HBase 官方文档

Revision History	
Revision 0.90.4	
配置，数据模型使用入门	

Abstract

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.

我(译者)熟悉Hbase的源代码，从事Hbase的开发运维工作，如果有什么地方不清楚，欢迎一起讨论。邮箱yankaycom@gmail.com

---

Table of Contents

List of Tables

序

这本书是 HBase 的官方指南。 版本为 *0.90.4*.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 1.3, “慢速开始(相对快速开始)” 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-0.90.4.tar.gz`.

解压缩，然后进入到那个要解压的目录.

```
$ tar xfz hbase-0.90.4.tar.gz
```

```
$ cd hbase-0.90.4
```

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 1.3, “慢速开始(相对快速开始)”，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。

1.3. 慢速开始(相对快速开始)

1.3.1. 需要的软件

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。

1.3.1.1. java

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

1.3.1.2. hadoop

该版本的Hbase只可以运行在Hadoop 0.20.x，不可以运行于hadoop 0.21.x (0.22.x也不行). HBase运行在没有持久同步功能的HDFS上会丢失数据。 Hadoop 0.20.2 和 Hadoop 0.20.203.0就没有这个功能。现在只有 branch-0.20-append 补丁有这个功能[1]. 现在官方的发行版都没有这个功能，所以你要自己打这个补丁。推荐看 Michael Noll 写的详细的说明, Building an Hadoop 0.20.x version for HBase 0.90.2.

你还可以用 Cloudera's CDH3. CDH 打了这个补丁 (CDH3 betas 就可以满足; b2, b3, or b4).

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.

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.

1.3.1.3. 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").

1.3.1.4. 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.

1.3.1.5. 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!

1.3.1.6. `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]

1.3.1.6.1. 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!

1.3.1.7. 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]

1.3.1.8. 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

1.3.2. HBase运行模式:单机和分布式

HBase有两个运行模式: Section 1.3.2.1, “单机模式” 和 Section 1.3.2.2, “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.

1.3.2.1. 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。

1.3.2.2. 分布式模式

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

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

在我们安装之后，你需要确认你的*伪分布式模式*或者 *完全分布式模式*的配置是否正确。这两个模式可以使用同一个验证脚本Section 1.3.2.3, “运行和确认你的安装”。

1.3.2.2.1. 伪分布式模式

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

你确认HDFS安装成功之后，就可以先编辑 `conf/hbase-site.xml`。在这个文件你可以加入自己的配置，这个配置会覆盖 Section 3.1.1, “HBase 默认配置” and Section 1.3.2.2.2.3, “HDFS客户端配置”. 运行Hbase需要设置`hbase.rootdir` 属性.该属性是指Hbase在HDFS中使用的目录的位置。例如，要想 `/hbase` 目录，让namenode 监听locahost的9000端口，只有一份数据拷贝(HDFS默认是3份拷贝)。可以在 `hbase-site.xml` 写上如下内容

```
<configuration>

  ...

  <property>

    <name>hbase.rootdir</name>

    <value>hdfs://localhost:9000/hbase</value>

    <description>The directory shared by RegionServers.

    </description>

  </property>

  <property>

    <name>dfs.replication</name>

    <value>1</value>

    <description>The replication count for HLog & HFile storage. Should not
be greater than HDFS datanode count.

    </description>

  </property>

  ...

</configuration>
```

Note

让Hbase自己创建 `hbase.rootdir` 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 1.3.2.3, “运行和确认你的安装” 来运行和确认你的伪分布式模式安装了。 [7]

1.3.2.2.2. 完全分布式模式

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

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

1.3.2.2.2.1. regionservers

完全分布式模式的还需要修改`conf/regionservers`. 在 Section 1.3.3.1.2, "`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.

1.3.2.2.2.2. 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

打个比方，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>
```

1.3.2.2.2.2.1. 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.

1.3.2.2.2.3. HDFS客户端配置

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

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

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

1.3.2.3. 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.

1.3.3. 配置例子

1.3.3.1. 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.

1.3.3.1.1. `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>
```

1.3.3.1.2. 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
```

1.3.3.1.3. 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.

Chapter 2. 升级

Table of Contents

参见 Section 1.3.1, “需要的软件”, 需要特别注意有关Hadoop 版本的信息.

2.1. 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 3.1.1, “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

Chapter 3. 配置

Table of Contents

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**.

3.1. hbase-site.xml 和 hbase-default.xml

正如Hadoop放置HDFS的配置文件hdfs-site.xml，Hbase的配置文件是conf/hbase-site.xml. 你可以在 Section 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。

3.1.1. HBase 默认配置

HBase 默认配置

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

3.2. `hbase-env.sh`

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

这个文件的改动系统Hbase重启才能生效。

3.3. `log4j.properties`

编辑这个文件可以改变Hbase的日志的级别，轮滚策略等等。

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

3.4. 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)。

3.5. 必须的配置

参见 Section 1.3.1, “需要的软件”. 这里列举了运行Hbase至少两个必须的配置: i.e. Section 1.3.1.6, “ `ulimit` 和 `nproc` ” 和 Section 1.3.1.7, “`dfs.datanode.max.xcievers`”.

3.6. 推荐的配置

3.6.1. `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。

3.6.2. `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。

3.6.3. 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.(译者注:原文到此为止，汗)

3.6.4. LZO 压缩

你可以考虑使用Lzo压缩，这个可以无缝集成，并且在大多数情况下可以提供性能。

Hbase是Apache的协议，而LZO是GPL的协议。Hbase不能自带LZO，因此LZO需要在安装Hbase之前安装。参见 使用 LZO 压缩介绍了如何在Hbase中使用LZO

一个常见的问题是，用户在一开始使用LZO的时候会很好，但是数月过去，管理员在给集群添加集群的时候，他们忘记了LZO的事情。在0.90.0版本之后，我们会运行失败，但也有可能不。 请你要阅读这一段[11].

还要在本书的尾部参见 Appendix B, *HBase 中的压缩* .

3.6.5. 更大的 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)。默认情况下单个Region是256MB.你可以设置为1G。有些人使用更大的，4G甚至更多。可以调整`hbase-site.xml`中的`hbase.hregion.max.filesize`属性.

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

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

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

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

3.7. 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>
```

3.7.1. 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.

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()
=> 1218920189000
```

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. 构建 HBase

Table of Contents

5.1. 将一个 HBase release 加入到 Apache's Maven Repository

可以参考 发布 Maven Artifacts的信息.要想让所有的组件正确运行，关键在于配置好mvn release plugin。确保你在运行**mvn release:perform**之前使用的是正确的分支版本。这点非常的重要，要手写`${HBASE_HOME}`下的release.properties文件，然后执行**release:perform**.。你需要编辑它，这样才能将他指向一个正确的SVN地址。(译者注：可以使用cloudera)

如果你出现了如下的问题，是因为你需要在pom.xml里编辑版本然后加上 `-SNAPSHOT` 。

```
[INFO] Scanning for projects...

[INFO] Searching repository for plugin with prefix: 'release'.

[INFO] ------------------------------------------------------------------------
--------

[INFO] Building HBase

[INFO]    task-segment: [release:prepare] (aggregator-style)

[INFO] ------------------------------------------------------------------------
--------

[INFO] [release:prepare {execution: default-cli}]

[INFO] ------------------------------------------------------------------------
--------

[ERROR] BUILD FAILURE

[INFO] ------------------------------------------------------------------------
--------

[INFO] You don't have a SNAPSHOT project in the reactor projects list.

[INFO] ------------------------------------------------------------------------
--------

[INFO] For more information, run Maven with the -e switch
```

```
[INFO] ------------------------------------------------------------------------
--------

[INFO] Total time: 3 seconds

[INFO] Finished at: Sat Mar 26 18:11:07 PDT 2011

[INFO] Final Memory: 35M/423M

[INFO] ------------------------------------------------------------------------
-------
```

Chapter 6. Developers

Table of Contents

6.1. IDEs

6.1.1. Eclipse

参见 HBASE-3678 Add Eclipse-based Apache Formatter to HBase Wiki可以看到一个eclipse的格式化文件，可以帮你把编码转换成符合Hbase的格式。 这个issue还包含有使用这个formatter的指导。

6.2. 单元测试

我们在Hbase中使用JUnit 4. 如果你希望跑一个最小化的HDFS, ZooKeeper, HBase, 或者 MapReduce 测试,可以checkout`HBaseTestingUtility`. Alex Baranau of Sematext 阐述了怎么使用它 HBase Case-Study: Using HBaseTestingUtility for Local Testing and Development (2010).

6.2.1. Mocito

有些时候你不需要运行一个完全的running server单元测试。比如一些操作 `org.apache.hadoop.hbase.Server` 实例的方法或者使用 `org.apache.hadoop.hbase.master.MasterServices` 接口而不是 `org.apache.hadoop.hbase.master.HMaster`类的应用. 这些情况下，你可以不必使用 mocked `Server` 实例. 比如:

（译者注：原文到此为止）

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.

7.1. 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 。

7.2. HBase Input MapReduce 例子

要想使HBase作为MapReduce的source,Job需要使用TableMapReduceUtil来配置，如下所示...

```
Job job = ...;

Scan scan = new Scan();

scan.setCaching(500);  // 1 is the default in Scan, which will be bad
for MapReduce jobs

scan.setCacheBlocks(false);

// Now set other scan attrs

...


TableMapReduceUtil.initTableMapperJob(

  tableName,             // input HBase table name

  scan,                  // Scan instance to control CF and attribute
selection

  MyMapper.class,               // mapper
```

```
  Text.class,            // reducer key
  LongWritable.class,    // reducer value
  job                    // job instance
  );
```

...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.
```

7.3. 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。

Chapter 8. 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.

8.1. Schema 创建

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

8.2. 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)

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。

8.3. 单调递增Row Keys/时序数据(log)

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。

8.4. 尽量最小化row和column的大小

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 12.3.4.2, "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 `

8.5. 版本的时间

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 11, *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。

Chapter 9. Metrics

Table of Contents

9.1. Metric 安装

参见 Metrics 可以获得一个enable Metrics emission的指导。

9.2. RegionServer Metrics

9.2.1. hbase.regionserver.blockCacheCount

内存中的Block cache item数量。这个是存储文件(HFiles)的缓存中的数量。

9.2.2. hbase.regionserver.blockCacheFree

内存中的Block cache memory 剩余 (单位 bytes).

9.2.3. hbase.regionserver.blockCacheHitRatio

Block cache 命中率(0 到 100). TODO: 描述当cacheBlocks=false时对这个值得影响

9.2.4. hbase.regionserver.blockCacheSize

内存中的Block cache 大小 (单位 bytes)

9.2.5. hbase.regionserver.compactionQueueSize

compaction队列的大小. 这个值是需要进行compaction的region数目

9.2.6. hbase.regionserver.fsReadLatency_avg_time

文件系统延迟 (ms). 这个值是平均读HDFS的延迟时间

9.2.7. hbase.regionserver.fsReadLatency_num_ops

TODO

9.2.8. hbase.regionserver.fsSyncLatency_avg_time

文件系统同步延迟(ms)

9.2.9. hbase.regionserver.fsSyncLatency_num_ops

TODO

9.2.10. hbase.regionserver.fsWriteLatency_avg_time

文件系统写延迟(ms)

9.2.11. hbase.regionserver.fsWriteLatency_num_ops

TODO

9.2.12. hbase.regionserver.memstoreSizeMB

所有的RegionServer的memstore大小 (MB)

9.2.13. hbase.regionserver.regions

RegionServer服务的regions数量

9.2.14. hbase.regionserver.requests

读写请求的全部数量。请求是指RegionServer的RPC数量，因此一次Get一个情况，一个带缓存的Scan也是一个请求。一个批量load是一个Hfile一个请求。

9.2.15. hbase.regionserver.storeFileIndexSizeMB

当前RegionServer的storefile索引的总大小(MB)

9.2.16. hbase.regionserver.stores

RegionServer打开的stores数量。一个stores对应一个column family。例如，一个表有3个region在这个RegionServer上，对应一个 column family就会有3个store.

9.2.17. hbase.regionserver.storeFiles

RegionServer打开的存储文件(HFile)数量。这个值一定大于等于store的数量。

Chapter 10. 跨集群复制

参见 跨集群复制.

Chapter 11. 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.

11.1. 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 11.1. 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>..."	

11.2. 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 11.2. 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 11.3. 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`。

11.3. 表

表是在schema声明的时候定义的。

11.4. 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。

11.5. 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。

11.6. 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。

11.7. 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].

11.7.1. 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。

11.7.1.1. 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.

11.7.1.2. 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
```

11.7.1.3. 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
```

11.7.1.4. 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。

11.7.1.4.1. 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);
```

11.7.1.4.2. 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);
```

11.7.1.5. 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。

11.7.2. 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。

11.7.2.1. 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，

11.7.2.2. 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

Chapter 12. 架构

Table of Contents

12.1. 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。

管理集群操作是经由HBaseAdmin发起的

12.1.1. 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.

12.1.2. 写缓冲和批量操作

若关闭了HTable中的 Section 13.6.1, “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.

12.1.3. Filters

Get 和 Scan实例可以使用 filters，这个过滤操作是运行在RegionServer上的。

12.2. Daemons

12.2.1. Master

12.2.2. RegionServer

12.3. Regions

本章节都是再讲Regions.

Note

Regions是由每个Column Family的Store组成。

12.3.1. 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。

12.3.2. 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加入到元信息中，再将他们加入到原本的RegionServer中，最后汇报Master.参见Section 3.6.6, “管理 Splitting”来手动管理切割操作。

12.3.3. Region负载均衡

当没有任何region在进行转换的时候，Hbase会定期执行一个load balance。他会将移动region进行集群的负载均衡。可以配置运行时间间隔。

12.3.4. 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.

12.3.4.1. 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。

12.3.4.2. StoreFile (HFile)

12.3.4.2.1. 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。

12.3.4.2.2. 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/D
SMP/4759508618286845475
```

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。

12.3.4.3. 压缩

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 3.6.6, “管理 Splitting”。

12.4. Write Ahead Log (WAL)

12.4.1. 目的

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 12.3.4, “Store”的Section 12.3.4.1, “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.

12.4.2. WAL Flushing

TODO (describe).

12.4.3. WAL Splitting

12.4.3.1. 当RegionServer宕掉的时候，如何恢复

TODO

12.4.3.2. `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。[19]

12.4.3.3. 如果处理一个发生在当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。[20]

---

[19] 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.

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

Chapter 13. 性能调优

Table of Contents

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.

13.1. Java

13.1.1. 垃圾收集和HBase

13.1.1.1. 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.

13.2. 配置

参见Section 3.6, “推荐的配置”.

13.2.1. 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”

13.2.2. 管理压缩

对于大型的系统，你需要考虑管理压缩和分割

13.2.3. 压缩

生产环境中的系统需要在column family的定义中使用Section 3.6.4, “LZO 压缩”之类的压缩。

13.2.4. 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)

13.2.5. hfile.block.cache.size

参见 hfile.block.cache.size. 对于RegionServer进程的内存设置。

13.2.6. 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。

13.2.7. 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。

13.2.8. 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。

13.2.9. hbase.hregion.memstore.block.multiplier

参见 hbase.hregion.memstore.block.multiplier. 如果有足够的RAM，提高这个值。

13.3. Column Families的数目

参见 Section 8.2, “ column families的数量 ”.

13.4. 数据聚集

如果你的数据总是往一个region写。你可以再看看处理时序数据 这一章.

13.5. 批量Loading

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

13.5.1. 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(BigInteg
er.valueOf(i)));

    byte[] b = String.format("%016x",
key).getBytes();

    splits[i] = b;

  }

  return splits;

}
```

13.6. HBase客户端

13.6.1. 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`

13.6.2. 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。

13.6.3. 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。

13.6.4. 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();
```

13.6.5. 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。

13.6.6. 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。

Chapter 14. Bloom Filters

Table of Contents

Bloom filters 是在 HBase-1200 Add bloomfilters上面开发的.[21][22] （译者注:Bloom Filter是一个算法，可以用来快速确认一个Row Key或者值是否在一个Hfile里面。）

14.1. 配置

可以在column family的选项的配置Blooms.可以通过Hbase Shell，也可以用Java代码操作`org.apache.hadoop.hbase.HColumnDescriptor`.

14.1.1. `HColumnDescriptor` 配置

使用 `HColumnDescriptor.setBloomFilterType(NONE | ROW | ROWCOL)`来控制每个column family的Blooms这种。默认值是 `NONE` ，如果值是`ROW`，就会在插入的时候去Hash这个row，加入到Bloom中去。如果值是`ROWCOL`，就Hash这个row,column family和column family qualifer。(译者注，ROW是哈希row key)

14.1.2. io.hfile.bloom.enabled 全局关闭开关

当有些东西出错的时候，Configuration中的io.hfile.bloom.enabled是一个关闭的开关。 默认是true.

14.1.3. io.hfile.bloom.error.rate

io.hfile.bloom.error.rate = 平均错误率。默认是1%.减少一半(如 .5%)，就意味着每个bloom entry加一个bit.

14.1.4. io.hfile.bloom.max.fold

io.hfile.bloom.max.fold = 保证最低的fold率。大多数人不该修改这个值，默认是7，就是说可以折叠到原本大小的1/128。参见 *Development Process*中的文档 BloomFilters in HBase获得更多关于这个配置的信息。

14.2. Bloom StoreFile footprint

Bloom filters在StoreFile加入了一个entry.包括 一般的FileInfo 数据结构和两个额外entries到StoreFile的元数据部分中。

14.2.1. StoreFile中的BloomFilter， FileInfo数据结构

14.2.1.1. BLOOM_FILTER_TYPE

FileInfo有一个 BLOOM_FILTER_TYPE entry，可以被设置为 NONE, ROW 或者 ROWCOL.

14.2.2. 在 StoreFile 元数据中的BloomFilter entries

14.2.2.1. BLOOM_FILTER_META

BLOOM_FILTER_META保存了Bloom的大小，使用的Hash算法等信息。他的大小的很小。 StoreFile.Reader加载的时候会缓存进去。

14.2.2.2. BLOOM_FILTER_DATA

BLOOM_FILTER_DATA是实际的bloomfiter数据。按需获取，保存在LRU缓存中(如果缓存是开启的，默认开启)。

---

[21] 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.

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

Chapter 15. Hbase的故障排除和Debug

Table of Contents

15.1. 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 1.3.1.6, “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 13.1.1.1, “长时间GC停顿”.

15.2. 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`

15.2.1. 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。

15.2.1.1. 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。

15.2.1.2. 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。

15.3. 工具

15.3.1. search-hadoop.com

search-hadoop.com将所有的 mailing lists 和 JIRA建立了索引。用它来找Hadoop/HBase的问题很方便。

15.3.2. 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)。

15.3.3. 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。

15.3.4. 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:+AggressiveOpts
-XX:+UseConcMarkSweepGC -XX:NewSize=64m -
XX:MaxNewSize=64m -
XX:CMSInitiatingOccupancyFraction=88 -
verbose:gc -XX:+PrintGCDetails -
XX:+PrintGCTimeStamps -
Xloggc:/export1/hadoop/logs/gc-hbase.log
-Dcom.sun.management.jmxremote.port=10102
-
Dcom.sun.management.jmxremote.authenticat
e=true -
Dcom.sun.management.jmxremote.ssl=false -
Dcom.sun.management.jmxremote.password.fi
le=/home/hadoop/hbase/conf/jmxremote.pass
word -Dcom.sun.management.jmxremote -
Dhbase.log.dir=/export1/hadoop/logs -
Dhbase.log.file=hbase-hadoop-
regionserver-sv4borg12.log -
Dhbase.home.dir=/home/hadoop/hbase -
Dhbase.id.str=hadoop -
Dhbase.root.logger=INFO,DRFA -
Djava.library.path=/home/hadoop/hbase/lib
/native/Linux-amd64-64 -classpath
/home/hadoop/hbase/bin/../conf:[many
jars]:/home/hadoop/hadoop/conf
org.apache.hadoop.hbase.regionserver.HReg
ionServer start
```

15.3.5. 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.AbstractQueued
Synchronizer$ConditionObject)

          at
java.util.concurrent.locks.LockSupport.pa
rkNanos(LockSupport.java:198)

          at
java.util.concurrent.locks.AbstractQueued
Synchronizer$ConditionObject.awaitNanos(A
bstractQueuedSynchronizer.java:1963)

          at
java.util.concurrent.LinkedBlockingQueue.
poll(LinkedBlockingQueue.java:395)

          at
org.apache.hadoop.hbase.regionserver.HReg
ionServer.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.inv
oke(Unknown Source)

          at
sun.reflect.DelegatingMethodAccessorImpl.
invoke(DelegatingMethodAccessorImpl.java:
25)

          at
java.lang.reflect.Method.invoke(Method.ja
va:597)

          at
org.apache.hadoop.io.retry.RetryInvocatio
nHandler.invokeMethod(RetryInvocationHand
ler.java:82)

          at
org.apache.hadoop.io.retry.RetryInvocatio
nHandler.invoke(RetryInvocationHandler.ja
va:59)

          at $Proxy1.complete(Unknown
Source)

          at
org.apache.hadoop.hdfs.DFSClient$DFSOutpu
tStream.closeInternal(DFSClient.java:3390
)

          - locked <0x00007f16cb14b470> (a
org.apache.hadoop.hdfs.DFSClient$DFSOutpu
tStream)

          at
org.apache.hadoop.hdfs.DFSClient$DFSOutpu
tStream.close(DFSClient.java:3304)

          at
org.apache.hadoop.fs.FSDataOutputStream$P
ositionCache.close(FSDataOutputStream.jav
a:61)
```

```
          at
org.apache.hadoop.fs.FSDataOutputStream.c
lose(FSDataOutputStream.java:86)

          at
org.apache.hadoop.hbase.io.hfile.HFile$Wr
iter.close(HFile.java:650)

          at
org.apache.hadoop.hbase.regionserver.Stor
eFile$Writer.close(StoreFile.java:853)

          at
org.apache.hadoop.hbase.regionserver.Stor
e.internalFlushCache(Store.java:467)

          - locked <0x00007f16d00e6f08> (a
java.lang.Object)

          at
org.apache.hadoop.hbase.regionserver.Stor
e.flushCache(Store.java:427)

          at
org.apache.hadoop.hbase.regionserver.Stor
e.access$100(Store.java:80)

          at
org.apache.hadoop.hbase.regionserver.Stor
e$StoreFlusherImpl.flushCache(Store.java:
1359)

          at
org.apache.hadoop.hbase.regionserver.HReg
ion.internalFlushcache(HRegion.java:907)

          at
org.apache.hadoop.hbase.regionserver.HReg
ion.internalFlushcache(HRegion.java:834)

          at
org.apache.hadoop.hbase.regionserver.HReg
ion.flushcache(HRegion.java:786)

          at
org.apache.hadoop.hbase.regionserver.MemS
toreFlusher.flushRegion(MemStoreFlusher.j
ava:250)

          at
org.apache.hadoop.hbase.regionserver.MemS
toreFlusher.flushRegion(MemStoreFlusher.j
ava:224)

          at
org.apache.hadoop.hbase.regionserver.MemS
toreFlusher.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..0x00007f16afefd9f0]

   java.lang.Thread.State: WAITING
(parking)

          at sun.misc.Unsafe.park(Native
Method)

          - parking to wait for
<0x00007f16cd3f8dd8> (a
java.util.concurrent.locks.AbstractQueued
Synchronizer$ConditionObject)

          at
java.util.concurrent.locks.LockSupport.pa
rk(LockSupport.java:158)

          at
java.util.concurrent.locks.AbstractQueued
Synchronizer$ConditionObject.await(Abstra
ctQueuedSynchronizer.java:1925)

          at
java.util.concurrent.LinkedBlockingQueue.
take(LinkedBlockingQueue.java:358)

          at
org.apache.hadoop.hbase.ipc.HBaseServer$H
andler.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.KeyV
alueHeap.<init>(KeyValueHeap.java:56)
```

```
          at
org.apache.hadoop.hbase.regionserver.Stor
eScanner.<init>(StoreScanner.java:79)

          at
org.apache.hadoop.hbase.regionserver.Stor
e.getScanner(Store.java:1202)

          at
org.apache.hadoop.hbase.regionserver.HReg
ion$RegionScanner.<init>(HRegion.java:220
9)

          at
org.apache.hadoop.hbase.regionserver.HReg
ion.instantiateInternalScanner(HRegion.ja
va:1063)

          at
org.apache.hadoop.hbase.regionserver.HReg
ion.getScanner(HRegion.java:1055)

          at
org.apache.hadoop.hbase.regionserver.HReg
ion.getScanner(HRegion.java:1039)

          at
org.apache.hadoop.hbase.regionserver.HReg
ion.getLastIncrement(HRegion.java:2875)

          at
org.apache.hadoop.hbase.regionserver.HReg
ion.incrementColumnValue(HRegion.java:297
8)

          at
org.apache.hadoop.hbase.regionserver.HReg
ionServer.incrementColumnValue(HRegionSer
ver.java:2433)

          at
sun.reflect.GeneratedMethodAccessor20.inv
oke(Unknown Source)

          at
sun.reflect.DelegatingMethodAccessorImpl.
invoke(DelegatingMethodAccessorImpl.java:
25)

          at
java.lang.reflect.Method.invoke(Method.ja
va:597)

          at
org.apache.hadoop.hbase.ipc.HBaseRPC$Serv
er.call(HBaseRPC.java:560)
```

```
        at
org.apache.hadoop.hbase.ipc.HBaseServer$H
andler.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
[0x00007f16b517d000..0x00007f16b517dbf0]

   java.lang.Thread.State: RUNNABLE

        at
sun.nio.ch.EPollArrayWrapper.epollWait(Na
tive Method)

        at
sun.nio.ch.EPollArrayWrapper.poll(EPollAr
rayWrapper.java:215)

        at
sun.nio.ch.EPollSelectorImpl.doSelect(EPo
llSelectorImpl.java:65)

        at
sun.nio.ch.SelectorImpl.lockAndDoSelect(S
electorImpl.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(SelectorIm
pl.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.r
ead(SocketInputStream.java:155)

          at
org.apache.hadoop.net.SocketInputStream.r
ead(SocketInputStream.java:128)

          at
java.io.FilterInputStream.read(FilterInpu
tStream.java:116)

          at
org.apache.hadoop.ipc.Client$Connection$P
ingInputStream.read(Client.java:304)

          at
java.io.BufferedInputStream.fill(Buffered
InputStream.java:218)

          at
java.io.BufferedInputStream.read(Buffered
InputStream.java:237)

          - locked <0x00007f1808539178> (a
java.io.BufferedInputStream)

          at
java.io.DataInputStream.readInt(DataInput
Stream.java:370)

          at
org.apache.hadoop.ipc.Client$Connection.r
eceiveResponse(Client.java:569)

          at
org.apache.hadoop.ipc.Client$Connection.r
un(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$DFSOutpu
tStream.processDatanodeError(DFSClient.ja
va:2636)

          at
org.apache.hadoop.hdfs.DFSClient$DFSOutpu
tStream.<init>(DFSClient.java:2832)

          at
org.apache.hadoop.hdfs.DFSClient.append(D
FSClient.java:529)

          at
org.apache.hadoop.hdfs.DistributedFileSys
tem.append(DistributedFileSystem.java:186
)

          at
org.apache.hadoop.fs.FileSystem.append(Fi
leSystem.java:530)

          at
org.apache.hadoop.hbase.util.FSUtils.reco
verFileLease(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.sp
litLogAfterStartup(HMaster.java:648)

          at
org.apache.hadoop.hbase.master.HMaster.jo
inCluster(HMaster.java:572)
```

```
          at
org.apache.hadoop.hbase.master.HMaster.ru
n(HMaster.java:503)
```

15.3.6. 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。

15.3.7. 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。

15.4. 客户端

15.4.1. ScannerTimeoutException

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。

15.5. RegionServer

15.5.1. 启动错误

15.5.1.1. 压缩链接错误

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(ClassLo
ader.java:1734)

        at
java.lang.Runtime.loadLibrary0(Runtime.ja
va: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.

15.5.2. 运行时错误

15.5.2.1. java.io.IOException...(Too many open files)

参见快速入门的章节ulimit and nproc configuration.

15.5.2.2. 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.

15.5.2.3. 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.

15.5.2.4. 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.HReg
ionServer: unable to report to master for
xxx milliseconds - retrying
```

或者看到了全GC压缩操作，你可能正在执行一个全GC。

15.5.2.5. "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。

15.5.3. 终止错误

15.6. Master

15.6.1. 启动错误

15.6.2. 终止错误

Appendix A. 工具

Table of Contents

这里我们列举一些Hbase管理，分析，修理和Debug的工具。

A.1. HBase hbck

用于Hbase安装的 *fsck*

在Hbase集群上运行 hbck

```
$ ./bin/hbase hbck
```

这个命令的输出是 *OK* 或者 *INCONSISTENCY*. 如果你的集群汇报inconsistencies，加上**-details** 看更多的详细信息。如果inconsistencies，多运行**hbck** 几次，因为inconsistencies可能是暂时的。 (集群正在启动或者region正在split)。加上**-fix**可以修复inconsistency(这是一个实验性的功能)

A.2. HFile 工具

参见 Section 12.3.4.2.2, “HFile工具”.

A.3. WAL Tools

A.3.1. HLog 工具

HLog的main方法提供了手动 切割和dump的方法。会把WALs或者splite的结果的内容保存在recovered.edits目录下

你可以使用

```
 $ ./bin/hbase
org.apache.hadoop.hbase.regionserver.wal.
HLog --dump
hdfs://example.org:9000/hbase/.logs/examp
le.org,60020,1283516293161/10.10.21.10%3A
60020.1283973724012
```

来获得一个WAL文件内容的文本化的Dump。如果返回码部位0，说明文件有错误，所有你可以用这个命令来看文件是否健康，将命令重定向到/dev/null，检查返回码就可以了。

相似的，你可以将一个log切割，运行如下命令：

```
 $ ./bin/hbase
org.apache.hadoop.hbase.regionserver.wal.
HLog --split
hdfs://example.org:9000/hbase/.logs/examp
le.org,60020,1283516293161/
```

A.4. 压缩工具

参见 Section A.4, “压缩工具”.

A.5. Node下线

你可以在Hbase的特定的节点上运行下面的脚本来停止RegionServer:

```
$ ./bin/hbase-daemon.sh stop regionserver
```

RegionServer会首先关闭所有的region然后把它自己关闭，在停止的过程中，RegionServer的会向Zookeeper报告说他已经过期了。master会发现

RegionServer已经死了，会把它当作崩溃的server来处理。他会将region分配到其他的节点上去。

在下线节点之前要停止Load Balancer

> 如果在运行load balancer的时候，一个节点要关闭， 则Load Balancer和Master的recovery可能会争夺这个要下线的Regionserver。为了避免这个问题，先将load balancer停止，参见下面的 Load Balancer.

RegionServer下线有一个缺点就是其中的Region会有好一会离线。Regions是被按顺序关闭的。如果一个server上有很多region,从第一个region会被下线，到最后一个region被关闭，并且Master确认他已经死了，该region才可以上线，整个过程要花很长时间。在Hbase 0.90.2中，我们加入了一个功能，可以让节点逐渐的摆脱他的负载，最后关闭。HBase 0.90.2加入了`graceful_stop.sh`脚本，可以这样用，

```
$ ./bin/graceful_stop.sh

Usage: graceful_stop.sh [--config &conf-
dir>] [--restart] [--reload] [--thrift]
[--rest] &hostname>

 thrift      If we should stop/start
thrift before/after the hbase stop/start

 rest        If we should stop/start rest
before/after the hbase stop/start

 restart     If we should restart after
graceful stop

 reload      Move offloaded regions back
on to the stopped server

 debug       Move offloaded regions back
on to the stopped server

 hostname    Hostname of server we are to
stop
```

要下线一台RegionServer可以这样做

```
$ ./bin/graceful_stop.sh HOSTNAME
```

这里的HOSTNAME是RegionServer的host you would decommission.

On HOSTNAME

传递到graceful_stop.sh的HOSTNAME必须和hbase使用的hostname一致，hbase用它来区分RegionServers。可以用master的UI来检查RegionServers的id。通常是hostname,也可能是FQDN。不管Hbase使用的哪一个，你可以将它传到 graceful_stop.sh脚本中去，目前他还不支持使用IP地址来推断hostname。所以使用IP就会发现server不在运行，也没有办法下线了。

graceful_stop.sh 脚本会一个一个将region从RegionServer中移除出去，以减少改RegionServer的负载。他会先移除一个region,然后再将这个region安置到一个新的地方，再移除下一个，直到全部移除。最后graceful_stop.sh脚本会让RegionServer **stop**.,Master会注意到RegionServer已经下线了，这个时候所有的region已经重新部署好。RegionServer就可以干干净净的结束，没有WAL日志需要分割。

Load Balancer

当执行**graceful_stop**脚本的时候，要将Region Load Balancer关掉(否则balancer和下线脚本会在region部署的问题上存在冲突):

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

```
true
```

```
0 row(s) in 0.3590 seconds
```

上面是将balancer关掉，要想开启：

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

```
false
```

```
0 row(s) in 0.3590 seconds
```

A.5.1. 依次重启

你还可以让这个脚本重启一个RegionServer,不改变上面的Region的位置。要想保留数据的位置，你可以依次重启(Rolling Restart),就像这样:

```
$ for i in `cat conf/regionservers|sort`;
do ./bin/graceful_stop.sh --restart --
reload --debug $i; done &> /tmp/log.txt &
```

Tail `/tmp/log.txt`来看脚本的运行过程.上面的脚本只对RegionServer进行操作。要确认load balancer已经关掉。还需要在之前更新master。下面是一段依次重启的伪脚本,你可以借鉴它：

1. 确认你的版本，保证配置已经rsync到整个集群中。如果版本是0.90.2，需要打上HBASE-3744 和HBASE-3756两个补丁。
2. 运行hbck确保你的集群是一致的

```
$ ./bin/hbase hbck
```

当发现不一致的时候，可以修复他。

3. 重启Master:

```
$ ./bin/hbase-daemon.sh stop
master; ./bin/hbase-daemon.sh
start master
```

4. 关闭region balancer:

```
$ echo "balance_switch false"
| ./bin/hbase
```

5. 在每个RegionServer上运行 `graceful_stop.sh`:

```
6. $ for i in `cat
   conf/regionservers|sort`;
   do ./bin/graceful_stop.sh --
   restart --reload --debug $i;
   done &> /tmp/log.txt &
```

```
```

如果你在RegionServer还开起来thrift和rest server。还需要加上--thrift or --rest 选项 (参见 `graceful_stop.sh` 脚本的用法).

7. 再次重启Master.这会把已经死亡的server列表清空，重新开启balancer.
8. 运行 hbck 保证集群是一直的

Appendix B. HBase中的压缩

Table of Contents

B.1. 测试压缩工具

HBase有一个用来测试压缩新的工具。要想运行它，输入`/bin/hbase org.apache.hadoop.hbase.util.CompressionTest`. 就会有提示这个工具的具体用法

B.2. `hbase.regionserver.codecs`

如果你的安装错误，就会测试不成功，或者无法启动。可以在你的`hbase-site.xml`加上配置 `hbase.regionserver.codecs` 值你需要的codecs。例如，如果 `hbase.regionserver.codecs` 的值是 `lzo,gz` 同时lzo不存在或者没有正确安装， RegionServer在启动的时候会提示配置错误。

当一台新机器加入到集群中的时候，管理员一定要注意，这台新机器有可能要安装特定的压缩解码器。

B.3. LZO

参见上面的 Section 3.6.4, “LZO 压缩”

B.4. GZIP

相对于LZO，GZIP的压缩率更高但是速度更慢。在某些特定情况下，压缩率是优先考量的。Java会使用Java自带的GZIP，除非Hadoop的本地库在CLASSPATH中。在这种情况下，最好使用本地压缩器。(如果本地库不存在，可以在Log看到很多*Got brand-new compressor*。参见Q:)

Appendix C. FAQ

.. 一般问题

Hbase还有其他的FAQs吗?

可以在Hbase的wiki HBase Wiki FAQ 和 Troubleshooting

HBase 支持 SQL吗?

不支持。可以通过Hive的SQL-ish来支持，该功能还在开
MapReduce的，对于低延迟的应用并不适合。参见Chap
客户端的例子。

HBase是如何工作在HDFS上的?

HDFS是一个为大文件设计的分布式文件系统。他的文
件系统，文件不支持快速的记录查找。另一方面，HBa
大表快速记录查找(更新)。这有时候会混淆概念。参见
Chapter 12, *架构*，来了解更多Hbase的目标。

为什么日志的最后一行是'2011-01-10 12:40:48,407 INFO org.apache.hadoop.io.compress.CodecPool: Got brand-new

因为我们没有使用本地的压缩类库。参见 HBASE-1900
hadoop 0.21 is released.将Hadoop的本地类库拷贝到Hbas

.. EC2

为什么我的连接EC2上的集群的远程Java连接不能工作?

根据用户列表，参见: Remote Java client connection into

3. 构建 HBase

当我build的时候，为什么遇到 Unable to find resource 'V

当我build的时候，为什么遇到 `Unable to find resour`

忽略他。这不是一个错误。这是officially ugly .

4. Runtime

为什么我在Hbase loading的是看到了停顿
为什么我的RegionServer会突然挂住？
为什么我看到RegionServer的数量是实际的两倍。一半

为什么我在Hbase loading的是看到了停顿

如果启用了压缩，参见用户列表 Long client pauses with

为什么我的RegionServer会突然挂住？

如果你使用了一个老式的JVM (< 1.6.0_u21?)?你可以看
BLOCKED但是没有一个hold着锁。参见 HBASE 3622 I
bug?). 在Hbase的`conf/hbase-env.sh`中的 `HBASE_OPTS`加

为什么我看到RegionServer的数量是实际的两倍。一半

修正你的DNS。在Hbase 0.92.x之前的版本，反向DNS和
HBASE 3431 Regionserver is not using the name given it b
listing of servers 获得详细信息.

5. 我如何在Hbase中建立

二级索引?

二级索引?

对于在Hbase中维护一个二级索引的问题，有一个用户
HBase, mail # user - Stargate+hbase的信息。

Appendix D. YCSB: 雅虎云服务 测试 和Hbase

TODO: YCSB不能很多的增加集群负载.

TODO: 如果给Hbase安装

Ted Dunning重做了YCSV,这个是用maven
管理了，加入了核实工作量的功能。参见
Ted Dunning's YCSB.

Index