestamp2]  [hostname][log-event][timestamp3]
```

… with separate rowkeys for each detailed event, but is re-written like this…

[hostname][log-event][timerange]

… and each of the above events are converted into columns stored with a time-offset relative to the beginning timerange (e.g., every 5 minutes). This is obviously a very advanced processing technique, but HBase makes this possible.

6.11.3. Case Study - Customer/Order

Assume that HBase is used to store customer and order information. There are two core record-types being ingested: a Customer record type, and Order record type.

The Customer record type would include all the things that you’d typically expect:

- Customer number
- Customer name
- Address (e.g., city, state, zip)
- Phone numbers, etc.

The Order record type would include things like:

- Customer number
- Order number
- Sales date
- A series of nested objects for shipping locations and line-items (see Section 6.11.3.2, “Order Object Design” for details)

Assuming that the combination of customer number and sales order uniquely identify an order, these two attributes will compose the rowkey, and specifically a composite key such as:

[customer number][order number]

… for a ORDER table. However, there are more design decisions to make: are the *raw* values the best choices for rowkeys?

The same design questions in the Log Data use-case confront us here. What is the keyspace of the customer number, and what is the format (e.g., numeric? alphanumeric?) As it is advantageous to use fixed-length keys in HBase, as well as keys that can support a reasonable spread in the keyspace, similar options appear:

Composite Rowkey With Hashes:

- [MD5 of customer number] = 16 bytes
- [MD5 of order number] = 16 bytes

Composite Numeric/Hash Combo Rowkey:

- [substituted long for customer number] = 8 bytes
- [MD5 of order number] = 16 bytes

6.11.3.1. Single Table? Multiple Tables?

A traditional design approach would have separate tables for CUSTOMER and SALES. Another option is to pack multiple record types into a single table (e.g., CUSTOMER++).

Customer Record Type Rowkey:

- [customer-id]
- [type] = type indicating ‘1’ for customer record type

Order Record Type Rowkey:

- [customer-id]
- [type] = type indicating ‘2’ for order record type
- [order]

The advantage of this particular CUSTOMER++ approach is that organizes many different record-types by customer-id (e.g., a single scan could get you everything about that customer). The disadvantage is that it’s not as easy to scan for a particular record-type.

6.11.3.2. Order Object Design

Now we need to address how to model the Order object. Assume that the class structure is as follows:

```
Order      ShippingLocation    (an Order can have multiple ShippingLocations)          LineItem             (a ShippingLoca
```

... there are multiple options on storing this data.

6.11.3.2.1. Completely Normalized

With this approach, there would be separate tables for ORDER, SHIPPING_LOCATION, and LINE_ITEM.

The ORDER table's rowkey was described above: Section 6.11.3, “Case Study - Customer/Order”

The SHIPPING_LOCATION's composite rowkey would be something like this:

- [order-rowkey]
- [shipping location number] (e.g., 1st location, 2nd, etc.)

The LINE_ITEM table's composite rowkey would be something like this:

- [order-rowkey]
- [shipping location number] (e.g., 1st location, 2nd, etc.)
- [line item number] (e.g., 1st lineitem, 2nd, etc.)

Such a normalized model is likely to be the approach with an RDBMS, but that's not your only option with HBase. The cons of such an approach is that to retrieve information about any Order, you will need:

- Get on the ORDER table for the Order
- Scan on the SHIPPING_LOCATION table for that order to get the ShippingLocation instances
- Scan on the LINE_ITEM for each ShippingLocation

... granted, this is what an RDBMS would do under the covers anyway, but since there are no joins in HBase you're just more aware of this fact.

6.11.3.2.2. Single Table With Record Types

With this approach, there would exist a single table ORDER that would contain

The Order rowkey was described above: Section 6.11.3, “Case Study - Customer/Order”

- [order-rowkey]
- [ORDER record type]

The ShippingLocation composite rowkey would be something like this:

- [order-rowkey]
- [SHIPPING record type]
- [shipping location number] (e.g., 1st location, 2nd, etc.)

The LineItem composite rowkey would be something like this:

- [order-rowkey]
- [LINE record type]
- [shipping location number] (e.g., 1st location, 2nd, etc.)
- [line item number] (e.g., 1st lineitem, 2nd, etc.)

6.11.3.2.3. Denormalized

A variant of the Single Table With Record Types approach is to denormalize and flatten some of the object hierarchy, such as collapsing the ShippingLocation attributes onto each LineItem instance.

The LineItem composite rowkey would be something like this:

- [order-rowkey]
- [LINE record type]
- [line item number] (e.g., 1st lineitem, 2nd, etc. - care must be taken that there are unique across the entire order)

... and the LineItem columns would be something like this:

- itemNumber
- quantity
- price
- shipToLine1 (denormalized from ShippingLocation)
- shipToLine2 (denormalized from ShippingLocation)
- shipToCity (denormalized from ShippingLocation)
- shipToState (denormalized from ShippingLocation)
- shipToZip (denormalized from ShippingLocation)

The pros of this approach include a less complex object heirarchy, but one of the cons is that updating gets more complicated in case any of this information changes.

6.11.3.2.4. Object BLOB

With this approach, the entire Order object graph is treated, in one way or another, as a BLOB. For example, the ORDER table's rowkey was described above: Section 6.11.3, “Case Study - Customer/Order”, and a single column called "order" would contain an object that could be deserialized that contained a container Order, ShippingLocations, and LineItems.

There are many options here: JSON, XML, Java Serialization, Avro, Hadoop Writables, etc. All of them are variants of the same approach: encode the object graph to a byte-array. Care should be taken with this approach to ensure backward compatibilty in case the object model changes such that older persisted structures can still be read back out of HBase.

Pros are being able to manage complex object graphs with minimal I/O (e.g., a single HBase Get per Order in this example), but the cons include the aforementioned warning about backward compatiblity of serialization, language dependencies of serialization (e.g., Java Serialization only works with Java clients), the fact that you have to deserialize the entire object to get any piece of information inside the BLOB, and the difficulty in getting frameworks like Hive to work with custom objects like this.

6.11.4. Case Study - "Tall/Wide/Middle" Schema Design Smackdown

This section will describe additional schema design questions that appear on the dist-list, specifically about tall and wide tables. These are general guidelines and not laws - each application must consider its own needs.

6.11.4.1. Rows vs. Versions

A common question is whether one should prefer rows or HBase's built-in-versioning. The context is typically where there are "a lot" of versions of a row to be retained (e.g., where it is significantly above the HBase default of 3 max versions). The rows-approach would require storing a timstamp in some portion of the rowkey so that they would not overwite with each successive update.

Preference: Rows (generally speaking).

6.11.4.2. Rows vs. Columns

Another common question is whether one should prefer rows or columns. The context is typically in extreme cases of wide tables, such as having 1 row with 1 million attributes, or 1 million rows with 1 columns apiece.

Preference: Rows (generally speaking). To be clear, this guideline is in the context is in extremely wide cases, not in the standard use-case where one needs to store a few dozen or hundred columns. But there is also a middle path between these two options, and that is "Rows as Columns."

6.11.4.3. Rows as Columns

The middle path between Rows vs. Columns is packing data that would be a separate row into columns, for certain rows. OpenTSDB is the best example of this case where a single row represents a defined time-range, and then discrete events are treated as columns. This approach is often more complex, and may require the additional complexity of re-writing your data, but has the advantage of being I/O efficient. For an overview of this approach, see ???.

6.11.5. Case Study - List Data

The following is an exchange from the user dist-list regarding a fairly common question: how to handle per-user list data in Apache HBase.

*** QUESTION ***

We're looking at how to store a large amount of (per-user) list data in HBase, and we were trying to figure out what kind of access pattern made the most sense. One option is store the majority of the data in a key, so we could have something like:

```
<FixedWidthUserName><FixedWidthValueId1>:"" (no value)  <FixedWidthUserName><FixedWidthValueId2>:"" (no value)  <FixedWidthUserNa
```

The other option we had was to do this entirely using:

```
<FixedWidthUserName><FixedWidthPageNum0>:<FixedWidthLength><FixedIdNextPageNum><ValueId1><ValueId2><ValueId3>...  <FixedWidthUser
```

where each row would contain multiple values. So in one case reading the first thirty values would be:

```
scan { STARTROW => 'FixedWidthUsername' LIMIT => 30}
```

And in the second case it would be

```
get 'FixedWidthUserName\x00\x00\x00\x00'
```

The general usage pattern would be to read only the first 30 values of these lists, with infrequent access reading deeper into the lists. Some users would have <= 30 total values in these lists, and some users would have millions (i.e. power-law distribution)

The single-value format seems like it would take up more space on HBase, but would offer some improved retrieval / pagination flexibility. Would there be any significant performance advantages to be able to paginate via gets vs paginating with scans?

My initial understanding was that doing a scan should be faster if our paging size is unknown (and caching is set appropriately), but that gets should be faster if we'll always need the same page size. I've ended up hearing different people tell me opposite things about performance. I assume the page sizes would be relatively consistent, so for most use cases we could guarantee that we only wanted one page of data in the fixed-page-length case. I would also assume that we would have infrequent updates, but may have inserts into the middle of these lists (meaning we'd need to update all subsequent rows).

Thanks for help / suggestions / follow-up questions.

*** ANSWER ***

If I understand you correctly, you're ultimately trying to store triples in the form "user, valueid, value", right? E.g., something like:

```
"user123, firstname, Paul",  "user234, lastname, Smith"
```

(But the usernames are fixed width, and the valueids are fixed width).

And, your access pattern is along the lines of: "for user X, list the next 30 values, starting with valueid Y". Is that right? And these values should be returned sorted by valueid?

The tl;dr version is that you should probably go with one row per user+value, and not build a complicated intra-row pagination scheme on your own unless you're really sure it is needed.

Your two options mirror a common question people have when designing HBase schemas: should I go "tall" or "wide"? Your first schema is "tall": each row represents one value for one user, and so there are many rows in the table for each user; the row key is user + valueid, and there would be (presumably) a single column qualifier that means "the value". This is great if you want to scan over rows in sorted order by row key (thus my question above, about whether these ids are sorted correctly). You can start a scan at any user+valueid, read the next 30, and be done. What you're giving up is the ability to have transactional guarantees around all the rows for one user, but it doesn't sound like you need that. Doing it this way is generally recommended (see here #schema.smackdown).

Your second option is "wide": you store a bunch of values in one row, using different qualifiers (where the qualifier is the valueid). The simple way to do that would be to just store ALL values for one user in a single row. I'm guessing you jumped to the "paginated" version because you're assuming that storing millions of columns in a single row would be bad for performance, which may or may not be true; as long as you're not trying to do too much in a single request, or do things like scanning over and returning all of the cells in the row, it shouldn't be fundamentally worse. The client has methods that allow you to get specific slices of columns.

Note that neither case fundamentally uses more disk space than the other; you're just "shifting" part of the identifying information for a value either to the left (into the row key, in option one) or to the right (into the column qualifiers in option 2). Under the covers, every key/value still stores the whole row key, and column family name. (If this is a bit confusing, take an hour and watch Lars George's excellent video about understanding HBase schema design: http://www.youtube.com/watch?v=_HLoH_PgrLk).

A manually paginated version has lots more complexities, as you note, like having to keep track of how many things are in each page, re-shuffling if new values are inserted, etc. That seems significantly more complex. It might have some slight speed advantages (or disadvantages!) at extremely high throughput, and the only way to really know that would be to try it out. If you don't have time to build it both ways and compare, my advice would be to start with the simplest option (one row per user+value). Start simple and iterate! :)

6.12. 业务和性能配置选项

参考 the Performance section Section 11.6, “Schema Design” for more information operational and performance schema design options, such as Bloom Filters, Table-configured regionsizes, compression, and blocksizes.

Chapter 7. HBase 和 MapReduce

Table of Contents

关于 HBase 和 MapReduce详见 javadocs. 下面是一些附加的帮助文档. MapReduce的更多信息 (如，通用框架), 参考 Hadoop MapReduce Tutorial.

7.1. Map-Task 分割

7.1.1 默认 HBase MapReduce 分割器(Splitter)

当 MapReduce 任务的HBase 表使用TableInputFormat为数据源格式的时候,他的splitter会给这个table的每个region一个map。因此，如果一个table有100个region，就有100个map-tasks，不论需需要scan多少个列族 。

7.1.2. 自定义分割器

iv>

For those interested in implementing custom splitters, see the method getSplits in TableInputFormatBase. That is where the logic for map-task assignment resides.

7.2. HBase MapReduce 例子

7.2.1 HBase MapReduce 读取例子

下面是使用HBase 作为源的MapReduce读取示例。特别是仅有Mapper实例，没有Reducer。Mapper什么也不产生。

如下所示...

```
Configuration config = HBaseConfiguration.create();
Job job = new Job(config, "ExampleRead");
job.setJarByClass(MyReadJob.class);     // class that contains mapper

Scan scan = new Scan();
scan.setCaching(500);        // 1 is the default in Scan, which will be bad for MapReduce jobs
scan.setCacheBlocks(false);  // don't set to true for MR jobs
// set other scan attrs
...

TableMapReduceUtil.initTableMapperJob(
  tableName,        // input HBase table name
  scan,             // Scan instance to control CF and attribute selection
  MyMapper.class,   // mapper
  null,             // mapper output key
  null,             // mapper output value
  job);
job.setOutputFormatClass(NullOutputFormat.class);   // because we aren't emitting anything from mapper

boolean b = job.waitForCompletion(true);
if (!b) {
  throw new IOException("error with job!");
}
```

...mapper需要继承于TableMapper...

```
public class MyMapper extends TableMapper<Text, LongWritable> {
public void map(ImmutableBytesWritable row, Result value, Context context)
throws InterruptedException, IOException {
// process data for the row from the Result instance.
```

7.2.2. HBase MapReduce 读/写 示例

下面是使用HBase 作为源和目标的MapReduce示例. 本示例简单从一个表复制到另一个表。

```
Configuration config = HBaseConfiguration.create();
Job job = new Job(config,"ExampleReadWrite");
job.setJarByClass(MyReadWriteJob.class);    // class that contains mapper

Scan scan = new Scan();
scan.setCaching(500);        // 1 is the default in Scan, which will be bad for MapReduce jobs
scan.setCacheBlocks(false);  // don't set to true for MR jobs
// set other scan attrs

TableMapReduceUtil.initTableMapperJob(
        sourceTable,      // input table
        scan,             // Scan instance to control CF and attribute selection
        MyMapper.class,   // mapper class
        null,             // mapper output key
        null,             // mapper output value
        job);
TableMapReduceUtil.initTableReducerJob(
        targetTable,      // output table
        null,             // reducer class
        job);
job.setNumReduceTasks(0);

boolean b = job.waitForCompletion(true);
if (!b) {
    throw new IOException("error with job!");
}
```

`TableMapReduceUtil`做了什么需要解释, 特别是对 reducer. TableOutputFormat 作为 outputFormat 类, 几个参数在config中设置(e.g., TableOutputFormat.OUTPUT_TABLE), 同时设置reducer output key 到 `ImmutableBytesWritable` 和 reducer value到 `Writable`. 这可以编

程时设置到job和conf，但TableMapReduceUtil 使其变简单.

下面是 mapper示例, 创建一个 Put，匹配输入的 Result 并提交. Note: 这是 CopyTable 工具做的.

```
public static class MyMapper extends TableMapper<ImmutableBytesWritable, Put>  {

        public void map(ImmutableBytesWritable row, Result value, Context context) throws IOException, InterruptedException {
                // this example is just copying the data from the source table...
                context.write(row, resultToPut(row,value));
        }

        private static Put resultToPut(ImmutableBytesWritable key, Result result) throws IOException {
                Put put = new Put(key.get());
                for (KeyValue kv : result.raw()) {
                        put.add(kv);
                }
                return put;
        }
}
```

这不是真正的 reducer 步骤, 所以 TableOutputFormat 处理发送 Put 到目标表.

这仅是示例, 开发者可以选择不使用TableOutputFormat并自己连接到目标表。

7.2.3. HBase MapReduce Read/Write 多表输出示例

TODO: MultiTableOutputFormat 示例.

7.2.4. HBase MapReduce 汇总到 HBase 示例

下面是使用HBase 作为源和目标的MapReduce示例，具有汇总步骤。本示例计算一个表中值的个数，并将汇总的计数输出到另一个表。

```
Configuration config = HBaseConfiguration.create();
Job job = new Job(config,"ExampleSummary");
job.setJarByClass(MySummaryJob.class);     // class that contains mapper and reducer

Scan scan = new Scan();
scan.setCaching(500);        // 1 is the default in Scan, which will be bad for MapReduce jobs
scan.setCacheBlocks(false);  // don't set to true for MR jobs
// set other scan attrs

TableMapReduceUtil.initTableMapperJob(
        sourceTable,        // input table
        scan,               // Scan instance to control CF and attribute selection
        MyMapper.class,     // mapper class
        Text.class,         // mapper output key
        IntWritable.class,  // mapper output value
        job);
TableMapReduceUtil.initTableReducerJob(
        targetTable,        // output table
        MyTableReducer.class,    // reducer class
        job);
job.setNumReduceTasks(1);   // at least one, adjust as required

boolean b = job.waitForCompletion(true);
if (!b) {
        throw new IOException("error with job!");
}
```

本示例mapper，将一个列的一个字符串值作为汇总值。该值作为key在mapper中生成。 IntWritable 代表一个实例计数。

```
public static class MyMapper extends TableMapper<Text, IntWritable>  {

        private final IntWritable ONE = new IntWritable(1);
        private Text text = new Text();

        public void map(ImmutableBytesWritable row, Result value, Context context) throws IOException, InterruptedException {
                String val = new String(value.getValue(Bytes.toBytes("cf"), Bytes.toBytes("attr1")));
                text.set(val);     // we can only emit Writables...

                context.write(text, ONE);
        }
}
```

在 reducer, "ones" 被统计 (和其他 MR 示例一样), 产生一个 Put.

```
public static class MyTableReducer extends TableReducer<Text, IntWritable, ImmutableBytesWritable>  {

        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
                int i = 0;
                for (IntWritable val : values) {
                        i += val.get();
                }
                Put put = new Put(Bytes.toBytes(key.toString()));
                put.add(Bytes.toBytes("cf"), Bytes.toBytes("count"), Bytes.toBytes(i));

                context.write(null, put);
        }
}
```

7.2.5. HBase MapReduce 汇总到文件示例

This very similar to the summary example above, with exception that this is using HBase as a MapReduce source but HDFS as the sink. The differences are in the job setup and in the reducer. The mapper remains the same.

```
Configuration config = HBaseConfiguration.create();
Job job = new Job(config,"ExampleSummaryToFile");
job.setJarByClass(MySummaryFileJob.class);     // class that contains mapper and reducer
```

```
Scan scan = new Scan();
scan.setCaching(500);        // 1 is the default in Scan, which will be bad for MapReduce jobs
scan.setCacheBlocks(false);  // don't set to true for MR jobs
// set other scan attrs

TableMapReduceUtil.initTableMapperJob(
        sourceTable,        // input table
        scan,               // Scan instance to control CF and attribute selection
        MyMapper.class,     // mapper class
        Text.class,         // mapper output key
        IntWritable.class,  // mapper output value
        job);
job.setReducerClass(MyReducer.class);    // reducer class
job.setNumReduceTasks(1);    // at least one, adjust as required
FileOutputFormat.setOutputPath(job, new Path("/tmp/mr/mySummaryFile"));  // adjust directories as required

boolean b = job.waitForCompletion(true);
if (!b) {
        throw new IOException("error with job!");
}
```

As stated above, the previous Mapper can run unchanged with this example. As for the Reducer, it is a "generic" Reducer instead of extending TableMapper and emitting Puts.

```
 public static class MyReducer extends Reducer<Text, IntWritable, Text, IntWritable>  {

        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
                int i = 0;
                for (IntWritable val : values) {
                        i += val.get();
                }
                context.write(key, new IntWritable(i));
        }
}
```

7.2.6. HBase MapReduce 没有Reducer时汇总到 HBase

It is also possible to perform summaries without a reducer - if you use HBase as the reducer.

An HBase target table would need to exist for the job summary. The HTable method `incrementColumnValue` would be used to atomically increment values. From a performance perspective, it might make sense to keep a Map of values with their values to be incremeneted for each map-task, and make one update per key at during the `cleanup` method of the mapper. However, your milage may vary depending on the number of rows to be processed and unique keys.

In the end, the summary results are in HBase.

7.2.7. HBase MapReduce 汇总到 RDBMS

有时更合适产生汇总到 RDBMS.这种情况下,可以将汇总直接通过一个自定义的reducer输出到 RDBMS 。 `setup` 方法可以连接到 RDBMS (连接信息可以通过context的自定义参数传递)， cleanup 可以关闭连接.

关键需要理解job的多个reducer会影响汇总实现，必须在reducer中进行设计。无论是一个recucer还是多个reducer。不管对错, 依赖于你的用例。认识到多个reducer分配到job，需要创建多个并发的RDBMS连接-可以扩充，但仅在一个点。

```
 public static class MyRdbmsReducer extends Reducer<Text, IntWritable, Text, IntWritable>  {

        private Connection c = null;

        public void setup(Context context) {
                // create DB connection...
        }

        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
                // do summarization
                // in this example the keys are Text, but this is just an example
        }

        public void cleanup(Context context) {
                // close db connection
        }

}
```

最后，汇总的结果被写入到 RDBMS 表.

7.3. 在一个MapReduce Job中访问其他的HBase Tables

尽管现有的框架允许一个HBase table作为一个MapReduce job的输入，其他的HBase table可以同时作为普通的表被访问。例如在一个MapReduce的job中，可以在Mapper的setup方法中创建HTable实例。

```
public class MyMapper extends TableMapper<Text, LongWritable> {
  private HTable myOtherTable;

  @Override
  public void setup(Context context) {
    myOtherTable = new HTable("myOtherTable");
  }
```

7.4. 预测执行

通常建议关掉针对HBase的MapReduce job的预测执行(speculative execution)功能。这个功能也可以用每个Job的配置来完成。对于整个集群，使用预测执行意味着双倍的运算量。这可不是你所希望的。

参考 Section 2.8.2.9, “Speculative Execution” 获取更多信息。

Chapter 8. HBase安全

Table of Contents

8.1. 安全客户端访问HBase

新版 HBase (>= 0.92) 支持客户端可选 SASL 认证.

这里描述如何设置HBase 和 HBase 客户端，以安全连接到HBase 资源.

8.1.1. 先决条件

HBase 必须使用 安全Hadoop/HBase的新 maven 配置文件： `-P security`. Secure Hadoop dependent classes are separated under a pseudo-module in the security/ directory and are only included if built with the secure Hadoop profile.

You need to have a working Kerberos KDC.

A HBase configured for secure client access is expected to be running on top of a secured HDFS cluster. HBase must be able to authenticate to HDFS services. HBase needs Kerberos credentials to interact with the Kerberos-enabled HDFS daemons. Authenticating a service should be done using a keytab file. The procedure for creating keytabs for HBase service is the same as for creating keytabs for Hadoop. Those steps are omitted here. Copy the resulting keytab files to wherever HBase Master and RegionServer processes are deployed and make them readable only to the user account under which the HBase daemons will run.

A Kerberos principal has three parts, with the form `username/fully.qualified.domain.name@YOUR-REALM.COM`. We recommend using `hbase` as the username portion.

The following is an example of the configuration properties for Kerberos operation that must be added to the `hbase-site.xml` file on every server machine in the cluster. Required for even the most basic interactions with a secure Hadoop configuration, independent of HBase security.

```
<property>
  <name>hbase.regionserver.kerberos.principal</name>
  <value>hbase/_HOST@YOUR-REALM.COM</value>
</property>
<property>
  <name>hbase.regionserver.keytab.file</name>
  <value>/etc/hbase/conf/keytab.krb5</value>
</property>
<property>
  <name>hbase.master.kerberos.principal</name>
  <value>hbase/_HOST@YOUR-REALM.COM</value>
</property>
<property>
  <name>hbase.master.keytab.file</name>
  <value>/etc/hbase/conf/keytab.krb5</value>
</property>
```

Each HBase client user should also be given a Kerberos principal. This principal should have a password assigned to it (as opposed to a keytab file). The client principal's `maxrenewlife` should be set so that it can be renewed enough times for the HBase client process to complete. For example, if a user runs a long-running HBase client process that takes at most 3 days, we might create this user's principal within `kadmin` with: `addprinc -maxrenewlife 3days`

Long running daemons with indefinite lifetimes that require client access to HBase can instead be configured to log in from a keytab. For each host running such daemons, create a keytab with `kadmin` or `kadmin.local`. The procedure for creating keytabs for HBase service is the same as for creating keytabs for Hadoop. Those steps are omitted here. Copy the resulting keytab files to where the client daemon will execute and make them readable only to the user account under which the daemon will run.

8.1.2. 安全操作的服务器端配置

增加下列内容到 `hbase-site.xml` file on every server machine in the cluster:

```
<property>
  <name>hbase.security.authentication</name>
  <value>kerberos</value>
</property>
<property>
  <name>hbase.security.authorization</name>
  <value>true</value>
</property>
<property>
  <name>hbase.rpc.engine</name>
  <value>org.apache.hadoop.hbase.ipc.SecureRpcEngine</value>
</property>
<property>
<name>hbase.coprocessor.region.classes</name>
  <value>org.apache.hadoop.hbase.security.token.TokenProvider</value>
</property>
```

A full shutdown and restart of HBase service is required when deploying these configuration changes.

8.1.3. 客户端安全操作配置

每个客户端增加下列内容到 `hbase-site.xml`:

```
      <property>
        <name>hbase.security.authentication</name>
        <value>kerberos</value>
      </property>
      <property>
        <name>hbase.rpc.engine</name>
        <value>org.apache.hadoop.hbase.ipc.SecureRpcEngine</value>
      </property>
```

The client environment must be logged in to Kerberos from KDC or keytab via the `kinit` command before communication with the HBase cluster will be possible.

Be advised that if the `hbase.security.authentication` and `hbase.rpc.engine` properties in the client- and server-side site files do not match, the client will not be able to communicate with the cluster.

Once HBase is configured for secure RPC it is possible to optionally configure encrypted communication. To do so, 增加下列内容到 `hbase-site.xml` file on every client:

```
      <property>
        <name>hbase.rpc.protection</name>
        <value>privacy</value>
      </property>
```

This configuration property can also be set on a per connection basis. Set it in the `Configuration` supplied to `HTable`:

```
      Configuration conf = HBaseConfiguration.create();
      conf.set("hbase.rpc.protection", "privacy");
      HTable table = new HTable(conf, tablename);
```

Expect a ~10% performance penalty for encrypted communication.

8.1.4. 客户端安全操作配置 - Thrift 网关

每个Thrift网关增加下列内容到 `hbase-site.xml`:

```
    <property>
      <name>hbase.thrift.keytab.file</name>
      <value>/etc/hbase/conf/hbase.keytab</value>
    </property>
    <property>
      <name>hbase.thrift.kerberos.principal</name>
      <value>$USER/_HOST@HADOOP.LOCALDOMAIN</value>
    </property>
```

Substitute the appropriate credential and keytab for $USER and $KEYTAB respectively.

The Thrift gateway will authenticate with HBase using the supplied credential. No authentication will be performed by the Thrift gateway itself. All client access via the Thrift gateway will use the Thrift gateway's credential and have its privilege.

8.1.5. 客户端安全操作配置 - REST 网关

每个REST网关增加下列内容到 `hbase-site.xml`:

```
    <property>
      <name>hbase.rest.keytab.file</name>
      <value>$KEYTAB</value>
    </property>
    <property>
      <name>hbase.rest.kerberos.principal</name>
      <value>$USER/_HOST@HADOOP.LOCALDOMAIN</value>
    </property>
```

Substitute the appropriate credential and keytab for $USER and $KEYTAB respectively.

The REST gateway will authenticate with HBase using the supplied credential. No authentication will be performed by the REST gateway itself. All client access via the REST gateway will use the REST gateway's credential and have its privilege.

It should be possible for clients to authenticate with the HBase cluster through the REST gateway in a pass-through manner via SPEGNO HTTP authentication. This is future work.

8.2. 访问控制

Newer releases of HBase (>= 0.92) support optional access control list (ACL-) based protection of resources on a column family and/or table basis.

This describes how to set up Secure HBase for access control, with an example of granting and revoking user permission on table resources provided.

8.2.1. 先决条件

You must configure HBase for secure operation. Refer to the section "安全客户端访问HBase" and complete all of the steps described there.

You must also configure ZooKeeper for secure operation. Changes to ACLs are synchronized throughout the cluster using ZooKeeper. Secure authentication to ZooKeeper must be enabled or otherwise it will be possible to subvert HBase access control via direct client access to ZooKeeper. Refer to the section on secure ZooKeeper configuration and complete all of the steps described there.

8.2.2. 概述

With Secure RPC and Access Control enabled, client access to HBase is authenticated and user data is private unless access has been explicitly granted. Access to data can be granted at a table or per column family basis.

However, the following items have been left out of the initial implementation for simplicity:

1. Row-level or per value (cell): This would require broader changes for storing the ACLs inline with rows. It is a future goal.
2. Push down of file ownership to HDFS: HBase is not designed for the case where files may have different permissions than the HBase system principal. Pushing file ownership down into HDFS would necessitate changes to core code. Also, while HDFS file ownership would make applying quotas easy, and possibly make bulk imports more straightforward, it is not clear that it would offer a more secure setup.
3. HBase managed "roles" as collections of permissions: We will not model "roles" internally in HBase to begin with. We instead allow group names to be granted permissions, which allows external modeling of roles via group membership. Groups are created and manipulated externally to HBase, via the Hadoop group mapping service.

Access control mechanisms are mature and fairly standardized in the relational database world. The HBase implementation approximates current convention, but HBase has a simpler feature set than relational databases, especially in terms of client operations. We don't distinguish between an insert (new record) and update (of existing record), for example, as both collapse down into a Put. Accordingly, the important operations condense to four permissions: READ, WRITE, CREATE, and ADMIN.

Operation To Permission
MappingPermissionOperationReadGetExistsScanWritePutDeleteLock/UnlockRowIncrementColumnValueCheckAndDelete/PutFlushCompactCreateCreate
CompactGrantRevokeShutdown

Permissions can be granted in any of the following scopes, though CREATE and ADMIN permissions are effective only at table scope.

- Table
  - Read: User can read from any column family in table
  - Write: User can write to any column family in table
  - Create: User can alter table attributes; add, alter, or drop column families; and drop the table.
  - Admin: User can alter table attributes; add, alter, or drop column families; and enable, disable, or drop the table. User can also trigger region (re)assignments or relocation.
- Column Family
  - Read: User can read from the column family
  - Write: User can write to the column family

There is also an implicit global scope for the superuser.

The superuser is a principal, specified in the HBase site configuration file, that has equivalent access to HBase as the 'root' user would on a UNIX derived system. Normally this is the principal that the HBase processes themselves authenticate as. Although future versions of HBase Access Control may support multiple superusers, the superuser privilege will always include the principal used to run the HMaster process. Only the superuser is allowed to create tables, switch the balancer on or off, or take other actions with global consequence. Furthermore, the superuser has an implicit grant of all permissions to all resources.

Tables have a new metadata attribute: OWNER, the user principal who owns the table. By default this will be set to the user principal who creates the table, though it may be changed at table creation time or during an alter operation by setting or changing the OWNER table attribute. Only a single user principal can own a table at a given time. A table owner will have all permissions over a given table.

8.2.3. Server-side Configuration for Access Control

Enable the AccessController coprocessor in the cluster configuration and restart HBase. The restart can be a rolling one. Complete the restart of all Master and RegionServer processes before setting up ACLs.

To enable the AccessController, modify the `hbase-site.xml` file on every server machine in the cluster to look like:

```
      <property>
        <name>hbase.coprocessor.master.classes</name>
        <value>org.apache.hadoop.hbase.security.access.AccessController</value>
      </property>
      <property>
      <name>hbase.coprocessor.region.classes</name>
        <value>org.apache.hadoop.hbase.security.token.TokenProvider,
        org.apache.hadoop.hbase.security.access.AccessController</value>
      </property>
```

8.2.4. Shell Enhancements for Access Control

The HBase shell has been extended to provide simple commands for editing and updating user permissions. The following commands have been added for access control list management:

Grant

```
    grant <user> <permissions> <table> [ <column family> [ <column qualifier> ] ]
```

`<permissions>` is zero or more letters from the set "RWCA": READ('R'), WRITE('W'), CREATE('C'), ADMIN('A').

Note: Grants and revocations of individual permissions on a resource are both accomplished using the `grant` command. A separate `revoke` command is also provided by the shell, but this is for fast revocation of all of a user's access rights to a given resource only.

Revoke

```
    revoke <user> <table> [ <column family> [ <column qualifier> ] ]
```

Alter

The `alter` command has been extended to allow ownership assignment:

```
      alter 'tablename', {OWNER => 'username'}
```

User Permission

The `user_permission` command shows all access permissions for the current user for a given table:

```
      user_permission <table>
```

8.3. Secure Bulk Load

Bulk loading in secure mode is a bit more involved than normal setup, since the client has to transfer the ownership of the files generated from the mapreduce job to HBase. Secure bulk loading is implemented by a coprocessor, named SecureBulkLoadEndpoint. SecureBulkLoadEndpoint uses a staging directory "hbase.bulkload.staging.dir", which defaults to /tmp/hbase-staging/. The algorithm is as follows.

- Create an hbase owned staging directory which is world traversable (-rwx--x--x, 711) /tmp/hbase-staging.
- A user writes out data to his secure output directory: /user/foo/data
- A call is made to hbase to create a secret staging directory which is globally readable/writable (-rwxrwxrwx, 777): /tmp/hbase-staging/averylongandrandomdirectoryname
- The user makes the data world readable and writable, then moves it into the random staging directory, then calls bulkLoadHFiles()

Like delegation tokens the strength of the security lies in the length and randomness of the secret directory.

You have to enable the secure bulk load to work properly. You can modify the hbase-site.xml file on every server machine in the cluster and add the SecureBulkLoadEndpoint class to the list of regionserver coprocessors:

```
      <property>          <name>hbase.bulkload.staging.dir</name>          <value>/tmp/hbase-staging</value>          </property>
```

Chapter 9. 架构

Table of Contents

9.1. 概述

9.1.1. NoSQL?

HBase是一种 "NoSQL" 数据库. "NoSQL"是一个通用词表示数据库不是RDBMS ，后者支持 SQL 作为主要访问手段。有许多种 NoSQL 数据库: BerkeleyDB 是本地 NoSQL 数据库例子, 而 HBase 是大型分布式数据库。 技术上来说, HBase 更像是"数据存储(Data Store)" 多于 "数据库(Data Base)"。因为缺少很多RDBMS特性, 如列类型，第二索引，触发器，高级查询语言等.

然而, HBase 有许多特征同时支持线性化和模块化扩充。 HBase 集群通过增加RegionServers进行扩充。 它可以放在普通的服务器中。例如，如果集群从10个扩充到20个RegionServer，存储空间和处理容量都同时翻倍。 RDBMS 也能很好扩充， 但仅对一个点 - 特别是对一个单独数据库服务器的大小 - 同时，为了更好的性能，需要特殊的硬件和存储设备。 HBase 特性：

- 强一致性读写: HBase 不是 "最终一致性(eventually consistent)" 数据存储. 这让它很适合高速计数聚合类任务。
- 自动分片(Automatic sharding): HBase 表通过region分布在集群中。数据增长时，region会自动分割并重新分布。
- RegionServer 自动故障转移
- Hadoop/HDFS 集成: HBase 支持本机外HDFS 作为它的分布式文件系统。
- MapReduce: HBase 通过MapReduce支持大并发处理， HBase 可以同时做源和目标.

- Java 客户端 API: HBase 支持易于使用的 Java API 进行编程访问.
- Thrift/REST API: HBase 也支持Thrift 和 REST 作为非Java 前端.
- Block Cache 和 Bloom Filters: 对于大容量查询优化， HBase支持 Block Cache 和 Bloom Filters。
- 运维管理: HBase提供内置网页用于运维视角和JMX 度量.

9.1.2. 什么时候用 HBase?

HBase不适合所有问题.

首先，确信有足够多数据，如果有上亿或上千亿行数据，HBase是很好的备选。 如果只有上千或上百万行，则用传统的RDBMS可能是更好的选择。因为所有数据可以在一两个节点保存，集群其他节点可能闲置。

其次，确信可以不依赖所有RDBMS的额外特性 (e.g., 列数据类型, 第二索引, 事物,高级查询语言等.) 一个建立在RDBMS上应用，如不能仅通过改变一个JDBC驱动移植到HBase。相对于移植， 需考虑从RDBMS 到 HBase是一次完全的重新设计。

第三， 确信你有足够硬件。甚至 HDFS 在小于5个数据节点时，干不好什么事情 (根据如 HDFS 块复制具有缺省值 3), 还要加上一个NameNode.

HBase 能在单独的笔记本上运行良好。但这应仅当成开发配置。

9.1.3. HBase 和 Hadoop/HDFS 的区别?

HDFS 是分布式文件系统，适合保存大文件。官方宣称它并非普通用途文件系统，不提供文件的个别记录的快速查询。 另一方面，HBase基于HDFS且提供大表的记录快速查找(和更新)。这有时可能引起概念混乱。 HBase 内部将数据放到索引好的 "存储文件(StoreFiles)" ，以便高速查询。存储文件位于 HDFS中。参考Chapter 5, *数据模型* 和该章其他内容获取更多HBase如何归档的信息。

9.2. 目录表(Catalog Tables)

目录表 -ROOT- 和 .META. 作为 HBase 表存在。他们被HBase shell的 `list` 命令过滤掉了， 但他们和其他表一样存在。

9.2.1. ROOT

-ROOT- 保存 .META. 表存在哪里的踪迹. -ROOT- 表结构如下:

Key:

- .META. region key (`.META.,,1`)

Values:

- `info:regioninfo` (序列化.META.的 HRegionInfo 实例)
- `info:server` (保存 .META.的RegionServer的server:port)
- `info:serverstartcode` (保存 .META.的RegionServer进程的启动时间)

9.2.2. META

.META. 保存系统中所有region列表。 .META.表结构如下:

Key:

- Region key 格式 (`[table],[region start key],[region id]`)

Values:

- `info:regioninfo` (序列化.META.的 HRegionInfo 实例)
- `info:server` (保存 .META.的RegionServer的server:port)
- `info:serverstartcode` (保存 .META.的RegionServer进程的启动时间)

当表在分割过程中，会创建额外的两列, `info:splitA` 和 `info:splitB` 代表两个女儿 region. 这两列的值同样是序列化HRegionInfo 实例. region最终分割完毕后，这行会删除。

HRegionInfo的备注: 空 key 用于指示表的开始和结束。具有空开始键值的region是表内的首region。 如果 region 同时有空起始和结束key，说明它是表内的唯一region。

在需要编程访问(希望不要)目录元数据时，参考 Writables 工具.

9.2.3. 启动时序

META 地址首先在ROOT 中设置。META 会更新 server 和 startcode 的值.

需要 region-RegionServer 分配信息, 参考 Section 9.7.2, “Region-RegionServer 分配”.

9.3. 客户端

HBase客户端的 HTable类负责寻找相应的RegionServers来处理行。他是先查询 `.META.` 和 `-ROOT` 目录表。然后再确定region的位置。定位到所需要的区域后，客户端会*直接* 去访问相应的region(不经过master)，发起读写请求。这些信息会缓存在客户端，这样就不用每发起一个请求就去查一下。如果一个region已经废弃(原因可能是master load balance或者RegionServer死了)，客户端就会重新进行这个步骤，决定要去访问的新的地址。

参考 Section 9.5.2, “Runtime Impact” for more information about the impact of the Master on HBase Client communication.

管理集群操作是经由HBaseAdmin发起的

9.3.1. 连接

关于连接的配置信息，参见Section 3.7, “连接HBase集群的客户端配置和依赖”.

HTable不是线程安全的。建议使用同一个HBaseConfiguration实例来创建HTable实例。这样可以共享ZooKeeper和socket实例。例如，最好这样做：

```
HBaseConfiguration conf = HBaseConfiguration.create();
HTable table1 = new HTable(conf, "myTable");
HTable table2 = new HTable(conf, "myTable");
```

而不是这样：

```
HBaseConfiguration conf1 = HBaseConfiguration.create();
HTable table1 = new HTable(conf1, "myTable");
HBaseConfiguration conf2 = HBaseConfiguration.create();
HTable table2 = new HTable(conf2, "myTable");
```

如果你想知道的更多的关于HBase客户端connection的知识，可以参照： HConnectionManager.

9.3.1.1. 连接池

对需要高端多线程访问的应用 (如网页服务器或应用服务器需要在一个JVM服务很多应用线程)，参考 HTablePool.

9.3.2. 写缓冲和批量操作

若关闭了HTable中的 Section 11.10.4, “AutoFlush”，`Put`操作会在写缓冲填满的时候向RegionServer发起请求。默认情况下，写缓冲是2MB.在Htable被废弃之前，要调用`close()`, `flushCommits()`操作，这样写缓冲就不会丢失。

要想更好的细粒度控制 `Put`或`Delete`的批量操作，可以参考Htable中的batch 方法.

9.3.3. 外部客户端

关于非Java客户端和定制协议信息，在 Chapter 10, *外部 API*

9.3.4. 行锁

行锁 在客户端 API中仍然存在， *但是* 不鼓励使用，因为管理不好，会锁定整个RegionServer.

这里是优秀ticket HBASE-2332 从终端移除该特性。

9.4. 客户端请求过滤器

Get 和 Scan 实例可以用 filters 配置，以应用于 RegionServer.

过滤器可能会搞混，因为有很多类型的过滤器， 最好通过理解过滤器功能组来了解他们。

9.4.1. 结构(Structural)过滤器

结构过滤器包含其他过滤器

9.4.1.1. FilterList

FilterList 代表一个过滤器列表，过滤器间具有 `FilterList.Operator.MUST_PASS_ALL` 或 `FilterList.Operator.MUST_PASS_ONE` 关系。下面示例展示两个过滤器的'或'关系(检查同一属性的'my value' 或'my other value').

```
FilterList list = new FilterList(FilterList.Operator.MUST_PASS_ONE);
SingleColumnValueFilter filter1 = new SingleColumnValueFilter(
        cf,
        column,
        CompareOp.EQUAL,
        Bytes.toBytes("my value")
        );
list.add(filter1);
SingleColumnValueFilter filter2 = new SingleColumnValueFilter(
        cf,
        column,
        CompareOp.EQUAL,
        Bytes.toBytes("my other value")
        );
list.add(filter2);
scan.setFilter(list);
```

9.4.2. 列值

9.4.2.1. SingleColumnValueFilter

SingleColumnValueFilter 用于测试列值相等 (`CompareOp.EQUAL`), 不等 (`CompareOp.NOT_EQUAL`),或范围 (e.g., `CompareOp.GREATER`). 下面示例检查列值和字符串'my value' 相等...

```
SingleColumnValueFilter filter = new SingleColumnValueFilter(
        cf,
        column,
        CompareOp.EQUAL,
        Bytes.toBytes("my value")
        );
scan.setFilter(filter);
```

9.4.3. 列值比较器

过滤器包内有好几种比较器类需要特别提及。这些比较器和其他过滤器一起使用, 如 Section 9.4.2.1, “SingleColumnValueFilter”.

9.4.3.1. RegexStringComparator

RegexStringComparator 支持值比较的正则表达式 。

```
RegexStringComparator comp = new RegexStringComparator("my.");   // any value that starts with 'my'
SingleColumnValueFilter filter = new SingleColumnValueFilter(
        cf,
        column,
        CompareOp.EQUAL,
        comp
        );
scan.setFilter(filter);
```

参考 Oracle JavaDoc 了解 supported RegEx patterns in Java.

9.4.3.2. SubstringComparator

SubstringComparator 用于检测一个子串是否存在于值中。大小写不敏感。

```
SubstringComparator comp = new SubstringComparator("y val");   // looking for 'my value'
SingleColumnValueFilter filter = new SingleColumnValueFilter(
        cf,
        column,
        CompareOp.EQUAL,
        comp
        );
scan.setFilter(filter);
```

9.4.3.3. BinaryPrefixComparator

参考 BinaryPrefixComparator.

9.4.3.4. BinaryComparator

参考 BinaryComparator.

9.4.4. 键值元数据

由于HBase 采用键值对保存内部数据，键值元数据过滤器评估一行的键是否存在(如 ColumnFamily:Column qualifiers) , 对应前节所述值的情况。

9.4.4.1. FamilyFilter

FamilyFilter 用于过滤列族。 通常，在Scan中选择ColumnFamilie优于在过滤器中做。

9.4.4.2. QualifierFilter

QualifierFilter 用于基于列名(即 Qualifier)过滤.

9.4.4.3. ColumnPrefixFilter

ColumnPrefixFilter 可基于列名(即Qualifier)前缀过滤。

A ColumnPrefixFilter seeks ahead to the first column matching the prefix in each row and for each involved column family. It can be used to efficiently get a subset of the columns in very wide rows.

Note: The same column qualifier can be used in different column families. This filter returns all matching columns.

Example: Find all columns in a row and family that start with "abc"

```
HTableInterface t = ...;
byte[] row = ...;
byte[] family = ...;
byte[] prefix = Bytes.toBytes("abc");
Scan scan = new Scan(row, row); // (optional) limit to one row
scan.addFamily(family); // (optional) limit to one family
Filter f = new ColumnPrefixFilter(prefix);
scan.setFilter(f);
scan.setBatch(10); // set this if there could be many columns returned
ResultScanner rs = t.getScanner(scan);
for (Result r = rs.next(); r != null; r = rs.next()) {
  for (KeyValue kv : r.raw()) {
    // each kv represents a column
  }
}
rs.close();
```

9.4.4.4. MultipleColumnPrefixFilter

MultipleColumnPrefixFilter 和 ColumnPrefixFilter 行为差不多，但可以指定多个前缀。

Like ColumnPrefixFilter, MultipleColumnPrefixFilter efficiently seeks ahead to the first column matching the lowest prefix and also seeks past ranges of columns between prefixes. It can be used to efficiently get discontinuous sets of columns from very wide rows.

Example: Find all columns in a row and family that start with "abc" or "xyz"

```
HTableInterface t = ...;
byte[] row = ...;
byte[] family = ...;
byte[][] prefixes = new byte[][] {Bytes.toBytes("abc"), Bytes.toBytes("xyz")};
Scan scan = new Scan(row, row); // (optional) limit to one row
scan.addFamily(family); // (optional) limit to one family
Filter f = new MultipleColumnPrefixFilter(prefixes);
scan.setFilter(f);
scan.setBatch(10); // set this if there could be many columns returned
ResultScanner rs = t.getScanner(scan);
for (Result r = rs.next(); r != null; r = rs.next()) {
  for (KeyValue kv : r.raw()) {
    // each kv represents a column
  }
}
rs.close();
```

9.4.4.5. ColumnRangeFilter

ColumnRangeFilter 可以进行高效内部扫描。

A ColumnRangeFilter can seek ahead to the first matching column for each involved column family. It can be used to efficiently get a 'slice' of the columns of a very wide row. i.e. you have a million columns in a row but you only want to look at columns bbbb-bbdd.

Note: The same column qualifier can be used in different column families. This filter returns all matching columns.

Example: Find all columns in a row and family between "bbbb" (inclusive) and "bbdd" (inclusive)

```
HTableInterface t = ...;
byte[] row = ...;
byte[] family = ...;
byte[] startColumn = Bytes.toBytes("bbbb");
byte[] endColumn = Bytes.toBytes("bbdd");
Scan scan = new Scan(row, row); // (optional) limit to one row
scan.addFamily(family); // (optional) limit to one family
Filter f = new ColumnRangeFilter(startColumn, true, endColumn, true);
scan.setFilter(f);
scan.setBatch(10); // set this if there could be many columns returned
ResultScanner rs = t.getScanner(scan);
for (Result r = rs.next(); r != null; r = rs.next()) {
  for (KeyValue kv : r.raw()) {
    // each kv represents a column
  }
}
rs.close();
```

Note: HBase 0.92 引入

9.4.5. RowKey

9.4.5.1. RowFilter

通常认为行选择时Scan采用 startRow/stopRow 方法比较好。然而 RowFilter 也可以用。

9.4.6. Utility

9.4.6.1. FirstKeyOnlyFilter

This is primarily used for rowcount jobs. 参考 FirstKeyOnlyFilter.

9.5. 主服务器

`HMaster` is the implementation of the Master Server. The Master server is responsible for monitoring all RegionServer instances in the cluster, and is the interface for all metadata changes. In a distributed cluster, the Master typically runs on the Section 9.9.1, “NameNode”.

9.5.1. Startup Behavior

If run in a multi-Master environment, all Masters compete to run the cluster. If the active Master loses it's lease in ZooKeeper (or the Master shuts down), then then the remaining Masters jostle to take over the Master role.

9.5.2. Runtime Impact

A common dist-list question is what happens to an HBase cluster when the Master goes down. Because the HBase client talks directly to the RegionServers, the cluster can still function in a "steady state." Additionally, per Section 9.2, “Catalog Tables” ROOT and META exist as HBase tables (i.e., are not resident in the Master). However, the Master controls critical functions such as RegionServer failover and completing region splits. So while the cluster can still run *for a time* without the Master, the Master should be restarted as soon as possible.

9.5.3. Interface

The methods exposed by `HMasterInterface` are primarily metadata-oriented methods:

- Table (createTable, modifyTable, removeTable, enable, disable)
- ColumnFamily (addColumn, modifyColumn, removeColumn)
- Region (move, assign, unassign)

For example, when the `HBaseAdmin` method `disableTable` is invoked, it is serviced by the Master server.

9.5.4. 进程

Master 后台运行几种线程:

9.5.4.1. LoadBalancer

Periodically, and when there are not any regions in transition, a load balancer will run and move regions around to balance cluster load. 参考 Section 2.8.3.1, “Balancer” for configuring this property.

参考 Section 9.7.2, “Region-RegionServer Assignment” for more information on region assignment.

9.5.4.2. CatalogJanitor

Periodically checks and cleans up the .META. table. 参考 Section 9.2.2, “META” for more information on META.

9.6. RegionServer

HRegionServer is the RegionServer implementation. It is responsible for serving and managing regions. In a distributed cluster, a RegionServer runs on a Section 9.9.2, “DataNode”.

9.6.1. 接口

The methods exposed by HRegionRegionInterface contain both data-oriented and region-maintenance methods:

- Data (get, put, delete, next, etc.)
- Region (splitRegion, compactRegion, etc.)

For example, when the HBaseAdmin method majorCompact is invoked on a table, the client is actually iterating through all regions for the specified table and requesting a major compaction directly to each region.

9.6.2. 进程

RegionServer 后台运行几种线程:

9.6.2.1. CompactSplitThread

检查分割并处理最小紧缩。

9.6.2.2. MajorCompactionChecker

检查主紧缩。

9.6.2.3. MemStoreFlusher

周期将写到内存存储的内容刷到文件存储。

9.6.2.4. LogRoller

周期检查RegionServer 的 HLog.

9.6.3. 协处理器

协处理器在0.92版添加。 有一个详细帖子 Blog Overview of CoProcessors 供参考。文档最终会放到本参考手册，但该blog是当前能获取的大部分信息。

9.6.4. 块缓存

9.6.4.1. 设计

The Block Cache is an LRU cache that contains three levels of block priority to allow for scan-resistance and in-memory ColumnFamilies:

- Single access priority: The first time a block is loaded from HDFS it normally has this priority and it will be part of the first group to be considered during evictions. The advantage is that scanned blocks are more likely to get evicted than blocks that are getting more usage.
- Mutli access priority: If a block in the previous priority group is accessed again, it upgrades to this priority. It is thus part of the second group considered during evictions.
- In-memory access priority: If the block's family was configured to be "in-memory", it will be part of this priority disregarding the number of times it was accessed. Catalog tables are configured like this. This group is the last one considered during evictions.

For more information, see the LruBlockCache source

9.6.4.2. 使用

Block caching is enabled by default for all the user tables which means that any read operation will load the LRU cache. This might be good for a large number of use cases, but further tunings are usually required in order to achieve better performance. An important concept is the working set size, or WSS, which is: "the amount of memory needed to compute the answer to a problem". For a website, this would be the data that's needed to answer the queries over a short amount of time.

The way to calculate how much memory is available in HBase for caching is:

```
            number of region servers * heap size * hfile.block.cache.size * 0.85
```

The default value for the block cache is 0.25 which represents 25% of the available heap. The last value (85%) is the default acceptable loading factor in the LRU cache after which eviction is started. The reason it is included in this equation is that it would be unrealistic to say that it is possible to use 100% of the available memory since this would make the process blocking from the point where it loads new blocks. Here are some examples:

- One region server with the default heap size (1GB) and the default block cache size will have 217MB of block cache available.
- 20 region servers with the heap size set to 8GB and a default block cache size will have 34GB of block cache.
- 100 region servers with the heap size set to 24GB and a block cache size of 0.5 will have about 1TB of block cache.

Your data isn't the only resident of the block cache, here are others that you may have to take into account:

- Catalog tables: The -ROOT- and .META. tables are forced into the block cache and have the in-memory priority which means that they are harder to evict. The former never uses more than a few hundreds of bytes while the latter can occupy a few MBs (depending on the number of regions).
- HFiles indexes: HFile is the file format that HBase uses to store data in HDFS and it contains a multi-layered index in order seek

to the data without having to read the whole file. The size of those indexes is a factor of the block size (64KB by default), the size of your keys and the amount of data you are storing. For big data sets it's not unusual to see numbers around 1GB per region server, although not all of it will be in cache because the LRU will evict indexes that aren't used.
- Keys: Taking into account only the values that are being stored is missing half the picture since every value is stored along with its keys (row key, family, qualifier, and timestamp). 参考 Section 6.3.2, “Try to minimize row and column sizes”.
- Bloom filters: Just like the HFile indexes, those data structures (when enabled) are stored in the LRU.

Currently the recommended way to measure HFile indexes and bloom filters sizes is to look at the region server web UI and checkout the relevant metrics. For keys, sampling can be done by using the HFile command line tool and look for the average key size metric.

It's generally bad to use block caching when the WSS doesn't fit in memory. This is the case when you have for example 40GB available across all your region servers' block caches but you need to process 1TB of data. One of the reasons is that the churn generated by the evictions will trigger more garbage collections unnecessarily. Here are two use cases:

- Fully random reading pattern: This is a case where you almost never access the same row twice within a short amount of time such that the chance of hitting a cached block is close to 0. Setting block caching on such a table is a waste of memory and CPU cycles, more so that it will generate more garbage to pick up by the JVM. For more information on monitoring GC, see Section 12.2.3, “JVM Garbage Collection Logs”.
- Mapping a table: In a typical MapReduce job that takes a table in input, every row will be read only once so there's no need to put them into the block cache. The Scan object has the option of turning this off via the setCaching method (set it to false). You can still keep block caching turned on on this table if you need fast random read access. An example would be counting the number of rows in a table that serves live traffic, caching every block of that table would create massive churn and would surely evict data that's currently in use.

9.6.5. 预写日志 (WAL)

9.6.5.1. Purpose

每个RegionServer会将更新(Puts, Deletes) 先记录到预写日志中(WAL)，然后将其更新在Section 9.7.5, “Store”的Section 9.7.5.1, “MemStore”里面。这样就保证了HBase的写的可靠性。如果没有WAL,当RegionServer宕掉的时候，MemStore还没有flush，StoreFile还没有保存，数据就会丢失。HLog 是HBase的一个WAL实现，一个RegionServer有一个HLog实例。

WAL 保存在HDFS 的 `/hbase/.logs/` 里面，每个region一个文件。

要想知道更多的信息，可以访问维基百科 Write-Ahead Log 的文章.

9.6.5.2. WAL Flushing

TODO (describe).

9.6.5.3. WAL Splitting

9.6.5.3.1. 当RegionServer宕掉的时候，如何恢复

When a RegionServer crashes, it will lose its ephemeral lease in ZooKeeper...TODO

9.6.5.3.2. `hbase.hlog.split.skip.errors`

默认设置为 `true`,在split执行中发生的任何错误会被记录，有问题的WAL会被移动到HBase `rootdir`目录下的`.corrupt`目录，接着进行处理。如果设置为 `false`，异常会被抛出，split会记录错误。[23]

9.6.5.3.3. 如何处理一个发生在当RegionServers' WALs 分割时候的EOFExceptions异常

如果我们在分割日志的时候发生EOF,就是`hbase.hlog.split.skip.errors`设置为 `false`，我们也会进行处理。一个EOF会发生在一行一行读取Log，但是Log中最后一行似乎只写了一半就停止了。如果在处理过程中发生了EOF，我们还会继续处理，除非这个文件是要处理的最后一个文件。[24]

[23] 参考 HBASE-2958 When hbase.hlog.split.skip.errors is set to false, we fail the split but thats it. We need to do more than just fail split if this flag is set.

[24] 要想知道背景知识, 参见HBASE-2643 Figure how to deal with eof splitting logs

9.7. 区域

区域是表获取和分布的基本元素，由每个列族的一个库(Store)组成。对象层级图如下：

```
Table       (HBase table)
    Region       (Regions for the table)
         Store          (Store per ColumnFamily for each Region for the table)
              MemStore           (MemStore for each Store for each Region for the table)
              StoreFile          (StoreFiles for each Store for each Region for the table)
                    Block              (Blocks within a StoreFile within a Store for each Region for the table)
```

关于HBase文件写到HDFS的描述，参考 Section 12.7.2, “浏览 HDFS的 HBase 对象”.

9.7.1. Region 大小

Region的大小是一个棘手的问题，需要考量如下几个因素。

- Regions是可用性和分布式的最基本单位

- HBase通过将region切分在许多机器上实现分布式。也就是说，你如果有16GB的数据，只分了2个region， 你却有20台机器，有18台就浪费了。
- region数目太多就会造成性能下降，现在比以前好多了。但是对于同样大小的数据，700个region比3000个要好。
- region数目太少就会妨碍可扩展性，降低并行能力。有的时候导致压力不够分散。这就是为什么，你向一个10节点的HBase集群导入200MB的数据，大部分的节点是idle的。
- RegionServer中1个region和10个region索引需要的内存量没有太多的差别。

最好是使用默认的配置，可以把热的表配小一点(或者受到split热点的region把压力分散到集群中)。如果你的cell的大小比较大(100KB或更大)，就可以把region的大小调到1GB。

参考 Section 2.5.2.6, “更大区域” 获取配置更多信息.

9.7.2. 区域-区域服务器分配

本节描述区域如何分配到区域服务器。

9.7.2.1. 启动

当HBase启动时，区域分配如下(短版本):

1. 启动时主服务器调用`AssignmentManager`.
2. `AssignmentManager` 在META 中查找已经存在的区域分配。
3. 如果区域分配还有效(如 RegionServer 还在线) ，那么分配继续保持。
4. 如果区域分配失效，`LoadBalancerFactory` 被调用来分配区域。 `DefaultLoadBalancer` 将随机分配区域到RegionServer.
5. META 随 RegionServer 分配更新(如果需要) ， RegionServer 启动区域开启代码(RegionServer 启动时进程)

9.7.2.2. 故障转移

当区域服务器出故障退出时 (短版本):

1. 区域立即不可获取，因为区域服务器退出。
2. 主服务器会检测到区域服务器退出。
3. 区域分配会失效并被重新分配，如同启动时序。

9.7.2.3. 区域负载均衡

区域可以定期移动，见 Section 9.5.4.1, “LoadBalancer”.

9.7.3. 区域-区域服务器本地化

Over time, Region-RegionServer locality is achieved via HDFS block replication. The HDFS client does the following by default when choosing locations to write replicas:

1. First replica is written to local node
2. Second replica is written to another node in same rack
3. Third replica is written to a node in another rack (if sufficient nodes)

Thus, HBase eventually achieves locality for a region after a flush or a compaction. In a RegionServer failover situation a RegionServer may be assigned regions with non-local StoreFiles (because none of the replicas are local), however as new data is written in the region, or the table is compacted and StoreFiles are re-written, they will become "local" to the RegionServer.

For more information, see HDFS Design on Replica Placement and also Lars George's blog on HBase and HDFS locality.

9.7.4. 区域分割

区域服务器的分割操作是不可见的，因为Master不会参与其中。区域服务器切割region的步骤是，先将该region下线，然后切割，将其子region加入到META元信息中，再将他们加入到原本的区域服务器中，最后汇报Master.参见Section 2.8.2.7, “管理 Splitting”来手动管理切割操作(以及为何这么做)。

9.7.4.1. 自定义分割策略

缺省分割策略可以被重写，采用自定义RegionSplitPolicy (HBase 0.94+).一般自定义分割策略应该扩展HBase的缺省分割策略: ConstantSizeRegionSplitPolicy.

策略可以HBaseConfiguration 全局使用，或基于每张表:

```
HTableDescriptor myHtd = ...;
myHtd.setValue(HTableDescriptor.SPLIT_POLICY, MyCustomSplitPolicy.class.getName());
```

9.7.5. 存储

一个存储包含了一个内存存储(MemStore)和若干个文件存储(StoreFile--HFile).一个存储可以定位到一个列族中的一个区.

9.7.5.1. MemStore

MemStores是Store中的内存Store,可以进行修改操作。修改的内容是KeyValues。当flush的是，现有的memstore会生成快照，然后清空。在执行快照的时候，HBase会继续接收修改操作，保存在memstore外面，直到快照完成。

9.7.5.2. StoreFile (HFile)

文件存储是数据存在的地方。

9.7.5.2.1. HFile Format

*hfile*文件格式是基于BigTable [2006]论文中的SSTable。构建在Hadoop的tfile 上面(直接使用了tfile的单元测试和压缩工具)。 Schubert Zhang 的博客HFile: A Block-Indexed File Format to Store Sorted Key-Value Pairs详细介绍了HBases的hfile。Matteo Bertozzi也做了详细的介绍HBase I/O: HFile。

For more information, see the HFile source code. Also see Appendix E, *HFile format version 2* for information about the HFile v2 format that was included in 0.92.

9.7.5.2.2. HFile 工具

要想看到hfile内容的文本化版本，你可以使用`org.apache.hadoop.hbase.io.hfile.HFile` 工具。可以这样用:

```
$ ${HBASE_HOME}/bin/hbase org.apache.hadoop.hbase.io.hfile.HFile
```

例如，你想看文件 `hdfs://10.81.47.41:9000/hbase/TEST/1418428042/DSMP/4759508618286845475`的内容, 就执行如下的命令:

```
 $ ${HBASE_HOME}/bin/hbase org.apache.hadoop.hbase.io.hfile.HFile -v -f hdfs://10.81.47.41:9000/hbase/TEST/1418428042/DSMP/47595
```

如果你没有输入-v,就仅仅能看到一个hfile的汇总信息。其他功能的用法可以看`HFile`的文档。

9.7.5.2.3. StoreFile Directory Structure on HDFS

For more information of what StoreFiles look like on HDFS with respect to the directory structure, see Section 12.7.2, “Browsing HDFS for HBase Objects”.

9.7.5.3. Blocks

StoreFiles are composed of blocks. The blocksize is configured on a per-ColumnFamily basis.

Compression happens at the block level within StoreFiles. For more information on compression, see Appendix C, *Compression In HBase*.

For more information on blocks, see the HFileBlock source code.

9.7.5.4. KeyValue

The KeyValue class is the heart of data storage in HBase. KeyValue wraps a byte array and takes offsets and lengths into passed array at where to start interpreting the content as KeyValue.

The KeyValue format inside a byte array is:

- keylength
- valuelength
- key
- value

The Key is further decomposed as:

- rowlength
- row (i.e., the rowkey)
- columnfamilylength
- columnfamily
- columnqualifier
- timestamp
- keytype (e.g., Put, Delete, DeleteColumn, DeleteFamily)

KeyValue instances are *not* split across blocks. For example, if there is an 8 MB KeyValue, even if the block-size is 64kb this KeyValue will be read in as a coherent block. For more information, see the KeyValue source code.

9.7.5.4.1. Example

To emphasize the points above, examine what happens with two Puts for two different columns for the same row:

- Put #1: `rowkey=row1, cf:attr1=value1`
- Put #2: `rowkey=row1, cf:attr2=value2`

Even though these are for the same row, a KeyValue is created for each column:

Key portion for Put #1:

- rowlength `------------> 4`
- row `-----------------> row1`
- columnfamilylength `---> 2`
- columnfamily `--------> cf`
- columnqualifier `------> attr1`
- timestamp `-----------> server time of Put`
- keytype `-------------> Put`

Key portion for Put #2:

- rowlength `------------> 4`
- row `-----------------> row1`
- columnfamilylength `---> 2`
- columnfamily `--------> cf`
- columnqualifier `------> attr2`
- timestamp `-----------> server time of Put`
- keytype `-------------> Put`

It is critical to understand that the rowkey, ColumnFamily, and column (aka columnqualifier) are embedded within the KeyValue instance. The longer these identifiers are, the bigger the KeyValue is.

9.7.5.5. 紧缩

有两种类型的紧缩：次紧缩和主紧缩。minor紧缩通常会将数个小的相邻的文件合并成一个大的。Minor不会删除打上删除标记的数据，也不会删除过期的数据，Major紧缩会删除过期的数据。有些时候minor紧缩就会将一个store中的全部文件紧缩，实际上这个时候他本身就是一个major压缩。对于一个minor紧缩是如何紧缩的，可以参见ascii diagram in the Store source code.

在执行一个major紧缩之后，一个store只会有一个sotrefile,通常情况下这样可以提供性能。注意：major紧缩将会将store中的数据全部重写，在一个负载很大的系统中，这个操作是很伤的。所以在大型系统中，通常会自己Section 2.8.2.8, “管理 Splitting”。

紧缩 *不会* 进行分区合并。参考 Section 14.2.2, “Merge” 获取更多合并的信息。

9.7.5.5.1. Compaction File Selection

To understand the core algorithm for StoreFile selection, there is some ASCII-art in the Store source code that will serve as useful reference. It has been copied below:

```
/* normal skew:
 *
 *         older ----> newer
 *     _
 *    | |   _
 *    | |  | |   _
 *  --|-|- |-|- |-|---_-------_-------  minCompactSize
 *    | |  | |  | |  | |  _  | |
 *    | |  | |  | |  | | | | | |
 *    | |  | |  | |  | | | | | |
 */
```

Important knobs:

- `hbase.store.compaction.ratio` Ratio used in compaction file selection algorithm (default 1.2f).
- `hbase.hstore.compaction.min` (.90 hbase.hstore.compactionThreshold) (files) Minimum number of StoreFiles per Store to be selected for a compaction to occur (default 2).
- `hbase.hstore.compaction.max` (files) Maximum number of StoreFiles to compact per minor compaction (default 10).
- `hbase.hstore.compaction.min.size` (bytes) Any StoreFile smaller than this setting with automatically be a candidate for compaction. Defaults to `hbase.hregion.memstore.flush.size` (128 mb).
- `hbase.hstore.compaction.max.size` (.92) (bytes) Any StoreFile larger than this setting with automatically be excluded from compaction (default Long.MAX_VALUE).

The minor compaction StoreFile selection logic is size based, and selects a file for compaction when the file <= sum(smaller_files) * `hbase.hstore.compaction.ratio`.

9.7.5.5.2. Minor Compaction File Selection - Example #1 (Basic Example)

This example mirrors an example from the unit test `TestCompactSelection`.

- `hbase.store.compaction.ratio` = 1.0f
- `hbase.hstore.compaction.min` = 3 (files)
- `hbase.hstore.compaction.max` = 5 (files)
- `hbase.hstore.compaction.min.size` = 10 (bytes)
- `hbase.hstore.compaction.max.size` = 1000 (bytes)

The following StoreFiles exist: 100, 50, 23, 12, and 12 bytes apiece (oldest to newest). With the above parameters, the files that would be selected for minor compaction are 23, 12, and 12.

Why?

- 100 --> No, because sum(50, 23, 12, 12) * 1.0 = 97.
- 50 --> No, because sum(23, 12, 12) * 1.0 = 47.
- 23 --> Yes, because sum(12, 12) * 1.0 = 24.
- 12 --> Yes, because the previous file has been included, and because this does not exceed the the max-file limit of 5
- 12 --> Yes, because the previous file had been included, and because this does not exceed the the max-file limit of 5.

9.7.5.5.3. Minor Compaction File Selection - Example #2 (Not Enough Files To Compact)

This example mirrors an example from the unit test `TestCompactSelection`.

- `hbase.store.compaction.ratio` = 1.0f
- `hbase.hstore.compaction.min` = 3 (files)
- `hbase.hstore.compaction.max` = 5 (files)
- `hbase.hstore.compaction.min.size` = 10 (bytes)
- `hbase.hstore.compaction.max.size` = 1000 (bytes)

The following StoreFiles exist: 100, 25, 12, and 12 bytes apiece (oldest to newest). With the above parameters, the files that would be selected for minor compaction are 23, 12, and 12.

Why?

- 100 --> No, because sum(25, 12, 12) * 1.0 = 47
- 25 --> No, because sum(12, 12) * 1.0 = 24
- 12 --> No. Candidate because sum(12) * 1.0 = 12, there are only 2 files to compact and that is less than the threshold of 3
- 12 --> No. Candidate because the previous StoreFile was, but there are not enough files to compact

9.7.5.5.4. Minor Compaction File Selection - Example #3 (Limiting Files To Compact)

This example mirrors an example from the unit test `TestCompactSelection`.

- `hbase.store.compaction.ratio` = 1.0f
- `hbase.hstore.compaction.min` = 3 (files)
- `hbase.hstore.compaction.max` = 5 (files)
- `hbase.hstore.compaction.min.size` = 10 (bytes)
- `hbase.hstore.compaction.max.size` = 1000 (bytes)

The following StoreFiles exist: 7, 6, 5, 4, 3, 2, and 1 bytes apiece (oldest to newest). With the above parameters, the files that would be selected for minor compaction are 7, 6, 5, 4, 3.

Why?

- 7 --> Yes, because sum(6, 5, 4, 3, 2, 1) * 1.0 = 21. Also, 7 is less than the min-size
- 6 --> Yes, because sum(5, 4, 3, 2, 1) * 1.0 = 15. Also, 6 is less than the min-size.
- 5 --> Yes, because sum(4, 3, 2, 1) * 1.0 = 10. Also, 5 is less than the min-size.
- 4 --> Yes, because sum(3, 2, 1) * 1.0 = 6. Also, 4 is less than the min-size.
- 3 --> Yes, because sum(2, 1) * 1.0 = 3. Also, 3 is less than the min-size.
- 2 --> No. Candidate because previous file was selected and 2 is less than the min-size, but the max-number of files to compact has been reached.
- 1 --> No. Candidate because previous file was selected and 1 is less than the min-size, but max-number of files to compact has been reached.

9.7.5.5.5. Impact of Key Configuration Options

`hbase.store.compaction.ratio`. A large ratio (e.g., 10) will produce a single giant file. Conversely, a value of .25 will produce behavior similar to the BigTable compaction algorithm - resulting in 4 StoreFiles.

`hbase.hstore.compaction.min.size`. Because this limit represents the "automatic include" limit for all StoreFiles smaller than this value, this value may need to be adjusted downwards in write-heavy environments where many 1 or 2 mb StoreFiles are being flushed, because every file will be targeted for compaction and the resulting files may still be under the min-size and require further compaction, etc.

[25] For description of the development process -- why static blooms rather than dynamic -- and for an overview of the unique properties that pertain to blooms in HBase, as well as possible future directions, see the *Development Process* section of the document BloomFilters in HBase attached to HBase-1200.

[26] The bloom filters described here are actually version two of blooms in HBase. In versions up to 0.19.x, HBase had a dynamic bloom option based on work done by the European Commission One-Lab Project 034819. The core of the HBase bloom work was later pulled up into Hadoop to implement org.apache.hadoop.io.BloomMapFile. Version 1 of HBase blooms never worked that well. Version 2 is a rewrite from scratch though again it starts with the one-lab work.

9.8. 批量装载(Bulk Loading)

9.8.1. 概述

HBase 有好几种方法将数据装载到表。最直接的方式即可以通过MapReduce任务，也可以通过普通客户端API。但是这都不是高效方法。

批量装载特性采用 MapReduce 任务，将表数据输出为HBase的内部数据格式，然后可以将产生的存储文件直接装载到运行的集群中。批量装载比简单使用 HBase API 消耗更少的CPU和网络资源。

9.8.2. 批量装载架构

HBase 批量装载过程包含两个主要步骤。

9.8.2.1. 通过MapReduce 任务准备数据

批量装载第一步，从MapReduce任务通过`HFileOutputFormat`产生HBase数据文件(StoreFiles) 。输出数据为HBase的内部数据格式，以便随后装载到集群更高效。

为了处理高效， `HFileOutputFormat` 必须比配置为每个HFile适合在一个分区内。为了做到这一点，输出将被批量装载到HBase的任务，使用Hadoop 的`TotalOrderPartitioner` 类来分开map输出为分开的键空间区间。对应于表内每个分区(region)的键空间。

`HFileOutputFormat` 包含一个方便的函数, `configureIncrementalLoad()`, 可以基于表当前分区边界自动设置`TotalOrderPartitioner`。

9.8.2.2. 完成数据装载

After the data has been prepared using `HFileOutputFormat`, it is loaded into the cluster using `completebulkload`. This command line tool iterates through the prepared data files, and for each one determines the region the file belongs to. It then contacts the appropriate Region Server which adopts the HFile, moving it into its storage directory and making the data available to clients.

If the region boundaries have changed during the course of bulk load preparation, or between the preparation and completion steps, the `completebulkloads` utility will automatically split the data files into pieces corresponding to the new boundaries. This process is not optimally efficient, so users should take care to minimize the delay between preparing a bulk load and importing it into the cluster, especially if other clients are simultaneously loading data through other means.

9.8.3. 采用completebulkload 工具导入准备的数据

After a data import has been prepared, either by using the `importtsv` tool with the "`importtsv.bulk.output`" option or by some other MapReduce job using the `HFileOutputFormat`, the `completebulkload` tool is used to import the data into the running cluster.

The `completebulkload` tool simply takes the output path where `importtsv` or your MapReduce job put its results, and the table name to import into. For example:

```
$ hadoop jar hbase-VERSION.jar completebulkload [-c /path/to/hbase/config/hbase-site.xml] /user/todd/myoutput mytable
```

The `-c config-file` option can be used to specify a file containing the appropriate hbase parameters (e.g., hbase-site.xml) if not supplied already on the CLASSPATH (In addition, the CLASSPATH must contain the directory that has the zookeeper configuration file if zookeeper is NOT managed by HBase).

Note: If the target table does not already exist in HBase, this tool will create the table automatically.

This tool will run quickly, after which point the new data will be visible in the cluster.

9.8.4. 参考

For more information about the referenced utilities, see Section 14.1.9, “ImportTsv” and Section 14.1.10, “CompleteBulkLoad”.

9.8.5. 高级使用

Although the `importtsv` tool is useful in many cases, advanced users may want to generate data programatically, or import data from other formats. To get started doing so, dig into `ImportTsv.java` and check the JavaDoc for HFileOutputFormat.

The import step of the bulk load can also be done programatically. 参考 the `LoadIncrementalHFiles` class 获取更多信息。

9.9. HDFS

由于 HBase 在 HDFS 上运行(每个存储文件也被写为HDFS的文件)，必须理解 HDFS 结构，特别是它如何存储文件，处理故障转移，备份块。

参考 Hadoop 文档 HDFS Architecture 获取更多信息。

9.9.1. NameNode

NameNode 负责维护文件系统元数据。参考上述HDFS结构链接获取更多信息。

9.9.2. DataNode

DataNode 负责存储HDFS 块。 参考上述HDFS结构链接获取更多信息。

[23] 参考 HBASE-2958 When hbase.hlog.split.skip.errors is set to false, we fail the split but thats it. We need to do more than just fail split if this flag is set.

[24] For background, see HBASE-2643 Figure how to deal with eof splitting logs

[25] For description of the development process -- why static blooms rather than dynamic -- and for an overview of the unique properties that pertain to blooms in HBase, as well as possible future directions, see the *Development Process* section of the documentBloomFilters in HBase attached to HBase-1200.

[26] The bloom filters described here are actually version two of blooms in HBase. In versions up to 0.19.x, HBase had a dynamic bloom option based on work done by the European Commission One-Lab Project 034819. The core of the HBase bloom work was later pulled up into Hadoop to implement org.apache.hadoop.io.BloomMapFile. Version 1 of HBase blooms never worked that well. Version 2 is a rewrite from scratch though again it starts with the one-lab work.

Chapter 10. 外部 API

Table of Contents

This chapter will cover access to HBase either through non-Java languages, or through custom protocols.

10.1. 非Java 语言和 JVM 通话

当前本话题大部分文档在 HBase Wiki. 参考 Thrift API Javadoc.

10.2. REST

当前 REST大部分文档在 HBase Wiki on REST.

10.3. Thrift

当前 Thrift大部分文档在 HBase Wiki on Thrift.

10.3.1. 过滤器语言

10.3.1.1. 用例

注意: 本特性在 HBase 0.92 中加入。

This allows the user to perform server-side filtering when accessing HBase over Thrift. The user specifies a filter via a string. The string is parsed on the server to construct the filter

10.3.1.2. 通用过滤字符串语法

A simple filter expression is expressed as: `“FilterName (argument, argument, ... , argument)”`

You must specify the name of the filter followed by the argument list in parenthesis. Commas separate the individual arguments

If the argument represents a string, it should be enclosed in single quotes.

If it represents a boolean, an integer or a comparison operator like <, >, != etc. it should not be enclosed in quotes

The filter name must be one word. All ASCII characters are allowed except for whitespace, single quotes and parenthesis.

The filter’s arguments can contain any ASCII character. `If single quotes are present in the argument, they must be escaped by a preceding single quote`

10.3.1.3. Compound Filters and Operators

Currently, two binary operators – AND/OR and two unary operators – WHILE/SKIP are supported.

Note: the operators are all in uppercase

AND – as the name suggests, if this operator is used, the key-value must pass both the filters

OR – as the name suggests, if this operator is used, the key-value must pass at least one of the filters

SKIP – For a particular row, if any of the key-values don’t pass the filter condition, the entire row is skipped

WHILE - For a particular row, it continues to emit key-values until a key-value is reached that fails the filter condition

Compound Filters: Using these operators, a hierarchy of filters can be created. For example: `“(Filter1 AND Filter2) OR (Filter3 AND Filter4)”`

10.3.1.4. Order of Evaluation

Parenthesis have the highest precedence. The SKIP and WHILE operators are next and have the same precedence.The AND operator has the next highest precedence followed by the OR operator.

For example:

A filter string of the form:`“Filter1 AND Filter2 OR Filter3”` will be evaluated as:`“(Filter1 AND Filter2) OR Filter3”`

A filter string of the form:`“Filter1 AND SKIP Filter2 OR Filter3”` will be evaluated as:`“(Filter1 AND (SKIP Filter2)) OR Filter3”`

10.3.1.5. 比较运算符

比较运算符可以是下面之一:

1. LESS (<)
2. LESS_OR_EQUAL (<=)
3. EQUAL (=)
4. NOT_EQUAL (!=)
5. GREATER_OR_EQUAL (>=)
6. GREATER (>)
7. NO_OP (no operation)

客户端应该使用 (<, <=, =, !=, >, >=) 来表达比较操作.

10.3.1.6. 比较器(Comparator)

比较器可以是下面之一:

1. **BinaryComparator** - This lexicographically compares against the specified byte array using Bytes.compareTo(byte[], byte[])
2. **BinaryPrefixComparator** - This lexicographically compares against a specified byte array. It only compares up to the length of this byte array.
3. **RegexStringComparator** - This compares against the specified byte array using the given regular expression. Only EQUAL and NOT_EQUAL comparisons are valid with this comparator
4. **SubStringComparator** - This tests if the given substring appears in a specified byte array. The comparison is case insensitive. Only EQUAL and NOT_EQUAL comparisons are valid with this comparator

The general syntax of a comparator is: `ComparatorType:ComparatorValue`

The ComparatorType for the various comparators is as follows:

1. **BinaryComparator** - binary

2. **BinaryPrefixComparator** - binaryprefix

3. **RegexStringComparator** - regexstring

4. **SubStringComparator** - substring

The ComparatorValue can be any value.

Example1: `>, 'binary:abc'` will match everything that is lexicographically greater than "abc"

Example2: `=, 'binaryprefix:abc'` will match everything whose first 3 characters are lexicographically equal to "abc"

Example3: `!=, 'regexstring:ab*yz'` will match everything that doesn't begin with "ab" and ends with "yz"

Example4: `=, 'substring:abc123'` will match everything that begins with the substring "abc123"

10.3.1.7. PHP 客户端编程使用过滤器示例

```
<? $_SERVER['PHP_ROOT'] = realpath(dirname(__FILE__).'/..');
   require_once $_SERVER['PHP_ROOT'].'/flib/__flib.php';
   flib_init(FLIB_CONTEXT_SCRIPT);
   require_module('storage/hbase');
   $hbase = new HBase('<server_name_running_thrift_server>', <port on which thrift server is running>);
   $hbase->open();
   $client = $hbase->getClient();
   $result = $client->scannerOpenWithFilterString('table_name', "(PrefixFilter ('row2') AND (QualifierFilter (>=, 'binary:xyz'))
   $to_print = $client->scannerGetList($result,1);
   while ($to_print) {
      print_r($to_print);
      $to_print = $client->scannerGetList($result,1);
    }
   $client->scannerClose($result);
?>
```

10.3.1.8. 过滤字符串示例

- `“PrefixFilter (‘Row’) AND PageFilter (1) AND FirstKeyOnlyFilter ()”` will return all key-value pairs that match the following conditions:

  1) The row containing the key-value should have prefix “Row”

  2) The key-value must be located in the first row of the table

  3) The key-value pair must be the first key-value in the row

- `“(RowFilter (=, ‘binary:Row 1’) AND TimeStampsFilter (74689, 89734)) OR ColumnRangeFilter (‘abc’, true, ‘xyz’, false))”` will return all key-value pairs that match both the following conditions:

  1) The key-value is in a row having row key “Row 1”

  2) The key-value must have a timestamp of either 74689 or 89734.

  Or it must match the following condition:

  1) The key-value pair must be in a column that is lexicographically >= abc and < xyz

- `“SKIP ValueFilter (0)”` will skip the entire row if any of the values in the row is not 0

10.3.1.9. 独有过滤器语法

1. **KeyOnlyFilter**

   Description: This filter doesn’t take any arguments. It returns only the key component of each key-value.

   Syntax: KeyOnlyFilter ()

   Example: "KeyOnlyFilter ()"

2. **FirstKeyOnlyFilter**

   Description: This filter doesn’t take any arguments. It returns only the first key-value from each row.

   Syntax: FirstKeyOnlyFilter ()

   Example: "FirstKeyOnlyFilter ()"

3. **PrefixFilter**

   Description: This filter takes one argument – a prefix of a row key. It returns only those key-values present in a row that starts with the specified row prefix

   Syntax: PrefixFilter (‘<row_prefix>’)

   Example: "PrefixFilter (‘Row’)"

4. **ColumnPrefixFilter**

   Description: This filter takes one argument – a column prefix. It returns only those key-values present in a column that starts with the specified column prefix. The column prefix must be of the form: `“qualifier”`

   Syntax: ColumnPrefixFilter(‘<column_prefix>’)

Example: "ColumnPrefixFilter('Col')"

5. **MultipleColumnPrefixFilter**

   Description: This filter takes a list of column prefixes. It returns key-values that are present in a column that starts with any of the specified column prefixes. Each of the column prefixes must be of the form: `"qualifier"`

   Syntax: MultipleColumnPrefixFilter('<column_prefix>', '<column_prefix>', …, '<column_prefix>')

   Example: "MultipleColumnPrefixFilter('Col1', 'Col2')"

6. **ColumnCountGetFilter**

   Description: This filter takes one argument – a limit. It returns the first limit number of columns in the table

   Syntax: ColumnCountGetFilter ('<limit>')

   Example: "ColumnCountGetFilter (4)"

7. **PageFilter**

   Description: This filter takes one argument – a page size. It returns page size number of rows from the table.

   Syntax: PageFilter ('<page_size>')

   Example: "PageFilter (2)"

8. **ColumnPaginationFilter**

   Description: This filter takes two arguments – a limit and offset. It returns limit number of columns after offset number of columns. It does this for all the rows

   Syntax: ColumnPaginationFilter('<limit>', '<offest>')

   Example: "ColumnPaginationFilter (3, 5)"

9. **InclusiveStopFilter**

   Description: This filter takes one argument – a row key on which to stop scanning. It returns all key-values present in rows up to and including the specified row

   Syntax: InclusiveStopFilter('<stop_row_key>')

   Example: "InclusiveStopFilter ('Row2')"

10. **TimeStampsFilter**

    Description: This filter takes a list of timestamps. It returns those key-values whose timestamps matches any of the specified timestamps

    Syntax: TimeStampsFilter (<timestamp>, <timestamp>, ... ,<timestamp>)

    Example: "TimeStampsFilter (5985489, 48895495, 58489845945)"

11. **RowFilter**

    Description: This filter takes a compare operator and a comparator. It compares each row key with the comparator using the compare operator and if the comparison returns true, it returns all the key-values in that row

    Syntax: RowFilter (<compareOp>, '<row_comparator>')

    Example: "RowFilter (<=, 'xyz)"

12. **Family Filter**

    Description: This filter takes a compare operator and a comparator. It compares each qualifier name with the comparator using the compare operator and if the comparison returns true, it returns all the key-values in that column

    Syntax: QualifierFilter (<compareOp>, '<qualifier_comparator>')

    Example: "QualifierFilter (=, 'Column1')"

13. **QualifierFilter**

    Description: This filter takes a compare operator and a comparator. It compares each qualifier name with the comparator using the compare operator and if the comparison returns true, it returns all the key-values in that column

    Syntax: QualifierFilter (<compareOp>,'<qualifier_comparator>')

    Example: "QualifierFilter (=,'Column1')"

14. **ValueFilter**

    Description: This filter takes a compare operator and a comparator. It compares each value with the comparator using the compare operator and if the comparison returns true, it returns that key-value

    Syntax: ValueFilter (<compareOp>,'<value_comparator>')

Example: "ValueFilter (!=, 'Value')"

15. **DependentColumnFilter**

    Description: This filter takes two arguments – a family and a qualifier. It tries to locate this column in each row and returns all key-values in that row that have the same timestamp. If the row doesn't contain the specified column – none of the key-values in that row will be returned.

    The filter can also take an optional boolean argument – dropDependentColumn. If set to true, the column we were depending on doesn't get returned.

    The filter can also take two more additional optional arguments – a compare operator and a value comparator, which are further checks in addition to the family and qualifier. If the dependent column is found, its value should also pass the value check and then only is its timestamp taken into consideration

    Syntax: DependentColumnFilter ('<family>', '<qualifier>', <boolean>, <compare operator>, '<value comparator')

    Syntax: DependentColumnFilter ('<family>', '<qualifier>', <boolean>)

    Syntax: DependentColumnFilter ('<family>', '<qualifier>')

    Example: "DependentColumnFilter ('conf', 'blacklist', false, >=, 'zebra')"

    Example: "DependentColumnFilter ('conf', 'blacklist', true)"

    Example: "DependentColumnFilter ('conf', 'blacklist')"

16. **SingleColumnValueFilter**

    Description: This filter takes a column family, a qualifier, a compare operator and a comparator. If the specified column is not found – all the columns of that row will be emitted. If the column is found and the comparison with the comparator returns true, all the columns of the row will be emitted. If the condition fails, the row will not be emitted.

    This filter also takes two additional optional boolean arguments – filterIfColumnMissing and setLatestVersionOnly

    If the filterIfColumnMissing flag is set to true the columns of the row will not be emitted if the specified column to check is not found in the row. The default value is false.

    If the setLatestVersionOnly flag is set to false, it will test previous versions (timestamps) too. The default value is true.

    These flags are optional and if you must set neither or both

    Syntax: SingleColumnValueFilter(<compare operator>, '<comparator>', '<family>', '<qualifier>', <filterIfColumnMissing_boolean>, <latest_version_boolean>)

    Syntax: SingleColumnValueFilter(<compare operator>, '<comparator>', '<family>', '<qualifier>)

    Example: "SingleColumnValueFilter (<=, 'abc','FamilyA', 'Column1', true, false)"

    Example: "SingleColumnValueFilter (<=, 'abc','FamilyA', 'Column1')"

17. **SingleColumnValueExcludeFilter**

    Description: This filter takes the same arguments and behaves same as SingleColumnValueFilter – however, if the column is found and the condition passes, all the columns of the row will be emitted except for the tested column value.

    Syntax: SingleColumnValueExcludeFilter(<compare operator>, '<comparator>', '<family>', '<qualifier>', <latest_version_boolean>, <filterIfColumnMissing_boolean>)

    Syntax: SingleColumnValueExcludeFilter(<compare operator>, '<comparator>', '<family>', '<qualifier>')

    Example: "SingleColumnValueExcludeFilter ('<=', 'abc','FamilyA', 'Column1', 'false', 'true')"

    Example: "SingleColumnValueExcludeFilter ('<=', 'abc', 'FamilyA', 'Column1')"

18. **ColumnRangeFilter**

    Description: This filter is used for selecting only those keys with columns that are between minColumn and maxColumn. It also takes two boolean variables to indicate whether to include the minColumn and maxColumn or not.

    If you don't want to set the minColumn or the maxColumn – you can pass in an empty argument.

    Syntax: ColumnRangeFilter ('<minColumn>', <minColumnInclusive_bool>, '<maxColumn>', <maxColumnInclusive_bool>)

    Example: "ColumnRangeFilter ('abc', true, 'xyz', false)"

10.4. C/C++ Apache HBase Client

Facebook的 Chip Turner 写了个纯 C/C++ 客户端。 Check it out.

Chapter 11. 性能调优

Table of Contents

11.1. 操作系统

11.1.1. 内存

RAM, RAM, RAM. 不要饿着 HBase.

11.1.2. 64-bit

使用 64-bit 平台(和64-bit JVM).

11.1.3. 交换区

小心交换，将交换区设为0。

11.2. 网络

也许，避免网络问题降低Hadoop和HBase性能的最重要因素就是所使用的交换机硬件。在项目范围内，集群大小翻倍或三倍甚至更多时，早期的决定可能导致主要的问题。

要考虑的重要事项:

- 设备交换机容量
- 系统连接数量
- 上行容量

11.2.1. 单交换机

单交换机配置最重要的因素，是硬件交换容量，所有系统连接到交换机产生的流量的处理能力。一些低价硬件商品，相对全交换机，具有较低交换能力。

11.2.2. 多交换机

多交换机在系统结构中是潜在陷阱。低价硬件的最常用配置是1Gbps上行连接到另一个交换机。 该常被忽略的窄点很容易成为集群通讯的瓶颈。特别是MapReduce任务通过该上行连接同时读写大量数据时，会导致饱和。

缓解该问题很简单，可以通过多种途径完成:

- 针对要创建的集群容量，采用合适硬件。
- 采用更大单交换机配置，如单48 端口相较2x 24 端口为优。
- 配置上行端口聚合(port trunking)来利用多网络接口增加交换机带宽。(译者注：port trunk：
  将交换机上的多个端口在物理上连接起来，在逻辑上捆绑在一起，形成一个拥有较大带宽的端口，组成一个干路，以达到平衡负载和提供备份线路，扩充带宽的E
)

11.2.3. 多机架

多机架配置带来多交换机同样的潜在问题。导致性能降低的原因主要来自两个方面:

- 较低的交换机容量性能
- 到其他机架的上行链路不足

如果机架上的交换机有合适交换容量，可以处理所有主机全速通信，那么下一个问题就是如何自动导航更多的交错在机架中的集群。最简单的避免横跨多机架问题的办法，是采用端口聚合来创建到其他机架的捆绑的上行的连接。然而该方法下行侧，是潜在被使用的端口开销。举例：从机架A到机架B创建 8Gbps 端口通道，采用24端口中的8个来和其他机架互通，ROI(投资回报率)很低。采用太少端口意味着不能从集群中传出最多的东西。

机架间采用10Gbe 链接将极大增加性能，确保交换机都支持10Gbe 上行连接或支持扩展卡，后者相对上行连接，允许你节省机器端口。

11.2.4. 网络接口

所有网络接口功能正常吗？你确定？参考故障诊断用例：Section 13.3.1, “Case Study #1 (Performance Issue On A Single Node)”.

可以从 wiki Performance Tuning看起。这个文档讲了一些主要的影响性能的方面:RAM, 压缩, JVM 设置, 等等。然后，可以看看下面的补充内容。

打开RPC-level日志

在区域服务器打开RPC-level的日志对于深度的优化是有好处的。一旦打开，日志将喷涌而出。所以不建议长时间打开，只能看一小段时间。要想启用RPC-level的职责，可以使用区域服务器 UI点击*Log Level*。将

org.apache.hadoop.ipc 的日志级别设为DEBUG。然后tail 区域服务器的日志，进行分析。

要想关闭，只要把日志级别设为INFO就可以了.

11.3. Java

11.3.1. 垃圾收集和Apache HBase

11.3.1.1. 长时间GC停顿

在这个PPT Avoiding Full GCs with MemStore-Local Allocation Buffers, Todd Lipcon描述了在HBase中常见的两种"世界停止"式的GC操作，尤其是在加载的时候。一种是CMS失败的模式(译者注:CMS是一种GC的算法)，另一种是老一代的堆碎片导致的。要想定位第一种，只要将CMS执行的时间提前就可以了，加入-XX:CMSInitiatingOccupancyFraction参数，把值调低。可以先从60%和70%开始(这个值调的越低，触发的GC次数就越多，消耗的CPU时间就越长)。要想定位第二种错误，Todd加入了一个实验性的功能，在HBase 0.90.x中这个是要明确指定的(在0.92.x中，这个是默认项)，将你的Configuration中的hbase.hregion.memstore.mslab.enabled设置为true。详细信息，可以看这个PPT. [27]. Be aware that when enabled, each MemStore instance will occupy at least an MSLAB instance of memory. If you have thousands of regions or lots of regions each with many column families, this allocation of MSLAB may be responsible for a good portion of your heap allocation and in an extreme case cause you to OOME. Disable MSLAB in this case, or lower the amount of memory it uses or float less regions per server.

GC日志的更多信息，参考 Section 12.2.3, "JVM 垃圾收集日志".

11.4. 配置

参见Section 2.8.2, "推荐的配置".

11.4.1. Regions的数目

HBase中region的数目可以根据Section 3.6.5, "更大的 Regions"调整.也可以参见 Section 12.3.1, "Region大小"

11.4.2. 管理紧缩

对于大型的系统，你需要考虑管理紧缩和分割

11.4.3. hbase.regionserver.handler.count

参见hbase.regionserver.handler.count.这个参数的本质是设置一个RegsionServer可以同时处理多少请求。 如果定的太高，吞吐量反而会降低;如果定的太低，请求会被阻塞，得不到响应。你可以打开RPC-level日志读Log，来决定对于你的集群什么值是合适的。(请求队列也是会消耗内存的)

11.4.4. hfile.block.cache.size

参见 hfile.block.cache.size. 对于区域服务器进程的内存设置。

11.4.5. hbase.regionserver.global.memstore.upperLimit

参见 hbase.regionserver.global.memstore.upperLimit. 这个内存设置是根据区域服务器的需要来设定。

11.4.6. hbase.regionserver.global.memstore.lowerLimit

参见 hbase.regionserver.global.memstore.lowerLimit. 这个内存设置是根据区域服务器的需要来设定。

11.4.7. hbase.hstore.blockingStoreFiles

参见hbase.hstore.blockingStoreFiles. 如果在区域服务器的Log中block,提高这个值是有帮助的。

11.4.8. hbase.hregion.memstore.block.multiplier

参见 hbase.hregion.memstore.block.multiplier. 如果有足够的RAM，提高这个值。

11.5. ZooKeeper

配置ZooKeeper信息，请参考 Section 2.5, "ZooKeeper" ，参看关于使用专用磁盘部分。

11.6. 模式设计

11.6.1. 列族的数目

参见 Section 6.2, " On the number of column families ".

11.6.2. 键和属性长度

参考 Section 6.3.2, "Try to minimize row and column sizes". 参考 Section 11.6.7.1, "However..." 获取压缩申请终止(compression caveats)

11.6.3. 表的区域大小

区域大小可以通过基于每张表设置，当某些表需要与缺省设置的区域大小不同时，通过 HTableDescriptor 的setFileSize 的事件设置。

参考 Section 11.4.1, "Number of Regions" 获取更多信息。

11.6.4. 布隆过滤(Bloom Filters)

布隆过滤可以每列族单独启用。使用 HColumnDescriptor.setBloomFilterType(NONE | ROW | ROWCOL) 对列族单独启用布隆。Default = NONE 没有布隆过滤。对 ROW，行键的哈希在每次插入行时将被添加到布隆。对 ROWCOL，行键 + 列族 + 列族修饰的哈希将在每次插入行时添加到布隆。

参考 HColumnDescriptor 和 Section 9.7.6, “布隆过滤(Bloom Filters)” 获取更多信息。

11.6.5. 列族块大小

The blocksize can be configured for each ColumnFamily in a table, and this defaults to 64k. Larger cell values require larger blocksizes. There is an inverse relationship between blocksize and the resulting StoreFile indexes (i.e., if the blocksize is doubled then the resulting indexes should be roughly halved).

参考 HColumnDescriptor and Section 9.7.5, “Store”获取更多信息。

11.6.6. 内存中的列族

ColumnFamilies can optionally be defined as in-memory. Data is still persisted to disk, just like any other ColumnFamily. In-memory blocks have the highest priority in the Section 9.6.4, “Block Cache”, but it is not a guarantee that the entire table will be in memory.

参考 HColumnDescriptor 获取更多信息。

11.6.7. 压缩

生产系统应该采用列族压缩定义。 参考 Appendix C, *Compression In HBase* 获取更多信息。

11.6.7.1. 然而...

Compression deflates data *on disk*. When it's in-memory (e.g., in the MemStore) or on the wire (e.g., transferring between RegionServer and Client) it's inflated. So while using ColumnFamily compression is a best practice, but it's not going to completely eliminate the impact of over-sized Keys, over-sized ColumnFamily names, or over-sized Column names.

参考 Section 6.3.2, “Try to minimize row and column sizes” on for schema design tips, and Section 9.7.5.4, “KeyValue” for more information on HBase stores data internally.

11.7. HBase 通用模式

11.7.1. 常量

人们刚开始使用HBase时，趋向于写如下的代码：

```
Get get = new Get(rowkey);  Result r = htable.get(get);  byte[] b = r.getValue(Bytes.toBytes("cf"), Bytes.toBytes("attr"));  // r
```

然而，特别是在循环内(和 MapReduce 工作内)，将列族和列名转为字节数组代价昂贵。最好使用字节数组常量，如下：

```
public static final byte[] CF = "cf".getBytes();  public static final byte[] ATTR = "attr".getBytes();  ...  Get get = new Get(ro
```

11.8. 写到 HBase

11.8.1. 批量装载

如果可以的话，尽量使用批量导入工具，参见 Section 9.8, “批量装载”.否则就要详细看看下面的内容。

11.8.2. 表创建: 预创建区域(Region)

默认情况下HBase创建表会新建一个区域。执行批量导入，意味着所有的client会写入这个区域，直到这个区域足够大，以至于分裂。一个有效的提高批量导入的性能的方式，是预创建空的区域。最好稍保守一点，因为过多的区域会实实在在的降低性能。下面是一个预创建区域的例子。(注意：这个例子里需要根据应用的key进行调整。):

```
public static boolean createTable(HBaseAdmin admin, HTableDescriptor table, byte[][] splits)
throws IOException {
  try {
    admin.createTable( table, splits );
    return true;
  } catch (TableExistsException e) {
    logger.info("table " + table.getNameAsString() + " already exists");
    // the table already exists...
    return false;
  }
}

public static byte[][] getHexSplits(String startKey, String endKey, int numRegions) {
  byte[][] splits = new byte[numRegions-1][];
  BigInteger lowestKey = new BigInteger(startKey, 16);
  BigInteger highestKey = new BigInteger(endKey, 16);
  BigInteger range = highestKey.subtract(lowestKey);
  BigInteger regionIncrement = range.divide(BigInteger.valueOf(numRegions));
  lowestKey = lowestKey.add(regionIncrement);
  for(int i=0; i < numRegions-1;i++) {
    BigInteger key = lowestKey.add(regionIncrement.multiply(BigInteger.valueOf(i)));
    byte[] b = String.format("%016x", key).getBytes();
    splits[i] = b;
  }
  return splits;
}
```

11.8.3. 表创建: 延迟log刷写

Puts的缺省行为使用 Write Ahead Log (WAL)，会导致 `HLog` 编辑立即写盘。如果采用延迟刷写，WAL编辑会保留在内存中，直到刷写周期来临。好处是集中和异步写HLog，潜在问题是如果RegionServer退出，没有刷写的日志将丢失。但这也比Puts时不使用WAL安全多了。

延迟log刷写可以通过 HTableDescriptor 在表上设置，`hbase.regionserver.optionallogflushinterval`缺省值是1000ms.

11.8.4. HBase 客户端: 自动刷写

当你进行大量的Put的时候，要确认你的HTable的setAutoFlush是关闭着的。否则的话，每执行一个Put就要想区域服务器发一个请求。通过 `htable.add(Put)` 和 `htable.add( <List> Put)`来将Put添加到写缓冲中。如果 `autoFlush = false`，要等到写缓冲都填满的时候才会发起请求。要想显式的发起请求，可以调用`flushCommits`。在`HTable`实例上进行的`close`操作也会发起`flushCommits`

11.8.5. HBase 客户端: 在Puts上关闭WAL

一个经常讨论的在`Puts`上增加吞吐量的选项是调用 `writeToWAL(false)`。关闭它意味着 RegionServer 不再将 `Put` 写到 Write Ahead Log, 仅写到内存。然而后果是如果出现 RegionServer 失败，将*导致数据丢失*。如果调用 `writeToWAL(false)` ，需保持高度警惕。你会发现实际上基本没有不同，如果你的负载很好的分布到集群中。

通常而言，最好对Puts使用WAL， 而增加负载吞吐量与使用 bulk loading 替代技术有关。

11.8.6. HBase 客户端: RegionServer 成组写入

In addition to using the writeBuffer, grouping `Puts` by RegionServer can reduce the number of client RPC calls per writeBuffer flush. There is a utility `HTableUtil` currently on TRUNK that does this, but you can either copy that or implement your own verison for those still on 0.90.x or earlier.

11.8.7. MapReduce: 跳过 Reducer

When writing a lot of data to an HBase table from a MR job (e.g., with TableOutputFormat), and specifically where Puts are being emitted from the Mapper, skip the Reducer step. When a Reducer step is used, all of the output (Puts) from the Mapper will get spooled to disk, then sorted/shuffled to other Reducers that will most likely be off-node. It's far more efficient to just write directly to HBase.

For summary jobs where HBase is used as a source and a sink, then writes will be coming from the Reducer step (e.g., summarize values then write out result). This is a different processing problem than from the the above case.

11.8.8. Anti-Pattern: One Hot Region

If all your data is being written to one region at a time, then re-read the section on processing timeseries data.

Also, if you are pre-splitting regions and all your data is *still* winding up in a single region even though your keys aren't monotonically increasing, confirm that your keyspace actually works with the split strategy. There are a variety of reasons that regions may appear "well split" but won't work with your data. As the HBase client communicates directly with the RegionServers, this can be obtained via HTable.getRegionLocation.

参考 Section 11.8.2, “ Table Creation: Pre-Creating Regions ”, as well as Section 11.4, “HBase Configurations”

11.9. 从HBase读

11.9.1. Scan 缓存

如果HBase的输入源是一个MapReduce Job，要确保输入的Scan的`setCaching`值要比默认值0要大。使用默认值就意味着map-task每一行都会去请求一下region-server。可以把这个值设为500，这样就可以一次传输500行。当然这也是需要权衡的，过大的值会同时消耗客户端和服务端很大的内存，不是越大越好。

11.9.1.1. Scan Caching in MapReduce Jobs

Scan settings in MapReduce jobs deserve special attention. Timeouts can result (e.g., UnknownScannerException) in Map tasks if it takes longer to process a batch of records before the client goes back to the RegionServer for the next set of data. This problem can occur because there is non-trivial processing occuring per row. If you process rows quickly, set caching higher. If you process rows more slowly (e.g., lots of transformations per row, writes), then set caching lower.

Timeouts can also happen in a non-MapReduce use case (i.e., single threaded HBase client doing a Scan), but the processing that is often performed in MapReduce jobs tends to exacerbate this issue.

11.9.2. Scan 属性选择

当Scan用来处理大量的行的时候(尤其是作为MapReduce的输入)，要注意的是选择了什么字段。如果调用了 `scan.addFamily`，这个列族的所有属性都会返回。如果只是想过滤其中的一小部分，就指定那几个column，否则就会造成很大浪费，影响性能。

11.9.3. MapReduce - 输入分割

For MapReduce jobs that use HBase tables as a source, if there a pattern where the "slow" map tasks seem to have the same Input Split (i.e., the RegionServer serving the data), see the Troubleshooting Case Study in Section 13.3.1, “Case Study #1 (Performance Issue On A Single Node)”.

11.9.4. 关闭 ResultScanners

这与其说是提高性能，倒不如说是避免发生性能问题。如果你忘记了关闭ResultScanners，会导致RegionServer出现问题。所以一定要把ResultScanner包含在try/catch 块中...

```
Scan scan = new Scan();
// set attrs...
ResultScanner rs = htable.getScanner(scan);
try {
  for (Result r = rs.next(); r != null; r = rs.next()) {
  // process result...
} finally {
  rs.close();  // always close the ResultScanner!
}
htable.close();
```

11.9.5. 块缓存

Scan实例可以在RegionServer中使用块缓存，可以由`setCacheBlocks`方法控制。如果Scan是MapReduce的输入源，要将这个值设置为`false`。对于经常读到的行，就建议使用块缓冲。

11.9.6. 行键的负载优化

当scan一个表的时候， 如果仅仅需要行键（不需要no families, qualifiers, values 和 timestamps）,在加入FilterList的时候，要使用Scanner的`setFilter`方法的时候，要填上`MUST_PASS_ALL`操作参数(译者注：相当于And操作符)。一个FilterList要包含一个FirstKeyOnlyFilter 和一个 KeyOnlyFilter.通过这样的filter组合，就算在最坏的情况下，RegionServer只会从磁盘读一个值，同时最小化客户端的网络带宽占用。

11.9.7. 并发: 监测数据扩散

当优化大量读取时，监测数据扩散到目标表。如果目标表含区域太少，读取时感觉像只有很少节点提供服务一样。

参考 Section 11.9.2, “ Table Creation: Pre-Creating Regions ”, 及 Section 11.4, “HBase Configurations”

11.9.8. 布隆过滤(Bloom Filters)

启用布隆过滤可以节省必须读磁盘过程，可以有助于改进读取延迟。

Bloom filters 在 HBase-1200 Add bloomfilters中开发。[28][29]

参考 Section 11.6.4, “布隆过滤(Bloom Filters)”.

11.9.8.1. Bloom StoreFile footprint

Bloom filters add an entry to the StoreFile general FileInfo data structure and then two extra entries to the StoreFilemetadata section.

11.9.8.1.1. BloomFilter in the StoreFile FileInfo data structure

FileInfo has a BLOOM_FILTER_TYPE entry which is set to NONE, ROW or ROWCOL.

11.9.8.1.2. BloomFilter entries in StoreFile metadata

BLOOM_FILTER_META holds Bloom Size, Hash Function used, etc. Its small in size and is cached on StoreFile.Reader load

BLOOM_FILTER_DATA is the actual bloomfilter data. Obtained on-demand. Stored in the LRU cache, if it is enabled (Its enabled by default).

11.9.8.2. 布隆过滤(Bloom Filter) 配置

11.9.8.2.1. io.hfile.bloom.enabled 全局杀死开关

配置文件的io.hfile.bloom.enabled 是一个当出错时的杀死开关。Default = true.

11.9.8.2.2. io.hfile.bloom.error.rate

io.hfile.bloom.error.rate = 平均误报率(average false positive rate). 缺省 = 1%. 降低率为 ½ (如 .5%) == +1 位每布隆入口。

11.9.8.2.3. io.hfile.bloom.max.fold

io.hfile.bloom.max.fold = 保证最小折叠速率(guaranteed minimum fold rate). 大多时候不要管. Default = 7, 或压缩到原来大小的至少 1/128. 想获取更多本选项的意义，参看本文档 *开发进程* 节 BloomFilters in HBase

11.10. 从HBase删除

11.10.1. 将 HBase 表当 Queues用

HBase tables are sometimes used as queues. In this case, special care must be taken to regularly perform major compactions on tables used in this manner. As is documented in Chapter 5, *Data Model*, marking rows as deleted creates additional StoreFiles which then need to be processed on reads. Tombstones only get cleaned up with major compactions.

参考 Section 9.7.5.5, “Compaction” 和 HBaseAdmin.majorCompact.

11.10.2. 删除的 RPC 行为

Be aware that htable.delete(Delete) doesn't use the writeBuffer. It will execute an RegionServer RPC with each invocation. For a large number of deletes, consider htable.delete(List).

参考 http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/HTable.html#delete%28org.apache.hadoop.hbase.client.Delete%29

11.11. HDFS

由于 HBase 在 Section 9.9, “HDFS” 上运行， it is important to understand how it works and how it affects HBase.

11.11.1. Current Issues With Low-Latency Reads

The original use-case for HDFS was batch processing. As such, there low-latency reads were historically not a priority. With the increased adoption of HBase this is changing, and several improvements are already in development. 参考 the Umbrella Jira Ticket for HDFS Improvements for HBase.

11.11.2. Leveraging local data

Since Hadoop 1.0.0 (also 0.22.1, 0.23.1, CDH3u3 and HDP 1.0) via HDFS-2246, it is possible for the DFSClient to take a "short circuit" and read directly from disk instead of going through the DataNode when the data is local. What this means for HBase is that the RegionServers can read directly off their machine's disks instead of having to open a socket to talk to the DataNode, the former being generally much faster[30]. Also see HBase, mail # dev - read short circuit thread for more discussion around short circuit reads.

To enable "short circuit" reads, you must set two configurations. First, the hdfs-site.xml needs to be amended. Set the property dfs.block.local-path-access.user to be the *only* user that can use the shortcut. This has to be the user that started HBase. Then in hbase-site.xml, set dfs.client.read.shortcircuit to be true

For optimal performance when short-circuit reads are enabled, it is recommended that HDFS checksums are disabled. To maintain data integrity with HDFS checksums disabled, HBase can be configured to write its own checksums into its datablocks and verify against these. See Section 11.4.9, “hbase.regionserver.checksum.verify”.

The DataNodes need to be restarted in order to pick up the new configuration. Be aware that if a process started under another username than the one configured here also has the shortcircuit enabled, it will get an Exception regarding an unauthorized access but the data will still be read.

11.11.3. Performance Comparisons of HBase vs. HDFS

A fairly common question on the dist-list is why HBase isn't as performant as HDFS files in a batch context (e.g., as a MapReduce source or sink). The short answer is that HBase is doing a lot more than HDFS (e.g., reading the KeyValues, returning the most current row or specified timestamps, etc.), and as such HBase is 4-5 times slower than HDFS in this processing context. Not that there isn't room for improvement (and this gap will, over time, be reduced), but HDFS will always be faster in this use-case.

11.12. Amazon EC2

Performance questions are common on Amazon EC2 environments because it is a shared environment. You will not see the same throughput as a dedicated server. In terms of running tests on EC2, run them several times for the same reason (i.e., it's a shared environment and you don't know what else is happening on the server).

If you are running on EC2 and post performance questions on the dist-list, please state this fact up-front that because EC2 issues are practically a separate class of performance issues.

11.13. Case Studies

For Performance and Troubleshooting Case Studies, see Chapter 13, *Case Studies*.

[27] The latest jvms do better regards fragmentation so make sure you are running a recent release. Read down in the message,Identifying concurrent mode failures caused by fragmentation.

[28] For description of the development process -- why static blooms rather than dynamic -- and for an overview of the unique properties that pertain to blooms in HBase, as well as possible future directions, see the *Development Process* section of the document BloomFilters in HBase attached to HBase-1200.

[29] The bloom filters described here are actually version two of blooms in HBase. In versions up to 0.19.x, HBase had a dynamic bloom option based on work done by the European Commission One-Lab Project 034819. The core of the HBase bloom work was later pulled up into Hadoop to implement org.apache.hadoop.io.BloomMapFile. Version 1 of HBase blooms never worked that well. Version 2 is a rewrite from scratch though again it starts with the one-lab work.

[30] See JD's Performance Talk

Chapter 12. HBase的故障排除和Debug

Table of Contents

12.1. 一般准则

总是先从主服务器的日志开始(TODO: 哪些行?)。通常情况下，他总是一行一行的重复信息。如果不是这样，说明有问题，可以Google或是用search-hadoop.com来搜索遇到的异常。

错误很少仅仅单独出现在HBase中，通常是某一个地方出了问题，引起各处大量异常和调用栈跟踪信息。遇到这样的错误，最好的办法是往上查日志，找到最初的异常。例如区域服务器会在退出的时候打印一些度量信息。Grep这个*转储*应该可以找到最初的异常信息。

区域服务器的自杀是很“正常”的。当一些事情发生错误的，他们就会自杀。如果ulimit和xcievers(最重要的两个设定，详见Section 2.2.5, “`ulimit` 和 `nproc`”)没有修改，HDFS将无法运转正常，在HBase看来，HDFS死掉了。假想一下，你的MySQL突然无法访问它的文件系统，他会怎么做。同样的事情会发生在HBase和HDFS上。还有一个造成区域服务器切腹自杀的常见的原因是，他们执行了一个长时间的GC操作，这个时间超过了ZooKeeper的会话时长。关于GC停顿的详细信息，参见Todd Lipcon的 3 part blog post by Todd Lipcon 和上面的 Section 11.3.1.1, “长时间GC停顿”.

12.2. Logs

重要日志的位置(<user>是启动服务的用户，<hostname> 是机器的名字)

NameNode: `$HADOOP_HOME/logs/hadoop-<user>-namenode-<hostname>.log`

DataNode: `$HADOOP_HOME/logs/hadoop-<user>-datanode-<hostname>.log`

JobTracker: `$HADOOP_HOME/logs/hadoop-<user>-jobtracker-<hostname>.log`

TaskTracker: `$HADOOP_HOME/logs/hadoop-<user>-jobtracker-<hostname>.log`

HMaster: `$HBASE_HOME/logs/hbase-<user>-master-<hostname>.log`

RegionServer: `$HBASE_HOME/logs/hbase-<user>-regionserver-<hostname>.log`

ZooKeeper: `TODO`

12.2.1. Log 位置

对于单节点模式，Log都会在一台机器上，但是对于生产环境，都会运行在一个集群上。

12.2.1.1. NameNode

NameNode的日志在NameNode server上。HBase Master 通常也运行在NameNode server上，ZooKeeper通常也是这样。

对于小一点的机器，JobTracker也通常运行在NameNode server上面。

12.2.1.2. DataNode

每一台DataNode server有一个HDFS的日志，Region有一个HBase日志。

每个DataNode server还有一份TaskTracker的日志，来记录MapReduce的Task信息。

12.2.2. 日志级别

12.2.2.1. 启用 RPC级别日志

Enabling the RPC-level logging on a RegionServer can often given insight on timings at the server. Once enabled, the amount of log spewed is voluminous. It is not recommended that you leave this logging on for more than short bursts of time. To enable RPC-level logging, browse to the RegionServer UI and click on *Log Level*. Set the log level to `DEBUG` for the package `org.apache.hadoop.ipc` (Thats right, for `hadoop.ipc`, NOT, `hbase.ipc`). Then tail the RegionServers log. Analyze.

To disable, set the logging level back to `INFO` level.

12.2.3. JVM 垃圾收集日志

HBase is memory intensive, and using the default GC you can see long pauses in all threads including the *Juliet Pause* aka "GC of Death". To help debug this or confirm this is happening GC logging can be turned on in the Java virtual machine.

To enable, in `hbase-env.sh` add:

```
export HBASE_OPTS="-XX:+UseConcMarkSweepGC -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -Xloggc:/home/hadoop/hbase/lo
```

Adjust the log directory to wherever you log. Note: The GC log does NOT roll automatically, so you'll have to keep an eye on it so it doesn't fill up the disk.

At this point you should see logs like so:

```
64898.952: [GC [1 CMS-initial-mark: 2811538K(3055704K)] 2812179K(3061272K), 0.0007360 secs] [Times: user=0.00 sys=0.00, real=0.0
64898.953: [CMS-concurrent-mark-start]
64898.971: [GC 64898.971: [ParNew: 5567K->576K(5568K), 0.0101110 secs] 2817105K->2812715K(3061272K), 0.0102200 secs] [Times: use
```

In this section, the first line indicates a 0.0007360 second pause for the CMS to initially mark. This pauses the entire VM, all threads for that period of time.

The third line indicates a "minor GC", which pauses the VM for 0.0101110 seconds - aka 10 milliseconds. It has reduced the "ParNew" from about 5.5m to 576k. Later on in this cycle we see:

```
64901.445: [CMS-concurrent-mark: 1.542/2.492 secs] [Times: user=10.49 sys=0.33, real=2.49 secs]
64901.445: [CMS-concurrent-preclean-start]
64901.453: [GC 64901.453: [ParNew: 5505K->573K(5568K), 0.0062440 secs] 2868746K->2864292K(3061272K), 0.0063360 secs] [Times: use
64901.476: [GC 64901.476: [ParNew: 5563K->575K(5568K), 0.0072510 secs] 2869283K->2864837K(3061272K), 0.0073320 secs] [Times: use
64901.500: [GC 64901.500: [ParNew: 5517K->573K(5568K), 0.0120390 secs] 2869780K->2865267K(3061272K), 0.0121150 secs] [Times: use
64901.529: [GC 64901.529: [ParNew: 5507K->569K(5568K), 0.0086240 secs] 2870200K->2865742K(3061272K), 0.0087180 secs] [Times: use
64901.554: [GC 64901.555: [ParNew: 5516K->575K(5568K), 0.0107130 secs] 2870689K->2866291K(3061272K), 0.0107820 secs] [Times: use
64901.578: [CMS-concurrent-preclean: 0.070/0.133 secs] [Times: user=0.48 sys=0.01, real=0.14 secs]
64901.578: [CMS-concurrent-abortable-preclean-start]
64901.584: [GC 64901.584: [ParNew: 5504K->571K(5568K), 0.0087270 secs] 2871220K->2866830K(3061272K), 0.0088220 secs] [Times: use
64901.609: [GC 64901.609: [ParNew: 5512K->569K(5568K), 0.0063370 secs] 2871771K->2867322K(3061272K), 0.0064230 secs] [Times: use
64901.615: [CMS-concurrent-abortable-preclean: 0.007/0.037 secs] [Times: user=0.13 sys=0.00, real=0.03 secs]
64901.616: [GC[YG occupancy: 645 K (5568 K)]64901.616: [Rescan (parallel) , 0.0020210 secs]64901.618: [weak refs processing, 0.0
64901.621: [CMS-concurrent-sweep-start]
```

The first line indicates that the CMS concurrent mark (finding garbage) has taken 2.4 seconds. But this is a _concurrent_ 2.4 seconds, Java has not been paused at any point in time.

There are a few more minor GCs, then there is a pause at the 2nd last line:

```
64901.616: [GC[YG occupancy: 645 K (5568 K)]64901.616: [Rescan (parallel) , 0.0020210 secs]64901.618: [weak refs processing, 0.0
```

The pause here is 0.0049380 seconds (aka 4.9 milliseconds) to 'remark' the heap.

At this point the sweep starts, and you can watch the heap size go down:

```
64901.637: [GC 64901.637: [ParNew: 5501K->569K(5568K), 0.0097350 secs] 2871958K->2867441K(3061272K), 0.0098370 secs] [Times: use
...  lines removed ...
64904.936: [GC 64904.936: [ParNew: 5532K->568K(5568K), 0.0070720 secs] 1365024K->1360689K(3061272K), 0.0071930 secs] [Times: use
64904.953: [CMS-concurrent-sweep: 2.030/3.332 secs] [Times: user=9.57 sys=0.26, real=3.33 secs]
```

At this point, the CMS sweep took 3.332 seconds, and heap went from about ~ 2.8 GB to 1.3 GB (approximate).

The key points here is to keep all these pauses low. CMS pauses are always low, but if your ParNew starts growing, you can see minor GC pauses approach 100ms, exceed 100ms and hit as high at 400ms.

This can be due to the size of the ParNew, which should be relatively small. If your ParNew is very large after running HBase for a while, in one example a ParNew was about 150MB, then you might have to constrain the size of ParNew (The larger it is, the longer the collections take but if its too small, objects are promoted to old gen too quickly). In the below we constrain new gen size to 64m.

Add this to HBASE_OPTS:

```
export HBASE_OPTS="-XX:NewSize=64m -XX:MaxNewSize=64m <cms options from above> <gc logging options from above>"
```

For more information on GC pauses, see the 3 part blog post by Todd Lipcon and Section 11.3.1.1, "Long GC pauses" above.

12.3. 资源

12.3.1. search-hadoop.com

search-hadoop.com 索引了全部邮件列表，很适合做历史检索。有问题时先在这里查询，因为别人可能已经遇到过你的问题。

12.3.2. 邮件列表

Ask a question on the HBase mailing lists. The 'dev' mailing list is aimed at the community of developers actually building HBase and for features currently under development, and 'user' is generally used for questions on released versions of HBase. Before going to the mailing list, make sure your question has not already been answered by searching the mailing list archives first. Use Section 12.3.1, "search-hadoop.com". Take some time crafting your question[31]; a quality question that includes all context and exhibits evidence the author has tried to find answers in the manual and out on lists is more likely to get a prompt response.

12.3.3. IRC

#hbase on irc.freenode.net

12.3.4. JIRA

JIRA 在处理 Hadoop/HBase相关问题时也很有帮助。

12.4. 工具

12.4.1. 内置工具

12.4.1.1. 主服务器Web接口

主服务器启动了一个缺省端口是 60010的web接口。

The Master web UI lists created tables and their definition (e.g., ColumnFamilies, blocksize, etc.). Additionally, the available RegionServers in the cluster are listed along with selected high-level metrics (requests, number of regions, usedHeap, maxHeap). The Master web UI allows navigation to each RegionServer's web UI.

12.4.1.2. 区域服务器Web接口

区域服务器启动了一个缺省端口是 60030的web接口。

The RegionServer web UI lists online regions and their start/end keys, as well as point-in-time RegionServer metrics (requests, regions, storeFileIndexSize, compactionQueueSize, etc.).

参考 Section 14.4, “HBase Metrics” 获取更多度量信息。

12.4.1.3. zkcli

zkcli是一个研究 ZooKeeper相关问题的有用工具。调用:

```
./hbase zkcli -server host:port <cmd> <args>
```

命令 (和参数) :

```
        connect host:port
        get path [watch]
        ls path [watch]
        set path data [version]
        delquota [-n|-b] path
        quit
        printwatches on|off
        create [-s] [-e] path data acl
        stat path [watch]
        close
        ls2 path [watch]
        history
        listquota path
        setAcl path acl
        getAcl path
        sync path
        redo cmdno
        addauth scheme auth
        delete path [version]
        setquota -n|-b val path
```

12.4.2. 外部工具

12.4.2.1 tail

tail是一个命令行工具，可以用来看日志的尾巴。加入的"-f"参数后，就会在数据更新的时候自己刷新。用它来看日志很方便。例如，一个机器需要花很多时间来启动或关闭，你可以tail他的master log(也可以是region server的log)。

12.4.2.2 top

top是一个很重要的工具来看你的机器各个进程的资源占用情况。下面是一个生产环境的例子:

```
top - 14:46:59 up 39 days, 11:55,  1 user,  load average: 3.75, 3.57, 3.84
Tasks: 309 total,   1 running, 308 sleeping,   0 stopped,   0 zombie
Cpu(s):  4.5%us,  1.6%sy,  0.0%ni, 91.7%id,  1.4%wa,  0.1%hi,  0.6%si,  0.0%st
Mem:  24414432k total, 24296956k used,   117476k free,     7196k buffers
Swap: 16008732k total,  14348k used, 15994384k free, 11106908k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM     TIME+  COMMAND
15558 hadoop    18  -2 3292m 2.4g 3556 S   79 10.4   6523:52 java
13268 hadoop    18  -2 8967m 8.2g 4104 S   21 35.1   5170:30 java
 8895 hadoop    18  -2 1581m 497m 3420 S   11  2.1   4002:32 java
…
```

这里你可以看到系统的load average在最近5分钟是3.75，意思就是说这5分钟里面平均有3.75个线程在CPU时间的等待队列里面。通常来说，最完美的情况是这个值和CPU和核数相等，比这个值低意味着资源闲置，比这个值高就是过载了。这是一个重要的概念，要想理解的更多，可以看这篇文章 http://www.linuxjournal.com/article/9001.

处理负载，我们可以看到系统已经几乎使用了他的全部RAM，其中大部分都是用于OS cache(这是一件好事).Swap只使用了一点点KB,这正是我们期望的，如果数值很高的话，就意味着在进行交换，这对Java程序的性能是致命的。另一种检测交换的方法是看Load average是否过高(load average过高还可能是磁盘损坏或者其它什么原因导致的)。

默认情况下进程列表不是很有用，我们可以看到3个Java进程使用了111%的CPU。要想知道哪个进程是什么，可以输入"c"，每一行就会扩展信息。输入“1”可以显示CPU的每个核的具体状况。

12.4.2.3 jps

`jps`是JDK集成的一个工具，可以用来看当前用户的Java进程id。(如果是root,可以看到所有用户的id)，例如:

```
hadoop@sv4borg12:~$ jps
1322 TaskTracker
17789 HRegionServer
27862 Child
1158 DataNode
25115 HQuorumPeer
2950 Jps
19750 ThriftServer
18776 jmx
```

按顺序看

- Hadoop TaskTracker,管理本地的Task
- HBase RegionServer,提供region的服务
- Child, 一个 MapReduce task,无法看出详细类型
- Hadoop DataNode, 管理blocks
- HQuorumPeer, ZooKeeper集群的成员
- Jps, 就是这个进程
- ThriftServer, 当thrif启动后，就会有这个进程
- jmx, 这个是本地监控平台的进程。你可以不用这个。

你可以看到这个进程启动是全部命令行信息。

```
hadoop@sv4borg12:~$ ps aux | grep HRegionServer
hadoop   17789  155 35.2 9067824 8604364 ?      S<l  Mar04 9855:48 /usr/java/jdk1.6.0_14/bin/java -Xmx8000m -XX:+DoEscapeAnalysis
```

12.4.2.4 jstack

`jstack` 是一个最重要(除了看Log)的java工具，可以看到具体的Java进程的在做什么。可以先用Jps看到进程的Id,然后就可以用jstack。他会按线程的创建顺序显示线程的列表，还有这个线程在做什么。下面是例子：

这个主线程是一个RegionServer正在等master返回什么信息。

```
      "regionserver60020" prio=10 tid=0x0000000040ab4000 nid=0x45cf waiting on condition [0x00007f16b6a96000..0x00007f16b6a96a70
   java.lang.Thread.State: TIMED_WAITING (parking)
                at sun.misc.Unsafe.park(Native Method)
                - parking to wait for  <0x00007f16cd5c2f30> (a java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionOb
                at java.util.concurrent.locks.LockSupport.parkNanos(LockSupport.java:198)
                at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.awaitNanos(AbstractQueuedSynchronizer.j
                at java.util.concurrent.LinkedBlockingQueue.poll(LinkedBlockingQueue.java:395)
                at org.apache.hadoop.hbase.regionserver.HRegionServer.run(HRegionServer.java:647)
                at java.lang.Thread.run(Thread.java:619)

                The MemStore flusher thread that is currently flushing to a file:
"regionserver60020.cacheFlusher" daemon prio=10 tid=0x0000000040f4e000 nid=0x45eb in Object.wait() [0x00007f16b5b86000..0x00007f
   java.lang.Thread.State: WAITING (on object monitor)
                at java.lang.Object.wait(Native Method)
                at java.lang.Object.wait(Object.java:485)
                at org.apache.hadoop.ipc.Client.call(Client.java:803)
                - locked <0x00007f16cb14b3a8> (a org.apache.hadoop.ipc.Client$Call)
                at org.apache.hadoop.ipc.RPC$Invoker.invoke(RPC.java:221)
                at $Proxy1.complete(Unknown Source)
                at sun.reflect.GeneratedMethodAccessor38.invoke(Unknown Source)
                at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
                at java.lang.reflect.Method.invoke(Method.java:597)
                at org.apache.hadoop.io.retry.RetryInvocationHandler.invokeMethod(RetryInvocationHandler.java:82)
                at org.apache.hadoop.io.retry.RetryInvocationHandler.invoke(RetryInvocationHandler.java:59)
                at $Proxy1.complete(Unknown Source)
                at org.apache.hadoop.hdfs.DFSClient$DFSOutputStream.closeInternal(DFSClient.java:3390)
                - locked <0x00007f16cb14b470> (a org.apache.hadoop.hdfs.DFSClient$DFSOutputStream)
                at org.apache.hadoop.hdfs.DFSClient$DFSOutputStream.close(DFSClient.java:3304)
                at org.apache.hadoop.fs.FSDataOutputStream$PositionCache.close(FSDataOutputStream.java:61)
                at org.apache.hadoop.fs.FSDataOutputStream.close(FSDataOutputStream.java:86)
                at org.apache.hadoop.hbase.io.hfile.HFile$Writer.close(HFile.java:650)
                at org.apache.hadoop.hbase.regionserver.StoreFile$Writer.close(StoreFile.java:853)
                at org.apache.hadoop.hbase.regionserver.Store.internalFlushCache(Store.java:467)
                - locked <0x00007f16d00e6f08> (a java.lang.Object)
                at org.apache.hadoop.hbase.regionserver.Store.flushCache(Store.java:427)
                at org.apache.hadoop.hbase.regionserver.Store.access$100(Store.java:80)
                at org.apache.hadoop.hbase.regionserver.Store$StoreFlusherImpl.flushCache(Store.java:1359)
                at org.apache.hadoop.hbase.regionserver.HRegion.internalFlushcache(HRegion.java:907)
                at org.apache.hadoop.hbase.regionserver.HRegion.internalFlushcache(HRegion.java:834)
                at org.apache.hadoop.hbase.regionserver.HRegion.flushcache(HRegion.java:786)
                at org.apache.hadoop.hbase.regionserver.MemStoreFlusher.flushRegion(MemStoreFlusher.java:250)
                at org.apache.hadoop.hbase.regionserver.MemStoreFlusher.flushRegion(MemStoreFlusher.java:224)
                at org.apache.hadoop.hbase.regionserver.MemStoreFlusher.run(MemStoreFlusher.java:146)
```

一个处理线程是在等一些东西(例如put, delete, scan...):

```
"IPC Server handler 16 on 60020" daemon prio=10 tid=0x00007f16b011d800 nid=0x4a5e waiting on condition [0x00007f16afefd000..0x00
   java.lang.Thread.State: WAITING (parking)
                at sun.misc.Unsafe.park(Native Method)
                - parking to wait for  <0x00007f16cd3f8dd8> (a java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionOb
                at java.util.concurrent.locks.LockSupport.park(LockSupport.java:158)
                at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.await(AbstractQueuedSynchronizer.java:1
                at java.util.concurrent.LinkedBlockingQueue.take(LinkedBlockingQueue.java:358)
                at org.apache.hadoop.hbase.ipc.HBaseServer$Handler.run(HBaseServer.java:1013)
```

有一个线程正在忙，在递增一个counter(这个阶段是正在创建一个scanner来读最新的值):

```
"IPC Server handler 66 on 60020" daemon prio=10 tid=0x00007f16b006e800 nid=0x4a90 runnable [0x00007f16acb77000..0x00007f16acb77c
   java.lang.Thread.State: RUNNABLE
                at org.apache.hadoop.hbase.regionserver.KeyValueHeap.<init>(KeyValueHeap.java:56)
                at org.apache.hadoop.hbase.regionserver.StoreScanner.<init>(StoreScanner.java:79)
                at org.apache.hadoop.hbase.regionserver.Store.getScanner(Store.java:1202)
                at org.apache.hadoop.hbase.regionserver.HRegion$RegionScanner.<init>(HRegion.java:2209)
                at org.apache.hadoop.hbase.regionserver.HRegion.instantiateInternalScanner(HRegion.java:1063)
                at org.apache.hadoop.hbase.regionserver.HRegion.getScanner(HRegion.java:1055)
                at org.apache.hadoop.hbase.regionserver.HRegion.getScanner(HRegion.java:1039)
                at org.apache.hadoop.hbase.regionserver.HRegion.getLastIncrement(HRegion.java:2875)
                at org.apache.hadoop.hbase.regionserver.HRegion.incrementColumnValue(HRegion.java:2978)
                at org.apache.hadoop.hbase.regionserver.HRegionServer.incrementColumnValue(HRegionServer.java:2433)
                at sun.reflect.GeneratedMethodAccessor20.invoke(Unknown Source)
                at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
                at java.lang.reflect.Method.invoke(Method.java:597)
                at org.apache.hadoop.hbase.ipc.HBaseRPC$Server.call(HBaseRPC.java:560)
                at org.apache.hadoop.hbase.ipc.HBaseServer$Handler.run(HBaseServer.java:1027)
```

还有一个线程在从HDFS获取数据。

```
"IPC Client (47) connection to sv4borg9/10.4.24.40:9000 from hadoop" daemon prio=10 tid=0x00007f16a02d0000 nid=0x4fa3 runnable [
   java.lang.Thread.State: RUNNABLE
                at sun.nio.ch.EPollArrayWrapper.epollWait(Native Method)
                at sun.nio.ch.EPollArrayWrapper.poll(EPollArrayWrapper.java:215)
                at sun.nio.ch.EPollSelectorImpl.doSelect(EPollSelectorImpl.java:65)
                at sun.nio.ch.SelectorImpl.lockAndDoSelect(SelectorImpl.java:69)
                - locked <0x00007f17d5b68c00> (a sun.nio.ch.Util$1)
                - locked <0x00007f17d5b68be8> (a java.util.Collections$UnmodifiableSet)
                - locked <0x00007f1877959b50> (a sun.nio.ch.EPollSelectorImpl)
                at sun.nio.ch.SelectorImpl.select(SelectorImpl.java:80)
                at org.apache.hadoop.net.SocketIOWithTimeout$SelectorPool.select(SocketIOWithTimeout.java:332)
                at org.apache.hadoop.net.SocketIOWithTimeout.doIO(SocketIOWithTimeout.java:157)
                at org.apache.hadoop.net.SocketInputStream.read(SocketInputStream.java:155)
                at org.apache.hadoop.net.SocketInputStream.read(SocketInputStream.java:128)
                at java.io.FilterInputStream.read(FilterInputStream.java:116)
                at org.apache.hadoop.ipc.Client$Connection$PingInputStream.read(Client.java:304)
                at java.io.BufferedInputStream.fill(BufferedInputStream.java:218)
                at java.io.BufferedInputStream.read(BufferedInputStream.java:237)
                - locked <0x00007f1808539178> (a java.io.BufferedInputStream)
                at java.io.DataInputStream.readInt(DataInputStream.java:370)
                at org.apache.hadoop.ipc.Client$Connection.receiveResponse(Client.java:569)
                at org.apache.hadoop.ipc.Client$Connection.run(Client.java:477)
```

这里是一个RegionServer死了，master正在试着恢复。

```
"LeaseChecker" daemon prio=10 tid=0x00000000407ef800 nid=0x76cd waiting on condition [0x00007f6d0eae2000..0x00007f6d0eae2a70]
--
   java.lang.Thread.State: WAITING (on object monitor)
                at java.lang.Object.wait(Native Method)
                at java.lang.Object.wait(Object.java:485)
                at org.apache.hadoop.ipc.Client.call(Client.java:726)
                - locked <0x00007f6d1cd28f80> (a org.apache.hadoop.ipc.Client$Call)
                at org.apache.hadoop.ipc.RPC$Invoker.invoke(RPC.java:220)
                at $Proxy1.recoverBlock(Unknown Source)
                at org.apache.hadoop.hdfs.DFSClient$DFSOutputStream.processDatanodeError(DFSClient.java:2636)
                at org.apache.hadoop.hdfs.DFSClient$DFSOutputStream.<init>(DFSClient.java:2832)
                at org.apache.hadoop.hdfs.DFSClient.append(DFSClient.java:529)
                at org.apache.hadoop.hdfs.DistributedFileSystem.append(DistributedFileSystem.java:186)
                at org.apache.hadoop.fs.FileSystem.append(FileSystem.java:530)
                at org.apache.hadoop.hbase.util.FSUtils.recoverFileLease(FSUtils.java:619)
                at org.apache.hadoop.hbase.regionserver.wal.HLog.splitLog(HLog.java:1322)
                at org.apache.hadoop.hbase.regionserver.wal.HLog.splitLog(HLog.java:1210)
                at org.apache.hadoop.hbase.master.HMaster.splitLogAfterStartup(HMaster.java:648)
                at org.apache.hadoop.hbase.master.HMaster.joinCluster(HMaster.java:572)
                at org.apache.hadoop.hbase.master.HMaster.run(HMaster.java:503)
```

12.4.2.5 OpenTSDB

OpenTSDB是一个Ganglia的很好的替代品，因为他使用HBase来存储所有的时序而不需要采样。使用OpenTSDB来监控你的HBase是一个很好的实践

这里有一个例子，集群正在同时进行上百个紧缩，严重影响了IO性能。(TODO: 在这里插入compactionQueueSize的图片)

给集群构建一个图表监控是一个很好的实践。包括集群和每台机器。这样就可以快速定位到问题。例如，在StumbleUpon，每个机器有一个图表监控，包括OS和HBase，涵盖所有的重要的信息。你也可以登录到机器上，获取更多的信息。

12.4.2.6 clusterssh+top

clusterssh+top,感觉是一个穷人用的监控系统，但是他确实很有效，当你只有几台机器的是，很好设置。启动clusterssh后，你就会每台机器有个终端，还有一个终端，你在这个终端的操作都会反应到其他的每一个终端上。 这就意味着，你在一天机器执行“top”,集群中的所有机器都会给你全部的top信息。你还可以这样tail全部的log，等等。

12.5. 客户端

HBase 客户端的更多信息， 参考 Section 9.3, “Client”.

12.5.1. ScannerTimeoutException 或 UnknownScannerException

当从客户端到RegionServer的RPC请求超时。例如如果Scan.setCacheing的值设置为500，RPC请求就要去获取500行的数据，每500次`.next()`操作获取一次。因为数据是以大块的形式传到客户端的，就可能造成超时。将这个 serCacheing的值调小是一个解决办法，但是这个值要是设的太小就会影响性能。

参考 Section 11.10.1, “Scan Caching”.

12.5.2. 普通操作时，Shell 或客户端应用抛出很多不太重要的异常

Since 0.20.0 the default log level for `org.apache.hadoop.hbase.*`is DEBUG.

On your clients, edit `$HBASE_HOME/conf/log4j.properties` and change this: `log4j.logger.org.apache.hadoop.hbase=DEBUG` to this: `log4j.logger.org.apache.hadoop.hbase=INFO`, or even `log4j.logger.org.apache.hadoop.hbase=WARN`.

12.5.3. 压缩时客户端长时暂停

这是一个在HBase区列表中经常被问的问题。场景一般是客户端正在插入大量数据到相对未优化的HBase集群中时发生。压缩正好加重暂停，尽管这不是问题源头。

参考 Section 11.10.2, “ Table Creation: Pre-Creating Regions ” ，关于预先创建区域的模式部分，并确认表没有在单个区域中启动。

参考 Section 11.4, “HBase Configurations” 获取集群配置相关信息, 特别是 `hbase.hstore.blockingStoreFiles`, `hbase.hregion.memstore.block.multiplier`, `MAX_FILESIZE` (region size), 和 `MEMSTORE_FLUSHSIZE`.

A slightly longer explanation of why pauses can happen is as follows: Puts are sometimes blocked on the MemStores which are blocked by the flusher thread which is blocked because there are too many files to compact because the compactor is given too many small files to compact and has to compact the same data repeatedly. This situation can occur even with minor compactions. Compounding this situation, HBase doesn't compress data in memory. Thus, the 64MB that lives in the MemStore could become a 6MB file after compression - which results in a smaller StoreFile. The upside is that more data is packed into the same region, but performance is achieved by being able to write larger files - which is why HBase waits until the flushize before writing a new StoreFile. And smaller StoreFiles become targets for compaction. Without compression the files are much bigger and don't need as much compaction, however this is at the expense of I/O.

For additional information, see this thread on Long client pauses with compression.

12.5.4. ZooKeeper 客户端连接错误

错误类似于...

```
11/07/05 11:26:41 WARN zookeeper.ClientCnxn: Session 0x0 for server null,
 unexpected error, closing socket connection and attempting reconnect
 java.net.ConnectException: Connection refused: no further information
        at sun.nio.ch.SocketChannelImpl.checkConnect(Native Method)
        at sun.nio.ch.SocketChannelImpl.finishConnect(Unknown Source)
        at org.apache.zookeeper.ClientCnxn$SendThread.run(ClientCnxn.java:1078)
 11/07/05 11:26:43 INFO zookeeper.ClientCnxn: Opening socket connection to
 server localhost/127.0.0.1:2181
 11/07/05 11:26:44 WARN zookeeper.ClientCnxn: Session 0x0 for server null,
 unexpected error, closing socket connection and attempting reconnect
 java.net.ConnectException: Connection refused: no further information
        at sun.nio.ch.SocketChannelImpl.checkConnect(Native Method)
        at sun.nio.ch.SocketChannelImpl.finishConnect(Unknown Source)
        at org.apache.zookeeper.ClientCnxn$SendThread.run(ClientCnxn.java:1078)
 11/07/05 11:26:45 INFO zookeeper.ClientCnxn: Opening socket connection to
 server localhost/127.0.0.1:2181
```

... 要么是 ZooKeeper 不在了，或网络不可达问题。

工具 Section 12.4.1.3, “zkcli” 可以帮助调查 ZooKeeper 问题。

12.5.5. 客户端内存耗尽，但堆大小看起来不太变化(off-heap/direct heap 在增长)

You are likely running into the issue that is described and worked through in the mail thread HBase, mail # user - Suspected memory leak and continued over in HBase, mail # dev - FeedbackRe: Suspected memory leak. A workaround is passing your client-side JVM a reasonable value for `-XX:MaxDirectMemorySize`. By default, the `MaxDirectMemorySize` is equal to your `-Xmx` max heapsize setting (if `-Xmx` is set). Try seting it to something smaller (for example, one user had success setting it to `1g` when they had a client-side heap of `12g`). If you set it too small, it will bring on `FullGCs` so keep it a bit hefty. You want to make this setting client-side only especially if you are running the new experiemental server-side off-heap cache since this feature depends on being able to use big direct buffers (You may have to keep separate client-side and server-side config dirs).

12.5.6. 客户端变慢，在调用管理方法(flush, compact, 等)时发生

该客户端问题在 HBASE-5073 版本 0.90.6中修订。 客户端的ZooKeeper 内存泄露，而客户端被管理API的额外调用产生的ZooKeeper事件连续调用。

12.5.7. 安全客户端不能连接 ([由 GSSException 引起: No valid credentials provided (Mechanism level: Failed to find any Kerberos tgt)])

There can be several causes that produce this symptom.

First, check that you have a valid Kerberos ticket. One is required in order to set up communication with a secure Apache HBase cluster. Examine the ticket currently in the credential cache, if any, by running the klist command line utility. If no ticket is listed, you must obtain a ticket by running the kinit command with either a keytab specified, or by interactively entering a password for the desired principal.

Then, consult the Java Security Guide troubleshooting section. The most common problem addressed there is resolved by setting javax.security.auth.useSubjectCredsOnly system property value to false.

Because of a change in the format in which MIT Kerberos writes its credentials cache, there is a bug in the Oracle JDK 6 Update 26 and earlier that causes Java to be unable to read the Kerberos credentials cache created by versions of MIT Kerberos 1.8.1 or higher. If you have this problematic combination of components in your environment, to work around this problem, first log in with kinit and then immediately refresh the credential cache with kinit -R. The refresh will rewrite the credential cache without the problematic formatting.

Finally, depending on your Kerberos configuration, you may need to install the Java Cryptography Extension, or JCE. Insure the JCE jars are on the classpath on both server and client systems.

You may also need to download the unlimited strength JCE policy files. Uncompress and extract the downloaded file, and install the policy jars into <java-home>/lib/security.

12.6. MapReduce

12.6.1. 你认为自己在用集群, 实际上你在用本地(Local)

如下的调用栈在使用 ImportTsv时发生，但同样的事可以在错误配置的任何任务中发生。

```
    WARN mapred.LocalJobRunner: job_local_0001
java.lang.IllegalArgumentException: Can't read partitions file
       at org.apache.hadoop.hbase.mapreduce.hadoopbackport.TotalOrderPartitioner.setConf(TotalOrderPartitioner.java:111)
       at org.apache.hadoop.util.ReflectionUtils.setConf(ReflectionUtils.java:62)
       at org.apache.hadoop.util.ReflectionUtils.newInstance(ReflectionUtils.java:117)
       at org.apache.hadoop.mapred.MapTask$NewOutputCollector.<init>(MapTask.java:560)
       at org.apache.hadoop.mapred.MapTask.runNewMapper(MapTask.java:639)
       at org.apache.hadoop.mapred.MapTask.run(MapTask.java:323)
       at org.apache.hadoop.mapred.LocalJobRunner$Job.run(LocalJobRunner.java:210)
Caused by: java.io.FileNotFoundException: File _partition.lst does not exist.
       at org.apache.hadoop.fs.RawLocalFileSystem.getFileStatus(RawLocalFileSystem.java:383)
       at org.apache.hadoop.fs.FilterFileSystem.getFileStatus(FilterFileSystem.java:251)
       at org.apache.hadoop.fs.FileSystem.getLength(FileSystem.java:776)
       at org.apache.hadoop.io.SequenceFile$Reader.<init>(SequenceFile.java:1424)
       at org.apache.hadoop.io.SequenceFile$Reader.<init>(SequenceFile.java:1419)
       at org.apache.hadoop.hbase.mapreduce.hadoopbackport.TotalOrderPartitioner.readPartitions(TotalOrderPartitioner.java:296)
```

.. 看到调用栈的关键部分了吗？就是...

```
       at org.apache.hadoop.mapred.LocalJobRunner$Job.run(LocalJobRunner.java:210)
```

LocalJobRunner 意思就是任务跑在本地，不在集群。

参考 http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/mapreduce/package-summary.html#classpath for more information on HBase MapReduce jobs and classpaths.

12.7. NameNode

NameNode 更多信息, 参考 Section 9.9, “HDFS”.

12.7.1. 表和区域的HDFS 工具

要确定HBase 用了HDFS多大空间，可在NameNode使用 hadoop shell命令，例如...

```
hadoop fs -dus /hbase/
```

...返回全部HBase对象磁盘占用的情况。

```
hadoop fs -dus /hbase/myTable
```

...返回HBase表'myTable'磁盘占用的情况。

```
hadoop fs -du /hbase/myTable
```

...返回HBase的'myTable'表的各区域列表的磁盘占用情况。

更多关于 HDFS shell 命令的信息，参考 HDFS 文件系统 Shell 文档.

12.7.2. 浏览 HDFS ，查看 HBase 对象

有时需要浏览HDFS上的 HBase对象 。对象包括WALs (Write Ahead Logs), 表，区域，存储文件等。最简易的方法是在NameNode web应用中查看，端口 50070。NameNode web 应用提供到集群中所有 DataNode 的链接，可以无缝浏览。

存储在HDFS集群中的HBase表的目录结构是...

```
/hbase
     /<Table>             (Tables in the cluster)
          /<Region>           (Regions for the table)
               /<ColumnFamiy>      (ColumnFamilies for the Region for the table)
                    /<StoreFile>        (StoreFiles for the ColumnFamily for the Regions for the table)
```

HDFS 中的HBase WAL目录结构是..

```
/hbase
     /.logs
          /<RegionServer>    (RegionServers)
               /<HLog>           (WAL HLog files for the RegionServer)
```

参考HDFS User Guide 获取其他非Shell诊断工具如fsck.

12.7.2.1. 用例

查询HDFS，获取HBase对象的两个通常用例是研究未紧缩表的程度。如果每个列族有大量存储文件(StoreFile)，这表示需要主紧缩。另外，在主紧缩之后，如果存储文件的比较小，意味着表的列族要减少。

12.8. 网络

12.8.1. 网络峰值(Network Spikes)

如果看到周期性网络峰值，你可能需要检查compactionQueues，是不是主紧缩正在进行。

参考 Section 2.8.2.8, “Managed Compactions” ，获取更多管理紧缩的信息。

12.8.2. 回环IP(Loopback IP)

HBase 希望回环 IP 地址是 127.0.0.1. 参考开始章节 Section 2.2.3, “Loopback IP”.

12.8.3. 网络接口

所有网络接口是否正常？你确定吗？参考故障诊断用例研究 Section 12.14, “Case Studies”.

12.9. 区域服务器

RegionServer 的更多信息，参考 Section 9.6, “RegionServer”.

12.9.1. 启动错误

12.9.1.1. 主服务器启动了，但区域服务器没有启动

主服务器相信区域服务器有IP地址127.0.0.1 - 这是 localhost 并被解析到主服务器自己的localhost.

区域服务器错误的通知主服务器他们的IP地址是127.0.0.1.

修改区域服务器的 /etc/hosts 从...

```
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1               fully.qualified.regionservername regionservername  localhost.localdomain localhost
::1             localhost6.localdomain6 localhost6
```

... 改到 (将主名称从localhost中移掉)...

```
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1               localhost.localdomain localhost
::1             localhost6.localdomain6 localhost6
```

12.9.1.2. Compression Link Errors

因为LZO压缩算法需要在集群中的每台机器都要安装，这是一个启动失败的常见错误。如果你获得了如下信息

```
11/02/20 01:32:15 ERROR lzo.GPLNativeCodeLoader: Could not load native gpl library
java.lang.UnsatisfiedLinkError: no gplcompression in java.library.path
        at java.lang.ClassLoader.loadLibrary(ClassLoader.java:1734)
        at java.lang.Runtime.loadLibrary0(Runtime.java:823)
        at java.lang.System.loadLibrary(System.java:1028)
```

就意味着你的压缩库出现了问题。参见配置章节的 LZO compression configuration.

12.9.2. 运行时错误

12.9.2.1. RegionServer Hanging

Are you running an old JVM (< 1.6.0_u21?)? When you look at a thread dump, does it look like threads are BLOCKED but no one holds the lock all are blocked on? 参考 HBASE 3622 Deadlock in HBaseServer (JVM bug?). Adding -XX:+UseMembar to the HBase HBASE_OPTS in conf/hbase-env.sh may fix it.

Also, are you using Section 9.3.4, “RowLocks”? These are discouraged because they can lock up the RegionServers if not managed properly.

12.9.2.2. java.io.IOException...打开太多文件(Too many open files)

如果看到如下消息...

```
2010-09-13 01:24:17,336 WARN org.apache.hadoop.hdfs.server.datanode.DataNode:
Disk-related IOException in BlockReceiver constructor. Cause is java.io.IOException: Too many open files
        at java.io.UnixFileSystem.createFileExclusively(Native Method)
        at java.io.File.createNewFile(File.java:883)
```

... 参见快速入门的章节 ulimit and nproc configuration.

12.9.2.3. xceiverCount 258 exceeds the limit of concurrent xcievers 256

这个时常会出现在DataNode的日志中。

参见快速入门章节的 xceivers configuration.

12.9.2.4. 系统不稳定,DataNode或者其他系统进程有 "java.lang.OutOfMemoryError: unable to create new native thread in exceptions"的错误

参见快速入门章节的 ulimit and nproc configuration.. 最新的Linux发行版缺省值是 1024 - 这对HBase实在太小了。

12.9.2.5. DFS不稳定或者RegionServer租期超时

如果你收到了如下的消息:

```
2009-02-24 10:01:33,516 WARN org.apache.hadoop.hbase.util.Sleeper: We slept xxx ms, ten times longer than scheduled: 10000
2009-02-24 10:01:33,516 WARN org.apache.hadoop.hbase.util.Sleeper: We slept xxx ms, ten times longer than scheduled: 15000
2009-02-24 10:01:36,472 WARN org.apache.hadoop.hbase.regionserver.HRegionServer: unable to report to master for xxx milliseconds
```

... 或者看到了全GC压缩操作，你可能正在执行一个全GC。

12.9.2.6. "No live nodes contain current block" and/or YouAreDeadException

这个错误有可能是OS的文件句柄溢出，也可能是网络故障导致节点无法访问。

参见快速入门章节 ulimit and nproc configuration，检查你的网络。

12.9.2.7. ZooKeeper 会话超时事件

Master or RegionServers shutting down with messages like those in the logs:

```
WARN org.apache.zookeeper.ClientCnxn: Exception
closing session 0x278bd16a96000f to sun.nio.ch.SelectionKeyImpl@355811ec
java.io.IOException: TIMED OUT
       at org.apache.zookeeper.ClientCnxn$SendThread.run(ClientCnxn.java:906)
WARN org.apache.hadoop.hbase.util.Sleeper: We slept 79410ms, ten times longer than scheduled: 5000
INFO org.apache.zookeeper.ClientCnxn: Attempting connection to server hostname/IP:PORT
INFO org.apache.zookeeper.ClientCnxn: Priming connection to java.nio.channels.SocketChannel[connected local=/IP:PORT remote=host
INFO org.apache.zookeeper.ClientCnxn: Server connection successful
WARN org.apache.zookeeper.ClientCnxn: Exception closing session 0x278bd16a96000d to sun.nio.ch.SelectionKeyImpl@3544d65e
java.io.IOException: Session Expired
       at org.apache.zookeeper.ClientCnxn$SendThread.readConnectResult(ClientCnxn.java:589)
       at org.apache.zookeeper.ClientCnxn$SendThread.doIO(ClientCnxn.java:709)
       at org.apache.zookeeper.ClientCnxn$SendThread.run(ClientCnxn.java:945)
ERROR org.apache.hadoop.hbase.regionserver.HRegionServer: ZooKeeper session expired
```

The JVM is doing a long running garbage collecting which is pausing every threads (aka "stop the world"). Since the RegionServer's local ZooKeeper client cannot send heartbeats, the session times out. By design, we shut down any node that isn't able to contact the ZooKeeper ensemble after getting a timeout so that it stops serving data that may already be assigned elsewhere.

- Make sure you give plenty of RAM (in `hbase-env.sh`), the default of 1GB won't be able to sustain long running imports.
- Make sure you don't swap, the JVM never behaves well under swapping.
- Make sure you are not CPU starving the RegionServer thread. For example, if you are running a MapReduce job using 6 CPU-intensive tasks on a machine with 4 cores, you are probably starving the RegionServer enough to create longer garbage collection pauses.
- Increase the ZooKeeper session timeout

If you wish to increase the session timeout, add the following to your `hbase-site.xml` to increase the timeout from the default of 60 seconds to 120 seconds.

```
<property>
    <name>zookeeper.session.timeout</name>
    <value>1200000</value>
</property>
<property>
    <name>hbase.zookeeper.property.tickTime</name>
    <value>6000</value>
</property>
```

Be aware that setting a higher timeout means that the regions served by a failed RegionServer will take at least that amount of time to be transfered to another RegionServer. For a production system serving live requests, we would instead recommend setting it lower than 1 minute and over-provision your cluster in order the lower the memory load on each machines (hence having less garbage to collect per machine).

If this is happening during an upload which only happens once (like initially loading all your data into HBase), consider bulk loading.

参考 Section 12.11.2, "ZooKeeper, The Cluster Canary" for other general information about ZooKeeper troubleshooting.

12.9.2.8. 无服务区域异常(NotServingRegionException)

This exception is "normal" when found in the RegionServer logs at DEBUG level. This exception is returned back to the client and then the client goes back to .META. to find the new location of the moved region.

However, if the NotServingRegionException is logged ERROR, then the client ran out of retries and something probably wrong.

12.9.2.9. 区域列示先是域名，然后IP

修复 DNS. HBase 0.92.x以前的版本，反向 DNS 需要和正向查询相同答案。 参考 HBASE 3431 RegionServer is not using the name given it by the master; double entry in master listing of servers 获取详细细节。

12.9.2.10. 大量类似 '2011-01-10 12:40:48,407 INFO org.apache.hadoop.io.compress.CodecPool: Got brand-new compressor' 日志消息

没有采用本地压缩库版本。 参考 HBASE-1900 Put back native support when hadoop 0.21 is released。从hadoop的HBase库目录复制本地库或建立链接到正确位置，该消息将消失。

12.9.2.11. Server handler X on 60020 caught: java.nio.channels.ClosedChannelException

If you see this type of message it means that the region server was trying to read/send data from/to a client but it already went away. Typical causes for this are if the client was killed (you see a storm of messages like this when a MapReduce job is killed or fails) or if the client receives a SocketTimeoutException. It's harmless, but you should consider digging in a bit more if you aren't doing something to trigger them.

12.9.3. 终止错误

12.10. Master

Master 更多信息, 参考 Section 9.5, “Master”.

12.10.1. 启动错误

12.10.1.1. Master says that you need to run the hbase migrations script

Upon running that, the hbase migrations script says no files in root directory.

HBase expects the root directory to either not exist, or to have already been initialized by hbase running a previous time. If you create a new directory for HBase using Hadoop DFS, this error will occur. Make sure the HBase root directory does not currently exist or has been initialized by a previous run of HBase. Sure fire solution is to just use Hadoop dfs to delete the HBase root and let HBase create and initialize the directory itself.

12.10.2. 终止错误

12.11. ZooKeeper

12.11.1. 启动错误

12.11.1.1. 找不到地址: xyz in list of ZooKeeper quorum servers

A ZooKeeper server wasn't able to start, throws that error. xyz is the name of your server.

This is a name lookup problem. HBase tries to start a ZooKeeper server on some machine but that machine isn't able to find itself in the hbase.zookeeper.quorumconfiguration.

Use the hostname presented in the error message instead of the value you used. If you have a DNS server, you can set hbase.zookeeper.dns.interface andhbase.zookeeper.dns.nameserver in hbase-site.xml to make sure it resolves to the correct FQDN.

12.11.2. ZooKeeper, The Cluster Canary

ZooKeeper is the cluster's "canary in the mineshaft". It'll be the first to notice issues if any so making sure its happy is the short-cut to a humming cluster.

参考 ZooKeeper Operating Environment Troubleshooting 页。 It has suggestions and tools for checking disk and networking performance; i.e. the operating environment your ZooKeeper and HBase are running in.

Additionally, the utility Section 12.4.1.3, “zkcli” may help investigate ZooKeeper issues.

12.12. Amazon EC2

12.12.1. ZooKeeper 在 Amazon EC2上看起来不工作?

HBase does not start when deployed as Amazon EC2 instances. Exceptions like the below appear in the Master and/or RegionServer logs:

```
2009-10-19 11:52:27,030 INFO org.apache.zookeeper.ClientCnxn: Attempting
connection to server ec2-174-129-15-236.compute-1.amazonaws.com/10.244.9.171:2181
2009-10-19 11:52:27,032 WARN org.apache.zookeeper.ClientCnxn: Exception
closing session 0x0 to sun.nio.ch.SelectionKeyImpl@656dc861
java.net.ConnectException: Connection refused
```

Security group policy is blocking the ZooKeeper port on a public address. Use the internal EC2 host names when configuring the ZooKeeper quorum peer list.

12.12.2. Amazon EC2 上不稳定?

关于 HBase 和Amazon EC2 的问题，经常在HBase 讨论列表上被问起。搜索旧线索，使用 Search Hadoop

12.12.3. 远程Java连接到EC2集群不工作

参考 Andrew回复，更新在用户列表：Remote Java client connection into EC2 instance.

12.13. HBase 和 Hadoop 版本问题

12.13.1. 当想在adoop-0.20.205.x (或 hadoop-1.0.x)运行 0.90.x 时报NoClassDefFoundError

HBase 0.90.x does not ship with hadoop-0.20.205.x, etc. To make it run, you need to replace the hadoop jars that HBase shipped with in its `lib` directory with those of the Hadoop you want to run HBase on. If even after replacing Hadoop jars you get the below exception:

```
sv4r6s38: Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/commons/configuration/Configuration
sv4r6s38:       at org.apache.hadoop.metrics2.lib.DefaultMetricsSystem.<init>(DefaultMetricsSystem.java:37)
sv4r6s38:       at org.apache.hadoop.metrics2.lib.DefaultMetricsSystem.<clinit>(DefaultMetricsSystem.java:34)
sv4r6s38:       at org.apache.hadoop.security.UgiInstrumentation.create(UgiInstrumentation.java:51)
sv4r6s38:       at org.apache.hadoop.security.UserGroupInformation.initialize(UserGroupInformation.java:209)
sv4r6s38:       at org.apache.hadoop.security.UserGroupInformation.ensureInitialized(UserGroupInformation.java:177)
sv4r6s38:       at org.apache.hadoop.security.UserGroupInformation.isSecurityEnabled(UserGroupInformation.java:229)
sv4r6s38:       at org.apache.hadoop.security.KerberosName.<clinit>(KerberosName.java:83)
sv4r6s38:       at org.apache.hadoop.security.UserGroupInformation.initialize(UserGroupInformation.java:202)
sv4r6s38:       at org.apache.hadoop.security.UserGroupInformation.ensureInitialized(UserGroupInformation.java:177)
```

you need to copy under `hbase/lib`, the `commons-configuration-X.jar` you find in your Hadoop's `lib` directory. That should fix the above complaint.

12.14. 用例研究

性能和故障诊断用例，参考 Chapter 13, *Case Studies*.

[31]参考获取答案

Chapter 13. 用例研究

Table of Contents

13.1. 概述

This chapter will describe a variety of performance and troubleshooting case studies that can provide a useful blueprint on diagnosing cluster issues.

For more information on Performance and Troubleshooting, see Chapter 11, *Performance Tuning* and Chapter 12, *Troubleshooting and Debugging HBase*.

13.2. 模式设计

13.2.1. 数据列表

The following is an exchange from the user dist-list regarding a fairly common question: how to handle per-user list data in HBase.

*** QUESTION ***

We're looking at how to store a large amount of (per-user) list data in HBase, and we were trying to figure out what kind of access pattern made the most sense. One option is store the majority of the data in a key, so we could have something like:

```
<FixedWidthUserName><FixedWidthValueId1>:"" (no value)
<FixedWidthUserName><FixedWidthValueId2>:"" (no value)
<FixedWidthUserName><FixedWidthValueId3>:"" (no value)
```

The other option we had was to do this entirely using:

```
<FixedWidthUserName><FixedWidthPageNum0>:<FixedWidthLength><FixedIdNextPageNum><ValueId1><ValueId2><ValueId3>...
<FixedWidthUserName><FixedWidthPageNum1>:<FixedWidthLength><FixedIdNextPageNum><ValueId1><ValueId2><ValueId3>...
```

where each row would contain multiple values. So in one case reading the first thirty values would be:

```
scan { STARTROW => 'FixedWidthUsername' LIMIT => 30}
```

And in the second case it would be

```
get 'FixedWidthUserName\x00\x00\x00\x00'
```

The general usage pattern would be to read only the first 30 values of these lists, with infrequent access reading deeper into the lists. Some users would have <= 30 total values in these lists, and some users would have millions (i.e. power-law distribution)

The single-value format seems like it would take up more space on HBase, but would offer some improved retrieval / pagination flexibility. Would there be any significant performance advantages to be able to paginate via gets vs paginating with scans?

My initial understanding was that doing a scan should be faster if our paging size is unknown (and caching is set appropriately), but that gets should be faster if we'll always need the same page size. I've ended up hearing different people tell me opposite things about performance. I assume the page sizes would be relatively consistent, so for most use cases we could guarantee that we only wanted one page of data in the fixed-page-length case. I would also assume that we would have infrequent updates, but may have inserts into the middle of these lists (meaning we'd need to update all subsequent rows).

Thanks for help / suggestions / follow-up questions.

*** ANSWER ***

If I understand you correctly, you're ultimately trying to store triples in the form "user, valueid, value", right? E.g., something like:

```
"user123, firstname, Paul",
"user234, lastname, Smith"
```

(But the usernames are fixed width, and the valueids are fixed width).

And, your access pattern is along the lines of: "for user X, list the next 30 values, starting with valueid Y". Is that right? And these values should be returned sorted by valueid?

The tl;dr version is that you should probably go with one row per user+value, and not build a complicated intra-row pagination scheme on your own unless you're really sure it is needed.

Your two options mirror a common question people have when designing HBase schemas: should I go "tall" or "wide"? Your first schema is "tall": each row represents one value for one user, and so there are many rows in the table for each user; the row key is user + valueid, and there would be (presumably) a single column qualifier that means "the value". This is great if you want to scan over rows in sorted order by row key (thus my question above, about whether these ids are sorted correctly). You can start a scan at any user+valueid, read the next 30, and be done. What you're giving up is the ability to have transactional guarantees around all the rows for one user, but it doesn't sound like you need that. Doing it this way is generally recommended (see here #schema.smackdown).

Your second option is "wide": you store a bunch of values in one row, using different qualifiers (where the qualifier is the valueid). The simple way to do that would be to just store ALL values for one user in a single row. I'm guessing you jumped to the "paginated" version because you're assuming that storing millions of columns in a single row would be bad for performance, which may or may not be true; as long as you're not trying to do too much in a single request, or do things like scanning over and returning all of the cells in the row, it shouldn't be fundamentally worse. The client has methods that allow you to get specific slices of columns.

Note that neither case fundamentally uses more disk space than the other; you're just "shifting" part of the identifying information for a value either to the left (into the row key, in option one) or to the right (into the column qualifiers in option 2). Under the covers, every key/value still stores the whole row key, and column family name. (If this is a bit confusing, take an hour and watch Lars George's excellent video about understanding HBase schema design: http://www.youtube.com/watch?v=_HLoH_PgrLk).

A manually paginated version has lots more complexities, as you note, like having to keep track of how many things are in each page, re-shuffling if new values are inserted, etc. That seems significantly more complex. It might have some slight speed advantages (or disadvantages!) at extremely high throughput, and the only way to really know that would be to try it out. If you don't have time to build it both ways and compare, my advice would be to start with the simplest option (one row per user+value). Start simple and iterate! :)

13.3. 性能/故障诊断

13.3.1. 用例 #1 (单节点性能问题)

13.3.1.1.场景

Following a scheduled reboot, one data node began exhibiting unusual behavior. Routine MapReduce jobs run against HBase tables which regularly completed in five or six minutes began taking 30 or 40 minutes to finish. These jobs were consistently found to be waiting on map and reduce tasks assigned to the troubled data node (e.g., the slow map tasks all had the same Input Split). The situation came to a head during a distributed copy, when the copy was severely prolonged by the lagging node.

13.3.1.2. 硬件

Datanodes:

- Two 12-core processors
- Six Enerprise SATA disks
- 24GB of RAM
- Two bonded gigabit NICs

Network:

- 10 Gigabit top-of-rack switches
- 20 Gigabit bonded interconnects between racks.

13.3.1.3. 假设

13.3.1.3.1. HBase "热点" 区域

We hypothesized that we were experiencing a familiar point of pain: a "hot spot" region in an HBase table, where uneven key-space distribution can funnel a huge number of requests to a single HBase region, bombarding the RegionServer process and cause slow response time. Examination of the HBase Master status page showed that the number of HBase requests to the troubled node was almost zero. Further, examination of the HBase logs showed that there were no region splits, compactions, or other region transitions in progress. This effectively ruled out a "hot spot" as the root cause of the observed slowness.

13.3.1.3.2. HBase 分区具有非本地数据

Our next hypothesis was that one of the MapReduce tasks was requesting data from HBase that was not local to the datanode, thus forcing HDFS to request data blocks from other servers over the network. Examination of the datanode logs showed that there were very few blocks being requested over the network, indicating that the HBase region was correctly assigned, and that the majority of the necessary data was located on the node. This ruled out the possibility of non-local data causing a slowdown.

13.3.1.3.3. Excessive I/O Wait Due To Swapping Or An Over-Worked Or Failing Hard Disk

After concluding that the Hadoop and HBase were not likely to be the culprits, we moved on to troubleshooting the datanode's hardware. Java, by design, will periodically scan its entire memory space to do garbage collection. If system memory is heavily overcommitted, the Linux kernel may enter a vicious cycle, using up all of its resources swapping Java heap back and forth from disk to RAM as Java tries to run garbage collection. Further, a failing hard disk will often retry reads and/or writes many times before giving up and returning an error. This can manifest as high iowait, as running processes wait for reads and writes to complete. Finally, a disk nearing the upper edge of its performance envelope will begin to cause iowait as it informs the kernel that it cannot accept any more data, and the kernel queues incoming data into the dirty write pool in memory. However, using `vmstat(1)` and `free(1)`, we could see that no swap was being used, and the amount of disk IO was only a few kilobytes per second.

13.3.1.3.4. Slowness Due To High Processor Usage

Next, we checked to see whether the system was performing slowly simply due to very high computational load. `top(1)` showed that the system load was higher than normal, but `vmstat(1)` and `mpstat(1)` showed that the amount of processor being used for actual computation was low.

13.3.1.3.5. Network Saturation (The Winner)

Since neither the disks nor the processors were being utilized heavily, we moved on to the performance of the network interfaces. The datanode had two gigabit ethernet adapters, bonded to form an active-standby interface. `ifconfig(8)` showed some unusual anomalies, namely interface errors, overruns, framing errors. While not unheard of, these kinds of errors are exceedingly rare on modern hardware which is operating as it should:

```
$ /sbin/ifconfig bond0
bond0  Link encap:Ethernet  HWaddr 00:00:00:00:00:00
inet addr:10.x.x.x  Bcast:10.x.x.255  Mask:255.255.255.0
UP BROADCAST RUNNING MASTER MULTICAST  MTU:1500  Metric:1
RX packets:2990700159 errors:12 dropped:0 overruns:1 frame:6          <--- Look Here! Errors!
TX packets:3443518196 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:2416328868676 (2.4 TB)  TX bytes:3464991094001 (3.4 TB)
```

These errors immediately lead us to suspect that one or more of the ethernet interfaces might have negotiated the wrong line speed. This was confirmed both by running an ICMP ping from an external host and observing round-trip-time in excess of 700ms, and by running `ethtool(8)` on the members of the bond interface and discovering that the active interface was operating at 100Mbs/, full duplex.

```
$ sudo ethtool eth0
Settings for eth0:
Supported ports: [ TP ]
Supported link modes:   10baseT/Half 10baseT/Full
                       100baseT/Half 100baseT/Full
                       1000baseT/Full
Supports auto-negotiation: Yes
Advertised link modes:  10baseT/Half 10baseT/Full
                       100baseT/Half 100baseT/Full
                       1000baseT/Full
Advertised pause frame use: No
Advertised auto-negotiation: Yes
Link partner advertised link modes:  Not reported
Link partner advertised pause frame use: No
Link partner advertised auto-negotiation: No
Speed: 100Mb/s                                     <--- Look Here!  Should say 1000Mb/s!
Duplex: Full
Port: Twisted Pair
PHYAD: 1
Transceiver: internal
Auto-negotiation: on
MDI-X: Unknown
Supports Wake-on: umbg
Wake-on: g
Current message level: 0x00000003 (3)
Link detected: yes
```

In normal operation, the ICMP ping round trip time should be around 20ms, and the interface speed and duplex should read, "1000MB/s", and, "Full", respectively.

13.3.1.4. 结论

After determining that the active ethernet adapter was at the incorrect speed, we used the `ifenslave(8)` command to make the standby interface the active interface, which yielded an immediate improvement in MapReduce performance, and a 10 times improvement in network throughput:

On the next trip to the datacenter, we determined that the line speed issue was ultimately caused by a bad network cable, which was replaced.

13.3.2. 用例 #2 (性能研究 2012)

Investigation results of a self-described "we're not sure what's wrong, but it seems slow" problem. http://gbif.blogspot.com/2012/03/hbase-performance-evaluation-continued.html

13.3.3. Case Study #3 (Performance Research 2010))

Investigation results of general cluster performance from 2010. Although this research is on an older version of the codebase, this writeup is still very useful in terms of approach. http://hstack.org/hbase-performance-testing/

13.3.4. Case Study #4 (xcievers Config)

Case study of configuring `xceivers`, and diagnosing errors from mis-configurations. http://www.larsgeorge.com/2012/03/hadoop-hbase-and-xceivers.html

参考 also Section 2.3.2, "`dfs.datanode.max.xcievers`".

Chapter 14. HBase 运维管理

Table of Contents

This chapter will cover operational tools and practices required of a running HBase cluster. The subject of operations is related to the topics of Chapter 12, *Troubleshooting and Debugging HBase*, Chapter 11, *Performance Tuning*, and Chapter 2, *Configuration* but is a distinct topic in itself.

14.1. HBase 工具和实用程序

这里我们给出HBase工具列表，可用于管理，分析，修复和调试。

14.1.1. 驱动

There is a `Driver` class that is executed by the HBase jar can be used to invoke frequently accessed utilities. For example,

```
HADOOP_CLASSPATH=`${HBASE_HOME}/bin/hbase classpath` ${HADOOP_HOME}/bin/hadoop jar ${HBASE_HOME}/hbase-VERSION.jar
```

... will return...

```
An example program must be given as the first argument.
Valid program names are:
  completebulkload: Complete a bulk data load.
  copytable: Export a table from local cluster to peer cluster
  export: Write table data to HDFS.
  import: Import data written by Export.
  importtsv: Import data in TSV format.
  rowcounter: Count rows in HBase table
  verifyrep: Compare the data from tables in two different clusters. WARNING: It doesn't work for incrementColumnValues'd cells
```

... for allowable program names.

14.1.2. HBase hbck

An *fsck* for your HBase install

To run hbck against your HBase cluster run

```
$ ./bin/hbase hbck
```

At the end of the commands output it prints *OK* or *INCONSISTENCY*. If your cluster reports inconsistencies, pass **-details** to see more detail emitted. If inconsistencies, run **hbck** a few times because the inconsistency may be transient (e.g. cluster is starting up or a region is splitting). Passing **-fix** may correct the inconsistency (This latter is an experimental feature).

For more information, see Appendix B, *hbck In Depth*.

14.1.3. HFile 工具

参考 Section 9.7.5.2.2, “HFile Tool”.

14.1.4. WAL 工具

14.1.4.1. `HLog` tool

The main method on `HLog` offers manual split and dump facilities. Pass it WALs or the product of a split, the content of the `recovered.edits`. directory.

You can get a textual dump of a WAL file content by doing the following:

```
 $ ./bin/hbase org.apache.hadoop.hbase.regionserver.wal.HLog --dump hdfs://example.org:8020/hbase/.logs/example.org,60020,128351
```

The return code will be non-zero if issues with the file so you can test wholesomeness of file by redirecting `STDOUT` to `/dev/null` and testing the program return.

Similarly you can force a split of a log file directory by doing:

```
 $ ./bin/hbase org.apache.hadoop.hbase.regionserver.wal.HLog --split hdfs://example.org:8020/hbase/.logs/example.org,60020,12835
```

14.1.4.1.1. `HLogPrettyPrinter`

`HLogPrettyPrinter` is a tool with configurable options to print the contents of an HLog.

14.1.5. Compression Tool

参考 Section C.1, “CompressionTest Tool”.

14.1.6. CopyTable

CopyTable is a utility that can copy part or of all of a table, either to the same cluster or another cluster. The usage is as follows:

```
$ bin/hbase org.apache.hadoop.hbase.mapreduce.CopyTable [--starttime=X] [--endtime=Y] [--new.name=NEW] [--peer.adr=ADR] tablenam
```

Options:

- `starttime` Beginning of the time range. Without endtime means starttime to forever.
- `endtime` End of the time range. Without endtime means starttime to forever.
- `versions` Number of cell versions to copy.
- `new.name` New table's name.
- `peer.adr` Address of the peer cluster given in the format hbase.zookeeper.quorum:hbase.zookeeper.client.port:zookeeper.znode.parent
- `families` Comma-separated list of ColumnFamilies to copy.
- `all.cells` Also copy delete markers and uncollected deleted cells (advanced option).

Args:

- tablename Name of table to copy.

Example of copying 'TestTable' to a cluster that uses replication for a 1 hour window:

```
$ bin/hbase org.apache.hadoop.hbase.mapreduce.CopyTable
--starttime=1265875194289 --endtime=1265878794289
--peer.adr=server1,server2,server3:2181:/hbase TestTable
```

Scanner Caching

Caching for the input Scan is configured via `hbase.client.scanner.caching` in the job configuration.

参考 Jonathan Hsieh's Online HBase Backups with CopyTable blog post for more on **CopyTable**.

14.1.7. 导出

导出实用工具可以将表的内容输出成HDFS的序列化文件，如下调用：

```
$ bin/hbase org.apache.hadoop.hbase.mapreduce.Export <tablename> <outputdir> [<versions> [<starttime> [<endtime>]]]
```

Note: caching for the input Scan is configured via `hbase.client.scanner.caching` in the job configuration.

14.1.8. 导入

导入实用工具可以加载导出的数据回到HBase，如下调用：

```
$ bin/hbase org.apache.hadoop.hbase.mapreduce.Import <tablename> <inputdir>
```

14.1.9. ImportTsv

ImportTsv is a utility that will load data in TSV format into HBase. It has two distinct usages: loading data from TSV format in HDFS into HBase via Puts, and preparing StoreFiles to be loaded via the `completebulkload`.

To load data via Puts (i.e., non-bulk loading):

```
$ bin/hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.columns=a,b,c <tablename> <hdfs-inputdir>
```

To generate StoreFiles for bulk-loading:

```
$ bin/hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.columns=a,b,c -Dimporttsv.bulk.output=hdfs://storefile-outpu
```

These generated StoreFiles can be loaded into HBase via Section 14.1.10, “CompleteBulkLoad”.

14.1.9.1. ImportTsv 选项

Running ImportTsv with no arguments prints brief usage information:

```
Usage: importtsv -Dimporttsv.columns=a,b,c <tablename> <inputdir>

Imports the given input directory of TSV data into the specified table.

The column names of the TSV data must be specified using the -Dimporttsv.columns
option. This option takes the form of comma-separated column names, where each
column name is either a simple column family, or a columnfamily:qualifier. The special
column name HBASE_ROW_KEY is used to designate that this column should be used
as the row key for each imported record. You must specify exactly one column
to be the row key, and you must specify a column name for every column that exists in the
input data.

By default importtsv will load data directly into HBase. To instead generate
HFiles of data to prepare for a bulk data load, pass the option:
  -Dimporttsv.bulk.output=/path/for/output
  Note: if you do not use this option, then the target table must already exist in HBase

Other options that may be specified with -D include:
  -Dimporttsv.skip.bad.lines=false - fail if encountering an invalid line
  '-Dimporttsv.separator=|' - eg separate on pipes instead of tabs
  -Dimporttsv.timestamp=currentTimeAsLong - use the specified timestamp for the import
  -Dimporttsv.mapper.class=my.Mapper - A user-defined Mapper to use instead of org.apache.hadoop.hbase.mapreduce.TsvImporterMapp
```

14.1.9.2. ImportTsv 示例

For example, assume that we are loading data into a table called 'datatsv' with a ColumnFamily called 'd' with two columns "c1" and "c2".

Assume that an input file exists as follows:

```
row1    c1      c2
row2    c1      c2
row3    c1      c2
```

```
row4    c1      c2
row5    c1      c2
row6    c1      c2
row7    c1      c2
row8    c1      c2
row9    c1      c2
row10   c1      c2
```

For ImportTsv to use this imput file, the command line needs to look like this:

```
 HADOOP_CLASSPATH=`${HBASE_HOME}/bin/hbase classpath` ${HADOOP_HOME}/bin/hadoop jar ${HBASE_HOME}/hbase-VERSION.jar importtsv -I
```

... and in this example the first column is the rowkey, which is why the HBASE_ROW_KEY is used. The second and third columns in the file will be imported as "d:c1" and "d:c2", respectively.

14.1.9.3. ImportTsv Warning

If you have preparing a lot of data for bulk loading, make sure the target HBase table is pre-split appropriately.

14.1.9.4. 参考

For more information about bulk-loading HFiles into HBase, see Section 9.8, “Bulk Loading”

14.1.10. CompleteBulkLoad

`completebulkload` 实用工具可以将产生的存储文件移动到HBase表。该工具经常和Section 14.1.9, “ImportTsv” 的输出联合使用。

两种方法调用该工具，带显式类名或通过驱动：

```
$ bin/hbase org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFiles <hdfs://storefileoutput> <tablename>
```

.. 通过驱动..

```
HADOOP_CLASSPATH=`${HBASE_HOME}/bin/hbase classpath` ${HADOOP_HOME}/bin/hadoop jar ${HBASE_HOME}/hbase-VERSION.jar completebulkl
```

批量导入 HFiles 到 HBase的更多信息， 参考 Section 9.8, “Bulk Loading”.

14.1.11. WALPlayer

WALPlayer 实用工具可以重放 WAL 文件到 HBase.

The WAL can be replayed for a set of tables or all tables, and a timerange can be provided (in milliseconds). The WAL is filtered to this set of tables. The output can optionally be mapped to another set of tables.

WALPlayer can also generate HFiles for later bulk importing, in that case only a single table and no mapping can be specified.

Invoke via:

```
$ bin/hbase org.apache.hadoop.hbase.mapreduce.WALPlayer [options] <wal inputdir> <tables> [<tableMappings>]>
```

For example:

```
$ bin/hbase org.apache.hadoop.hbase.mapreduce.WALPlayer /backuplogdir oldTable1,oldTable2 newTable1,newTable2
```

14.1.12. RowCounter

RowCounter 实用工具可以统计表的行数。这是一个好工具，如果担心元数据可能存在不一致，可以用于确认HBase可以读取表的所有分块。

```
$ bin/hbase org.apache.hadoop.hbase.mapreduce.RowCounter <tablename> [<column1> <column2>...]
```

Note: caching for the input Scan is configured via `hbase.client.scanner.caching` in the job configuration.

14.2. 区域管理

14.2.1. 主紧缩

主紧缩可以通过HBase shell 或 HBaseAdmin.majorCompact 进行。

注意：主紧缩不进行区域合并。更多关于紧缩的信息，参考 Section 9.7.5.5, “Compaction”

14.2.2. 合并

Merge is a utility that can merge adjoining regions in the same table (see org.apache.hadoop.hbase.util.Merge).

```
$ bin/hbase org.apache.hbase.util.Merge <tablename> <region1> <region2>
```

If you feel you have too many regions and want to consolidate them, Merge is the utility you need. Merge must run be done when the cluster is down. 参考 the O'Reilly HBase Book for an example of usage.

Additionally, there is a Ruby script attached to HBASE-1621 for region merging.

14.3. 节点管理

14.3.1. 节点下线

你可以在HBase的特定的节点上运行下面的脚本来停止RegionServer:

```
$ ./bin/hbase-daemon.sh stop regionserver
```

RegionServer会首先关闭所有的region然后把它自己关闭，在停止的过程中，RegionServer的会向Zookeeper报告说他已经过期了。master会发现RegionServer已经死了，会把它当作崩溃的server来处理。他会将region分配到其他的节点上去。

在下线节点之前要停止Load Balancer

如果在运行load balancer的时候，一个节点要关闭， 则Load Balancer和Master的recovery可能会争夺这个要下线的Regionserver。为了避免这个问题，先将load balancer停止，参见下面的 Load Balancer.

RegionServer下线有一个缺点就是其中的Region会有好一会离线。Regions是被按顺序关闭的。如果一个server上有很多region,从第一个region会被下线，到最后一个region被关闭，并且Master确认他已经死了，该region才可以上线，整个过程要花很长时间。在HBase 0.90.2中，我们加入了一个功能，可以让节点逐渐的摆脱他的负载，最后关闭。HBase 0.90.2加入了 `graceful_stop.sh`脚本，可以这样用，

```
$ ./bin/graceful_stop.sh
Usage: graceful_stop.sh [--config &conf-dir>] [--restart] [--reload] [--thrift] [--rest] &hostname>
 thrift      If we should stop/start thrift before/after the hbase stop/start
 rest        If we should stop/start rest before/after the hbase stop/start
 restart     If we should restart after graceful stop
 reload      Move offloaded regions back on to the stopped server
 debug       Move offloaded regions back on to the stopped server
 hostname    Hostname of server we are to stop
```

要下线一台RegionServer可以这样做

```
$ ./bin/graceful_stop.sh HOSTNAME
```

这里的`HOSTNAME`是RegionServer的host you would decommission.

On `HOSTNAME`

传递到`graceful_stop.sh`的`HOSTNAME`必须和hbase使用的hostname一致，hbase用它来区分RegionServers。可以用master的UI来检查RegionServers的id。通常是hostname,也可能是FQDN。不管HBase使用的哪一个，你可以将它传到`graceful_stop.sh`脚本中去，目前他还不支持使用IP地址来推断hostname。所以使用IP就会发现server不在运行，也没有办法下线了。

`graceful_stop.sh` 脚本会一个一个将region从RegionServer中移除出去，以减少改RegionServer的负载。他会先移除一个region,然后再将这个region安置到一个新的地方，再移除下一个，直到全部移除。最后`graceful_stop.sh`脚本会让RegionServer **stop**.,Master会注意到RegionServer已经下线了，这个时候所有的region已经重新部署好。RegionServer就可以干干净净的结束，没有WAL日志需要分割。

Load Balancer

当执行**graceful_stop**脚本的时候，要将Region Load Balancer关掉(否则balancer和下线脚本会在region部署的问题上存在冲突):

```
hbase(main):001:0> balance_switch false
true
0 row(s) in 0.3590 seconds
```

上面是将balancer关掉，要想开启：

```
hbase(main):001:0> balance_switch true
false
0 row(s) in 0.3590 seconds
```

14.3.2. 依次重启

你还可以让这个脚本重启一个RegionServer,不改变上面的Region的位置。要想保留数据的位置，你可以依次重启(Rolling Restart),就像这样:

```
$ for i in `cat conf/regionservers|sort`; do ./bin/graceful_stop.sh --restart --reload --debug $i; done &> /tmp/log.txt &
```

Tail `/tmp/log.txt`来看脚本的运行过程.上面的脚本只对RegionServer进行操作。要确认load balancer已经关掉。还需要在之前更新master。下面是一段依次重启的伪脚本,你可以借鉴它：

1. 确认你的版本，保证配置已经rsync到整个集群中。如果版本是0.90.2，需要打上HBASE-3744 和 HBASE-3756两个补丁。

2. 运行hbck确保你的集群是一致的

   ```
   $ ./bin/hbase hbck
   ```

   当发现不一致的时候，可以修复他。

3. 重启Master:

   ```
   $ ./bin/hbase-daemon.sh stop master; ./bin/hbase-daemon.sh start master
   ```

4. 关闭region balancer:

```
$ echo "balance_switch false" | ./bin/hbase
```

5. 在每个RegionServer上运行`graceful_stop.sh`:

   ```
   $ for i in `cat conf/regionservers|sort`; do ./bin/graceful_stop.sh --restart --reload --debug $i; done &> /tmp/log.t
   ```

   如果你在RegionServer还开起来thrift和rest server。还需要加上--thrift or --rest 选项 (参见 `graceful_stop.sh` 脚本的用法).

6. 再次重启Master.这会把已经死亡的server列表清空，重新开启balancer.

7. 运行 hbck 保证集群是一直的

14.4. HBase 度量

14.4.1. 度量安装

参见 Metrics 可以获得一个enable Metrics emission的指导。

14.4.2. 区域服务器度量

14.4.2.1. `hbase.regionserver.blockCacheCount`

内存中的Block cache item数量。这个是存储文件(HFiles)的缓存中的数量。

14.4.2.2. `hbase.regionserver.blockCacheEvictedCount`

Number of blocks that had to be evicted from the block cache due to heap size constraints.

14.4.2.3. `hbase.regionserver.blockCacheFree`

内存中的Block cache memory 剩余 (单位 bytes).

14.4.2.4. `hbase.regionserver.blockCacheHitCachingRatio`

Block cache hit caching ratio (0 to 100). The cache-hit ratio for reads configured to look in the cache (i.e., cacheBlocks=true).

14.4.2.5. `hbase.regionserver.blockCacheHitCount`

Number of blocks of StoreFiles (HFiles) read from the cache.

14.4.2.6. `hbase.regionserver.blockCacheHitRatio`

Block cache 命中率(0 到 100). Includes all read requests, although those with cacheBlocks=false will always read from disk and be counted as a "cache miss"

14.4.2.7. `hbase.regionserver.blockCacheMissCount`

StoreFiles (HFiles)请求的未在缓存中的分块数量。

14.4.2.8. `hbase.regionserver.blockCacheSize`

内存中的Block cache 大小 (单位 bytes). i.e., memory in use by the BlockCache

14.4.2.9. `hbase.regionserver.compactionQueueSize`

compaction队列的大小. 这个值是需要进行compaction的region数目

14.4.2.10. `hbase.regionserver.flushQueueSize`

Number of enqueued regions in the MemStore awaiting flush.

14.4.2.11. `hbase.regionserver.fsReadLatency_avg_time`

文件系统延迟 (ms). 这个值是平均读HDFS的延迟时间

14.4.2.12. `hbase.regionserver.fsReadLatency_num_ops`

文件系统读操作。

14.4.2.13. `hbase.regionserver.fsSyncLatency_avg_time`

文件系统同步延迟(ms). Latency to sync the write-ahead log records to the filesystem.

14.4.2.14. `hbase.regionserver.fsSyncLatency_num_ops`

Number of operations to sync the write-ahead log records to the filesystem.

14.4.2.15. `hbase.regionserver.fsWriteLatency_avg_time`

文件系统写延迟(ms). Total latency for all writers, including StoreFiles and write-head log.

14.4.2.16. `hbase.regionserver.fsWriteLatency_num_ops`

Number of filesystem write operations, including StoreFiles and write-ahead log.

14.4.2.17. `hbase.regionserver.memstoreSizeMB`

所有的RegionServer的memstore大小 (MB)

14.4.2.18. `hbase.regionserver.regions`

RegionServer服务的regions数量

14.4.2.19. `hbase.regionserver.requests`

读写请求的全部数量。请求是指RegionServer的RPC数量，因此一次Get一个请求，但一个缓存设为1000的Scan也会在每次调用'next'时导致一个请求。一个批量load是一个Hfile一个请求。

14.4.2.20. `hbase.regionserver.storeFileIndexSizeMB`

当前RegionServer的storefile索引的总大小(MB)

14.4.2.21. `hbase.regionserver.stores`

RegionServer打开的stores数量。一个stores对应一个列族。例如，一个包含列族的表有3个region在这个RegionServer上，对应一个 列族就会有3个store.

14.4.2.22. `hbase.regionserver.storeFiles`

RegionServer打开的存储文件(HFile)数量。这个值一定大于等于store的数量。

14.5. HBase 监控

14.5.1. 概述

下面的度量方法对每个区域服务器的宏观监控被证明是是最重要的，特别是在像 OpenTSDB这样的系统中。如果你的集群具有性能问题，你可能得参考本组信息。

HBase:

- Requests
- Compactions queue

OS:

- IO Wait
- User CPU

Java:

- GC

HBase度量的更多信息，参考 Section 14.4, “HBase Metrics”.

14.5.2. 查询太慢的日志

HBase查询太慢的日志由可分析的 JSON结构描述。客户端操作 (Gets, Puts, Deletes, 等)的属性， 要么运行太久，或产生输出太多。“运行太久”和“输出太多”的门限可配置，如后面所述。输出产生在主区域服务器日志中，以便和其他日志事件一起发现更多细节。它也前置区分标签`(responseTooSlow)`, `(responseTooLarge)`, `(operationTooSlow)`和`(operationTooLarge)`，以便当用户只希望看到慢查询时，用grep过滤。

14.5.2.1. 配置

有两个配置节可用于调整查询太慢日志的门限。

- `hbase.ipc.warn.response.time` 不被记录太慢日志的查询执行的最大毫秒数(millisecond) 。缺省10000, 即 10 秒。可设 -1 禁止通过时间长短记入日志。
- `hbase.ipc.warn.response.size` 不被记录日志的查询可返回的最大字节数。缺省 100 MB，可设为 -1 禁止通过大小记入日志。

14.5.2.2. 度量

查询太慢日志暴露给了JMX 度量。

- `hadoop.regionserver_rpc_slowResponse` 是一个全局度量，反射所有超时记入日志的响应。
- `hadoop.regionserver_rpc_methodName.aboveOneSec` 一个度量，反射所有超过一秒时间的响应。

14.5.2.3. 输出

输出以操作做标签，如 `(operationTooSlow)` 。如果调用是客户端操作，如 Put, Get, 或 Delete，会暴露详细指纹信息。否则，标签为 `(responseTooSlow)` ，也同样提供可分析的JSON 输出，但具有较少细节信息，完全依赖于RPC自身的超时和超量设置。 `TooLarge` 代替 `TooSlow` 如果响应大小引起日志记录。在大小和时长都引起日志记录时，也是 `TooLarge` 后置。

14.5.2.4. 示例

```
2011-09-08 10:01:25,824 WARN org.apache.hadoop.ipc.HBaseServer: (operationTooSlow): {"tables":{"riley2":{"puts":[{"totalColumns"
```

注意，在"tables"结构里的所有东西，是MultiPut的指纹打印的输出。其余的信息是RPC相关的，如处理时间和客户端IP/port。 客户

端的其他操作的模式和通用结构与此相同，但根据单个操作的类型会有一定的不同。如果调用不是客户端操作，则指纹细节信息将完全没有。

对本示例而言，指出了缓慢的原因可能是简单的超大 (100MB) multiput，通过 "vlen" 即 value length告知，multiPut中的每个put的域有该信息。

14.6. 集群复制

参见 集群复制.

14.7. HBase 备份

有两种通常策略进行 HBase 备份：停止整个集群再备份，和在正在使用的集群上备份。 每一种途径都有优缺点。

更多信息，参考Sematext的Blog HBase Backup Options.

14.7.1. 全停止备份

一些环境可以容忍暂时停止 HBase 集群，如用于后台容量分析，并不提供前台页面。 好处是 NameNode/Master 和 RegionServers是停止的， 不会有任何机会丢失正在处理改变的保存文件或元数据。明显的坏处是集群被 关闭。步骤包括：

14.7.1.1. 停止 HBase

14.7.1.2. Distcp

Distcp 既用于将HDFS里面的HBase 目录下的内容拷贝到当前集群的另一个目录，也可以拷贝到另一个集群。

注意： Distcp 工作的情形是集群关闭，没有正在改变的文件。Distcp 不推荐用于正工作着的集群。

14.7.1.3. 恢复 (如有必要)

通过distcp，从 HDFS备份的数据被拷贝到 '真实' 的hbase 目录。 复制动作产生新的 HDFS 元数据， 所以并不需要从备份的 NameNode 元数据恢复，因为是通过distcp从一个特定的 HDFS 目录 (如，HBase 部分)复制， 不是整个HDFS 文件系统。

14.7.2. 工作集群备份 - Replication

这种方法假设有另一个集群。参考HBase的 replication 。

14.7.3. 工作集群备份 - CopyTable

14.1.6节, “CopyTable” 工具，即可用于将一个表复制到同集群的另一个表，也可将表复制到另一个集群的另一个表。

由于集群在工作，有丢失正在改变的数据的风险。

14.7.4. 工作集群备份 - Export

14.1.7节, “Export” 是一种将 HDFS 内容导出到同一集群的方法。 恢复数据， 14.1.8节, “Import” 工具可以使用。

由于集群在工作，有丢失正在改变的数据的风险。

14.8. 容量计划

14.8.1.存储

一个常见问题是HBase管理员需要估算一个HBase集群要用多大存储量。可以通过几个方面去考虑，最重要的是集群要加载什么数据。 开始于对HBase内部处理数据(KeyValue)的可靠了解。

14.8.1.1. KeyValue

HBase storage will be dominated by KeyValues. 参考 Section 9.7.5.4, “KeyValue” and Section 6.3.2, “Try to minimize row and column sizes” for how HBase stores data internally.

It is critical to understand that there is a KeyValue instance for every attribute stored in a row, and the rowkey-length, ColumnFamily name-length and attribute lengths will drive the size of the database more than any other factor.

14.8.1.2. StoreFiles and Blocks

KeyValue instances are aggregated into blocks, and the blocksize is configurable on a per-ColumnFamily basis. Blocks are aggregated into StoreFile's. 参考 Section 9.7, “Regions”.

14.8.1.3. HDFS Block Replication

Because HBase runs on top of HDFS, factor in HDFS block replication into storage calculations.

14.8.2. 区域

Another common question for HBase administrators is determining the right number of regions per RegionServer. This affects both storage and hardware planning. 参考 Section 11.4.1, “Number of Regions”.

Chapter 15. 创建和部署HBase

This chapter will be of interest only to those building and developing HBase (i.e., as opposed to just downloading the latest distribution).

15.1. HBase 版本库

15.1.1. SVN

```
svn co http://svn.apache.org/repos/asf/hbase/trunk hbase-core-trunk
```

15.1.2. Git

```
git clone git://git.apache.org/hbase.git
```

15.2. IDEs

15.2.1. Eclipse

15.2.1.1. Code Formatting

参考 HBASE-3678 Add Eclipse-based Apache Formatter to HBase Wiki for an Eclipse formatter to help ensure your code conforms to HBase'y coding convention. The issue includes instructions for loading the attached formatter.

In addition to the automatic formatting, make sure you follow the style guidelines explained in Section 15.10.5, “Common Patch Feedback”

Also, no @author tags - that's a rule. Quality Javadoc comments are appreciated. And include the Apache license.

15.2.1.2. Subversive Plugin

Download and install the Subversive plugin.

Set up an SVN Repository target from Section 15.1.1, “SVN”, then check out the code.

15.2.1.3. Git Plugin

If you cloned the project via git, download and install the Git plugin (EGit). Attach to your local git repo (via the Git Repositories window) and you'll be able to see file revision history, generate patches, etc.

15.2.1.4. HBase Project Setup in Eclipse

The easiest way is to use the m2eclipse plugin for Eclipse. Eclipse Indigo or newer has m2eclipse built-in, or it can be found here:http://www.eclipse.org/m2e/. M2Eclipse provides Maven integration for Eclipse - it even lets you use the direct Maven commands from within Eclipse to compile and test your project.

To import the project, you merely need to go to File->Import...Maven->Existing Maven Projects and then point Eclipse at the HBase root directory; m2eclipse will automatically find all the hbase modules for you.

If you install m2eclipse and import HBase in your workspace, you will have to fix your eclipse Build Path. Remove `target` folder, add `target/generated-jamon` and `target/generated-sources/java` folders. You may also remove from your Build Path the exclusions on the `src/main/resources` and `src/test/resources` to avoid error message in the console 'Failed to execute goal org.apache.maven.plugins:maven-antrun-plugin:1.6:run (default) on project hbase: 'An Ant BuildException has occured: Replace: source file .../target/classes/hbase-default.xml doesn't exist'. This will also reduce the eclipse build cycles and make your life easier when developing.

15.2.1.5. Import into eclipse with the command line

For those not inclined to use m2eclipse, you can generate the Eclipse files from the command line. First, run (you should only have to do this once):

```
mvn clean install -DskipTests
```

and then close Eclipse and execute...

```
mvn eclipse:eclipse
```

... from your local HBase project directory in your workspace to generate some new `.project` and `.classpath`files. Then reopen Eclipse, and import the .project file in the HBase directory to a workspace.

15.2.1.6. Maven Classpath Variable

The `M2_REPO` classpath variable needs to be set up for the project. This needs to be set to your local Maven repository, which is usually `~/.m2/repository`

If this classpath variable is not configured, you will see compile errors in Eclipse like this...

```
Description     Resource        Path    Location        Type
The project cannot be built until build path errors are resolved        hbase           Unknown Java Problem
Unbound classpath variable: 'M2_REPO/asm/asm/3.1/asm-3.1.jar' in project 'hbase'        hbase           Build path      Build Pa
Unbound classpath variable: 'M2_REPO/com/github/stephenc/high-scale-lib/high-scale-lib/1.1.1/high-scale-lib-1.1.1.jar' in projec
Unbound classpath variable: 'M2_REPO/com/google/guava/guava/r09/guava-r09.jar' in project 'hbase'       hbase           Build pa
Unbound classpath variable: 'M2_REPO/com/google/protobuf/protobuf-java/2.3.0/protobuf-java-2.3.0.jar' in project 'hbase'
```

15.2.1.7. Eclipse Known Issues

Eclipse will currently complain about `Bytes.java`. It is not possible to turn these errors off.

```
Description     Resource        Path    Location        Type
Access restriction: The method arrayBaseOffset(Class) from the type Unsafe is not accessible due to restriction on required libr
Access restriction: The method arrayIndexScale(Class) from the type Unsafe is not accessible due to restriction on required libr
Access restriction: The method getLong(Object, long) from the type Unsafe is not accessible due to restriction on required libra
```

15.2.1.8. Eclipse - More Information

For additional information on setting up Eclipse for HBase development on Windows, see Michael Morello's blog on the topic.

15.3. 创建HBase

This section will be of interest only to those building HBase from source.

15.3.1. Building in snappy compression support

Pass `-Dsnappy` to trigger the snappy maven profile for building snappy native libs into hbase.

15.3.2. Building the HBase tarball

Do the following to build the HBase tarball. Passing the -Drelease will generate javadoc and run the RAT plugin to verify licenses on source.

```
% MAVEN_OPTS="-Xmx2g" mvn clean site install assembly:single -Dmaven.test.skip -Prelease
```

15.3.3. Adding an HBase release to Apache's Maven Repository

Follow the instructions at Publishing Maven Artifacts. The 'trick' to making it all work is answering the questions put to you by the mvn release plugin properly, making sure it is using the actual branch AND before doing the **mvn release:perform** step, VERY IMPORTANT, check and if necessary hand edit the release.properties file that was put under `${HBASE_HOME}` by the previous step, **release:perform**. You need to edit it to make it point at right locations in SVN.

Use maven 3.0.x.

At the **mvn release:perform** step, before starting, if you are for example releasing hbase 0.92.0, you need to make sure the pom.xml version is 0.92.0-SNAPSHOT. This needs to be checked in. Since we do the maven release after actual release, I've been doing this checkin into a particular tag rather than into the actual release tag. So, say we released hbase 0.92.0 and now we want to do the release to the maven repository, in svn, the 0.92.0 release will be tagged 0.92.0. Making the maven release, copy the 0.92.0 tag to 0.92.0mvn. Check out this tag and change the version therein and commit.

Here is how I'd answer the questions at **release:prepare** time:

```
What is the release version for "HBase"? (org.apache.hbase:hbase) 0.92.0: :
What is SCM release tag or label for "HBase"? (org.apache.hbase:hbase) hbase-0.92.0: : 0.92.0mvnrelease
What is the new development version for "HBase"? (org.apache.hbase:hbase) 0.92.1-SNAPSHOT: :
[INFO] Transforming 'HBase'...
```

A strange issue I ran into was the one where the upload into the apache repository was being sprayed across multiple apache machines making it so I could not release. 参考 INFRA-4482 Why is my upload to mvn spread across multiple repositories?.

Here is my `~/.m2/settings.xml`.

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                      http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <servers>
    <!- To publish a snapshot of some part of Maven -->
    <server>
      <id>apache.snapshots.https</id>
      <username>YOUR_APACHE_ID
      </username>
      <password>YOUR_APACHE_PASSWORD
      </password>
    </server>
    <!-- To publish a website using Maven -->
    <!-- To stage a release of some part of Maven -->
    <server>
      <id>apache.releases.https</id>
      <username>YOUR_APACHE_ID
      </username>
      <password>YOUR_APACHE_PASSWORD
      </password>
    </server>
  </servers>
  <profiles>
    <profile>
      <id>apache-release</id>
      <properties>
    <gpg.keyname>YOUR_KEYNAME</gpg.keyname>
    <!--Keyname is something like this ... 00A5F21E... do gpg --list-keys to find it-->
    <gpg.passphrase>YOUR_KEY_PASSWORD
    </gpg.passphrase>
      </properties>
    </profile>
  </profiles>
</settings>
```

When you run **release:perform**, pass **-Papache-release** else it will not 'sign' the artifacts it uploads.

If you see run into the below, its because you need to edit version in the pom.xml and add `-SNAPSHOT` to the version (and commit).

```
[INFO] Scanning for projects...
[INFO] Searching repository for plugin with prefix: 'release'.
[INFO] ------------------------------------------------------------------------
```

```
[INFO] Building HBase
[INFO]    task-segment: [release:prepare] (aggregator-style)
[INFO] ------------------------------------------------------------------------
[INFO] [release:prepare {execution: default-cli}]
[INFO] ------------------------------------------------------------------------
[ERROR] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] You don't have a SNAPSHOT project in the reactor projects list.
[INFO] ------------------------------------------------------------------------
[INFO] For more information, run Maven with the -e switch
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3 seconds
[INFO] Finished at: Sat Mar 26 18:11:07 PDT 2011
[INFO] Final Memory: 35M/423M
[INFO] -----------------------------------------------------------------------
```

15.3.4. Build Gotchas

If you see `Unable to find resource 'VM_global_library.vm'`, ignore it. Its not an error. It is officially ugly though.

15.4. Adding an Apache HBase release to Apache's Maven Repository

Follow the instructions at Publishing Maven Artifacts after reading the below miscellaney.

You must use maven 3.0.x (Check by running **mvn -version**).

Let me list out the commands I used first. The sections that follow dig in more on what is going on. In this example, we are releasing the 0.92.2 jar to the apache maven repository.

```
  # First make a copy of the tag we want to release; presumes the release has been tagged already    # We do this because we need
```

Below is more detail on the commmands listed above.

At the **mvn release:perform** step, before starting, if you are for example releasing hbase 0.92.2, you need to make sure the pom.xml version is 0.92.2-SNAPSHOT. This needs to be checked in. Since we do the maven release after actual release, I've been doing this checkin into a copy of the release tag rather than into the actual release tag itself (presumes the release has been properly tagged in svn). So, say we released hbase 0.92.2 and now we want to do the release to the maven repository, in svn, the 0.92.2 release will be tagged 0.92.2. Making the maven release, copy the 0.92.2 tag to 0.92.2mvn. Check out this tag and change the version therein and commit.

Currently, the mvn release wants to go against trunk. I haven't figured how to tell it to do otherwise so I do the below hack. The hack comprises answering the questions put to you by the mvn release plugin properly, then immediately control-C'ing the build after the last question asked as the build release step starts to run. After control-C'ing it, You'll notice a release.properties in your build dir. Review it. Make sure it is using the proper branch -- it tends to use trunk rather than the 0.92.2mvn or whatever that you want it to use -- so hand edit the release.properties file that was put under ${HBASE_HOME} by the **release:perform** invocation. When done, resstart the **release:perform**.

Here is how I'd answer the questions at **release:prepare** time:

```
What is the release version for "HBase"? (org.apache.hbase:hbase) 0.92.2: :  What is SCM release tag or label for "HBase"? (org.a
```

When you run **release:perform**, pass **-Papache-release** else it will not 'sign' the artifacts it uploads.

A strange issue I ran into was the one where the upload into the apache repository was being sprayed across multiple apache machines making it so I could not release. SeeINFRA-4482 Why is my upload to mvn spread across multiple repositories?.

Here is my ~/.m2/settings.xml. This is read by the release plugin. The apache-release profile will pick up your gpg key setup from here if you've specified it into the file. The password can be maven encrypted as suggested in the "Publishing Maven Artifacts" but plain text password works too (just don't let anyone see your local settings.xml).

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"    xsi:schemaLo
```

If you see run into the below, its because you need to edit version in the pom.xml and add -SNAPSHOT to the version (and commit).

```
[INFO] Scanning for projects...  [INFO] Searching repository for plugin with prefix: 'release'.  [INFO] ------------------------
```

15.5. Generating the HBase Reference Guide

The manual is marked up using docbook. We then use the docbkx maven plugin to transform the markup to html. This plugin is run when you specify the **site** goal as in when you run **mvn site** or you can call the plugin explicitly to just generate the manual by doing **mvn docbkx:generate-html** (TODO: It looks like you have to run **mvn site**first because docbkx wants to include a transformed hbase-default.xml. Fix). When you run mvn site, we do the document generation twice, once to generate the multipage manual and then again for the single page manual (the single page version is easier to search).

15.6. Updating hbase.apache.org

15.6.1. Contributing to hbase.apache.org

The Apache HBase apache web site (including this reference guide) is maintained as part of the main Apache HBase source tree, under /src/main/docbkx and /src/main/site [30]. The former -- docbkx -- is this reference guide as a bunch of xml marked up using docbook; the latter is the hbase site (the navbars, the header, the layout, etc.), and some of the documentation, legacy pages mostly that are in the process of being merged into the docbkx tree that is converted to html by a maven plugin by the site build.

To contribute to the reference guide, edit these files under site or docbkx and submit them as a patch (see Section 15.11, "Submitting Patches"). Your Jira should contain a summary of the changes in each section (see HBASE-6081 for an example).

To generate the site locally while you're working on it, run:

```
mvn site
```

Then you can load up the generated HTML files in your browser (file are under /target/site).

15.6.2. Publishing hbase.apache.org

As of INFRA-5680 Migrate apache hbase website, to publish the website, build it, and then deploy it over a checkout ofhttps://svn.apache.org/repos/asf/hbase/hbase.apache.org/trunk. Finally, check it in. For example, if trunk is checked out out at /Users/stack/checkouts/trunk and the hbase website, hbase.apache.org, is checked out at /Users/stack/checkouts/hbase.apache.org/trunk, to update the site, do the following:

```
              # Build the site and deploy it to the checked out directory                # Getting the javadoc into site is a lit
```

Now check the deployed site by viewing in a brower, browse to file:////Users/stack/checkouts/hbase.apache.org/trunk/index.html and check all is good. If all checks out, commit it and your new build will show up immediately at http://hbase.apache.org

```
              $ cd /Users/stack/checkouts/hbase.apache.org/trunk                $ svn status                # Do an svn add of ar
```

15.7. 测试

Developers, at a minimum, should familiarize themselves with the unit test detail; unit tests in HBase have a character not usually seen in other projects.

15.7.1. HBase 模块

As of 0.96, HBase is split into multiple modules which creates "interesting" rules for how and where tests are written. If you are writting code for `hbase-server`, see Section 15.7.2, "Unit Tests" for how to write your tests; these tests can spin up a minicluster and will need to be categorized. For any other module, for example `hbase-common`, the tests must be strict unit tests and just test the class under test - no use of the HBaseTestingUtility or minicluster is allowed (or even possible given the dependency tree).

15.7.1.1. 在其他模块中运行测试

If the module you are developing in has no other dependencies on other HBase modules, then you can cd into that module and just run:

```
mvn test
```

which will just run the tests IN THAT MODULE. If there are other dependencies on other modules, then you will have run the command from the ROOT HBASE DIRECTORY. This will run the tests in the other modules, unless you specify to skip the tests in that module. For instance, to skip the tests in the hbase-server module, you would run:

```
mvn clean test -Dskip-server-tests
```

from the top level directory to run all the tests in modules other than hbase-server. Note that you can specify to skip tests in multiple modules as well as just for a single module. For example, to skip the tests in `hbase-server` and `hbase-common`, you would run:

```
mvn clean test -Dskip-server-tests -Dskip-common-tests
```

Also, keep in mind that if you are running tests in the `hbase-server` module you will need to apply the maven profiles discussed in Section 15.7.2.4, "Running tests" to get the tests to run properly.

15.7.2. 单元测试

Apache HBase unit tests are subdivided into four categories: small, medium, large, and integration with corresponding JUnit categories: SmallTests, MediumTests, LargeTests,IntegrationTests. JUnit categories are denoted using java annotations and look like this in your unit test code.

```
...   @Category(SmallTests.class)  public class TestHRegionInfo {    @Test    public void testCreateHRegionInfoName() throws Excep
```

The above example shows how to mark a unit test as belonging to the small category. All unit tests in HBase have a categorization.

The first three categories, small, medium, and large are for tests run when you type $ mvn test; i.e. these three categorizations are for HBase unit tests. The integration category is for not for unit tests but for integration tests. These are run when you invoke $ mvn verify. Integration tests are described in Section 15.7.5, "Integration Tests" and will not be discussed further in this section on HBase unit tests.

Apache HBase uses a patched maven surefire plugin and maven profiles to implement its unit test characterizations.

Read the below to figure which annotation of the set small, medium, and large to put on your new HBase unit test.

15.7.2.1. Small Tests

Small tests are executed in a shared JVM. We put in this category all the tests that can be executed quickly in a shared JVM. The maximum execution time for a small test is 15 seconds, and small tests should not use a (mini)cluster.

15.7.2.2. Medium Tests

Medium tests represent tests that must be executed before proposing a patch. They are designed to run in less than 30 minutes altogether, and are quite stable in their results. They are designed to last less than 50 seconds individually. They can use a cluster, and each of them is executed in a separate JVM.

15.7.2.3. Large Tests

Large tests are everything else. They are typically large-scale tests, regression tests for specific bugs, timeout tests, performance tests. They are executed before a commit on the pre-integration machines. They can be run on the developer machine as well.

15.7.2.4. Integration Tests

Integration tests are system level tests. See Section 15.7.5, "Integration Tests" for more info.

15.7.3. Running tests

Below we describe how to run the Apache HBase junit categories.

15.7.3.1. Default: small and medium category tests

Running

```
mvn test
```

will execute all small tests in a single JVM (no fork) and then medium tests in a separate JVM for each test instance. Medium tests are NOT executed if there is an error in a small test. Large tests are NOT executed. There is one report for small tests, and one report for medium tests if they are executed.

15.7.3.2. Running all tests

Running

```
mvn test -P runAllTests
```

will execute small tests in a single JVM then medium and large tests in a separate JVM for each test. Medium and large tests are NOT executed if there is an error in a small test. Large tests are NOT executed if there is an error in a small or medium test. There is one report for small tests, and one report for medium and large tests if they are executed.

15.7.3.3. Running a single test or all tests in a package

To run an individual test, e.g. MyTest, do

```
mvn test -Dtest=MyTest
```

You can also pass multiple, individual tests as a comma-delimited list:

```
mvn test -Dtest=MyTest1,MyTest2,MyTest3
```

You can also pass a package, which will run all tests under the package:

```
mvn test -Dtest=org.apache.hadoop.hbase.client.*
```

When -Dtest is specified, localTests profile will be used. It will use the official release of maven surefire, rather than our custom surefire plugin, and the old connector (The HBase build uses a patched version of the maven surefire plugin). Each junit tests is executed in a separate JVM (A fork per test class). There is no parallelization when tests are running in this mode. You will see a new message at the end of the -report: "[INFO] Tests are skipped". It's harmless. While you need to make sure the sum of Tests run: in the Results : section of test reports matching the number of tests you specified because no error will be reported when a non-existent test case is specified.

15.7.3.4. Other test invocation permutations

Running

```
mvn test -P runSmallTests
```

will execute "small" tests only, using a single JVM.

Running

```
mvn test -P runMediumTests
```

will execute "medium" tests only, launching a new JVM for each test-class.

Running

```
mvn test -P runLargeTests
```

will execute "large" tests only, launching a new JVM for each test-class.

For convenience, you can run

```
mvn test -P runDevTests
```

to execute both small and medium tests, using a single JVM.

15.7.3.5. Running tests faster

By default, $ mvn test -P runAllTests runs 5 tests in parallel. It can be increased on a developer's machine. Allowing that you can have 2 tests in parallel per core, and you need about 2Gb of memory per test (at the extreme), if you have an 8 core, 24Gb box, you can have 16 tests in parallel. but the memory available limits it to 12 (24/2), To run all tests with 12 tests in parallell, do this: **mvn test -P runAllTests -Dsurefire.secondPartThreadCount=12**. To increase the speed, you can as well use a ramdisk. You will need 2Gb of memory to run all tests. You will also need to delete the files between two test run. The typical way to configure a ramdisk on Linux is:

```
$ sudo mkdir /ram2G  sudo mount -t tmpfs -o size=2048M tmpfs /ram2G
```

You can then use it to run all HBase tests with the command: **mvn test -P runAllTests -Dsurefire.secondPartThreadCount=12 -Dtest.build.data.basedirectory=/ram2G**

15.7.3.6. hbasetests.sh

It's also possible to use the script **hbasetests.sh**. This script runs the medium and large tests in parallel with two maven instances, and provides a single report. This script does not use the hbase version of surefire so no parallelization is being done other than the two maven instances the script sets up. It must be executed from the directory which contains the pom.xml.

For example running

```
./dev-support/hbasetests.sh
```

will execute small and medium tests. Running

```
./dev-support/hbasetests.sh runAllTests
```

will execute all tests. Running

```
./dev-support/hbasetests.sh replayFailed
```

will rerun the failed tests a second time, in a separate jvm and without parallelisation.

15.7.3.7. Test Resource Checker

A custom Maven SureFire plugin listener checks a number of resources before and after each HBase unit test runs and logs its findings at the end of the test output files which can be found in target/surefire-reports per Maven module (Tests write test reports named for the test class into this directory. Check the *-out.txt files). The resources counted are the number of threads, the number of file descriptors, etc. If the number has increased, it adds a *LEAK?* comment in the logs. As you can have an HBase instance running in the background, some threads can be deleted/created without any specific action in the test. However, if the test does not work as expected, or if the test should not impact these resources, it's worth checking these log lines ...hbase.ResourceChecker(157): before... and ...hbase.ResourceChecker(157): after.... For example: 2012-09-26 09:22:15,315 INFO [pool-1-thread-1] hbase.ResourceChecker(157): after: regionserver.TestColumnSeeking#testReseeking Thread=65 (was 65), OpenFileDescriptor=107 (was 107), MaxFileDescriptor=10240 (was 10240), ConnectionCount=1 (was 1)

15.7.4. Writing Tests

15.7.4.1. General rules

- As much as possible, tests should be written as category small tests.
- All tests must be written to support parallel execution on the same machine, hence they should not use shared resources as fixed ports or fixed file names.
- Tests should not overlog. More than 100 lines/second makes the logs complex to read and use i/o that are hence not available for the other tests.
- Tests can be written with HBaseTestingUtility. This class offers helper functions to create a temp directory and do the cleanup, or to start a cluster.

15.7.4.2. Categories and execution time

- All tests must be categorized, if not they could be skipped.
- All tests should be written to be as fast as possible.
- Small category tests should last less than 15 seconds, and must not have any side effect.
- Medium category tests should last less than 50 seconds.
- Large category tests should last less than 3 minutes. This should ensure a good parallelization for people using it, and ease the analysis when the test fails.

15.7.4.3. Sleeps in tests

Whenever possible, tests should not use Thread.sleep, but rather waiting for the real event they need. This is faster and clearer for the reader. Tests should not do aThread.sleep without testing an ending condition. This allows understanding what the test is waiting for. Moreover, the test will work whatever the machine performance is. Sleep should be minimal to be as fast as possible. Waiting for a variable should be done in a 40ms sleep loop. Waiting for a socket operation should be done in a 200 ms sleep loop.

15.7.4.4. Tests using a cluster

Tests using a HRegion do not have to start a cluster: A region can use the local file system. Start/stopping a cluster cost around 10 seconds. They should not be started per test method but per test class. Started cluster must be shutdown using HBaseTestingUtility#shutdownMiniCluster, which cleans the directories. As most as possible, tests should use the default settings for the cluster. When they don't, they should document it. This will allow to share the cluster later.

15.7.5. Integration Tests

HBase integration/system tests are tests that are beyond HBase unit tests. They are generally long-lasting, sizeable (the test can be asked to 1M rows or 1B rows), targetable (they can take configuration that will point them at the ready-made cluster they are to run against; integration tests do not include cluster start/stop code), and verifying success, integration tests rely on public APIs only; they do not attempt to examine server internals asserting success/fail. Integration tests are what you would run when you need to more elaborate proofing of a release candidate beyond what unit tests can do. They are not generally run on the Apache Continuous Integration build server, however, some sites opt to run integration tests as a part of their continuous testing on an actual cluster.

Integration tests currently live under the src/test directory in the hbase-it submodule and will match the regex: **/IntegrationTest*.java. All integration tests are also annotated with @Category(IntegrationTests.class).

Integration tests can be run in two modes: using a mini cluster, or against an actual distributed cluster. Maven failsafe is used to run the tests using the mini cluster. IntegrationTestsDriver class is used for executing the tests against a distributed cluster. Integration tests SHOULD NOT assume that they are running against a mini cluster, and SHOULD NOT use private API's to access cluster state. To interact with the distributed or mini cluster uniformly, IntegrationTestingUtility, and HBaseClusterclasses, and public client API's can be used.

15.7.5.1. Running integration tests against mini cluster

HBase 0.92 added a verify maven target. Invoking it, for example by doing mvn verify, will run all the phases up to and including the verify phase via the maven failsafe plugin, running all the above mentioned HBase unit tests as well as tests that are in the HBase integration test group. After you have completed

```
mvn install -DskipTests
```

You can run just the integration tests by invoking:

```
cd hbase-it  mvn verify
```

If you just want to run the integration tests in top-level, you need to run two commands. First:

```
mvn failsafe:integration-test
```

This actually runs ALL the integration tests.

Note

This command will always output BUILD SUCCESS even if there are test failures.

At this point, you could grep the output by hand looking for failed tests. However, maven will do this for us; just use:

```
mvn failsafe:verify
```

The above command basically looks at all the test results (so don't remove the 'target' directory) for test failures and reports the results.

15.7.5.1.1. Running a subset of Integration tests

This is very similar to how you specify running a subset of unit tests (see above), but use the property it.test instead of test. To just run IntegrationTestClassXYZ.java, use:

```
mvn failsafe:integration-test -Dit.test=IntegrationTestClassXYZ
```

The next thing you might want to do is run groups of integration tests, say all integration tests that are named IntegrationTestClassX*.java:

```
mvn failsafe:integration-test -Dit.test=*ClassX*
```

This runs everything that is an integration test that matches *ClassX*. This means anything matching: "**/IntegrationTest*ClassX*". You can also run multiple groups of integration tests using comma-delimited lists (similar to unit tests). Using a list of matches still supports full regex matching for each of the groups.This would look something like:

```
mvn failsafe:integration-test -Dit.test=*ClassX*, *ClassY
```

15.7.5.2. Running integration tests against distributed cluster

If you have an already-setup HBase cluster, you can launch the integration tests by invoking the class IntegrationTestsDriver. You may have to run test-compile first. The configuration will be picked by the bin/hbase script.

```
mvn test-compile
```

Then launch the tests with:

```
bin/hbase [--config config_dir] org.apache.hadoop.hbase.IntegrationTestsDriver [-test=class_regex]
```

This execution will launch the tests under hbase-it/src/test, having @Category(IntegrationTests.class) annotation, and a name starting with IntegrationTests. If specified, class_regex will be used to filter test classes. The regex is checked against full class name; so, part of class name can be used. IntegrationTestsDriver uses Junit to run the tests. Currently there is no support for running integration tests against a distributed cluster using maven (see HBASE-6201).

The tests interact with the distributed cluster by using the methods in the DistributedHBaseCluster (implementing HBaseCluster) class, which in turn uses a pluggableClusterManager. Concrete implementations provide actual functionality for carrying out deployment-specific and environment-dependent tasks (SSH, etc). The defaultClusterManager is HBaseClusterManager, which uses SSH to remotely execute start/stop/kill/signal commands, and assumes some posix commands (ps, etc). Also assumes the user running the test has enough "power" to start/stop servers on the remote machines. By default, it picks up HBASE_SSH_OPTS, HBASE_HOME, HBASE_CONF_DIR from the env, and usesbin/hbase-daemon.sh to carry out the actions. Currently tarball deployments, deployments which uses hbase-daemons.sh, and Apache Ambari deployments are supported. /etc/init.d/ scripts are not supported for now, but it can be easily added. For other deployment options, a ClusterManager can be implemented and plugged in.

15.7.5.3. Destructive integration / system tests

In 0.96, a tool named ChaosMonkey has been introduced. It is modeled after the same-named tool by Netflix. Some of the tests use ChaosMonkey to simulate faults in the running cluster in the way of killing random servers, disconnecting servers, etc. ChaosMonkey can also be used as a stand-alone tool to run a (misbehaving) policy while you are running other tests.

ChaosMonkey defines Action's and Policy's. Actions are sequences of events. We have at least the following actions:

- Restart active master (sleep 5 sec)
- Restart random regionserver (sleep 5 sec)
- Restart random regionserver (sleep 60 sec)
- Restart META regionserver (sleep 5 sec)
- Restart ROOT regionserver (sleep 5 sec)
- Batch restart of 50% of regionservers (sleep 5 sec)
- Rolling restart of 100% of regionservers (sleep 5 sec)

Policies on the other hand are responsible for executing the actions based on a strategy. The default policy is to execute a random action every minute based on predefined action weights. ChaosMonkey executes predefined named policies until it is stopped. More than one policy can be active at any time.

To run ChaosMonkey as a standalone tool deploy your HBase cluster as usual. ChaosMonkey uses the configuration from the bin/hbase script, thus no extra configuration needs to be done. You can invoke the ChaosMonkey by running:

```
bin/hbase org.apache.hadoop.hbase.util.ChaosMonkey
```

This will output smt like:

```
12/11/19 23:21:57 INFO util.ChaosMonkey: Using ChaosMonkey Policy: class org.apache.hadoop.hbase.util.ChaosMonkey$PeriodicRandomA
```

As you can see from the log, ChaosMonkey started the default PeriodicRandomActionPolicy, which is configured with all the available actions, and ran RestartActiveMaster and RestartRandomRs actions. ChaosMonkey tool, if run from command line, will keep on running until the process is killed.

15.8. Maven Build Commands

All commands executed from the local HBase project directory.

Note: use Maven 3 (Maven 2 may work but we suggest you use Maven 3).

15.8.1. Compile

```
mvn compile
```

15.8.2. Running all or individual Unit Tests

See the Section 15.7.3, “Running tests” section above in Section 15.7.2, “Unit Tests”

15.8.3. Building against various hadoop versions.

As of 0.96, Apache HBase supports building against Apache Hadoop versions: 1.0.3, 2.0.0-alpha and 3.0.0-SNAPSHOT. By default, we will build with Hadoop-1.0.3. To change the version to run with Hadoop-2.0.0-alpha, you would run:

```
mvn -Dhadoop.profile=2.0 ...
```

That is, designate build with hadoop.profile 2.0. Pass 2.0 for hadoop.profile to build against hadoop 2.0. Tests may not all pass as of this writing so you may need to pass -DskipTests unless you are inclined to fix the failing tests.

Similarly, for 3.0, you would just replace the profile value. Note that Hadoop-3.0.0-SNAPSHOT does not currently have a deployed maven artificat - you will need to build and install your own in your local maven repository if you want to run against this profile.

In earilier verions of Apache HBase, you can build against older versions of Apache Hadoop, notably, Hadoop 0.22.x and 0.23.x. If you are running, for example HBase-0.94 and wanted to build against Hadoop 0.23.x, you would run with:

```
mvn -Dhadoop.profile=22 ...
```

15.9. Getting Involved

Apache HBase gets better only when people contribute!

As Apache HBase is an Apache Software Foundation project, see Appendix H, *HBase and the Apache Software Foundation* for more information about how the ASF functions.

15.9.1. Mailing Lists

Sign up for the dev-list and the user-list. See the mailing lists page. Posing questions - and helping to answer other people's questions - is encouraged! There are varying levels of experience on both lists so patience and politeness are encouraged (and please stay on topic.)

15.9.2. Jira

Check for existing issues in Jira. If it's either a new feature request, enhancement, or a bug, file a ticket.

15.9.2.1. Jira Priorities

The following is a guideline on setting Jira issue priorities:

- Blocker: Should only be used if the issue WILL cause data loss or cluster instability reliably.
- Critical: The issue described can cause data loss or cluster instability in some cases.
- Major: Important but not tragic issues, like updates to the client API that will add a lot of much-needed functionality or significant bugs that need to be fixed but that don't cause data loss.
- Minor: Useful enhancements and annoying but not damaging bugs.
- Trivial: Useful enhancements but generally cosmetic.

15.9.2.2. Code Blocks in Jira Comments

A commonly used macro in Jira is {code}. If you do this in a Jira comment...

```
{code}     code snippet  {code}
```

... Jira will format the code snippet like code, instead of a regular comment. It improves readability.

15.10. Developing

15.10.1. Codelines

Most development is done on TRUNK. However, there are branches for minor releases (e.g., 0.90.1, 0.90.2, and 0.90.3 are on the 0.90 branch).

If you have any questions on this just send an email to the dev dist-list.

15.10.2. Unit Tests

In HBase we use JUnit 4. If you need to run miniclusters of HDFS, ZooKeeper, HBase, or MapReduce testing, be sure to checkout the HBaseTestingUtility. Alex Baranau of Sematext describes how it can be used in HBase Case-Study: Using HBaseTestingUtility for Local Testing and Development (2010).

15.10.2.1. Mockito

Sometimes you don't need a full running server unit testing. For example, some methods can make do with a a org.apache.hadoop.hbase.Server instance or aorg.apache.hadoop.hbase.master.MasterServices Interface reference rather than a full-blown org.apache.hadoop.hbase.master.HMaster. In these cases, you maybe able to get away with a mocked Server instance. For example:

```
TODO...
```

15.10.3. Code Standards

See Section 15.2.1.1, “Code Formatting” and Section 15.11.5, “Common Patch Feedback”.

Also, please pay attention to the interface stability/audience classifications that you will see all over our code base. They look like this at the head of the class:

```
@InterfaceAudience.Public  @InterfaceStability.Stable
```

If the InterfaceAudience is Private, we can change the class (and we do not need to include a InterfaceStability mark). If a class is marked Public but its InterfaceStability is marked Unstable, we can change it. If it's marked Public/Evolving, we're allowed to change it but should try not to. If it's Public and Stable we can't change it without a deprecation path or with a really GREAT reason.

When you add new classes, mark them with the annotations above if publically accessible. If you are not cleared on how to mark your additions, ask up on the dev list.

This convention comes from our parent project Hadoop.

15.10.4. Invariants

We don't have many but what we have we list below. All are subject to challenge of course but until then, please hold to the rules of the road.

15.10.4.1. No permanent state in ZooKeeper

ZooKeeper state should transient (treat it like memory). If deleted, hbase should be able to recover and essentially be in the same state[31].

15.10.5. Running In-Situ

If you are developing Apache HBase, frequently it is useful to test your changes against a more-real cluster than what you find in unit tests. In this case, HBase can be run directly from the source in local-mode. All you need to do is run:

```
${HBASE_HOME}/bin/start-hbase.sh
```

This will spin up a full local-cluster, just as if you had packaged up HBase and installed it on your machine.

Keep in mind that you will need to have installed HBase into your local maven repository for the in-situ cluster to work properly. That is, you will need to run:

```
mvn clean install -DskipTests
```

to ensure that maven can find the correct classpath and dependencies. Generally, the above command is just a good thing to try running first, if maven is acting oddly.

15.11. Submitting Patches

If you are new to submitting patches to open source or new to submitting patches to Apache, I'd suggest you start by reading the On Contributing Patches page from Apache Commons Project. Its a nice overview that applies equally to the Apache HBase Project.

15.11.1. Create Patch

See the aforementioned Apache Commons link for how to make patches against a checked out subversion repository. Patch files can also be easily generated from Eclipse, for example by selecting "Team -> Create Patch". Patches can also be created by git diff and svn diff.

Please submit one patch-file per Jira. For example, if multiple files are changed make sure the selected resource when generating the patch is a directory. Patch files can reflect changes in multiple files.

Make sure you review Section 15.2.1.1, “Code Formatting” for code style.

15.11.2. Patch File Naming

The patch file should have the Apache HBase Jira ticket in the name. For example, if a patch was submitted for Foo.java, then a patch file called Foo_HBASE_XXXX.patch would be acceptable where XXXX is the Apache HBase Jira number.

If you generating from a branch, then including the target branch in the filename is advised, e.g., HBASE-XXXX-0.90.patch.

15.11.3. Unit Tests

Yes, please. Please try to include unit tests with every code patch (and especially new classes and large changes). Make sure unit tests pass locally before submitting the patch.

Also, see Section 15.10.2.1, “Mockito”.

If you are creating a new unit test class, notice how other unit test classes have classification/sizing annotations at the top and a static method on the end. Be sure to include these in any new unit test files you generate. See Section 15.7, “Tests” for more on how the annotations work.

15.11.4. Attach Patch to Jira

The patch should be attached to the associated Jira ticket "More Actions -> Attach Files". Make sure you click the ASF license inclusion, otherwise the patch can't be considered for inclusion.

Once attached to the ticket, click "Submit Patch" and the status of the ticket will change. Committers will review submitted patches for inclusion into the codebase. Please understand that not every patch may get committed, and that feedback will likely be provided on the patch. Fear not, though, because the Apache HBase community is helpful!

15.11.5. Common Patch Feedback

The following items are representative of common patch feedback. Your patch process will go faster if these are taken into account *before* submission.

See the Java coding standards for more information on coding conventions in Java.

15.11.5.1. Space Invaders

Rather than do this...

```
if ( foo.equals( bar ) ) {     // don't do this
```

... do this instead...

```
if (foo.equals(bar)) {
```

Also, rather than do this...

```
foo = barArray[ i ];     // don't do this
```

... do this instead...

```
foo = barArray[i];
```

15.11.5.2. Auto Generated Code

Auto-generated code in Eclipse often looks like this...

```
 public void readFields(DataInput arg0) throws IOException {    // don't do this     foo = arg0.readUTF();
```

... do this instead ...

```
 public void readFields(DataInput di) throws IOException {     foo = di.readUTF();
```

See the difference? 'arg0' is what Eclipse uses for arguments by default.

15.11.5.3. Long Lines

Keep lines less than 100 characters.

```
Bar bar = foo.veryLongMethodWithManyArguments(argument1, argument2, argument3, argument4, argument5, argument6, argument7, argume
```

... do something like this instead ...

```
Bar bar = foo.veryLongMethodWithManyArguments(   argument1, argument2, argument3,argument4, argument5, argument6, argument7, argu
```

15.11.5.4. Trailing Spaces

This happens more than people would imagine.

```
Bar bar = foo.getBar();     <--- imagine there's an extra space(s) after the semicolon instead of a line break.
```

Make sure there's a line-break after the end of your code, and also avoid lines that have nothing but whitespace.

15.11.5.5. Implementing Writable

Applies pre-0.96 only

In 0.96, HBase moved to protobufs. The below section on Writables applies to 0.94.x and previous, not to 0.96 and beyond.

Every class returned by RegionServers must implement Writable. If you are creating a new class that needs to implement this interface, don't forget the default constructor.

15.11.5.6. Javadoc

This is also a very common feedback item. Don't forget Javadoc!

Javadoc warnings are checked during precommit. If the precommit tool gives you a '-1', please fix the javadoc issue. Your patch won't be committed if it adds such warnings.

15.11.5.7. Findbugs

Findbugs is used to detect common bugs pattern. As Javadoc, it is checked during the precommit build up on Apache's Jenkins, and as with Javadoc, please fix them. You can run findbugs locally with 'mvn findbugs:findbugs': it will generate the findbugs files locally. Sometimes, you may have to write code smarter than Findbugs. You can annotate your code to tell Findbugs you know what you're doing, by annotating your class with:

```
@edu.umd.cs.findbugs.annotations.SuppressWarnings(                      value="HE_EQUALS_USE_HASHCODE",                      just
```

Note that we're using the apache licensed version of the annotations.

15.11.5.8. Javadoc - Useless Defaults

Don't just leave the @param arguments the way your IDE generated them. Don't do this...

```
  /**     *     * @param bar             <---- don't do this!!!!     * @return                <---- or this!!!!     */     public
```

... either add something descriptive to the @param and @return lines, or just remove them. But the preference is to add something descriptive and useful.

15.11.5.9. One Thing At A Time, Folks

If you submit a patch for one thing, don't do auto-reformatting or unrelated reformatting of code on a completely different area of code.

Likewise, don't add unrelated cleanup or refactorings outside the scope of your Jira.

15.11.5.10. Ambigious Unit Tests

Make sure that you're clear about what you are testing in your unit tests and why.

15.11.6. ReviewBoard

Larger patches should go through ReviewBoard.

For more information on how to use ReviewBoard, see the ReviewBoard documentation.

15.11.7. Committing Patches

Committers do this. See How To Commit in the Apache HBase wiki.

Commiters will also resolve the Jira, typically after the patch passes a build.

15.11.7.1. Committers are responsible for making sure commits do not break the build or tests

If a committer commits a patch it is their responsibility to make sure it passes the test suite. It is helpful if contributors keep an eye out that their patch does not break the hbase build and/or tests but ultimately, a contributor cannot be expected to be up on the particular vagaries and interconnections that occur in a project like hbase. A committer should.

[30] Before 0.95.0, site and reference guide were at src/docbkx and src/site respectively

[31] There are currently a few exceptions that we need to fix around whether a table is enabled or disabled

16. ZooKeeper

一个分布式运行的HBase依赖一个zookeeper集群。所有的节点和客户端都必须能够访问zookeeper。默认的情况下HBase会管理一个zookeep集群。这个集群会随着HBase的启动而启动。当然，你也可以自己管理一个zookeeper集群，但需要配置HBase。你需要修改conf/hbase-env.sh里面的HBASE_MANAGES_ZK 来切换。这个值默认是true的，作用是让HBase启动的时候同时也启动zookeeper.

当HBase管理zookeeper的时候，你可以通过修改zoo.cfg来配置zookeeper，一个更加简单的方法是在 conf/hbase-site.xml里面修改zookeeper的配置。Zookeep的配置是作为property写在 hbase-site.xml里面的。option的名字是 hbase.zookeeper.property. 打个比方， clientPort 配置在xml里面的名字是 hbase.zookeeper.property.clientPort. 所有的默认值都是HBase决定的，包括zookeeper, 参见 Section 2.3.1.1, "HBase 默认配置". 可以查找 hbase.zookeeper.property 前缀，找到关于zookeeper的配置。 [33]

对于zookeepr的配置，你至少要在 hbase-site.xml中列出zookeepr的ensemble servers，具体的字段是 hbase.zookeeper.quorum. 该这个字段的默认值是 localhost，这个值对于分布式应用显然是不可以的. (远程连接无法使用).

我需要运行几个zookeeper?

你运行一个zookeeper也是可以的，但是在生产环境中，你最好部署3，5，7个节点。部署的越多，可靠性就越高，当然只能部署奇数个，偶数个是不可以的。你需要给每个zookeeper 1G左右的内存，如果可能的话，最好有独立的磁盘。 (独立磁盘可以确保zookeeper是高性能的。).如果你的集群负载很重，不要把Zookeeper和RegionServer运行在同一台机器上面。就像DataNodes 和 TaskTrackers一样

举个例子，HBase管理着的ZooKeeper集群在节点 *rs{1,2,3,4,5}.example.com*, 监听2222 端口(默认是2181)，并确保conf/hbase-env.sh文件中 HBASE_MANAGE_ZK的值是 true ，再编辑 conf/hbase-site.xml 设置 hbase.zookeeper.property.clientPort 和 hbase.zookeeper.quorum。你还可以设置 hbase.zookeeper.property.dataDir属性来把ZooKeeper保存数据的目录地址改掉。默认值是 /tmp ，这里在重启的时候会被操作系统删掉，可以把它修改到 /user/local/zookeeper.

```
<configuration>
  ...
  <property>
    <name>hbase.zookeeper.property.clientPort</name>
    <value>2222</value>
    <description>Property from ZooKeeper's config zoo.cfg.
    The port at which the clients will connect.
    </description>
  </property>
  <property>
    <name>hbase.zookeeper.quorum</name>
    <value>rs1.example.com,rs2.example.com,rs3.example.com,rs4.example.com,rs5.example.com</value>
    <description>Comma separated list of servers in the ZooKeeper Quorum.
    For example, "host1.mydomain.com,host2.mydomain.com,host3.mydomain.com".
    By default this is set to localhost for local and pseudo-distributed modes
    of operation. For a fully-distributed setup, this should be set to a full
    list of ZooKeeper quorum servers. If HBASE_MANAGES_ZK is set in hbase-env.sh
    this is the list of servers which we will start/stop ZooKeeper on.
    </description>
  </property>
  <property>
    <name>hbase.zookeeper.property.dataDir</name>
    <value>/usr/local/zookeeper</value>
    <description>Property from ZooKeeper's config zoo.cfg.
    The directory where the snapshot is stored.
    </description>
  </property>
  ...
</configuration>
```

ZooKeeper 维护

Be sure to set up the data dir cleaner described under Zookeeper Maintenance else you could have 'interesting' problems a couple of months in; i.e. zookeeper could start dropping sessions if it has to run through a directory of hundreds of thousands of logs which is wont to do around leader reelection time -- a process rare but run on occasion whether because a machine is dropped or happens to hiccup.

16.1. 和已有的ZooKeeper一起使用

让HBase使用一个已有的不被HBase托管的Zookeep集群，需要设置 conf/hbase-env.sh文件中的HBASE_MANAGES_ZK 属性为 false

```
...
# Tell HBase whether it should manage it's own instance of Zookeeper or not.
export HBASE_MANAGES_ZK=false
```

接下来，指明Zookeeper的host和端口。可以在 hbase-site.xml中设置, 也可以在HBase的CLASSPATH下面加一个zoo.cfg配置文件。HBase 会优先加载 zoo.cfg 里面的配置，把hbase-site.xml里面的覆盖掉.

当HBase托管ZooKeeper的时候，Zookeeper集群的启动是HBase启动脚本的一部分。但现在，你需要自己去运行。你可以这样做

```
${HBASE_HOME}/bin/hbase-daemons.sh {start,stop} zookeeper
```

你可以用这条命令启动ZooKeeper而不启动HBase. HBASE_MANAGES_ZK 的值是 false， 如果你想在HBase重启的时候不重启ZooKeeper,你可以这样做

对于独立Zoopkeeper的问题，你可以在 Zookeeper启动得到帮助.

16.2. 通过ZooKeeper 的SASL 认证

新版 HBase (>= 0.92)将支持连接到 ZooKeeper Quorum 进行SASL 认证。(Zookeeper 3.4.0 以上可用).

这里描述如何设置 HBase，以同 ZooKeeper Quorum实现互相认证. ZooKeeper/HBase 互相认证 (HBASE-2418) 是 HBase安全配置所必不可少的一部分 (HBASE-3025)。 为简化说明, 本节忽略所必需的额外配置 (HDFS 安全和 Coprocessor 配置)。 推荐使用HBase内置 Zookeeper 配置 (相对独立 Zookeeper quorum) 以简化学习.

16.2.1. 操作系统预置

需要一个工作 Kerberos KDC 配置. 每个 $HOST 运行一个 ZooKeeper 服务器, 应该有个主要的 zookeeper/$HOST. 对每个主机，为zookeeper/$HOST添加一个 key (使用 kadmin 或 kadmin.local 工具的 ktadd 命令)， 将该key文件复制到 $HOST, 并设置仅对该 $HOST 上运行 zookeeper 的用户只读。注意文件位置, 我们将在下面以 $PATH_TO_ZOOKEEPER_KEYTAB 使用。

Similarly, for each $HOST that will run an HBase server (master or regionserver), you should have a principle: hbase/$HOST. For each host, add a keytab file called hbase.keytab containing a service key for hbase/$HOST, copy this file to $HOST, and make it readable only to the user that will run an HBase service on $HOST. Note the location of this file, which we will use below as $PATH_TO_HBASE_KEYTAB.

Each user who will be an HBase client should also be given a Kerberos principal. This principal should usually have a password assigned to it (as opposed to, as with the HBase servers, a keytab file) which only this user knows. The client's principal's maxrenewlife should be set so that it can be renewed enough so that the user can complete their HBase client processes. For example, if a user runs a long-running HBase client process that takes at most 3 days, we might create this user's principal within kadmin with: addprinc -maxrenewlife 3days. The Zookeeper client and server libraries manage their own ticket refreshment by running threads that wake up periodically to do the refreshment.

On each host that will run an HBase client (e.g. hbase shell), add the following file to the HBase home directory's conf directory:

```
                   Client {
                     com.sun.security.auth.module.Krb5LoginModule required
                     useKeyTab=false
                     useTicketCache=true;
                   };
```

We'll refer to this JAAS configuration file as $CLIENT_CONF below.

16.2.2. HBase内置的 Zookeeper 配置

每个节点要运行一个 zookeeper, 一个主服务, 或一个 regionserver,在HBASE_HOME的conf目录中创建如下所示的 JAAS 配置文件:

```
                   Server {
                     com.sun.security.auth.module.Krb5LoginModule required
                     useKeyTab=true
                     keyTab="$PATH_TO_ZOOKEEPER_KEYTAB"
                     storeKey=true
                     useTicketCache=false
                     principal="zookeeper/$HOST";
                   };
                   Client {
                     com.sun.security.auth.module.Krb5LoginModule required
                     useKeyTab=true
                     useTicketCache=false
                     keyTab="$PATH_TO_HBASE_KEYTAB"
                     principal="hbase/$HOST";
                   };
```

$PATH_TO_HBASE_KEYTAB 和 $PATH_TO_ZOOKEEPER_KEYTAB文件是上面创建的。 $HOST 该节点主机名.

Server 由 Zookeeper quorum 服务器使用, Client 节由 HBase master 和 regionserver使用。 The path to this file should be substituted for the text $HBASE_SERVER_CONF in the hbase-env.sh listing below.

The path to this file should be substituted for the text $CLIENT_CONF in the hbase-env.sh listing below.

Modify your hbase-env.sh to include the following:

```
export HBASE_OPTS="-Djava.security.auth.login.config=$CLIENT_CONF"
export HBASE_MANAGES_ZK=true
export HBASE_ZOOKEEPER_OPTS="-Djava.security.auth.login.config=$HBASE_SERVER_CONF"
export HBASE_MASTER_OPTS="-Djava.security.auth.login.config=$HBASE_SERVER_CONF"
export HBASE_REGIONSERVER_OPTS="-Djava.security.auth.login.config=$HBASE_SERVER_CONF"
```

where $HBASE_SERVER_CONF and $CLIENT_CONF are the full paths to the JAAS configuration files created above.

Modify your hbase-site.xml on each node that will run zookeeper, master or regionserver to contain:

```
<configuration>
  <property>
    <name>hbase.zookeeper.quorum</name>
    <value>$ZK_NODES</value>
  </property>
  <property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
  </property>
  <property>
    <name>hbase.zookeeper.property.authProvider.1</name>
    <value>org.apache.zookeeper.server.auth.SASLAuthenticationProvider</value>
  </property>
  <property>
    <name>hbase.zookeeper.property.kerberos.removeHostFromPrincipal</name>
    <value>true</value>
  </property>
  <property>
    <name>hbase.zookeeper.property.kerberos.removeRealmFromPrincipal</name>
    <value>true</value>
  </property>
</configuration>
```

where $ZK_NODES is the comma-separated list of hostnames of the Zookeeper Quorum hosts.

Start your hbase cluster by running one or more of the following set of commands on the appropriate hosts:

```
bin/hbase zookeeper start
bin/hbase master start
bin/hbase regionserver start
```

16.2.3. 外部 Zookeeper 配置

增加 JAAS 配置文件:

```
Client {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  useTicketCache=false
  keyTab="$PATH_TO_HBASE_KEYTAB"
  principal="hbase/$HOST";
};
```

$PATH_TO_HBASE_KEYTAB 是上面创建的keytab，以便 HBase 服务可以在本主机运行，$HOST 是该节点的 hostname . 将该配置放到 HBase home配置目录.在下面的 $HBASE_SERVER_CONF 进行引用.

修改 hbase-env.sh 增加如下项:

```
export HBASE_OPTS="-Djava.security.auth.login.config=$CLIENT_CONF"
export HBASE_MANAGES_ZK=false
export HBASE_MASTER_OPTS="-Djava.security.auth.login.config=$HBASE_SERVER_CONF"
export HBASE_REGIONSERVER_OPTS="-Djava.security.auth.login.config=$HBASE_SERVER_CONF"
```

修改每个节点的 hbase-site.xml 包含下面内容:

```
<configuration>
  <property>
    <name>hbase.zookeeper.quorum</name>
    <value>$ZK_NODES</value>
  </property>
  <property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
  </property>
</configuration>
```

$ZK_NODES是逗号分隔的Zookeeper Quorum主机名列表。

每个Zookeeper Quorum节点增加 zoo.cfg 包含下列内容:

```
  authProvider.1=org.apache.zookeeper.server.auth.SASLAuthenticationProvider
  kerberos.removeHostFromPrincipal=true
  kerberos.removeRealmFromPrincipal=true
```

在每个主机创建 JAAS 配置并包含:

```
Server {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  keyTab="$PATH_TO_ZOOKEEPER_KEYTAB"
  storeKey=true
```

```
                useTicketCache=false
                principal="zookeeper/$HOST";
              };
```

$HOST 每个Quorum 的主机名. 我们会在下面的$ZK_SERVER_CONF引用本文件的全路径名称.

在每个 Zookeeper Quorum主机启动 Zookeeper:

```
                SERVER_JVMFLAGS="-Djava.security.auth.login.config=$ZK_SERVER_CONF" bin/zkServer start
```

启动HBase 集群。在适当的节点运行下面的一到多个命令:

```
              bin/hbase master start
              bin/hbase regionserver start
```

16.2.4. Zookeeper 服务端认证日志输出

如果上面配置成功, 你应该可以看到如下所示的Zookeeper 服务器日志:

```
11/12/05 22:43:39 INFO zookeeper.Login: successfully logged in.
11/12/05 22:43:39 INFO server.NIOServerCnxnFactory: binding to port 0.0.0.0/0.0.0.0:2181
11/12/05 22:43:39 INFO zookeeper.Login: TGT refresh thread started.
11/12/05 22:43:39 INFO zookeeper.Login: TGT valid starting at:        Mon Dec 05 22:43:39 UTC 2011
11/12/05 22:43:39 INFO zookeeper.Login: TGT expires:                  Tue Dec 06 22:43:39 UTC 2011
11/12/05 22:43:39 INFO zookeeper.Login: TGT refresh sleeping until: Tue Dec 06 18:36:42 UTC 2011
..
11/12/05 22:43:59 INFO auth.SaslServerCallbackHandler:
  Successfully authenticated client: authenticationID=hbase/ip-10-166-175-249.us-west-1.compute.internal@HADOOP.LOCALDOMAIN;
  authorizationID=hbase/ip-10-166-175-249.us-west-1.compute.internal@HADOOP.LOCALDOMAIN.
11/12/05 22:43:59 INFO auth.SaslServerCallbackHandler: Setting authorizedID: hbase
11/12/05 22:43:59 INFO server.ZooKeeperServer: adding SASL authorization for authorizationID: hbase
```

16.2.5. Zookeeper 客户端认证日志输出

Zookeeper 客户端侧 (HBase master 或 regionserver), 应该可以看到如下所示的东西:

```
11/12/05 22:43:59 INFO zookeeper.ZooKeeper: Initiating client connection, connectString=ip-10-166-175-249.us-west-1.compute.inte
11/12/05 22:43:59 INFO zookeeper.ClientCnxn: Opening socket connection to server /10.166.175.249:2181
11/12/05 22:43:59 INFO zookeeper.RecoverableZooKeeper: The identifier of this process is 14851@ip-10-166-175-249
11/12/05 22:43:59 INFO zookeeper.Login: successfully logged in.
11/12/05 22:43:59 INFO client.ZooKeeperSaslClient: Client will use GSSAPI as SASL mechanism.
11/12/05 22:43:59 INFO zookeeper.Login: TGT refresh thread started.
11/12/05 22:43:59 INFO zookeeper.ClientCnxn: Socket connection established to ip-10-166-175-249.us-west-1.compute.internal/10.16
11/12/05 22:43:59 INFO zookeeper.Login: TGT valid starting at:        Mon Dec 05 22:43:59 UTC 2011
11/12/05 22:43:59 INFO zookeeper.Login: TGT expires:                  Tue Dec 06 22:43:59 UTC 2011
11/12/05 22:43:59 INFO zookeeper.Login: TGT refresh sleeping until: Tue Dec 06 18:30:37 UTC 2011
11/12/05 22:43:59 INFO zookeeper.ClientCnxn: Session establishment complete on server ip-10-166-175-249.us-west-1.compute.intern
```

16.2.6. 从头开始配置

在当前标准的 Amazon Linux AMI上测试通过. 先按上面的描述配置 KDC 和 主要节点. 然后获取代码并执行检测.

```
            git clone git://git.apache.org/hbase.git
            cd hbase
            mvn -Psecurity,localTests clean test -Dtest=TestZooKeeperACL
```

再按上面描述配置 HBase. 手工编辑target/cached_classpath.txt (如下)..

```
            bin/hbase zookeeper &
            bin/hbase master &
            bin/hbase regionserver &
```

16.2.7. 未来改进

16.2.7.1. 完善 target/cached_classpath.txt

必须在 target/cached_classpath.txt 重写标准 hadoop-core jar 文件为包含 HADOOP-7070 修改的版本。可以使用下列脚本完成:

```
              echo `find ~/.m2 -name "*hadoop-core*7070*SNAPSHOT.jar"` ':' `cat target/cached_classpath.txt` | sed 's/ //g'
              mv target/tmp.txt target/cached_classpath.txt
```

16.2.7.2. 用程序配置 JAAS

可以避免分离的 Hadoop 修复 HADOOP-7070 的jar文件.

16.2.7.3. 排除 kerberos.removeHostFromPrincipal 和 kerberos.removeRealmFromPrincipal

[33] 要查看全部 ZooKeeper 配置项列表，参考 ZooKeeper的 zoo.cfg. HBase 没有附带 zoo.cfg ，所以你需要查看合适的ZooKeeper下载的conf目录。

Chapter 17. 社区

Table of Contents

17.1. 决策

17.1.1. 特征分支

特征分支很容易制作。你可以不是代码提交者。只需提供需要添加到JIRA上的分支名称到开发者邮件列表，代码提交者会为你添加。然后你可以在你的特征分支下提交发行文件到Apache HBase (TM) JIRA。你的代码保留在别处——一定要公开以便观察到——你可以在开发邮件列表更新进度。当特征已经准备好提交时，3+1 位代码提交者将合并你的特征。[34]

17.1.2. 补丁 +1 政策

The below policy is something we put in place 09/2012. It is a suggested policy rather than a hard requirement. We want to try it first to see if it works before we cast it in stone.

Apache HBase is made of components. Components have one or more Section 17.2.1, “Component Owner”s. See the 'Description' field on the components JIRA page for who the current owners are by component.

Patches that fit within the scope of a single Apache HBase component require, at least, a +1 by one of the component's owners before commit. If owners are absent -- busy or otherwise -- two +1s by non-owners will suffice.

Patches that span components need at least two +1s before they can be committed, preferably +1s by owners of components touched by the x-component patch (TODO: This needs tightening up but I think fine for first pass).

Any -1 on a patch by anyone vetos a patch; it cannot be committed until the justification for the -1 is addressed.

17.2. 社区角色

17.2.1. 组件所有者

组件所有者列在Apache HBase JIRA components页的描述域内。所有者列在描述域而不是“组件领导者”域，因为后者只允许列一个人，而我们鼓励组件由多人所有。

Owners are volunteers who are (usually, but not necessarily) expert in their component domain and may have an agenda on how they think their Apache HBase component should evolve.

Duties include:

1. Owners will try and review patches that land within their component's scope.
2. If applicable, if an owner has an agenda, they will publish their goals or the design toward which they are driving their component

If you would like to be volunteer as a component owner, just write the dev list and we'll sign you up. Owners do not need to be committers.

[34] See HBase, mail # dev - Thoughts about large feature dev branches

附录 A. FAQ

A.1. 通用

什么情况下应该用HBase?

参考 the Section 9.1, “概述” in the Architecture chapter.

还有别的 HBase FAQ吗?

参考 the FAQ that is up on the wiki, HBase Wiki FAQ.

HBase 支持SQL吗?

事实上不支持。 SQL-ish support for HBase via Hive is in development, however Hive is based on MapReduce which is not generally suitable for low-latency requests. 参考 the Chapter 5, *Data Model* section for examples on the HBase client.

到哪里找到NoSQL/HBase的例子呢?

参考附录中 BigTable 论文链接 Appendix F, *Other Information About HBase* ， 及其他论文.

HBase历史如何?

参考 Appendix G, *HBase History*.

A.2. 结构

HBase 如何处理 Region-RegionServer 分配和本地化?

参考 Section 9.7, “Regions”.

A.3. 配置

How can I get started with my first cluster?

参考 Section 1.2, “Quick Start”.

Where can I learn about the rest of the configuration options?

参考 Chapter 2, *Configuration*.

A.4. 模式设计 / 数据访问

How should I design my schema in HBase?

参考 Chapter 5, *Data Model* and Chapter 6, *HBase and Schema Design*

How can I store (fill in the blank) in HBase?

参考 Section 6.5, “ Supported Datatypes ”.

How can I handle secondary indexes in HBase?

参考 Section 6.9, “ Secondary Indexes and Alternate Query Paths ”

Can I change a table's rowkeys?

This is a very common quesiton. You can't. 参考 Section 6.3.5, “Immutability of Rowkeys”.

What APIs does HBase support?

参考 Chapter 5, *Data Model*, Section 9.3, “Client” and Section 10.1, “非Java 语言和 JVM 通话”.

A.5. MapReduce

How can I use MapReduce with HBase?

参考 Chapter 7, *HBase and MapReduce*

A.6. 性能和问题定位

How can I improve HBase cluster performance?

参考 Chapter 11, *Performance Tuning*.

How can I troubleshoot my HBase cluster?

参考 Chapter 12, *Troubleshooting and Debugging HBase*.

A.7. Amazon EC2

I am running HBase on Amazon EC2 and...

EC2 issues are a special case. 参考 Troubleshooting Section 12.12, “Amazon EC2” and Performance Section 11.11, “Amazon EC2” sections.

A.8. 操作

How do I manage my HBase cluster?

参考 Chapter 14, *HBase Operational Management*

How do I back up my HBase cluster?

参考 Section 14.7, “HBase Backup”

A.9. HBase 实践

到哪里找到感兴趣的 HBase 相关视频和幻灯?

参考 Appendix F, *Other Information About HBase*

附录B. 深入hbck

Table of Contents

HBaseFsck (hbck) 是一个检查区域一致性和表完整性问题的工具，可以修复损坏的HBase. 它工作在两个基本模式-- 只读不一致性定位模式和多相读写修复模式。

B.1. 运行 hbck 定位不一致性

要检查HBase集群是否损坏，在HBase集群中运行 hbck：

```
$ ./bin/hbase hbck
```

命令输出结束的地方，会打印OK 或告诉你多少损坏(INCONSISTENCIES)出现。你也许想运行hbck几次，因为一些损坏可能是暂时的。(如集群正启动或区域正分裂)。 从运维来说，你可能希望有规律运行hbck 并在反复报告不一致性时设置报警(如通过 nagios) 。A run of hbck will report a list of inconsistencies along with a brief description of the regions and tables affected. The using the `-details` option will report more details including a representative listing of all the splits present in all the tables.

```
$ ./bin/hbase hbck -details
```

B.2. 不一致

If after several runs, inconsistencies continue to be reported, you may have encountered a corruption. These should be rare, but in the event they occur newer versions of HBase include the hbck tool enabled with automatic repair options.

There are two invariants that when violated create inconsistencies in HBase:

- HBase’s region consistency invariant is satisfied if every region is assigned and deployed on exactly one region server, and all places where this state kept is in accordance.
- HBase’s table integrity invariant is satisfied if for each table, every possible row key resolves to exactly one region.

Repairs generally work in three phases -- a read-only information gathering phase that identifies inconsistencies, a table integrity repair phase that restores the table integrity invariant, and then finally a region consistency repair phase that restores the region consistency invariant. Starting from version 0.90.0, hbck could detect region consistency problems report on a subset of possible table integrity problems. It also included the ability to automatically fix the most common inconsistency, region assignment and deployment consistency problems. This repair could be done by using the `-fix` command line option. These problems close regions if they are open on the wrong server or on multiple region servers and also assigns regions to region servers if they are not open.

Starting from HBase versions 0.90.7, 0.92.2 and 0.94.0, several new command line options are introduced to aid repairing a corrupted HBase. This hbck sometimes goes by the nickname “uberhbck”. Each particular version of uber hbck is compatible with the HBase’s of the same major version (0.90.7 uberhbck can repair a 0.90.4). However, versions <=0.90.6 and versions <=0.92.1 may require restarting the master or failing over to a backup master.

B.3. 本地修复

When repairing a corrupted HBase, it is best to repair the lowest risk inconsistencies first. These are generally region consistency repairs -- localized single region repairs, that only modify in-memory data, ephemeral zookeeper data, or patch holes in the META table. Region consistency requires that the HBase instance has the state of the region's data in HDFS (.regioninfo files), the region's row in the .META. table., and region's deployment/assignments on region servers and the master in accordance. Options for repairing region consistency include:

- `-fixAssignments` (equivalent to the 0.90 `-fix` option) repairs unassigned, incorrectly assigned or multiply assigned regions.
- `-fixMeta` which removes meta rows when corresponding regions are not present in HDFS and adds new meta rows if they regions are present in HDFS while not in META.

To fix deployment and assignment problems you can run this command:

```
$ ./bin/hbase hbck -fixAssignments
```

To fix deployment and assignment problems as well as repairing incorrect meta rows you can run this command:.

```
$ ./bin/hbase hbck -fixAssignments -fixMeta
```

There are a few classes of table integrity problems that are low risk repairs. The first two are degenerate (startkey == endkey) regions and backwards regions (startkey > endkey). These are automatically handled by sidelining the data to a temporary directory (/hbck/xxxx). The third low-risk class is hdfs region holes. This can be repaired by using the:

- `-fixHdfsHoles` option for fabricating new empty regions on the file system. If holes are detected you can use -fixHdfsHoles and should include -fixMeta and -fixAssignments to make the new region consistent.

```
$ ./bin/hbase hbck -fixAssignments -fixMeta -fixHdfsHoles
```

Since this is a common operation, we've added a the `-repairHoles` flag that is equivalent to the previous command:

```
$ ./bin/hbase hbck -repairHoles
```

If inconsistencies still remain after these steps, you most likely have table integrity problems related to orphaned or overlapping regions.

B.4. 区域叠加修复

Table integrity problems can require repairs that deal with overlaps. This is a riskier operation because it requires modifications to the file system, requires some decision making, and may require some manual steps. For these repairs it is best to analyze the output of a `hbck -details` run so that you isolate repairs attempts only upon problems the checks identify. Because this is riskier, there are safeguard that should be used to limit the scope of the repairs. WARNING: This is a relatively new and have only been tested on online but idle HBase instances (no reads/writes). Use at your own risk in an active production environment! The options for repairing table integrity violations include:

- `-fixHdfsOrphans` option for "adopting" a region directory that is missing a region metadata file (the .regioninfo file).
- `-fixHdfsOverlaps` ability for fixing overlapping regions

When repairing overlapping regions, a region's data can be modified on the file system in two ways: 1) by merging regions into a larger region or 2) by sidelining regions by moving data to "sideline" directory where data could be restored later. Merging a large number of regions is technically correct but could result in an extremely large region that requires series of costly compactions and splitting operations. In these cases, it is probably better to sideline the regions that overlap with the most other regions (likely the largest ranges) so that merges can happen on a more reasonable scale. Since these sidelined regions are already laid out in HBase's native directory and HFile format, they can be restored by using HBase's bulk load mechanism. The default safeguard thresholds are conservative. These options let you override the default thresholds and to enable the large region sidelining feature.

- `-maxMerge <n>` maximum number of overlapping regions to merge
- `-sidelineBigOverlaps` if more than maxMerge regions are overlapping, sideline attempt to sideline the regions overlapping with the most other regions.
- `-maxOverlapsToSideline <n>` if sidelining large overlapping regions, sideline at most n regions.

Since often times you would just want to get the tables repaired, you can use this option to turn on all repair options:

- `-repair` includes all the region consistency options and only the hole repairing table integrity options.

Finally, there are safeguards to limit repairs to only specific tables. For example the following command would only attempt to repair table TableFoo and TableBar.

```
$ ./bin/hbase/ hbck -repair TableFoo TableBar
```

B.4.1. 特例: Meta 没有正确分配

There are a few special cases that hbck can handle as well. Sometimes the meta table's only region is inconsistently assigned or deployed. In this case there is a special `-fixMetaOnly` option that can try to fix meta assignments.

```
$ ./bin/hbase hbck -fixMetaOnly -fixAssignments
```

B.4.2. 特例: HBase 版本文件丢失

HBase's data on the file system requires a version file in order to start. If this flie is missing, you can use the `-fixVersionFile` option to fabricating a new HBase version file. This assumes that the version of hbck you are running is the appropriate version for the HBase cluster.

B.4.3. 特例: Root 和 META 损毁.

最严重损坏场景是 ROOT 或 META 损坏，HBase 启动失败。这种情况可以使用OfflineMetaRepair 工具创建新的ROOT和META 区域和表。 This tool assumes that HBase is offline. It then marches through the existing HBase home directory, loads as much information from region metadata files (.regioninfo files) as possible from the file system. If the region metadata has proper table integrity, it sidelines the original root and meta table directories, and builds new ones with pointers to the region directories and their data.

```
$ ./bin/hbase org.apache.hadoop.hbase.util.OfflineMetaRepair
```

NOTE: This tool is not as clever as uberhbck but can be used to bootstrap repairs that uberhbck can complete. If the tool succeeds you should be able to start hbase and run online repairs if necessary.

Appendix C. HBase中的压缩

Table of Contents

C.1. 测试压缩工具

HBase有一个用来测试压缩新的工具。要想运行它，输入`/bin/hbase org.apache.hadoop.hbase.util.CompressionTest`.就会有提示这个工具的具体用法

C.2. `hbase.regionserver.codecs`

如果你的安装错误，就会测试不成功，或者无法启动。可以在你的`hbase-site.xml`加上配置 `hbase.regionserver.codecs` 值你需要的codecs。例如，如果 `hbase.regionserver.codecs` 的值是 `lzo,gz` 同时lzo不存在或者没有正确安装， RegionServer在启动的时候会提示配置错误。

当一台新机器加入到集群中的时候，管理员一定要注意，这台新机器有可能要安装特定的压缩解码器。

C.3. LZO

很不幸，HBase是Apache的协议，而LZO是GPL的协议。HBase不能自带LZO，因此LZO需要在安装HBase之前安装。参见 使用 LZO 压缩介绍了如何在HBase中使用LZO

一个常见的问题是，用户在一开始使用LZO的时候会很好，但是数月过去，管理员在给集群添加集群的时候，他们忘记了LZO的事情。在0.90.0版本之后，我们会运行失败，但也有可能不。

参考 Section C.2, “ hbase.regionserver.codecs ” for a feature to help protect against failed LZO install.

C.4. GZIP

相对于LZO，GZIP的压缩率更高但是速度更慢。在某些特定情况下，压缩率是优先考量的。Java会使用Java自带的GZIP，除非Hadoop的本地库在CLASSPATH中。在这种情况下，最好使用本地压缩器。(如果本地库不存在，可以在Log看到很多*Got brand-new compressor*。参见Q:)

C.5. SNAPPY

If snappy is installed, HBase can make use of it (courtesy of hadoop-snappy [29]).

1. Build and install snappy on all nodes of your cluster.
2. Use CompressionTest to verify snappy support is enabled and the libs can be loaded ON ALL NODES of your cluster:

```
$ hbase org.apache.hadoop.hbase.util.CompressionTest hdfs://host/path/to/hbase snappy
```

3. Create a column family with snappy compression and verify it in the hbase shell:

```
$ hbase> create 't1', { NAME => 'cf1', COMPRESSION => 'SNAPPY' }
hbase> describe 't1'
```

In the output of the "describe" command, you need to ensure it lists "COMPRESSION => 'SNAPPY'"

[29] 参考 Alejandro's note up on the list on difference between Snappy in Hadoop and Snappy in HBase

C.6. Changing Compression Schemes

A frequent question on the dist-list is how to change compression schemes for ColumnFamilies. This is actually quite simple, and can be done via an alter command. Because the compression scheme is encoded at the block-level in StoreFiles, the table does *not* need to be re-created and the data does *not* copied somewhere else. Just make sure the old codec is still available until you are sure that all of the old StoreFiles have been compacted.

Appendix D. YCSB: 雅虎云服务 测试 和HBase

TODO: YCSB不能很多的增加集群负载.

TODO: 如果给HBase安装

Ted Dunning重做了YCSV,这个是用maven管理了，加入了核实工作量的功能。参见 Ted Dunning's YCSB.

Appendix E. HFile format version 2

Table of Contents

E.1. Motivation

Note: this feature was introduced in HBase 0.92

We found it necessary to revise the HFile format after encountering high memory usage and slow startup times caused by large Bloom filters and block indexes in the region server. Bloom filters can get as large as 100 MB per HFile, which adds up to 2 GB when aggregated over 20 regions. Block indexes can grow as large as 6 GB in aggregate size over the same set of regions. A region is not considered opened until all of its block index data is loaded. Large Bloom filters produce a different performance problem: the first get request that requires a Bloom filter lookup will incur the latency of loading the entire Bloom filter bit array.

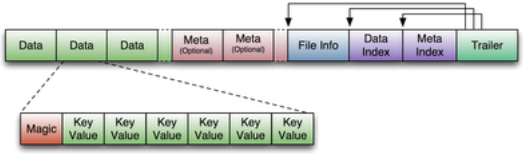
To speed up region server startup we break Bloom filters and block indexes into multiple blocks and write those blocks out as they fill up, which also reduces the HFile writer’s memory footprint. In the Bloom filter case, “filling up a block” means accumulating enough keys to efficiently utilize a fixed-size bit array, and in the block index case we accumulate an “index block” of the desired size. Bloom filter blocks and index blocks (we call these “inline blocks”) become interspersed with data blocks, and as a side effect we can no longer rely on the difference between block offsets to determine data block length, as it was done in version 1.

HFile is a low-level file format by design, and it should not deal with application-specific details such as Bloom filters, which are handled at StoreFile level. Therefore, we call Bloom filter blocks in an HFile "inline" blocks. We also supply HFile with an interface to write those inline blocks.

Another format modification aimed at reducing the region server startup time is to use a contiguous “load-on-open” section that has to be loaded in memory at the time an HFile is being opened. Currently, as an HFile opens, there are separate seek operations to read the trailer, data/meta indexes, and file info. To read the Bloom filter, there are two more seek operations for its “data” and “meta” portions. In version 2, we seek once to read the trailer and seek again to read everything else we need to open the file from a contiguous block.

E.2. HFile format version 1 overview

As we will be discussing the changes we are making to the HFile format, it is useful to give a short overview of the previous (HFile version 1) format. An HFile in the existing format is structured as follows: [30]



E.2.1. Block index format in version 1

The block index in version 1 is very straightforward. For each entry, it contains:

1. Offset (long)
2. Uncompressed size (int)
3. Key (a serialized byte array written using Bytes.writeByteArray)
    a. Key length as a variable-length integer (VInt)
    b. Key bytes

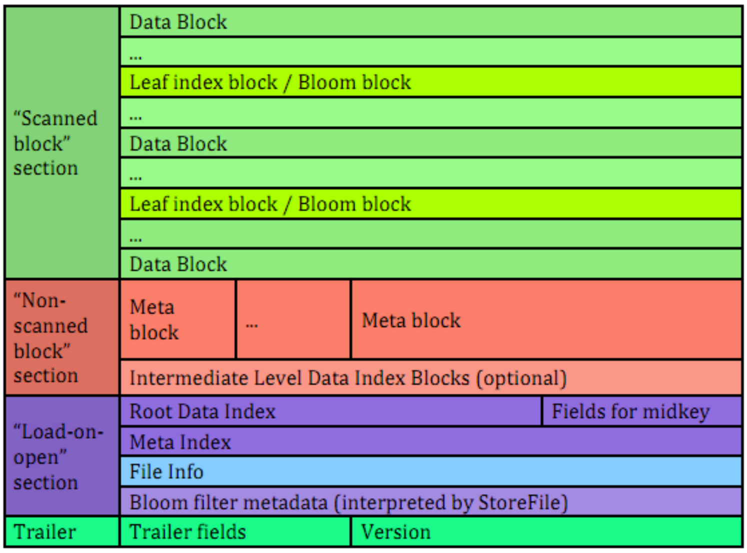
The number of entries in the block index is stored in the fixed file trailer, and has to be passed in to the method that reads the block index. One of the limitations of the block index in version 1 is that it does not provide the compressed size of a block, which turns out to be necessary for decompression. Therefore, the HFile reader has to infer this compressed size from the offset difference between blocks. We fix this limitation in version 2, where we store on-disk block size instead of uncompressed size, and get uncompressed size from the block header.

[30] Image courtesy of Lars George, hbase-architecture-101-storage.html.

E.3. HBase 带内嵌数据块的文件格式 (version 2)

E.3.1. 概述

The version of HBase introducing the above features reads both version 1 and 2 HFiles, but only writes version 2 HFiles. A version 2 HFile is structured as follows:



"Scanned block" section

Data Block

...

Leaf index block / Bloom block

...

Data Block

...

Leaf index block / Bloom block

...

Data Block

"Non-scanned block" section

Meta block

...

Meta block

Intermediate Level Data Index Blocks (optional)

"Load-on-open" section

Root Data Index

Fields for midkey

Meta Index

File Info

Bloom filter metadata (interpreted by StoreFile)

Trailer

Trailer fields

Version

E.3.2. 统一版本 2 的块格式

In the version 2 every block in the data section contains the following fields:

1. 8 bytes: Block type, a sequence of bytes equivalent to version 1's "magic records". Supported block types are:

   a. DATA – data blocks

   b. LEAF_INDEX – leaf-level index blocks in a multi-level-block-index

   c. BLOOM_CHUNK – Bloom filter chunks

   d. META – meta blocks (not used for Bloom filters in version 2 anymore)

   e. INTERMEDIATE_INDEX – intermediate-level index blocks in a multi-level blockindex

   f. ROOT_INDEX – root>level index blocks in a multi>level block index

   g. FILE_INFO – the "file info" block, a small key>value map of metadata

   h. BLOOM_META – a Bloom filter metadata block in the load>on>open section

   i. TRAILER – a fixed>size file trailer. As opposed to the above, this is not an HFile v2 block but a fixed>size (for each HFile version) data structure

   j. INDEX_V1 – this block type is only used for legacy HFile v1 block

2. Compressed size of the block's data, not including the header (int).

   Can be used for skipping the current data block when scanning HFile data.

3. Uncompressed size of the block's data, not including the header (int)

   This is equal to the compressed size if the compression algorithm is NON

4. File offset of the previous block of the same type (long)

   Can be used for seeking to the previous data/index block

5. Compressed data (or uncompressed data if the compression algorithm is NONE).

The above format of blocks is used in the following HFile sections:

1. Scanned block section. The section is named so because it contains all data blocks that need to be read when an HFile is scanned sequentially. Also contains leaf block index and Bloom chunk blocks.

2. Non-scanned block section. This section still contains unified-format v2 blocks but it does not have to be read when doing a sequential scan. This section contains "meta" blocks and intermediate-level index blocks.

We are supporting "meta" blocks in version 2 the same way they were supported in version 1, even though we do not store Bloom filter data in these blocks anymore.

E.3.3. Block index in version 2

There are three types of block indexes in HFile version 2, stored in two different formats (root and non-root):

1. Data index — version 2 multi-level block index, consisting of:

   a. Version 2 root index, stored in the data block index section of the file

   b. Optionally, version 2 intermediate levels, stored in the non%root format in the data index section of the file. Intermediate levels can only be present if leaf level blocks are present

    c. Optionally, version 2 leaf levels, stored in the non%root format inline with data blocks

2. Meta index — version 2 root index format only, stored in the meta index section of the file
3. Bloom index — version 2 root index format only, stored in the "load-on-open" section as part of Bloom filter metadata.

E.3.4. Root block index format in version 2

This format applies to:

1. Root level of the version 2 data index
2. Entire meta and Bloom indexes in version 2, which are always single-level.

A version 2 root index block is a sequence of entries of the following format, similar to entries of a version 1 block index, but storing on-disk size instead of uncompressed size.

1. Offset (long)

   This offset may point to a data block or to a deeper>level index block.

2. On-disk size (int)
3. Key (a serialized byte array stored using Bytes.writeByteArray)
    a. Key (VInt)
    b. Key bytes

A single-level version 2 block index consists of just a single root index block. To read a root index block of version 2, one needs to know the number of entries. For the data index and the meta index the number of entries is stored in the trailer, and for the Bloom index it is stored in the compound Bloom filter metadata.

For a multi-level block index we also store the following fields in the root index block in the load-on-open section of the HFile, in addition to the data structure described above:

1. Middle leaf index block offset
2. Middle leaf block on-disk size (meaning the leaf index block containing the reference to the "middle" data block of the file)
3. The index of the mid-key (defined below) in the middle leaf-level block.

These additional fields are used to efficiently retrieve the mid-key of the HFile used in HFile splits, which we define as the first key of the block with a zero-based index of (n – 1) / 2, if the total number of blocks in the HFile is n. This definition is consistent with how the mid-key was determined in HFile version 1, and is reasonable in general, because blocks are likely to be the same size on average, but we don't have any estimates on individual key/value pair sizes.

When writing a version 2 HFile, the total number of data blocks pointed to by every leaf-level index block is kept track of. When we finish writing and the total number of leaf-level blocks is determined, it is clear which leaf-level block contains the mid-key, and the fields listed above are computed. When reading the HFile and the mid-key is requested, we retrieve the middle leaf index block (potentially from the block cache) and get the mid-key value from the appropriate position inside that leaf block.

E.3.5. Non-root block index format in version 2

This format applies to intermediate-level and leaf index blocks of a version 2 multi-level data block index. Every non-root index block is structured as follows.

1. numEntries: the number of entries (int).
2. entryOffsets: the "secondary index" of offsets of entries in the block, to facilitate a quick binary search on the key (numEntries + 1 int values). The last value is the total length of all entries in this index block. For example, in a non-root index block with entry sizes 60, 80, 50 the "secondary index" will contain the following int array: {0, 60, 140, 190}.
3. Entries. Each entry contains:
    a. Offset of the block referenced by this entry in the file (long)
    b. On>disk size of the referenced block (int)
    c. Key. The length can be calculated from entryOffsets.

E.3.6. Bloom filters in version 2

In contrast with version 1, in a version 2 HFile Bloom filter metadata is stored in the load-on-open section of the HFile for quick startup.

1. A compound Bloom filter.
    a. Bloom filter version = 3 (int). There used to be a DynamicByteBloomFilter class that had the Bloom filter version number 2
    b. The total byte size of all compound Bloom filter chunks (long)
    c. Number of hash functions (int
    d. Type of hash functions (int)
    e. The total key count inserted into the Bloom filter (long)

f. The maximum total number of keys in the Bloom filter (long)

g. The number of chunks (int)

h. Comparator class used for Bloom filter keys, a UTF>8 encoded string stored using Bytes.writeByteArray

i. Bloom block index in the version 2 root block index format

E.3.7. File Info format in versions 1 and 2

The file info block is a serialized HBaseMapWritable (essentially a map from byte arrays to byte arrays) with the following keys, among others. StoreFile-level logic adds more keys to this.

hfile.LASTKEY	The last key of the file (byte array)
hfile.AVG_KEY_LEN	The average key length in the file (int)
hfile.AVG_VALUE_LEN	The average value length in the file (int)

File info format did not change in version 2. However, we moved the file info to the final section of the file, which can be loaded as one block at the time the HFile is being opened. Also, we do not store comparator in the version 2 file info anymore. Instead, we store it in the fixed file trailer. This is because we need to know the comparator at the time of parsing the load-on-open section of the HFile.

E.3.8. Fixed file trailer format differences between versions 1 and 2

The following table shows common and different fields between fixed file trailers in versions 1 and 2. Note that the size of the trailer is different depending on the version, so it is “fixed” only within one version. However, the version is always stored as the last four-byte integer in the file.

<table>
<tr><td>Version 1</td><td>Version 2</td></tr>
<tr><td colspan="2">File info offset (long)</td></tr>
<tr><td>Data index offset (long)</td><td>loadOnOpenOffset (long)

The offset of the section that we need toload when opening the file.</td></tr>
<tr><td colspan="2">Number of data index entries (int)</td></tr>
<tr><td>metaIndexOffset (long)

This field is not being used by the version 1 reader, so we removed it from version 2.</td><td>uncompressedDataIndexSize (long)

The total uncompressed size of the whole data block index, including root-level, intermediate-level, and leaf-level blocks.</td></tr>
<tr><td colspan="2">Number of meta index entries (int)</td></tr>
<tr><td colspan="2">Total uncompressed bytes (long)</td></tr>
<tr><td>numEntries (int)</td><td>numEntries (long)</td></tr>
<tr><td colspan="2">Compression codec: 0 = LZO, 1 = GZ, 2 = NONE (int)</td></tr>
<tr><td></td><td>The number of levels in the data block index (int)</td></tr>
<tr><td></td><td>firstDataBlockOffset (long)

The offset of the first first data block. Used when scanning.</td></tr>
<tr><td></td><td>lastDataBlockEnd (long)

The offset of the first byte after the last key/value data block. We don't need to go beyond this offset when scanning.</td></tr>
<tr><td>Version: 1 (int)</td><td>Version: 2 (int)</td></tr>
</table>

Appendix F. Other Information About HBase

Table of Contents

F.1. HBase Videos

Introduction to HBase

- Introduction to HBase by Todd Lipcon (Chicago Data Summit 2011).
- Introduction to HBase by Todd Lipcon (2010).

Building Real Time Services at Facebook with HBase by Jonathan Gray (Hadoop World 2011).

HBase and Hadoop, Mixing Real-Time and Batch Processing at StumbleUpon by JD Cryans (Hadoop World 2010).

F.2. HBase Presentations (Slides)

Advanced HBase Schema Design by Lars George (Hadoop World 2011).

Introduction to HBase by Todd Lipcon (Chicago Data Summit 2011).

Getting The Most From Your HBase Install by Ryan Rawson, Jonathan Gray (Hadoop World 2009).

F.3. HBase Papers

BigTable by Google (2006).

HBase and HDFS Locality by Lars George (2010).

No Relation: The Mixed Blessings of Non-Relational Databases by Ian Varley (2009).

F.4. HBase Sites

Cloudera's HBase Blog has a lot of links to useful HBase information.

- CAP Confusion is a relevant entry for background information on distributed storage systems.

HBase Wiki has a page with a number of presentations.

F.5. HBase Books

HBase: The Definitive Guide by Lars George.

F.6. Hadoop Books

Hadoop: The Definitive Guide by Tom White.

Appendix H. HBase and the Apache Software Foundation

Table of Contents

HBase is a project in the Apache Software Foundation and as such there are responsibilities to the ASF to ensure a healthy project.

H.1. ASF Development Process

参考 the Apache Development Process page for all sorts of information on how the ASF is structured (e.g., PMC, committers, contributors), to tips on contributing and getting involved, and how open-source works at ASF.

H.2. ASF Board Reporting

Once a quarter, each project in the ASF portfolio submits a report to the ASF board. This is done by the HBase project lead and the committers. 参考 ASF board reporting 获取更多信息。

Appendix I. Enabling Dapper-like Tracing in HBase

Table of Contents

HBASE-6449 added support for tracing requests through HBase, using the open source tracing library, HTrace. Setting up tracing is quite simple, however it currently requires some very minor changes to your client code (it would not be very difficult to remove this requirement).

I.1. SpanReceivers

The tracing system works by collecting information in structs called ‘Spans’. It is up to you to choose how you want to receive this information by implementing the SpanReceiver interface, which defines one method:

```
public void receiveSpan(Span span);
```

This method serves as a callback whenever a span is completed. HTrace allows you to use as many SpanReceivers as you want so you can easily send trace information to multiple destinations.

Configure what SpanReceivers you’d like to use by putting a comma separated list of the fully-qualified class name of classes implementing SpanReceiver in hbase-site.xml property: hbase.trace.spanreceiver.classes.

HBase includes a HBaseLocalFileSpanReceiver that writes all span information to local files in a JSON-based format. TheHBaseLocalFileSpanReceiver looks in hbase-site.xml for a hbase.trace.spanreceiver.localfilespanreceiver.filename property with a value describing the name of the file to which nodes should write their span information.

If you do not want to use the included HBaseLocalFileSpanReceiver, you are encouraged to write your own receiver (take a look at HBaseLocalFileSpanReceiver for an example). If you think others would benefit from your receiver, file a JIRA or send a pull request to HTrace.

I.2. Client Modifications

Currently, you must turn on tracing in your client code. To do this, you simply turn on tracing for requests you think are interesting, and turn it off when the request is done.

For example, if you wanted to trace all of your get operations, you change this:

```
HTable table = new HTable(...);  Get get = new Get(...);
```

into:

```
Span getSpan = Trace.startSpan(“doing get”, Sampler.ALWAYS);  try {    HTable table = new HTable(...);    Get get = new Get(...);
```

If you wanted to trace half of your ‘get’ operations, you would pass in:

```
new ProbabilitySampler(0.5)
```

in lieu of Sampler.ALWAYS to Trace.startSpan(). See the HTrace README for more information on Samplers.

Appendix J. 0.95 RPC Specification

Table of Contents

In 0.95, all client/server communication is done with protobuf’ed Messages rather than with Hadoop Writables. Our RPC wire format therefore changes. This document describes the client/server request/response protocol and our new RPC wire-format.

For what RPC is like in 0.94 and previous, see Benoît/Tsuna’s Unofficial Hadoop / HBase RPC protocol documentation. For more background on how we arrived at this spec., see HBase RPC: WIP

J.1. Goals

1. A wire-format we can evolve
2. A format that does not require our rewriting server core or radically changing its current architecture (for later).

J.2. TODO

1. List of problems with currently specified format and where we would like to go in a version2, etc. For example, what would we have to change if anything to move server async or to support streaming/chunking?
2. Diagram on how it works
3. A grammar that succinctly describes the wire-format. Currently we have these words and the content of the rpc protobuf idl but a grammar for the back and forth would help with groking rpc. Also, a little state machine on client/server interactions would help with understanding (and ensuring correct implementation).

J.3. RPC

The client will send setup information on connection establish. Thereafter, the client invokes methods against the remote server sending a protobuf Message and receiving a protobuf Message in response. Communication is synchronous. All back and forth is preceded by an int that has the total length of the request/response. Optionally, Cells(KeyValues) can be passed outside of protobufs in follow-behind Cell blocks (because we can’t protobuf megabytes of KeyValues or Cells). These CellBlocks are encoded and optionally compressed.

For more detail on the protobufs involved, see the RPC.proto file in trunk.

J.3.1. Connection Setup

Client initiates connection.

J.3.1.1. Client

On connection setup, client sends a preamble followed by a connection header.

J.3.1.1.1. <preamble>

```
<MAGIC 4 byte integer> <1 byte RPC Format Version> <1 byte auth type>[36]
```

E.g.: HBas0x000x80 -- 4 bytes of MAGIC -- ‘HBas’ -- plus one-byte of version, 0 in this case, and one byte, 0x80 (SIMPLE). of an auth type.

J.3.1.1.2. <Protobuf ConnectionHeader Message>

Has user info, and “protocol”, as well as the encoders and compression the client will use sending CellBlocks. CellBlock encoders and compressors are for the life of the connection. CellBlock encoders implement org.apache.hadoop.hbase.codec.Codec. CellBlocks may then also be compressed. Compressors implement org.apache.hadoop.io.compress.CompressionCodec. This protobuf is written using writeDelimited so is prefaced by a pb varint with its serialized length

J.3.1.2. Server

After client sends preamble and connection header, server does NOT respond if successful connection setup. No response means server is READY to accept requests and to give out response. If the version or authentication in the preamble is not agreeable or the server has trouble parsing the preamble, it will throw a org.apache.hadoop.hbase.ipc.FatalConnectionException explaining the error and will then disconnect. If the client in the connection header -- i.e. the protobuf'd Message that comes after the connection preamble -- asks for for a Service the server does not support or a codec the server does not have, again we throw a FatalConnectionException with explanation.

J.3.2. Request

After a Connection has been set up, client makes requests. Server responds.

A request is made up of a protobuf RequestHeader followed by a protobuf Message parameter. The header includes the method name and optionally, metadata on the optional CellBlock that may be following. The parameter type suits the method being invoked: i.e. if we are doing a getRegionInfo request, the protobuf Message param will be an instance of GetRegionInfoRequest. The response will be a GetRegionInfoResponse. The CellBlock is optionally used ferrying the bulk of the RPC data: i.e Cells/KeyValues.

J.3.2.1. Request Parts

J.3.2.1.1. <Total Length>

The request is prefaced by an int that holds the total length of what follows.

J.3.2.1.2. <Protobuf RequestHeader Message>

Will have call.id, trace.id, and method name, etc. including optional Metadata on the Cell block IFF one is following. Data is protobuf'd inline in this pb Message or optionally comes in the following CellBlock

J.3.2.1.3. <Protobuf Param Message>

If the method being invoked is getRegionInfo, if you study the Service descriptor for the client to regionserver protocol, you will find that the request sends a GetRegionInfoRequest protobuf Message param in this position.

J.3.2.1.4. <CellBlock>

An encoded and optionally compressed Cell block.

J.3.3. Response

Same as Request, it is a protobuf ResponseHeader followed by a protobuf Message response where the Message response type suits the method invoked. Bulk of the data may come in a following CellBlock.

J.3.3.1. Response Parts

J.3.3.1.1. <Total Length>

The response is prefaced by an int that holds the total length of what follows.

J.3.3.1.2. <Protobuf ResponseHeader Message>

Will have call.id, etc. Will include exception if failed processing. Optionally includes metadata on optional, IFF there is a CellBlock following.

J.3.3.1.3. <Protobuf Response Message>

Return or may be nothing if exception. If the method being invoked is getRegionInfo, if you study the Service descriptor for the client to regionserver protocol, you will find that the response sends a GetRegionInfoResponse protobuf Message param in this position.

J.3.3.1.4. <CellBlock>

An encoded and optionally compressed Cell block.

J.3.4. Exceptions

There are two distinct types. There is the request failed which is encapsulated inside the response header for the response. The connection stays open to receive new requests. The second type, the FatalConnectionException, kills the connection.

Exceptions can carry extra information. See the ExceptionResponse protobuf type. It has a flag to indicate do-no-retry as well as other miscellaneous payload to help improve client responsiveness.

J.3.5. CellBlocks

These are not versioned. Server can do the codec or it cannot. If new version of a codec with say, tighter encoding, then give it a new class name. Codecs will live on the server for all time so old clients can connect.

J.4. Notes

J.4.1. Constraints

In some part, current wire-format -- i.e. all requests and responses preceeded by a length -- has been dictated by current server non-async architecture.

J.4.2. One fat pb request or header+param

We went with pb header followed by pb param making a request and a pb header followed by pb response for now. Doing header+param rather than a single protobuf Message with both header and param content:

1. Is closer to what we currently have
2. Having a single fat pb requires extra copying putting the already pb'd param into the body of the fat request pb (and same making result)
3. We can decide whether to accept the request or not before we read the param; for example, the request might be low priority. As is, we read header+param in one go as server is currently implemented so this is a TODO.

The advantages are minor. If later, fat request has clear advantage, can roll out a v2 later.

J.4.3. Compression

Uses hadoops compression codecs.

[36] We need the auth method spec. here so the connection header is encoded if auth enabled.

Index