HBase 官方文档中文版

Revision History	
Revision 0.95-SNAPSHOT	2012-05-23T20:32
中文版翻译整理 周海汉	

译者：周海汉 基于 颜开翻译 整理更新。HBase新版 0.95 文档和0.90版相比，变化较大，补充更新了很多内容，章节调整较大。感谢盛大公司颜开的辛勤劳动！英文原文地址在此处。汉化最后更新请到此处浏览。还有很多没有翻译全的，以及链接错误，请愿意翻译的到此处报名并下载修改上传。贡献者将在此文档署名。谢谢！最终版将生成pdf供下载。

贡献者：

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

摘要

这是 Apache HBase的官方文档, Hbase是一个分布式,版本化(versioned)，构建在 Apache Hadoop和 Apache ZooKeeper上的列数据库.

目录

表列表

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。所以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 column family 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 column family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。To configure a deploy, edit a file of environment variables in conf/hbase-env.sh -- this configuration is used mostly by the launcher shell scripts getting the cluster off the ground -- and then add configuration to an XML file to do things like override HBase defaults, tell HBase what Filesystem to use, and the location of the ZooKeeper ensemble [1] .

When running in distributed mode, after you make an edit to an HBase configuration, make sure you copy the content of the conf directory to all nodes of the cluster. HBase will not do this for you. Use rsync.

[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.

2.1. 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。

2.2. 操作系统

2.2.1. 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").

2.2.2. 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.

2.2.3. 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
```

2.2.4. 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!

2.2.5. 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. 你还需要修改 hbase 用户的 nproc，如果过低会造成OutOfMemoryError异常。[2] [3].

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，所以你最好检查一下。[4]

2.2.5.1. 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!

2.2.6. 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 还虚拟一个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

2.3. hadoop

Please read all of this section

Please read this section to the end. Up front we wade through the weeds of Hadoop versions. Later we talk of what you must do in HBase to make it work w/ a particular Hadoop version.

除非运行在实现了持久化同步(sync)的HDFS上，HBase 将丢失所有数据。 Hadoop 0.20.2, Hadoop 0.20.203.0,及 Hadoop 0.20.204.0 不具有上述特性。当前Hadoop仅在Hadoop 0.20.205.x 或更高版本--包含hadoop 1.0.0 --具有持久化sync. Sync必须显式开启。即 dfs.support.append 同时在客户端和服务器端设为真，客户端： hbase-site.xml ，服务器端： 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>
```

修改后必须重启集群。 Ignore the chicken-little comment you'll find in the hdfs-default.xml in the description for thedfs.support.append configuration; it says it is not enabled because there are "... bugs in the 'append code' and is not supported in any production cluster.". This comment is stale, from another era, and while I'm sure there are bugs, the sync/append code has been running in production at large scale deploys and is on by default in the offerings of hadoop by commercial vendors

你还可以用 Cloudera's CDH3 或 MapR 。 Cloudera 的CDH3 是Apache hadoop 0.20.x的补丁增强，包含所有 branch-0.20-append 附加的持久化Sync. Use the released, most recent version of CDH3.

MapR includes a commercial, reimplementation of HDFS. It has a durable sync as well as some other interesting features that are not yet in Apache Hadoop. Their M3 product is free to use and unlimited.

因为Hbase建立在Hadoop之上，所以他用到了hadoop.jar,这个Jar在 lib 里面。这个jar是hbase自己打了branch-0.20-append补丁的hadoop.jar. Hadoop使用的hadoop.jar和Hbase使用的 必须 一致。所以你需要将 Hbase lib 目录下的hadoop.jar替换成Hadoop里面的那个，防止版本冲突。比方说CDH的版本没有HDFS-724而branch-0.20-append里面有，这个HDFS-724补丁修改了RPC协议。如果不替换，就会有版本冲突，继而造成严重的出错，Hadoop会看起来挂了。

我可以用Hbase里面的支持sync的hadoop.jar替代Hadoop里面的那个吗?

你可以这么干。详细可以参见这个邮件列表.

2.3.1. 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.

2.3.2. 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]

2.4. 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.

2.4.1. 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 ，所以的服务和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。

2.4.2. 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”。

2.4.2.1. 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 2.4.2.2.3, “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 目录，如果你自己建这个目录，会有一个warning，Hbase会试图在里面进行migration操作，但是缺少必须的文件。

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 2.4.3, “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]

2.4.2.1.1. Pseudo-distributed Configuration Files

The following are exmaple configuration files from a pseudo-distributed setup.

hdfs-site.xml

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

hbase-site.xml

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

2.4.2.1.2. Pseudo-distributed Extras

2.4.2.1.2.1. Startup

To start up the initial HBase cluster...

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

To start up an extra backup master(s) on the same server run...

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

... the '1' means use ports 60001 & 60011, and this backup master's logfile will be at logs/hbase-${USER}-1-master-${HOSTNAME}.log.

To startup multiple backup masters run...

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

You can start up to 9 backup masters (10 total).

To start up more regionservers...

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

where '1' means use ports 60201 & 60301 and its logfile will be at logs/hbase-${USER}-1-regionserver-${HOSTNAME}.log.

To add 4 more regionservers in addition to the one you just started by running...

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

This supports up to 99 extra regionservers (100 total).

2.4.2.1.2.2. Stop

Assuming you want to stop master backup # 1, run...

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

Note that bin/local-master-backup.sh stop 1 will try to stop the cluster along with the master.

To stop an individual regionserver, run...

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

2.4.2.2. 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>
```

2.4.2.2.1. 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.

2.4.2.2.2. ZooKeeper 和 HBase

2.4.2.2.3. 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.

2.5. 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 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。 [8]

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>
```

2.51. 使用现有的ZooKeeper例子

让Hbase使用一个现有的不被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.

2.5. ZooKeeper

A distributed HBase depends on a running ZooKeeper cluster. All participating nodes and clients need to be able to access the running ZooKeeper ensemble. HBase by default manages a ZooKeeper "cluster" for you. It will start and stop the ZooKeeper ensemble as part of the HBase start/stop process. You can also manage the ZooKeeper ensemble independent of HBase and just point HBase at the cluster it should use. To toggle HBase management of ZooKeeper, use the HBASE_MANAGES_ZK variable in conf/hbase-env.sh. This variable, which defaults to true, tells HBase whether to start/stop the ZooKeeper ensemble servers as part of HBase start/stop.

When HBase manages the ZooKeeper ensemble, you can specify ZooKeeper configuration using its native zoo.cfg file, or, the easier option is to just specify ZooKeeper options directly in conf/hbase-site.xml. A ZooKeeper configuration option can be set as a property in the HBase hbase-site.xml XML configuration file by prefacing the ZooKeeper option name with hbase.zookeeper.property. For example, the clientPort setting in ZooKeeper can be changed by setting the hbase.zookeeper.property.clientPort property. For all default values used by HBase, including ZooKeeper configuration, see Section 2.6.1.1, “HBase Default Configuration”. Look for the hbase.zookeeper.property prefix [13]

You must at least list the ensemble servers in hbase-site.xml using the hbase.zookeeper.quorum property. This property defaults to a single ensemble member at localhost which is not suitable for a fully distributed HBase. (It binds to the local machine only and remote clients will not be able to connect).

How many ZooKeepers should I run?

> You can run a ZooKeeper ensemble that comprises 1 node only but in production it is recommended that you run a ZooKeeper ensemble of 3, 5 or 7 machines; the more members an ensemble has, the more tolerant the ensemble is of host failures. Also, run an odd number of machines. In ZooKeeper, an even number of peers is supported, but it is normally not used because an even sized ensemble requires, proportionally, more peers to form a quorum than an odd sized ensemble requires. For example, an ensemble with 4 peers requires 3 to form a quorum, while an ensemble with 5 also requires 3 to form a quorum. Thus, an ensemble of 5 allows 2 peers to fail, and thus is more fault tolerant than the ensemble of 4, which allows only 1 down peer.
>
> Give each ZooKeeper server around 1GB of RAM, and if possible, its own dedicated disk (A dedicated disk is the best thing you can do to ensure a performant ZooKeeper ensemble). For very heavily loaded clusters, run ZooKeeper servers on separate machines from RegionServers (DataNodes and TaskTrackers).

For example, to have HBase manage a ZooKeeper quorum on nodes rs{1,2,3,4,5}.example.com, bound to port 2222 (the default is 2181) ensure HBASE_MANAGE_ZK is commented out or set to true in conf/hbase-env.sh and then edit conf/hbase-site.xml and set hbase.zookeeper.property.clientPort and hbase.zookeeper.quorum. You should also set hbase.zookeeper.property.dataDir to other than the default as the default has ZooKeeper persist data under /tmp which is often cleared on system restart. In the example below we have ZooKeeper persist to /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>
```

2.5.1. Using existing ZooKeeper ensemble

To point HBase at an existing ZooKeeper cluster, one that is not managed by HBase, set HBASE_MANAGES_ZK in conf/hbase-env.sh to false

```
...
# Tell HBase whether it should manage it's own instance of Zookeeper or not.
export HBASE_MANAGES_ZK=false
```

Next set ensemble locations and client port, if non-standard, in hbase-site.xml, or add a suitably configured zoo.cfg to HBase's CLASSPATH. HBase will prefer the configuration found in zoo.cfg over any settings in hbase-site.xml.

When HBase manages ZooKeeper, it will start/stop the ZooKeeper servers as a part of the regular start/stop scripts. If you would like to run ZooKeeper yourself, independent of HBase start/stop, you would do the following

```
${HBASE_HOME}/bin/hbase-daemons.sh {start,stop} zookeeper
```

Note that you can use HBase in this manner to spin up a ZooKeeper cluster, unrelated to HBase. Just make sure to set HBASE_MANAGES_ZK to false if you want it to stay up across HBase restarts so that when HBase shuts down, it doesn't take ZooKeeper down with it.

For more information about running a distinct ZooKeeper cluster, see the ZooKeeper Getting Started Guide. Additionally, see the ZooKeeper Wiki or the ZooKeeper documentation for more information on ZooKeeper sizing.

2.5.2. SASL Authentication with ZooKeeper

Newer releases of HBase (>= 0.92) will support connecting to a ZooKeeper Quorum that supports SASL authentication (which is available in Zookeeper versions 3.4.0 or later).

This describes how to set up HBase to mutually authenticate with a ZooKeeper Quorum. ZooKeeper/HBase mutual authentication (HBASE-2418) is required as part of a complete secure HBase configuration (HBASE-3025). For simplicity of explication, this section ignores additional configuration required (Secure HDFS and Coprocessor configuration). It's recommended to begin with an HBase-managed Zookeeper configuration (as opposed to a standalone Zookeeper quorum) for ease of learning.

2.5.2.1. Operating System Prerequisites

You need to have a working Kerberos KDC setup. For each $HOST that will run a ZooKeeper server, you should have a principle zookeeper/$HOST. For each such host, add a service key (using the kadmin or kadmin.local tool's ktadd command) for zookeeper/$HOST and copy this file to $HOST, and make it readable only to the user that will run zookeeper on $HOST. Note the location of this file, which we will use below as $PATH_TO_ZOOKEEPER_KEYTAB.

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.

2.5.2.2. HBase-managed Zookeeper Configuration

On each node that will run a zookeeper, a master, or a regionserver, create a JAAS configuration file in the conf directory of the node's HBASE_HOME directory that looks like the following:

```
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";
};
```

where the $PATH_TO_HBASE_KEYTAB and $PATH_TO_ZOOKEEPER_KEYTAB files are what you created above, and $HOST is the

hostname for that node.

The `Server` section will be used by the Zookeeper quorum server, while the `Client` section will be used by the HBase master and regionservers. 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
```

2.5.2.3. External Zookeeper Configuration

Add a JAAS configuration file that looks like:

```
Client {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  useTicketCache=false
  keyTab="$PATH_TO_HBASE_KEYTAB"
  principal="hbase/$HOST";
};
```

where the `$PATH_TO_HBASE_KEYTAB` is the keytab created above for HBase services to run on this host, and `$HOST` is the hostname for that node. Put this in the HBase home's configuration directory. We'll refer to this file's full pathname as `$HBASE_SERVER_CONF` below.

Modify your hbase-env.sh to include the following:

```
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"
```

Modify your `hbase-site.xml` on each node that will run a 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>
</configuration>
```

where `$ZK_NODES` is the comma-separated list of hostnames of the Zookeeper Quorum hosts.

Add a `zoo.cfg` for each Zookeeper Quorum host containing:

```
authProvider.1=org.apache.zookeeper.server.auth.SASLAuthenticationProvider
kerberos.removeHostFromPrincipal=true
```

```
kerberos.removeRealmFromPrincipal=true
```

Also on each of these hosts, create a JAAS configuration file containing:

```
Server {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  keyTab="$PATH_TO_ZOOKEEPER_KEYTAB"
  storeKey=true
  useTicketCache=false
  principal="zookeeper/$HOST";
};
```

where `$HOST` is the hostname of each Quorum host. We will refer to the full pathname of this file as `$ZK_SERVER_CONF` below.

Start your Zookeepers on each Zookeeper Quorum host with:

```
SERVER_JVMFLAGS="-Djava.security.auth.login.config=$ZK_SERVER_CONF" bin/zkServer start
```

Start your HBase cluster by running one or more of the following set of commands on the appropriate nodes:

```
bin/hbase master start
bin/hbase regionserver start
```

2.5.2.4. Zookeeper Server Authentication Log Output

If the configuration above is successful, you should see something similar to the following in your Zookeeper server logs:

```
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
```

2.5.2.5. Zookeeper Client Authentication Log Output

On the Zookeeper client side (HBase master or regionserver), you should see something similar to the following:

```
11/12/05 22:43:59 INFO zookeeper.ZooKeeper: Initiating client connection, connectString=ip-10-166-175-249.us-west-1.compute.internal:2181
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/10.166.175.249:
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.internal/10.166.
```

2.5.2.6. Configuration from Scratch

This has been tested on the current standard Amazon Linux AMI. First setup KDC and principals as described above. Next checkout code and run a sanity check.

```
git clone git://git.apache.org/hbase.git
cd hbase
mvn -Psecurity,localTests clean test -Dtest=TestZooKeeperACL
```

Then configure HBase as described above. Manually edit target/cached_classpath.txt (see below)..

```
bin/hbase zookeeper &
bin/hbase master &
bin/hbase regionserver &
```

2.5.2.7. Future improvements

2.5.2.7.1. Fix target/cached_classpath.txt

You must override the standard hadoop-core jar file from the `target/cached_classpath.txt` file with the version containing the HADOOP-7070 fix. You can use the following script to do this:

```
echo `find ~/.m2 -name "*hadoop-core*7070*SNAPSHOT.jar"` ':' `cat target/cached_classpath.txt` | sed 's/ //g' > target/t
mv target/tmp.txt target/cached_classpath.txt
```

2.5.2.7.2. Set JAAS configuration programmatically

This would avoid the need for a separate Hadoop jar that fixes HADOOP-7070.

2.5.2.7.3. Elimination of kerberos.removeHostFromPrincipal and kerberos.removeRealmFromPrincipal

[13] For the full list of ZooKeeper configurations, see ZooKeeper's zoo.cfg. HBase does not ship with a zoo.cfg so you will need to browse the conf directory in an appropriate ZooKeeper download.

2.6. 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.

2.6.1. 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 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。

2.6.1.1. 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(因翻译需要，被译者修改成中文注释).

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: /tmp/hbase-${user.name}

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。例如 region查询，Get操作，Update操作等等都可能发生错误，需要重试。这是最大重试错误的值。

默认： 10

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： 1

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.thread.splitcompactcheckfrequency

region server 多久执行一次split/compaction 检查.

默认： 20000

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regionserver有average + (average * slop)个region是会执行Rebalance

默认： 0

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.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：67108864

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：false

hbase.hregion.max.filesize

最大HStoreFile大小。若某个Column families的HStoreFile增长达到这个值，这个Hegion会被切割成两个。 Default: 256M.

默认：268435456

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.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。默认是20%，设置为0就是不分配。

默认： 0.2

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

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：

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

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.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: 2000

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

2.6.2. 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。

2.6.3. 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

2.6.4. 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>
```

2.6.4.1. 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.

[10] 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.

[11] 参见 Section B.2, “ hbase.regionserver.codecs ” 可以看到关于LZO安装的具体信息，帮助你放在安装失败。

[12] 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.

2.7. 配置示例

2.7.1. 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.

2.7.1.1. 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>
```

2.7.1.2. 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
```

2.7.1.3. 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.

[1] See CHANGES.txt in branch-0.20-append to see list of patches involved adding append on the Hadoop 0.20 branch.

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

[3] 这样的需求对于数据库应用来说是很常见的，例如Oracle。 Setting Shell Limits for the Oracle User in Short Guide to install Oracle 10 on Linux.

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

[5] 参见 Hadoop HDFS: Deceived by Xciever for an informative rant on xceivering.

[6] 这两个命名法来自于Hadoop.

[7] See Pseudo-distributed mode extras for notes on how to start extra Masters and RegionServers when running pseudo-distributed.

[8] For the full list of ZooKeeper configurations, see ZooKeeper's zoo.cfg. HBase does not ship with a zoo.cfg so you will need to browse the conf directory in an appropriate ZooKeeper download.

2.8. 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)。

2.8.1. Required Configurations

Review the Section 2.2, “Operating System” and Section 2.3, “Hadoop” sections.

2.8.2. Recommended Configurations

2.8.2.1. 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。

2.8.2.2. Number of ZooKeeper Instances

See Section 2.5, “ZooKeeper”.

2.8.2.3. 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。

You can get a sense of whether you have too little or too many handlers by Section 12.2.2.1, “Enabling RPC-level logging” on an individual RegionServer then tailing its logs (Queued requests consume memory).

2.8.2.4. 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.

2.8.2.5. 压缩

You should consider enabling ColumnFamily compression. There are several options that are near-frictionless and in most all cases boost performance by reducing the size of StoreFiles and thus reducing I/O.

See Appendix C, Compression In HBase for more information.

2.8.2.6. 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).

You may need to experiment with this setting based on your hardware configuration and application needs.

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.

2.8.2.7. 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ILong.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-split 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。

2.8.2.8. 管理 Compactions

A common administrative technique is to manage major compactions manually, rather than letting HBase do it. By default, HConstants.MAJOR_COMPACTION_PERIOD is one day and major compactions may kick in when you least desire it - especially on a busy system. To turn off automatic major compactions set the value to0.

It is important to stress that major compactions are absolutely necessary for StoreFile cleanup, the only variant is when they occur. They can be administered through the HBase shell, or via HBaseAdmin.

For more information about compactions and the compaction file selection process, see Section 9.7.5.5, “Compaction”

2.8.2.9. Speculative Execution

Speculative Execution of MapReduce tasks is on by default, and for HBase clusters it is generally advised to turn off Speculative Execution at a system-level unless you need it for a specific case, where it can be configured per-job. Set the
properties mapred.map.tasks.speculative.execution andmapred.reduce.tasks.speculative.execution to false.

2.8.3. Other Configurations

2.8.3.1. Balancer

The balancer is periodic operation run on the master to redistribute regions on the cluster. It is configured via hbase.balancer.period and defaults to 300000 (5 minutes).

See Section 9.5.4.1, “LoadBalancer” for more information on the LoadBalancer.

2.8.3.2. Disabling Blockcache

Do not turn off block cache (You'd do it by setting hbase.block.cache.size to zero). Currently we do not do well if you do this because the regionserver will spend all its time loading hfile indices over and over again. If your working set it such that block cache does you no good, at least size the block cache such that hfile indices will stay up in the cache (you can get a rough idea on the size you need by surveying regionserver UIs; you'll see index block size accounted near the top of the webpage).

---

[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.

2.9. Bloom Filter Configuration

2.9.1. io.hfile.bloom.enabled global kill switch

io.hfile.bloom.enabled in Configuration serves as the kill switch in case something goes wrong. Default = true.

2.9.2. io.hfile.bloom.error.rate

io.hfile.bloom.error.rate = average false positive rate. Default = 1%. Decrease rate by ½ (e.g. to .5%) == +1

bit per bloom entry.

2.9.3. io.hfile.bloom.max.fold

io.hfile.bloom.max.fold = guaranteed minimum fold rate. Most people should leave this alone. Default = 7, or can collapse to at least 1/128th of original size. See the Development Process section of the document BloomFilters in HBase for more on what this option means.

Chapter 3. 升级

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

3.1. 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 2.6.1.1, “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[9] .

---

[9] 参见 HBASE-3499 Users upgrading to 0.90.0 need to have their .META. table updated with the right MEMSTORE SIZE

3.2. Upgrading from 0.90.x to 0.92.x

Upgrade Guide

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).

3.2.1. You can't go back!

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.

3.2.2. MSLAB is ON by default

In 0.92.0, the hbase.hregion.memstore.mslab.enabled flag is set to true (See 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.

3.2.3. Distributed splitting is on by default

Previous, WAL logs on crash were split by the Master alone. In 0.92.0, log splitting is done by the cluster (See See “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.

3.2.4. Memory accounting is different now

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. See

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.

3.2.5. On the Hadoop version to use

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. See Section 2.3, “Hadoop”.

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

3.2.6. HBase 0.92.0 ships with 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).

3.2.7. Online alter is off by default

In 0.92.0, we’ve added an experimental online schema alter facility (See 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).

3.2.8. 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.

3.2.9. Security tarball

We now ship with two tarballs; secure and insecure HBase. Documentation on how to setup a secure HBase is on the way.

3.2.10. Experimental off-heap cache

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/

3.2.11. Changes in HBase replication

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.

3.2.12. RegionServer now aborts if OOME

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 (See [HBASE-4769] - ‘Abort RegionServer Immediately on OOME’)

3.2.13. HFile V2 and the “Bigger, Fewer” Tendency

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 (See “HBASE-1621 merge tool should work on online cluster, but disabled table”).

Chapter 4. The 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)).getTime(
```

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. Debug

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，所以的列是从属于某一个column family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row key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。所以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column family：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

一个列名是有它的column family前缀和qualifier连接而成。例如列contents:html是column family contents加冒号(:)加 qualifier 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column family 存储的。新的columns可以不经过声明直接加入一个column family.

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row key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

row key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. Column Family

在Hbase是column family一些列的集合。一个column family所有列成员是有着相同的前缀。比如，列courses:history 和courses:math都是 column family courses的成员.冒号(:)是column family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。column family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column family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column family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. Data Model Operations

The four primary data model operations are Get, Put, Scan, and Delete. Operations are applied via HTable instances.

5.7.1. Get

Get returns attributes for a specified row. Gets are executed via HTable.get.

5.7.2. Put

Put either adds new rows to a table (if the key is new) or can update existing rows (if the key already exists). Puts are executed via HTable.put (writeBuffer) or HTable.batch (non-writeBuffer).

5.7.3. Scans

Scan allow iteration over multiple rows for specified attributes.

The following is an example of a on an HTable table instance. Assume that a table is populated with rows with keys "row1", "row2", "row3", and then another set of rows with the keys "abc1", "abc2", and "abc3". The following example shows how startRow and stopRow can be applied to a Scan instance to return the rows beginning with "row".

```
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 removes a row from a table. Deletes are executed via HTable.delete.

HBase does not modify data in place, and so deletes are handled by creating new markers called tombstones. These tombstones, along with the dead values, are cleaned up on major compactions.

See Section 5.8.1.5, “Delete” for more information on deleting versions of columns, and see Section 9.7.5.5, “Compaction” for more information on compactions.

5.8. 版本

一个 {row, column, version} 元组是Hbase中的一个cell .但是有可能会有很多的cell的row和column是相同的，可以使用version来区分不同的cell.

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的 cell 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？[13]
- 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?[14]

下面我们介绍下在Hbase中版本是如何工作的。[15].

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

当你进行delete操作的是，有两种方式来确定要删除的版本。

- 删除所有比当前早的版本。
- 删除指定的版本。

一个删除操作可以删除一行，也可以是一个column family，或者仅仅删除一个column。你也可以删除指明的一个版本。若你没有指明，默认情况下是删除比当前时间早的版本。

删除操作的实现是创建一个删除标记。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。[16]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。

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. 删除标记误删Puts

删除标记操作可能会标记之后put的数据。[17].需要值得注意的是，当写下一个删除标记后，只有下一个major compaction操作发起之后，这个删除标记才会消失。设想一下，当你写下一个删除标记-“删除所有<= 时间T的数据”。但之后，你又执行了一个Put操作，版本<= T。这样就算这个Put发生在删除之后，他的数据也算是打上了删除标记。这个Put并不会失败，但是你需要注意的是这个操作没有任何作用。只有一个major compaction执行只有，一切才会恢复正常。如果你的Put操作一直使用升序的版本，这个错误就不会发生。但是也有可能出现这样的情况，你删除之后，

5.8.2.2. Major compactions 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。[18]”

---

[13] 目前，只有最新的那个是可以获取到的。.

[14] 可以

[15] See 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.

[16] 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。

[17] HBASE-2256

[18] See Garbage Collection in Bending time in HBase

5.9. Sort Order

All data model operations HBase return data in sorted order. First by row, then by ColumnFamily, followed by column qualifier, and finally timestamp (sorted in reverse, so newest records are returned first).

5.10. Column Metadata

There is no store of column metadata outside of the internal KeyValue instances for a ColumnFamily. Thus, while HBase can support not only a wide number of columns per row, but a heterogenous set of columns between rows as well, it is your responsibility to keep track of the column names.

The only way to get a complete set of columns that exist for a ColumnFamily is to process all the rows. For more information about how HBase stores data internally, see Section 9.7.5.4, “KeyValue”.

5.11. Joins

Whether HBase supports joins is a common question on the dist-list, and there is a simple answer: it doesn't, at not least in the way that RDBMS' support them (e.g., with equi-joins or outer-joins in SQL). As has been illustrated in this chapter, the read data model operations in HBase are Get and Scan.

However, that doesn't mean that equivalent join functionality can't be supported in your application, but you have to do it yourself. The two primary strategies are either denormalizing the data upon writing to HBase, or to have lookup tables and do the join between HBase tables in your application or MapReduce code (and as RDBMS' demonstrate, there are several strategies for this depending on the size of the tables, e.g., nested loops vs. hash-joins). So which is the best approach? It depends on what you are trying to do, and as such there isn't a single answer that works for every use case.

Chapter 6. HBase 的 Schema 设计

Table of Contents

有一个关于NSQL数据库的优点和确定的介绍， No Relation: The Mixed Blessings of Non-Relational Databases. 推荐看一看.

6.1. Schema 创建

可以使用HBaseAdmin或者Chapter 4, The HBase Shell 来创建和编辑Hbase的schemas

Tables must be disabled when making ColumnFamily modifications, for example..

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

See Section 2.6.4, “Client configuration and dependencies connecting to an HBase cluster” for more information about configuring client connections.

Note: online schema changes are supported in the 0.92.x codebase, but the 0.90.x codebase requires the table to be disabled.

6.1.1. Schema Updates

When changes are made to either Tables or ColumnFamilies (e.g., region size, block size), these changes take effect the next time there is a major compaction and the StoreFiles get re-written.

See Section 9.7.5, “Store” for more information on StoreFiles.

6.2. column families的数量

现在Hbase并不能很好的处理两个或者三个以上的column families，所以尽量让你的column families数量少一些。目前，flush和compaction操作是针对一个Region。所以当一个column family操作大量数据的时候会引发一个flush。那些不相关的column families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column family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column families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column family) 。 For more information on compactions, seeSection 9.7.5.5, “Compaction”.

尽量在你的应用中使用一个Column family。只有你的所有查询操作只访问一个column family的时候，可以引入第二个和第三个column family.例如，你有两个column family,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. Cardinality of ColumnFamilies

Where multiple ColumnFamilies exist in a single table, be aware of the cardinality (i.e., number of rows). If ColumnFamilyA has 1 million rows and ColumnFamilyB has 1 billion rows, ColumnFamilyA's data will likely be spread across many, many regions (and RegionServers). This makes mass scans for ColumnFamilyA less efficient.

6.3. Rowkey设计

6.3.1. 单调递增Row Keys/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，所以的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row和column的大小(为何我的StoreFile 指示很大？)

在Hbase中，值是作为一个cell保存在系统的中的，要定位一个cell,需要row,column name和timestamp.通常情况下，如果你的row和column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(recommended!)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value的随机访问，但是访问一个cell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block size，当然也可以使用更小的column name 。

Compression will also make for larger indices. See the thread a question storefileIndexSize up on the user mailing list.

Most of the time small inefficiencies don't matter all that much. Unfortunately, this is a case where they do. Whatever patterns are selected for ColumnFamilies, attributes, and rowkeys they could be repeated several billion times in your data.

See Section 9.7.5.4, “KeyValue” for more information on HBase stores data internally to see why this is important.

6.3.2.1. Column Families

Try to keep the ColumnFamily names as small as possible, preferably one character (e.g. "d" for data/default).

See Section 9.7.5.4, “KeyValue” for more information on HBase stores data internally to see why this is important.

6.3.2.2. Attributes

Although verbose attribute names (e.g., "myVeryImportantAttribute") are easier to read, prefer shorter attribute names (e.g., "via") to store in HBase.

See Section 9.7.5.4, “KeyValue” for more information on HBase stores data internally to see why this is important.

6.3.2.3. Rowkey Length

Keep them as short as is reasonable such that they can still be useful for required data access (e.g., Get vs. Scan). A short key that is useless for data access is not better than a longer key with better get/scan properties. Expect tradeoffs when designing rowkeys.

6.3.2.4. Byte Patterns

A long is 8 bytes. You can store an unsigned number up to 18,446,744,073,709,551,615 in those eight bytes. If you stored this number as a String -- presuming a byte per character -- you need nearly 3x the bytes.

Not convinced? Below is some sample code that you can run on your own.

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

6.3.3. Reverse Timestamps

A common problem in database processing is quickly finding the most recent version of a value. A technique using reverse timestamps as a part of the key can help greatly with a special case of this problem. Also found in the HBase chapter of Tom White's book Hadoop: The Definitive Guide (O'Reilly), the technique involves appending (Long.MAX_VALUE - timestamp) to the end of any key, e.g., [key][reverse_timestamp].

The most recent value for [key] in a table can be found by performing a Scan for [key] and obtaining the first record. Since HBase keys are in sorted order, this key sorts before any older row-keys for [key] and thus is first.

This technique would be used instead of using Section 6.4, “ Number of Versions ” where the intent is to hold onto all versions "forever" (or a very long time) and at the same time quickly obtain access to any other version by using the same Scan technique.

6.3.4. Rowkeys and ColumnFamilies

Rowkeys are scoped to ColumnFamilies. Thus, the same rowkey could exist in each ColumnFamily that exists in a table without collision.

6.3.5. Immutability of Rowkeys

Rowkeys cannot be changed. The only way they can be "changed" in a table is if the row is deleted and then re-inserted. This is a fairly common question on the HBase dist-list so it pays to get the rowkeys right the first time (and/or before you've inserted a lot of data).

6.4. 版本数量

6.4.1. Maximum Number of Versions

行的版本的数量是HColumnDescriptor设置的，每个column family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timestamp来区分、过早的版本会在执行major compaction的时候删除。这个版本的值可以根据具体的应用增加减少。

It is not recommended setting the number of max versions to an exceedingly high level (e.g., hundreds or more) unless those old values are very dear to you because this will greatly increase StoreFile size.

6.4.2. Minimum Number of Versions

Like maximum number of row versions, the minimum number of row versions to keep is configured per column family via HColumnDescriptor. The default for min versions is 0, which means the feature is disabled. The minimum number of row versions parameter is used together with the time-to-live parameter and can be combined with the number of row versions parameter to allow configurations such as "keep the last T minutes worth of data, at most N versions, but keep at least M versions around" (where M is the value for minimum number of row versions, M<N). This parameter should only be set when time-to-live is enabled for a column family and must be less than the number of row versions.

6.5. Supported Datatypes

HBase supports a "bytes-in/bytes-out" interface via Put and Result, so anything that can be converted to an array of bytes can be stored as a value. Input could be strings, numbers, complex objects, or even images as long as they can rendered as bytes.

There are practical limits to the size of values (e.g., storing 10-50MB objects in HBase would probably be too much to ask); search the mailling list for conversations on this topic. All rows in HBase conform to the Chapter 5, Data Model, and that includes versioning. Take that into consideration when making your design, as well as block size for the ColumnFamily.

6.5.1. Counters

One supported datatype that deserves special mention are "counters" (i.e., the ability to do atomic increments of numbers). See Increment in HTable.

Synchronization on counters are done on the RegionServer, not in the client.

6.6. Joins

If you have multiple tables, don't forget to factor in the potential for Section 5.11, "Joins" into the schema design.

6.7. Time To Live (TTL)

ColumnFamilies can set a TTL length in seconds, and HBase will automatically delete rows once the expiration time is reached. This applies to all versions of a row - even the current one. The TTL time encoded in the HBase for the row is specified in UTC.

See HColumnDescriptor for more information.

6.8. Keeping Deleted Cells

ColumnFamilies can optionally keep deleted cells. That means deleted cells can still be retrieved with Get or Scan operations, as long these operations have a time range specified that ends before the timestamp of any delete that would affect the cells. This allows for point in time queries even in the presence of deletes.

Deleted cells are still subject to TTL and there will never be more than "maximum number of versions" deleted cells. A new "raw" scan options returns all deleted rows and the delete markers.

See HColumnDescriptor for more information.

6.9. Secondary Indexes and Alternate Query Paths

This section could also be titled "what if my table rowkey looks like this but I also want to query my table like that." 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. Filter Query

Depending on the case, it may be appropriate to use Section 9.4, “Client Request Filters”. In this case, no secondary index is created. However, don't try a full-scan on a large table like this from an application (i.e., single-threaded client).

6.9.2. Periodic-Update Secondary Index

A secondary index could be created in an other table which is periodically updated via a MapReduce job. The job could be executed intra-day, but depending on load-strategy it could still potentially be out of sync with the main data table.

See Section 7.2.2, “HBase MapReduce Read/Write Example” for more information.

6.9.3. Dual-Write Secondary Index

Another strategy is to build the secondary index while publishing data to the cluster (e.g., write to data table, write to index table). If this is approach is taken after a data table already exists, then bootstrapping will be needed for the secondary index with a MapReduce job (see Section 6.9.2, “ Periodic-Update Secondary Index ”).

6.9.4. Summary Tables

Where time-ranges are very wide (e.g., year-long report) and where the data is voluminous, summary tables are a common approach. These would be generated with MapReduce jobs into another table.

See Section 7.2.4, “HBase MapReduce Summary to HBase Example” for more information.

6.9.5. Coprocessor Secondary Index

Coprocessors act like RDBMS triggers. These were added in 0.92. For more information, see Section 9.6.3, “Coprocessors”

6.10. Schema Design Smackdown

This section will describe common schema design questions that appear on the dist-list. These are general guidelines and not laws - each application must consider it's own needs.

6.10.1. 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).

6.10.2. 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.

6.11. Operational and Performance Configuration Options

See 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.

6.12. Constraints

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.

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. For more information about MapReduce (i.e., the framework in general), see the Hadoop MapReduce Tutorial.

7.1. Map-Task 分割

7.1.1 默认 HBase MapReduce 分割器(Splitter)

当 MapReduce job的HBase table 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，不论需要scan多少个column families 。

7.1.2. Custom Splitters

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

The following is an example of using HBase as a MapReduce source in read-only manner. Specifically, there is a Mapper instance but no Reducer, and nothing is being emitted from the Mapper. 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 Read/Write Example

The following is an example of using HBase both as a source and as a sink with MapReduce. This example will simply copy data from one table to another.

```
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!");
}
```

An explanation is required of what `TableMapReduceUtil` is doing, especially with the reducer. [TableOutputFormat](#) is being used as the outputFormat class, and several parameters are being set on the config (e.g., TableOutputFormat.OUTPUT_TABLE), as well as setting the reducer output key to `ImmutableBytesWritable` and reducer value to `Writable`. These could be set by the programmer on the job and conf, but `TableMapReduceUtil` tries to make things easier.

The following is the example mapper, which will create a `Put` and matching the input `Result` and emit it. Note: this is what the CopyTable utility does.

```
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;
        }
}
```

There isn't actually a reducer step, so `TableOutputFormat` takes care of sending the `Put` to the target table.

This is just an example, developers could choose not to use `TableOutputFormat` and connect to the target table themselves.

7.2.3. HBase MapReduce Read/Write Example With Multi-Table Output

TODO: example for `MultiTableOutputFormat`.

7.2.4. HBase MapReduce Summary to HBase Example

The following example uses HBase as a MapReduce source and sink with a summarization step. This example will count the number of distinct instances of a value in a table and write those summarized counts in another table.

```
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!");
}
```

In this example mapper a column with a String-value is chosen as the value to summarize upon. This value is used as the key to emit from the mapper, and an `IntWritable` represents an instance counter.

```
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);
        }
}
```

In the reducer, the "ones" are counted (just like any other MR example that does this), and then emits a `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 Summary to File Example

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 Summary to HBase Without Reducer

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 Summary to RDBMS

Sometimes it is more appropriate to generate summaries to an RDBMS. For these cases, it is possible to generate summaries directly to an RDBMS via a custom reducer. The `setup` method can connect to an RDBMS (the connection information can be passed via custom parameters in the context) and the cleanup method can close the connection.

It is critical to understand that number of reducers for the job affects the summarization implementation, and you'll have to design this into your reducer. Specifically, whether it is designed to run as a singleton (one reducer) or multiple reducers. Neither is right or wrong, it depends on your use-case. Recognize that the more reducers that are assigned to the job, the more simultaneous connections to the RDBMS will be created - this will scale, but only to a point.

```
 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
        }
```

```
}
```

In the end, the summary results are written to your RDBMS table/s.

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。

See Section 2.8.2.9, “Speculative Execution” for more information.

Chapter 8. Secure HBase

Table of Contents

8.1. Secure Client Access to HBase

Newer releases of HBase (>= 0.92) support optional SASL authentication of clients.

This describes how to set up HBase and HBase clients for connection to secure HBase resources.

8.1.1. Prerequisites

HBase must have been built using the new maven profile for secure Hadoop/HBase: `-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. Server-side Configuration for Secure Operation

Add the following to the `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. Client-side Configuration for Secure Operation

Add the following to the `hbase-site.xml` file on every client:

```
<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, add the following to the `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. Client-side Configuration for Secure Operation - Thrift Gateway

Add the following to the `hbase-site.xml` file for every Thrift gateway:

```
<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. Client-side Configuration for Secure Operation - REST Gateway

Add the following to the hbase-site.xml file for every REST gateway:

```
<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. Access Control

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. Prerequisites

You must configure HBase for secure operation. Refer to the section "Secure Client Access to 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. Overview

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/PutFlushCompact
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>
```

Chapter 9. 架构

Table of Contents

9.1. Overview

9.1.1. NoSQL?

HBase is a type of "NoSQL" database. "NoSQL" is a general term meaning that the database isn't an RDBMS which supports SQL as it's primary access language, but there are many types of NoSQL databases: BerkeleyDB is an example of a local NoSQL database, whereas HBase is very much a distributed database. Technically speaking, HBase is really more a "Data Store" than "Data Base" because it lacks many of the features you find in an RDBMS, such as typed columns, secondary indexes, triggers, and advanced query languages, etc.

However, HBase has many features which supports both linear and modular scaling. HBase clusters expand by adding RegionServers that are hosted on commodity class servers. If a cluster expands from 10 to 20 RegionServers, for example, it doubles both in terms of storage and as well as processing capacity. RDBMS can scale well, but only up to a point - specifically, the size of a single database server - and for the best performance requires specialized hardware and storage devices. HBase features of note are:

- Strongly consistent reads/writes: HBase is not an "eventually consistent" DataStore. This makes it very suitable for tasks such as high-speed counter aggregation.
- Automatic sharding: HBase tables are distributed on the cluster via regions, and regions are automatically split and re-distributed as your data grows.
- Automatic RegionServer failover
- Hadoop/HDFS Integration: HBase supports HDFS out of the box as it's distributed file system.
- MapReduce: HBase supports massively parallelized processing via MapReduce for using HBase as both source and sink.
- Java Client API: HBase supports an easy to use Java API for programmatic access.
- Thrift/REST API: HBase also supports Thrift and REST for non-Java front-ends.
- Block Cache and Bloom Filters: HBase supports a Block Cache and Bloom Filters for high volume query optimization.
- Operational Management: HBase provides build-in web-pages for operational insight as well as JMX metrics.

9.1.2. When Should I Use HBase?

HBase isn't suitable for every problem.

First, make sure you have enough data. If you have hundreds of millions or billions of rows, then HBase is a good candidate. If you only have a few thousand/million rows, then using a traditional RDBMS might be a better choice due to the fact that all of your data might wind up on a single node (or two) and the rest of the cluster may be sitting idle.

Second, make sure you can live without all the extra features that an RDBMS provides (e.g., typed columns, secondary indexes, transactions, advanced query languages, etc.) An application built against an RDBMS cannot be "ported" to HBase by simply changing a JDBC driver, for example. Consider moving from an RDBMS to HBase as a complete redesign as opposed to a port.

Third, make sure you have enough hardware. Even HDFS doesn't do well with anything less than 5 DataNodes (due to things such as HDFS block replication which has a default of 3), plus a NameNode.

HBase can run quite well stand-alone on a laptop - but this should be considered a development configuration only.

9.1.3. What Is The Difference Between HBase and Hadoop/HDFS?

HDFS is a distributed file system that is well suited for the storage of large files. It's documentation states that it is not, however, a general purpose file system, and does not provide fast individual record lookups in files. HBase, on the other hand, is built on top of HDFS and provides fast record lookups (and updates) for large tables. This can sometimes be a point of conceptual confusion. HBase internally puts your data in indexed "StoreFiles" that exist on HDFS for high-speed lookups. See the Chapter 5, Data Model and the rest of this chapter for more information on how HBase achieves its goals.

9.2. Catalog Tables

The catalog tables -ROOT- and .META. exist as HBase tables. They are are filtered out of the HBase shell's `list` command, but they are in fact tables just like any other.

9.2.1. ROOT

-ROOT- keeps track of where the .META. table is. The -ROOT- table structure is as follows:

Key:

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

Values:

- info:regioninfo (serialized HRegionInfo instance of .META.)
- info:server (server:port of the RegionServer holding .META.)
- info:serverstartcode (start-time of the RegionServer process holding .META.)

9.2.2. META

The .META. table keeps a list of all regions in the system. The .META. table structure is as follows:

Key:

- Region key of the format ([table],[region start key],[region id])

Values:

- info:regioninfo (serialized HRegionInfo instance for this region)
- info:server (server:port of the RegionServer containing this region)
- info:serverstartcode (start-time of the RegionServer process containing this region)

When a table is in the process of splitting two other columns will be created, info:splitA and info:splitB which represent the two daughter regions. The values for these columns are also serialized HRegionInfo instances. After the region has been split eventually this row will be deleted.

Notes on HRegionInfo: the empty key is used to denote table start and table end. A region with an empty start key is the first region in a table. If region has both an empty start and an empty end key, its the only region in the table

In the (hopefully unlikely) event that programmatic processing of catalog metadata is required, see the Writables utility.

9.2.3. Startup Sequencing

The META location is set in ROOT first. Then META is updated with server and startcode values.

For information on region-RegionServer assignment, see Section 9.7.2, “Region-RegionServer Assignment”.

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。

See 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. Connection Pooling

For applications which require high-end multithreaded access (e.g., web-servers or application servers that may serve many application threads in a single JVM), see HTablePool.

9.3.2. 写缓冲和批量操作

若关闭了HTable中的 Section 11.7.4, “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. External Clients

Information on non-Java clients and custom protocols is covered in Chapter 10, External APIs

9.3.4. RowLocks

RowLocks are still in the client API however they are discouraged because if not managed properly these can lock up the RegionServers.

There is an oustanding ticket HBASE-2332 to remove this feature from the client.

9.4. Client Request Filters

Get and Scan instances can be optionally configured with filters which are applied on the RegionServer.

Filters can be confusing because there are many different types, and it is best to approach them by understanding the groups of Filter functionality.

9.4.1. Structural

Structural Filters contain other Filters.

9.4.1.1. FilterList

FilterList represents a list of Filters with a relationship of `FilterList.Operator.MUST_PASS_ALL` or `FilterList.Operator.MUST_PASS_ONE` between the Filters. The following example shows an 'or' between two Filters (checking for either 'my value' or 'my other value' on the same attribute).

```
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. Column Value

9.4.2.1. SingleColumnValueFilter

SingleColumnValueFilter can be used to test column values for equivalence (`CompareOp.EQUAL`), inequality (`CompareOp.NOT_EQUAL`), or ranges (e.g., `CompareOp.GREATER`). The folowing is example of testing equivalence a column to a String value "my value"...

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

9.4.3. Column Value Comparators

There are several Comparator classes in the Filter package that deserve special mention. These Comparators are used in concert with other Filters, such as Section 9.4.2.1, “SingleColumnValueFilter”.

9.4.3.1. RegexStringComparator

RegexStringComparator supports regular expressions for value comparisons.

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

See the Oracle JavaDoc for supported RegEx patterns in Java.

9.4.3.2. SubstringComparator

SubstringComparator can be used to determine if a given substring exists in a value. The comparison is case-insensitive.

```
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

See BinaryPrefixComparator.

9.4.3.4. BinaryComparator

See BinaryComparator.

9.4.4. KeyValue Metadata

As HBase stores data internally as KeyValue pairs, KeyValue Metadata Filters evaluate the existence of keys (i.e., ColumnFamily:Column qualifiers) for a row, as opposed to values the previous section.

9.4.4.1. FamilyFilter

FamilyFilter can be used to filter on the ColumnFamily. It is generally a better idea to select ColumnFamilies in the Scan than to do it with a Filter.

9.4.4.2. QualifierFilter

QualifierFilter can be used to filter based on Column (aka Qualifier) name.

9.4.4.3. ColumnPrefixFilter

ColumnPrefixFilter can be used to filter based on the lead portion of Column (aka Qualifier) names.

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 behaves like ColumnPrefixFilter but allows specifying multiple prefixes.

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

A ColumnRangeFilter allows efficient intra row scanning.

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: Introduced in HBase 0.92

9.4.5. RowKey

9.4.5.1. RowFilter

It is generally a better idea to use the startRow/stopRow methods on Scan for row selection, however RowFilter can also be used.

9.4.6. Utility

9.4.6.1. FirstKeyOnlyFilter

This is primarily used for rowcount jobs. See FirstKeyOnlyFilter.

9.5. Master

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. Processes

The Master runs several background threads:

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. See Section 2.8.3.1, “Balancer” for configuring this property.

See 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. See 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. Interface

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. Processes

The RegionServer runs a variety of background threads:

9.6.2.1. CompactSplitThread

Checks for splits and handle minor compactions.

9.6.2.2. MajorCompactionChecker

Checks for major compactions.

9.6.2.3. MemStoreFlusher

Periodically flushes in-memory writes in the MemStore to StoreFiles.

9.6.2.4. LogRoller

Periodically checks the RegionServer's HLog.

9.6.3. Coprocessors

Coprocessors were added in 0.92. There is a thorough Blog Overview of CoProcessors posted. Documentation will eventually move to this reference guide, but the blog is the most current information available at this time.

9.6.4. Block Cache

9.6.4.1. Design

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. Usage

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). See 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. Write Ahead Log (WAL)

9.6.5.1. Purpose

每个RegionServer会将更新(Puts, Deletes) 先记录到Write Ahead Log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] See 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. Regions

Regions are the basic element of availability and distribution for tables, and are comprised of a Store per Column Family. The heirarchy of objects is as follows:

```
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)
```

For a description of what HBase files look like when written to HDFS, see Section 12.7.2, “Browsing HDFS for HBase Objects”.

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。

See Section 2.8.2.6, “Bigger Regions” for more information on configuration.

9.7.2. Region-RegionServer Assignment

This section describes how Regions are assigned to RegionServers.

9.7.2.1. Startup

When HBase starts regions are assigned as follows (short version):

1. The Master invokes the `AssignmentManager` upon startup.
2. The `AssignmentManager` looks at the existing region assignments in META.
3. If the region assignment is still valid (i.e., if the RegionServer is still online) then the assignment is kept.
4. If the assignment is invalid, then the `LoadBalancerFactory` is invoked to assign the region. The `DefaultLoadBalancer` will randomly assign the region to a RegionServer.
5. META is updated with the RegionServer assignment (if needed) and the RegionServer start codes (start time of the RegionServer process) upon region opening by the RegionServer.

9.7.2.2. Failover

When a RegionServer fails (short version):

1. The regions immediately become unavailable because the RegionServer is down.
2. The Master will detect that the RegionServer has failed.
3. The region assignments will be considered invalid and will be re-assigned just like the startup sequence.

9.7.2.3. Region Load Balancing

Regions can be periodically moved by the Section 9.5.4.1, “LoadBalancer”.

9.7.3. Region-RegionServer Locality

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. Region Splits

RegionServer的Splits操作是不可见的，因为Master不会参与其中。RegionServer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RegionServer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. Custom Split Policies

The default split policy can be overwritten using a custom RegionSplitPolicy (HBase 0.94+). Typically a custom split policy should extend HBase's default split policy: ConstantSizeRegionSplitPolicy.

The policy can set globally through the HBaseConfiguration used or on a per table basis:

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

9.7.5. Store

一个Store包含了一个MemStore和若干个StoreFile(HFile).一个Store可以定位到一个column family中的一个region.

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)

StoreFiles are where your data lives.

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's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/475950861828684
```

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. Compaction

有两种类型的压缩:minor和major。minor压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压缩会删除过期的数据。有些时候minor压缩就会将一个store中的全部文件压缩，实际上这个时候他本身就是一个major压缩。对于一个minor压缩是如何压缩的，可以参见ascii diagram in the Store source code.

在执行一个major压缩之后，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压缩将会将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”。

Compactions will not perform region merges. See Section 14.2.2, “Merge” for more information on region merging.

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.

9.7.6. Bloom Filters

Bloom filters were developed over in HBase-1200 Add bloomfilters.[25][26]

See also Section 11.6.4, “Bloom Filters” and Section 2.9, “Bloom Filter Configuration”.

9.7.6.1. Bloom StoreFile footprint

Bloom filters add an entry to the StoreFile general FileInfo data structure and then two extra entries to the StoreFile metadata section.

9.7.6.1.1. BloomFilter in the StoreFile FileInfo data structure

FileInfo has a BLOOM_FILTER_TYPE entry which is set to NONE, ROW or ROWCOL.

9.7.6.1.2. 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).

---

[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. Bulk Loading

9.8.1. Overview

HBase includes several methods of loading data into tables. The most straightforward method is to either use the TableOutputFormat class from a MapReduce job, or use the normal client APIs; however, these are not always the most efficient methods.

The bulk load feature uses a MapReduce job to output table data in HBase's internal data format, and then directly loads the generated StoreFiles into a running cluster. Using bulk load will use less CPU and network resources than simply using the HBase API.

9.8.2. Bulk Load Architecture

The HBase bulk load process consists of two main steps.

9.8.2.1. Preparing data via a MapReduce job

The first step of a bulk load is to generate HBase data files (StoreFiles) from a MapReduce job using HFileOutputFormat. This output format writes out data in HBase's internal storage format so that they can be later loaded very efficiently into the cluster.

In order to function efficiently, HFileOutputFormat must be configured such that each output HFile fits within a single region. In order to do this, jobs whose output will be bulk loaded into HBase use Hadoop's TotalOrderPartitioner class to partition the map output into disjoint ranges of the key space, corresponding to the key ranges of the regions in the table.

HFileOutputFormat includes a convenience function, configureIncrementalLoad(), which automatically sets up a TotalOrderPartitioner based on the current region boundaries of a table.

9.8.2.2. Completing the data load

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. Importing the prepared data using the completebulkload tool

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. See Also

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

9.8.5. Advanced Usage

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. See the LoadIncrementalHFiles class for more information.

9.9. HDFS

As HBase runs on HDFS (and each StoreFile is written as a file on HDFS), it is important to have an understanding of the HDFS Architecture especially in terms of how it stores files, handles failovers, and

replicates blocks.

See the Hadoop documentation on HDFS Architecture for more information.

9.9.1. NameNode

The NameNode is responsible for maintaining the filesystem metadata. See the above HDFS Architecture link for more information.

9.9.2. DataNode

The DataNodes are responsible for storing HDFS blocks. See the above HDFS Architecture link for more information.

Chapter 10. External APIs

Table of Contents

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

10.1. Non-Java Languages Talking to the JVM

Currently the documentation on this topic in the HBase Wiki. See also the Thrift API Javadoc.

10.2. REST

Currently most of the documentation on REST exists in the HBase Wiki on REST.

10.3. Thrift

Currently most of the documentation on Thrift exists in the HBase Wiki on Thrift.

10.3.1. Filter Language

10.3.1.1. Use Case

Note: this feature was introduced in HBase 0.92

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. General Filter String Syntax

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. Compare Operator

A compare operator can be any of the following:

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

The client should use the symbols (<, <=, =, !=, >, >=) to express compare operators.

10.3.1.6. Comparator

A comparator can be any of the following:

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. Example PHP Client Program that uses the Filter Language

```
<? $_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'))) AND (Tim
   $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. Example Filter Strings

- “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. Individual Filter Syntax

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)"

Chapter 11. 性能调优

Table of Contents

11.1. Operating System

11.1.1. Memory

RAM, RAM, RAM. Don't starve HBase.

11.1.2. 64-bit

Use a 64-bit platform (and 64-bit JVM).

11.1.3. Swapping

Watch out for swapping. Set swappiness to 0.

11.2. Network

Perhaps the most important factor in avoiding network issues degrading Hadoop and HBbase performance is the switching hardware that is used, decisions made early in the scope of the project can cause major problems when you double or triple the size of your cluster (or more).

Important items to consider:

- Switching capacity of the device
- Number of systems connected
- Uplink capacity

11.2.1. Single Switch

The single most important factor in this configuration is that the switching capacity of the hardware is capable of handling the traffic which can be generated by all systems connected to the switch. Some lower priced commodity hardware can have a slower switching capacity than could be utilized by a full switch.

11.2.2. Multiple Switches

Multiple switches are a potential pitfall in the architecture. The most common configuration of lower priced hardware is a simple 1Gbps uplink from one switch to another. This often overlooked pinch point can easily become a bottleneck for cluster communication. Especially with MapReduce jobs that are both reading and writing a lot of data the communication across this uplink could be saturated.

Mitigation of this issue is fairly simple and can be accomplished in multiple ways:

- Use appropriate hardware for the scale of the cluster which you're attempting to build.
- Use larger single switch configurations i.e. single 48 port as opposed to 2x 24 port
- Configure port trunking for uplinks to utilize multiple interfaces to increase cross switch bandwidth.

11.2.3. Multiple Racks

Multiple rack configurations carry the same potential issues as multiple switches, and can suffer performance degradation from two main areas:

- Poor switch capacity performance
- Insufficient uplink to another rack

If the the switches in your rack have appropriate switching capacity to handle all the hosts at full speed, the next most likely issue will be caused by homing more of your cluster across racks. The easiest way to avoid issues when spanning multiple racks is to use port trunking to create a bonded uplink to other racks. The downside of this method however, is in the overhead of ports that could potentially be used. An example of this is, creating an 8Gbps port channel from rack A to rack B, using 8 of your 24 ports to communicate between racks gives you a poor ROI, using too few however can mean you're not getting the most out of your cluster.

Using 10Gbe links between racks will greatly increase performance, and assuming your switches support a 10Gbe uplink or allow for an expansion card will allow you to save your ports for machines as opposed to uplinks.

11.2.4. Network Interfaces

Are all the network interfaces functioning correctly? Are you sure? See the Troubleshooting Case Study in 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RegionServer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RegionServer UI点击Log Level。将org.apache.hadoop.ipc 的日志级别设为DEBUG。然后tail RegionServer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描述列在Hbase中常见的两种stop-the-world的GC操作，尤其是在loading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.

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. 管理压缩

对于大型的系统，你需要考虑管理压缩和分割

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RegionServer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RegionServer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RegionServer的需要来设定。

11.4.7. hbase.hstore.blockingStoreFiles

参见hbase.hstore.blockingStoreFiles. 如果在RegionServer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. Schema Design

11.6.1. Column Families 的数目

参见 Section 6.2, “ On the number of column families ”.

11.6.2. Key and Attribute Lengths

See Section 6.3.2, “Try to minimize row and column sizes”. See also Section 11.6.7.1, “However...” for compression caveats.

11.6.3. Table RegionSize

The regionsize can be set on a per-table basis via setFileSize on HTableDescriptor in the event where certain tables require different regionsizes than the configured default regionsize.

See Section 11.4.1, “Number of Regions” for more information.

11.6.4. Bloom Filters

Bloom Filters can be enabled per-ColumnFamily. Use HColumnDescriptor.setBloomFilterType(NONE | ROW | ROWCOL) to enable blooms per Column Family. Default = NONE for no bloom filters. If ROW, the hash of the row will be added to the bloom on each insert. If ROWCOL, the hash of the row + column family + column family qualifier will be added to the bloom on each key insert.

See HColumnDescriptor and Section 9.7.6, “Bloom Filters” for more information.

11.6.5. ColumnFamily BlockSize

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).

See HColumnDescriptor and Section 9.7.5, “Store” for more information.

11.6.6. In-Memory ColumnFamilies

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.

See HColumnDescriptor for more information.

11.6.7. Compression

Production systems should use compression with their ColumnFamily definitions. See Appendix C, Compression In HBase for more information.

11.6.7.1. However...

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.

See 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. Writing to HBase

11.7.1. 批量 Loading

如果可以的话，尽量使用批量导入工具，参见 Section 9.8, “Bulk Loading”.否则就要详细看看下面的内容。

11.7.2. Table创建: 预创建Regions

默认情况下Hbase创建Table会新建一个region。执行批量导入，意味着所有的client会写入这个region，直到这个region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region。最好稍保守一点，因为过多的region会实实在在的降低性能。下面是一个预创建region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;
}
```

11.7.3. Table创建: Deferred Log Flush

The default behavior for Puts using the Write Ahead Log (WAL) is that HLog edits will be written immediately. If deferred log flush is used, WAL edits are kept in memory until the flush period. The benefit is aggregated and asynchronous HLog- writes, but the potential downside is that if the RegionServer goes down the yet-to-be-flushed edits are lost. This is safer, however, than not using WAL at all with Puts.

Deferred log flush can be configured on tables via HTableDescriptor. The default value of hbase.regionserver.optionallogflushinterval is 1000ms.

11.7.4. HBase 客户端: AutoFlush

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RegionServer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

11.7.5. HBase 客户端: Turn off WAL on Puts

A frequently discussed option for increasing throughput on Puts is to call writeToWAL(false). Turning this off means that the RegionServer will not write the Put to the Write Ahead Log, only into the memstore, HOWEVER the consequence is that if there is a RegionServer failure there will be data loss. If writeToWAL(false) is used, do so with extreme caution. You may find in actuality that it makes little difference if your load is well distributed across the cluster.

In general, it is best to use WAL for Puts, and where loading throughput is a concern to use bulk loading techniques instead.

11.7.6. HBase 客户端: Group Puts by RegionServer

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.

11.7.7. MapReduce: Skip The 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.

11.7.8. 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.

See Section 11.7.2, “ Table Creation: Pre-Creating Regions ”, as well as Section 11.4, “HBase Configurations”

11.8. Reading from HBase

11.8.1. Scan Caching

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。

11.8.1.1. 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.

11.8.2. 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column family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。

11.8.3. MapReduce - Input Splits

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)”.

11.8.4. 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();
```

11.8.5. 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。

11.8.6. Row Keys 的负载优化

当scan一个表的时候， 如果仅仅需要row key（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。

11.8.7. Concurrency: Monitor Data Spread

When performing a high number of concurrent reads, monitor the data spread of the target tables. If the target table(s) have too few regions then the reads could likely be served from too few nodes.

See Section 11.7.2, “ Table Creation: Pre-Creating Regions ”, as well as Section 11.4, “HBase Configurations”

11.9. Deleting from HBase

11.9.1. Using HBase Tables as Queues

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.

See also Section 9.7.5.5, “Compaction” and HBaseAdmin.majorCompact.

11.9.2. Delete RPC Behavior

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).

See http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/HTable.html#delete%28org.apache.hadoop.hbase.client.De

11.10. HDFS

Because HBase runs on Section 9.9, “HDFS” it is important to understand how it works and how it affects HBase.

11.10.1. 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. See the Umbrella Jira Ticket for HDFS Improvements for HBase.

11.10.2. 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.

11.11. 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.

11.12. Case Studies

For Performance and Troubleshooting Case Studies, see Chapter 13, Case Studies.

Chapter 12. Hbase的故障排除和Debug

Table of Contents

12.5. 客户端

For more information on the HBase client, see Section 9.3, “Client”.

12.5.1. ScannerTimeoutException or UnknownScannerException

This is thrown if the time between RPC calls from the client to RegionServer exceeds the scan timeout. For example, if Scan.setCaching is set to 500, then there will be an RPC call to fetch the next batch of rows every 500 .next() calls on the ResultScanner because data is being transferred in blocks of 500 rows to the client. Reducing the setCaching value may be an option, but setting this value too low makes for inefficient processing on numbers of rows.

See Section 11.8.1, “Scan Caching”.

12.5.2. Shell or client application throws lots of scary exceptions during normal operation

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. Long Client Pauses With Compression

This is a fairly frequent question on the HBase dist-list. The scenario is that a client is typically inserting a lot of data into a relatively un-optimized HBase cluster. Compression can exacerbate the pauses, although it is not the source of the problem.

See Section 11.7.2, “ Table Creation: Pre-Creating Regions ” on the pattern for pre-creating regions and confirm that the table isn't starting with a single region.

See Section 11.4, “HBase Configurations” for cluster configuration, particularly hbase.hstore.blockingStoreFiles, hbase.hregion.memstore.block.multiplier, MAX_FILESIZE (region size), and 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 Client Connection Errors

Errors like this...

```
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
```

... are either due to ZooKeeper being down, or unreachable due to network issues.

The utility Section 12.4.1.3, “zkcli” may help investigate ZooKeeper issues.

12.5.5. Client running out of memory though heap size seems to be stable (but the off-heap/direct heap keeps growing)

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. Client Slowdown When Calling Admin Methods (flush, compact, etc.)

This is a client issue fixed by HBASE-5073 in 0.90.6. There was a ZooKeeper leak in the client and the client was getting pummeled by ZooKeeper events with each additional invocation of the admin API.

12.5.7. Secure Client Cannot Connect ([Caused by GSSException: 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 HBase cluster. Examine the ticket currently in the credential cache, if any, by running the <tt>klist</tt> command line utility. If no ticket is listed, you must obtain a ticket by running the <tt>kinit</tt> command with either a keytab specified, or by interactively entering a password for the desired principal.

Then, consult the <a>Java Security Guide troubleshooting section. The most common problem addressed there is resolved by setting <tt>javax.security.auth.useSubjectCredsOnly</tt> system property value to <tt>false</tt>.

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 <tt>kinit</tt> and then immediately refresh the credential cache with <tt>kinit -R</tt>. The refresh will rewrite the credential cache without the problematic formatting.

Finally, depending on your Kerberos configuration, you may need to install the <a>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 <a>unlimited strength JCE policy files. Uncompress and extract the downloaded file, and install the policy jars into <tt><java-home>/lib/security</tt>.

12.1. 一般准则

首先可以看看master的log。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exception。

一个错误通常不是单独出现在Hbase中的，通常是某一个地方发生了异常，然后对其他的地方发生影响。到处都是exception和stack straces。遇到这样的错误，最好的办法是查日志，找到最初的异常。例如Region会在abort的时候打印一下信息。Grep这个Dump就有可能找到最初的异常信息。

RegionServer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RegionServer切腹(译者注:竟然用日文词)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session timeout。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. Log Levels

12.2.2.1. Enabling RPC-level logging

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 Garbage Collection Logs

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/logs/gc-hbase
```

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=0.00 secs]
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: user=0.07 sys
```

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: user=0.05 sys
64901.476: [GC 64901.476: [ParNew: 5563K->575K(5568K), 0.0072510 secs] 2869283K->2864837K(3061272K), 0.0073320 secs] [Times: user=0.05 sys
64901.500: [GC 64901.500: [ParNew: 5517K->573K(5568K), 0.0120390 secs] 2869780K->2865267K(3061272K), 0.0121150 secs] [Times: user=0.09 sys
64901.529: [GC 64901.529: [ParNew: 5507K->569K(5568K), 0.0086240 secs] 2870200K->2865742K(3061272K), 0.0087180 secs] [Times: user=0.05 sys
64901.554: [GC 64901.555: [ParNew: 5516K->575K(5568K), 0.0107130 secs] 2870689K->2866291K(3061272K), 0.0107820 secs] [Times: user=0.06 sys
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: user=0.05 sys
64901.609: [GC 64901.609: [ParNew: 5512K->569K(5568K), 0.0063370 secs] 2871771K->2867322K(3061272K), 0.0064230 secs] [Times: user=0.06 sys
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, 0.0027950 sec
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, 0.0027950 sec
```

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: user=0.05 sys
...  lines removed ...
64904.936: [GC 64904.936: [ParNew: 5532K->568K(5568K), 0.0070720 secs] 1365024K->1360689K(3061272K), 0.0071930 secs] [Times: user=0.05 sys
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. Resources

12.3.1. search-hadoop.com

search-hadoop.com indexes all the mailing lists and is great for historical searches. Search here first when you have an issue as its more than likely someone has already had your problem.

12.3.2. Mailing Lists

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[28]; 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 is also really helpful when looking for Hadoop/HBase-specific issues.

---

[28] See Getting Answers

12.4. 工具

12.4.1. Builtin Tools

12.4.1.1. Master Web Interface

The Master starts a web-interface on port 60010 by default.

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. RegionServer Web Interface

RegionServers starts a web-interface on port 60030 by default.

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.).

See Section 14.4, “HBase Metrics” for more information in metric definitions.

12.4.1.3. zkcli

`zkcli` is a very useful tool for investigating ZooKeeper-related issues. To invoke:

```
./hbase zkcli -server host:port <cmd> <args>
```

The commands (and arguments) are:

```
        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. External Tools

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 -XX:+Aggr
```

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$ConditionObject)
            at java.util.concurrent.locks.LockSupport.parkNanos(LockSupport.java:198)
            at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.awaitNanos(AbstractQueuedSynchronizer.java:1963)
            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..0x00007f16b5b87af0
   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..0x00007f16afef
   java.lang.Thread.State: WAITING (parking)
                at sun.misc.Unsafe.park(Native Method)
                - parking to wait for  <0x00007f16cd3f8dd8> (a java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject)
                at java.util.concurrent.locks.LockSupport.park(LockSupport.java:158)
                at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.await(AbstractQueuedSynchronizer.java:1925)
                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..0x00007f16acb77cf0]
   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 [0x00007f16
   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compaction，严重影响了IO性能。(TODO: 在这里插入compactionQueueSize的图片)(译者注:囧)

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 or 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。

See Section 11.8.1, “Scan Caching”.

12.5.2. Shell or client application throws lots of scary exceptions during normal operation

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. Long Client Pauses With Compression

This is a fairly frequent question on the HBase dist-list. The scenario is that a client is typically inserting a lot of data into a relatively un-optimized HBase cluster. Compression can exacerbate the pauses, although it is not the source of the problem.

See Section 11.7.2, “ Table Creation: Pre-Creating Regions ” on the pattern for pre-creating regions and confirm that the table isn't starting with a single region.

See Section 11.4, “HBase Configurations” for cluster configuration, particularly `hbase.hstore.blockingStoreFiles`, `hbase.hregion.memstore.block.multiplier`, `MAX_FILESIZE` (region size), and `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 Client Connection Errors

Errors like this...

```
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
```

... are either due to ZooKeeper being down, or unreachable due to network issues.

The utility Section 12.4.1.3, “zkcli” may help investigate ZooKeeper issues.

12.5.5. Client running out of memory though heap size seems to be stable (but the off-heap/direct heap keeps growing)

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. Client Slowdown When Calling Admin Methods (flush, compact, etc.)

This is a client issue fixed by HBASE-5073 in 0.90.6. There was a ZooKeeper leak in the client and the client was getting pummeled by ZooKeeper events with each additional invocation of the admin API.

12.5.7. Secure Client Cannot Connect ([Caused by GSSException: 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 HBase cluster. Examine the ticket currently in the credential cache, if any, by running the <tt>klist</tt> command line utility. If no ticket is listed, you must obtain a ticket by running the <tt>kinit</tt> command with either a keytab specified, or by interactively entering a password for the desired principal.

Then, consult the <a>Java Security Guide troubleshooting section. The most common problem addressed there is resolved by setting <tt>javax.security.auth.useSubjectCredsOnly</tt> system property value to <tt>false</tt>.

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 <tt>kinit</tt> and then immediately refresh the credential cache with <tt>kinit -R</tt>. The refresh will rewrite the credential cache without the problematic formatting.

Finally, depending on your Kerberos configuration, you may need to install the <a>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 <a>unlimited strength JCE policy files. Uncompress and extract the downloaded file, and install the policy jars into <tt><java-home>/lib/security</tt>.

12.6. MapReduce

12.6.1. You Think You're On The Cluster, But You're Actually Local

This following stacktrace happened using `ImportTsv`, but things like this can happen on any job with a mis-configuration.

```
    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)
```

.. see the critical portion of the stack? It's...

```
       at org.apache.hadoop.mapred.LocalJobRunner$Job.run(LocalJobRunner.java:210)
```

LocalJobRunner means the job is running locally, not on the cluster.

See 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

For more information on the NameNode, see Section 9.9, “HDFS”.

12.7.1. HDFS Utilization of Tables and Regions

To determine how much space HBase is using on HDFS use the hadoop shell commands from the NameNode. For example...

```
hadoop fs -dus /hbase/
```

...returns the summarized disk utilization for all HBase objects.

```
hadoop fs -dus /hbase/myTable
```

...returns the summarized disk utilization for the HBase table 'myTable'.

```
hadoop fs -du /hbase/myTable
```

...returns a list of the regions under the HBase table 'myTable' and their disk utilization.

For more information on HDFS shell commands, see the HDFS FileSystem Shell documentation.

12.7.2. Browsing HDFS for HBase Objects

Somtimes it will be necessary to explore the HBase objects that exist on HDFS. These objects could include the WALs (Write Ahead Logs), tables, regions, StoreFiles, etc. The easiest way to do this is with the NameNode web application that runs on port 50070. The NameNode web application will provide links to the all the DataNodes in the cluster so that they can be browsed seamlessly.

The HDFS directory structure of HBase tables in the cluster is...

```
/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)
```

The HDFS directory structure of HBase WAL is..

```
/hbase
     /.logs
          /<RegionServer>    (RegionServers)
               /<HLog>          (WAL HLog files for the RegionServer)
```

See the HDFS User Guide for other non-shell diagnostic utilities like fsck.

12.7.2.1. Use Cases

Two common use-cases for querying HDFS for HBase objects is research the degree of uncompaction of a table. If there are a large number of StoreFiles for each ColumnFamily it could indicate the need for a major compaction. Additionally, after a major compaction if the resulting StoreFile is "small" it could indicate the need for a reduction of ColumnFamilies for the table.

12.8. Network

12.8.1. Network Spikes

If you are seeing periodic network spikes you might want to check the compactionQueues to see if major compactions are happening.

See Section 2.8.2.8, “Managed Compactions” for more information on managing compactions.

12.8.2. Loopback IP

HBase expects the loopback IP Address to be 127.0.0.1. See the Getting Started section on Section 2.2.3, “Loopback IP”.

12.8.3. Network Interfaces

Are all the network interfaces functioning correctly? Are you sure? See the Troubleshooting Case Study in Section 12.14, “Case Studies”.

12.9. RegionServer

RegionServer 的更多信息，参考 Section 9.6, “RegionServer”.

12.9.1. 启动错误

12.9.1.1. Master Starts, But RegionServers Do Not

The Master believes the RegionServers have the IP of 127.0.0.1 - which is localhost and resolves to the

master's own localhost.

The RegionServers are erroneously informing the Master that their IP addresses are 127.0.0.1.

Modify /etc/hosts on the region servers, from...

```
# 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
```

... to (removing the master node's name from localhost)...

```
# 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? See 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...(Too many open files)

If you see log messages like this...

```
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.. The default on recent Linux distributions is 1024 - which is far too low for HBase.

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 - retryir
```

... 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 SessionExpired events

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$SendThread.run(ClientCnxn.java:906)
WARN org.apache.hadoop.hbase.util.Sleeper: We slept 79410ms, ten times longer than scheduled: 5000
INFO org.apache.zookeeper.ClientCnxn: Attempting connection to server hostname/IP:PORT
INFO org.apache.zookeeper.ClientCnxn: Priming connection to java.nio.channels.SocketChannel[connected local=/IP:PORT remote=hostname/IP:P(
INFO org.apache.zookeeper.ClientCnxn: Server connection successful
WARN org.apache.zookeeper.ClientCnxn: Exception closing session 0x278bd16a96000d to sun.nio.ch.SelectionKeyImpl@3544d65e
java.io.IOException: Session Expired
       at org.apache.zookeeper.ClientCnxn$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.

See Section 12.11.2, “ZooKeeper, The Cluster Canary” for other general information about ZooKeeper troubleshooting.

12.9.2.8. 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. Regions listed by domain name, then IP

Fix your DNS. In versions of HBase before 0.92.x, reverse DNS needs to give same answer as forward lookup. See HBASE 3431 RegionServer is not using the name given it by the master; double entry in master listing of servers for gorey details.

12.9.2.10. Logs flooded with '2011-01-10 12:40:48,407 INFO org.apache.hadoop.io.compress.CodecPool: Got brand-new compressor' messages

We are not using the native versions of compression libraries. See HBASE-1900 Put back native support when hadoop 0.21 is released. Copy the native libs from hadoop under hbase lib dir or symlink them into place and the message should go away.

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

For more information on the Master, see 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. Startup Errors

12.11.1.1. Could not find my address: 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.

See the ZooKeeper Operating Environment Troubleshooting page. 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 does not seem to work on Amazon EC2

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. Instability on Amazon EC2

Questions on HBase and Amazon EC2 come up frequently on the HBase dist-list. Search for old threads using Search Hadoop

12.12.3. Remote Java Connection into EC2 Cluster Not Working

See Andrew's answer here, up on the user list: Remote Java client connection into EC2 instance.

12.13. HBase and Hadoop version issues

12.13.1. NoClassDefFoundError when trying to run 0.90.x on hadoop-0.20.205.x (or hadoop-1.0.x)

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. Case Studies

For Performance and Troubleshooting Case Studies, see Chapter 13, Case Studies.

Chapter 13. Case Studies

Table of Contents

13.1. Overview

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. Schema Design

13.2.1. 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 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 http://hbase.apache.org/book.html#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. Performance/Troubleshooting

13.3.1. Case Study #1 (Performance Issue On A Single Node)

13.3.1.1. Scenario

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. Hardware

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. Hypotheses

13.3.1.3.1. HBase "Hot Spot" Region

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 Region With Non-Local Data

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. Resolution

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. Case Study #2 (Performance Research 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

See also Section 2.3.2, “dfs.datanode.max.xcievers”.

Chapter 14. HBase Operational Management

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 Tools and Utilities

Here we list HBase tools for administration, analysis, fixup, and debugging.

14.1.1. Driver

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 since the
```

... 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 Tool

See Section 9.7.5.2.2, “HFile Tool”.

14.1.4. WAL Tools

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,60020,1283516293161/10
```

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,60020,1283516293161/
```

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

See 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] tablename
```

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
```

Note: caching for the input Scan is configured via hbase.client.scanner.caching in the job configuration.

14.1.7. Export

Export is a utility that will dump the contents of table to HDFS in a sequence file. Invoke via:

```
$ 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. Import

Import is a utility that will load data that has been exported back into HBase. Invoke via:

```
$ 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-outputdir <tabl
```

These generated StoreFiles can be loaded into HBase via Section 14.1.10, “CompleteBulkLoad”.

14.1.9.1. ImportTsv Options

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.TsvImporterMapper
```

14.1.9.2. ImportTsv Example

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 -Dimporttsv.
```

... 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. See Also

For more information about bulk-loading HFiles into HBase, see Section 9.8, “Bulk Loading”

14.1.10. CompleteBulkLoad

The `completebulkload` utility will move generated StoreFiles into an HBase table. This utility is often used in conjunction with output from Section 14.1.9, “ImportTsv”.

There are two ways to invoke this utility, with explicit classname and via the driver:

```
$ bin/hbase org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFile <hdfs://storefileoutput> <tablename>
```

.. and via the Driver..

```
HADOOP_CLASSPATH=`${HBASE_HOME}/bin/hbase classpath` ${HADOOP_HOME}/bin/hadoop jar ${HBASE_HOME}/hbase-VERSION.jar completebulkload <hdfs:
```

For more information about bulk-loading HFiles into HBase, see Section 9.8, “Bulk Loading”.

14.1.11. WALPlayer

WALPlayer is a utility to replay WAL files into 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 is a utility that will count all the rows of a table. This is a good utility to use as a sanity check to ensure that HBase can read all the blocks of a table if there are any concerns of metadata inconsistency.

```
$ 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. Region Management

14.2.1. Major Compaction

Major compactions can be requested via the HBase shell or HBaseAdmin.majorCompact.

Note: major compactions do NOT do region merges. See Section 9.7.5.5, “Compaction” for more information about compactions.

14.2.2. Merge

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. See 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. Node Management

14.3.1. Node Decommission

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