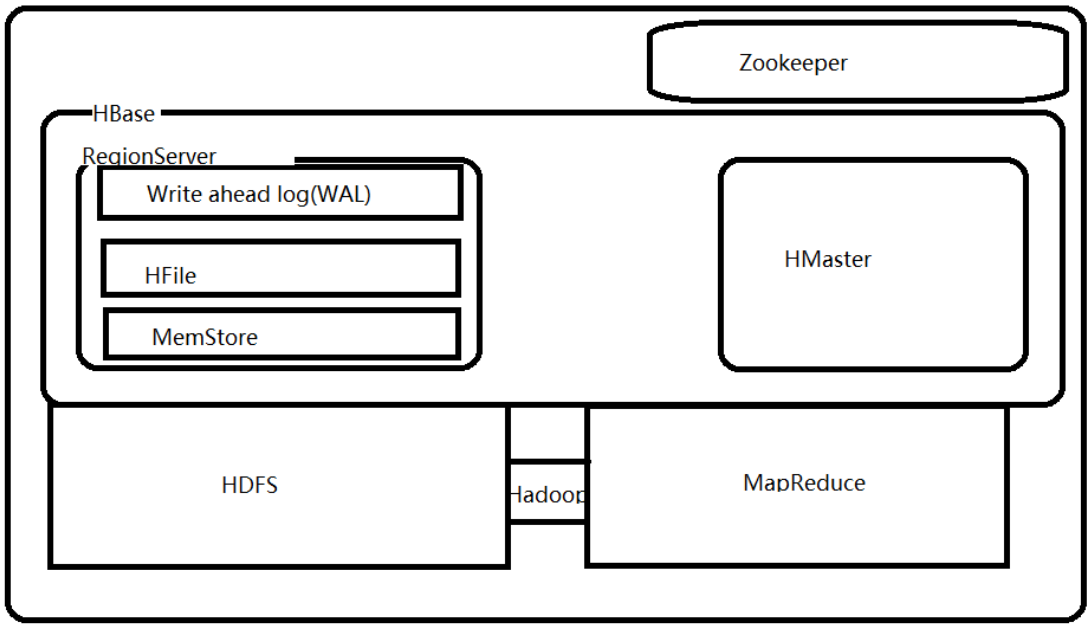
尚硅谷大数据技术之 HBase 基础

（作者：尽际）
官网：http://www.atguigu.com

一、HBase 的起源

HBase 的原型是 Google 的 BigTable 论文，受到了该论文思想的启发，目前作为 Hadoop 的子项目来开发维护，用于支持结构化的数据存储。
官方网站：http://hbase.apache.org
* 2006 年 Google 发表 BigTable 白皮书
* 2006 年开始开发 HBase
* 2008 年北京成功开奥运会，程序员默默地将 HBase 弄成了 Hadoop 的子项目
* 2010 年 HBase 成为 Apache 顶级项目
* 现在很多公司二次开发出了很多发行版本，你也开始使用了

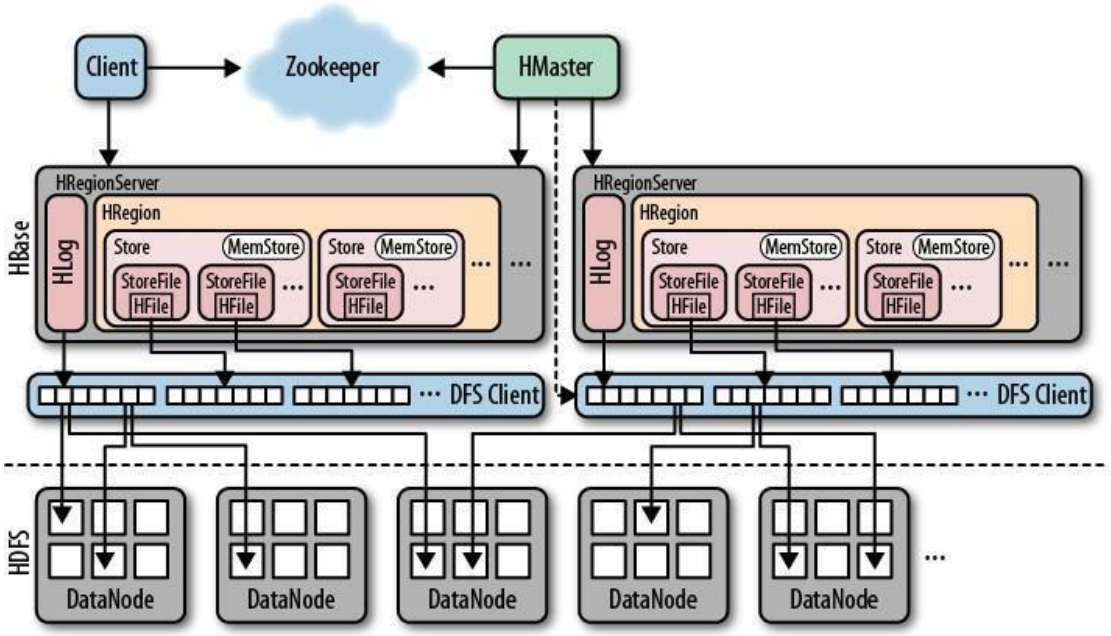
二、基于Hadoop的HBase架构



HBase 内置有 zookeeper，但一般我们会有其他的 Zookeeper 集群来监管 master 和 regionserver，Zookeeper 通过选举，保证任何时候，集群中只有一个活跃的 HMaster，HMaster 与 HRegionServer 启动时会向 ZooKeeper 注册，存储所有 HRegion 的寻址入口，实时监控 HRegionserver 的上线和下线信息。并实时通知给 HMaster，存储 HBase 的 schema 和 table 元数据，默认情况下，HBase 管理 ZooKeeper 实例，Zookeeper 的引入使得 HMaster 不再是单点故障。一般情况下会启动两个 HMaster，非 Active 的 HMaster 会定期的和 Active HMaster 通信以获取其最新状态，从而保证它是实时更新的，

因而如果启动了多个 HMaster 反而增加了 Active HMaster 的负担。

一个 RegionServer 可以包含多个 HRegion，每个 RegionServer 维护一个 HLog，和多个 HFiles 以及其对应的 MemStore。RegionServer 运行于 DataNode 上，数量可以与 DatNode 数量一致，请参考如下架构图：



三、RDBMS 与 HBase 的对比

3.1、关系型数据库

结构：

* 数据库以表的形式存在
* 支持 FAT、NTFS、EXT、文件系统
* 使用 Commit log 存储日志
* 参考系统是坐标系统
* 使用主键（PK）
* 支持分区
* 使用行、列、单元格

功能：

* 支持向上扩展
* 使用 SQL 查询
* 面向行，即每一行都是一个连续单元
* 数据总量依赖于服务器配置
* 具有 ACID 支持
* 适合结构化数据
* 传统关系型数据库一般都是中心化的

* 支持事务
* 支持 Join

3.2、HBase

结构：
* 数据库以 region 的形式存在
* 支持 HDFS 文件系统
* 使用 WAL（Write-Ahead Logs）存储日志
* 参考系统是 Zookeeper
* 使用行键（row key）
* 支持分片
* 使用行、列、列族和单元格

功能：
* 支持向外扩展
* 使用 API 和 MapReduce 来访问 HBase 表数据
* 面向列，即每一列都是一个连续的单元
* 数据总量不依赖具体某台机器，而取决于机器数量
* HBase 不支持 ACID（Atomicity、Consistency、Isolation、Durability）
* 适合结构化数据和非结构化数据
* 一般都是分布式的
* HBase 不支持事务
* 不支持 Join

四、HBase 特征简要

4.1、自动故障处理和负载均衡

HBase 运行在 HDFS 上，所以 HBase 中的数据以多副本形式存放，数据也服从分布式存放，数据的恢复也可以得到保障。另外，HMaster 和 RegionServer 也是多副本的。

4.2、自动分区

HBase 表是由分布在多个 RegionServer 中的 region 组成的，这些 RegionServer 又分布在不同的 DataNode 上，如果一个 region 增长到了一个阈值，为了负载均衡和减少 IO，HBase 可以自动或手动干预的将 region 切分为更小的 region，也称之为 subregion。

4.3、集成 Hadoop/HDFS

虽然 HBase 也可以运行在其他的分布式文件系统之上，但是与 HDFS 结合非常之方便，而且 HDFS 也非常之流行。

4.4、实时随机大数据访问

HBase 采用 log-structured merge-tree 作为内部数据存储架构，这种架构会周期性地将小文件合并成大文件以减少磁盘访问同时减少 NameNode 压力。

4.5、MapReduce

HBase 内建支持 MapReduce 框架，更加方便快速，并行的处理数据。Hbase 数据的增删改查以及 StoreFile 的合并都不依赖 MapReduce

4.6、Java API

HBase 提供原声的 Java API 支持，方便开发。

4.7、横向扩展

HBase 支持横向扩展，这就意味着如果现有服务器硬件性能出现瓶颈，不需要停掉现有集群提升硬件配置，而只需要在现有的正在运行的集群中添加新的机器节点即可，而且新的 RegionServer 一旦建立完毕，集群会开始重新调整。

4.8、列存储

HBase 是面向列存储的，每个列都单独存储，所以在 HBase 中列是连续存储的，而行不是。

4.9、HBase Shell

HBase 提供了交互式命令行工具可以进行创建表、添加数据、扫描数据、删除数据等操作和其他一些管理命令。

五、HBase 在集群中的定位

HBase 一种是作为存储的分布式文件系统，另一种是作为数据处理模型的 MR 框架。因为日常开发人员比较熟练的是结构化的数据进行处理，但是在 HDFS 直接存储的文件往往不具有结构化，所以催生出了 HBase 在 HDFS 上的操作。如果需要查询数据，只需要通过键值便可以成功访问。

六、HBase 内部存储架构

HBase 是由 row key，column family，column 和 cell 组成，row key 确定唯一的一行，column family 由若干 column 组成，column 是表的字段，cell 存储了实际的值或数据。

七、HBase 与 Hadoop

7.1、HDFS

* 为分布式存储提供文件系统
* 针对存储大尺寸的文件进行优化，不需要对 HDFS 上的文件进行随机读写
* 直接使用文件
* 数据模型不灵活
* 使用文件系统和处理框架
* 优化一次写入，多次读取的方式

7.2、HBase

* 提供表状的面向列的数据存储
* 针对表状数据的随机读写进行优化
* 使用 key-value 操作数据
* 提供灵活的数据模型
* 使用表状存储，支持 MapReduce，依赖 HDFS
* 优化了多次读，以及多次写

八、HBase的优缺点

8.1、优点

* 方便高效的压缩数据
* 支持快速数据检索
* 管理和配置简单，支持横向扩展，所以非常容易扩展
* 聚合查询性能非常高
* 可高效地进行分区，提供自动分区机制把大的 region 切分成小的 subregion

8.2、缺点

* 对 JOIN 以及多表合并数据的查询性能不好
* 更新过程中有大量的写入和删除操作，需要频繁合并和分裂，降低存储效率
* 对关系模型支持不好，分区和索引模式设计比较困难。

九、HBase的环境角色

9.1、HMaster

9.1.1、功能描述

* 监控 RegionServer
* 处理 RegionServer 故障转移
* 处理元数据的变更
* 处理 region 的分配或移除
* 在空闲时间进行数据的负载均衡
* 通过 Zookeeper 发布自己的位置给客户端

9.2、RegionServer

9.2.1、功能描述

* 负责存储 HBase 的实际数据
* 处理分配给它的 Region
* 刷新缓存到 HDFS
* 维护 HLog
* 执行压缩
* 负责处理 Region 分片

9.2.2、内含组件

*** Write-Ahead logs**
HBase 的修改记录，当对 HBase 读写数据的时候，数据不是直接写进磁盘，它会在内存中保留一段时间（时间以及数据量阈值可以设定）。如果机器突然原地爆炸，把数据保存在内存中会引起数据丢失，为了解决这个问题，数据会先写在一个叫做 Write-Ahead logfile 的文件中，然后再写入内存中。所以在系统出现故障的时候，数据可以通过这个日志文件重建。
*** HFile**
这是在磁盘上保存原始数据的实际的物理文件，是实际的存储文件。
*** Store**
HFile 存储在 Store 中，一个 Store 对应 HBase 表中的一个列族
*** MemStore**
顾名思义，就是内存存储，位于内存中，用来保存当前的数据操作，所以当数据保存在 WAL 中之后，RegsionServer 会在内存中存储键值对。
*** Region**

Hbase 表的分片，HBase 表会根据 RowKey 值被切分成不同的 region 存储在 RegionServer 中，在一个 RegionServer 中可以有多个不同的 region

9.3、Zookeeper

HMaster 与 HRegionServer 启动时会向 ZooKeeper 注册，存储所有 HRegion 的寻址入口，实时监控 HRegionserver 的上线和下线信息。并实时通知给 HMaster，存储 HBase 的 schema 和 table 元数据，默认情况下，HBase 管理 ZooKeeper 实例，Zookeeper 的引入使得 HMaster 不再是单点故障。一般情况下会启动两个 HMaster，非 Active 的 HMaster 会定期的和 Active HMaster 通信以获取其最新状态，从而保证它是实时更新的，因而如果启动了多个 HMaster 反而增加了 Active HMaster 的负担。

十、使用场景的探讨

10.1、何时使用

* 如果数据有很多列，且包含很多空字段
* 数据包含了不定数量的列
* 需要维护数据的版本
* 需要很高的横向扩展性
* 需要大量的压缩数据
* 需要大量的 I/O

一般而言数百万行的数据和频率不高的读写操作，是不需要 HBase 的，如果有几十亿列数据，同时在单位时间内有数以千、万记的读写操作，可以考虑 HBase。

10.2、何时不使用

* 数据总量不大时（比如就几个 G）
* 当需要 JOIN 以及关系型数据库的一些特性时
* 如果关系型数据库可以满足需求

十一、HBase 的安装与部署

10.1、Zookeeper 集群的正常部署并启动

```
$ /opt/modules/cdh/zookeeper-3.4.5-cdh5.3.6/bin/zkServer.sh start
```

10.2、Hadoop 集群的正常部署并启动

$ /opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/sbin/start-dfs.sh
$ /opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/sbin/start-yarn.sh

10.3、解压 HBase

$ tar -zxf /opt/softwares/hbase-0.98.6-cdh5.3.6.tar.gz -C /opt/modules/cdh/

10.4、修改 HBase 配置文件

10,.4.1、hbase-env.sh

```
# export JAVA_HOME=/usr/java/jdk1.6.0/
export JAVA_HOME=/opt/modules/jdk1.8.0_121
```

```
# export HBASE_MANAGES_ZK=true
export HBASE_MANAGES_ZK=false
```

10.4.2、hbase-site.xml

```
<configuration>
    <!-- 设置hbase的根地址，为NameNode所在位置-->
    <property>
        <name>hbase.rootdir</name>
        <value>hdfs://hadoop-senior01.itguigu.com:8020/hbase</value>
    </property>
    <!-- 使hbase运行于完全分布式-->
    <property>
        <name>hbase.cluster.distributed</name>
        <value>true</value>
    </property>
    <!-- HMaster的端口号-->
    <property>
        <name>hbase.master</name>
        <value>60000</value>
    </property>
    <!-- Zookeeper集群的地址列表，用逗号分割-->
    <property>
        <name>hbase.zookeeper.quorum</name>
        <value>hadoop-senior01.itguigu.com:2181,hadoop-senior02.itguigu.com:2181,hadoop-senior03.itguigu.com:2181</value>
    </property>
    <!-- zookeeper保存属性信息的文件，默认为/tmp 重启会丢失-->
    <property>
      <name>hbase.zookeeper.property.dataDir</name>
      <value>/opt/modules/cdh/zookeeper-3.4.5-cdh5.3.6/dataDir</value>
    </property>
</configuration>
```

10.4.3、regionservers

```
hadoop-senior01.itguigu.com
hadoop-senior02.itguigu.com
hadoop-senior03.itguigu.com
```

10.5、替换 HBase 根目录下的 lib 目录下的 jar 包，以解决兼容问题

* 删除原有 Jar 包

$ rm -rf /opt/modules/cdh/hbase-0.98.6-cdh5.3.6/lib/hadoop-*$ rm -rf lib/zookeeper-3.4.6.jar

（尖叫提示：如果 lib 目录下的 zookeeper 包不匹配也需要替换）

*** 拷贝新的 Jar 包**

这里涉及到的 jar 包大概是：

hadoop-annotations-2.5.0.jar
hadoop-auth-2.5.0-cdh5.3.6.jar
hadoop-client-2.5.0-cdh5.3.6.jar
hadoop-common-2.5.0-cdh5.3.6.jar
hadoop-hdfs-2.5.0-cdh5.3.6.jar
hadoop-mapreduce-client-app-2.5.0-cdh5.3.6.jar
hadoop-mapreduce-client-common-2.5.0-cdh5.3.6.jar
hadoop-mapreduce-client-core-2.5.0-cdh5.3.6.jar
hadoop-mapreduce-client-hs-2.5.0-cdh5.3.6.jar
hadoop-mapreduce-client-hs-plugins-2.5.0-cdh5.3.6.jar
hadoop-mapreduce-client-jobclient-2.5.0-cdh5.3.6.jar
hadoop-mapreduce-client-jobclient-2.5.0-cdh5.3.6-tests.jar
hadoop-mapreduce-client-shuffle-2.5.0-cdh5.3.6.jar
hadoop-yarn-api-2.5.0-cdh5.3.6.jar
hadoop-yarn-applications-distributedshell-2.5.0-cdh5.3.6.jar
hadoop-yarn-applications-unmanaged-am-launcher-2.5.0-cdh5.3.6.jar
hadoop-yarn-client-2.5.0-cdh5.3.6.jar
hadoop-yarn-common-2.5.0-cdh5.3.6.jar
hadoop-yarn-server-applicationhistoryservice-2.5.0-cdh5.3.6.jar
hadoop-yarn-server-common-2.5.0-cdh5.3.6.jar
hadoop-yarn-server-nodemanager-2.5.0-cdh5.3.6.jar
hadoop-yarn-server-resourcemanager-2.5.0-cdh5.3.6.jar
hadoop-yarn-server-tests-2.5.0-cdh5.3.6.jar
hadoop-yarn-server-web-proxy-2.5.0-cdh5.3.6.jar
zookeeper-3.4.5-cdh5.3.6.jar

我们可以通过 find 命令快速进行定位，例如我们可以执行：

$ find /opt/modules/ -name hadoop-hdfs-2.5.0-cdh5.3.6.jar

然后将查找出来的 Jar 包根据指定位置复制到 HBase 的 lib 目录下，在这里我给大家整合好到一个文件夹中了，请依次执行：

$ tar -zxf /opt/softwares/CDH_HadoopJar.tar.gz -C /opt/softwares/

$ cp -a /opt/softwares/HadoopJar/* /opt/modules/cdh/hbase-0.98.6-cdh5.3.6/lib/

10.6、将整理好的 HBase 安装目录 scp 到其他机器节点

$ scp -r /opt/modules/cdh/hbase-0.98.6-cdh5.3.6/ \
hadoop-senior02.itguigu.com:/opt/modules/cdh/

$ scp -r /opt/modules/cdh/hbase-0.98.6-cdh5.3.6/ \
hadoop-senior03.itguigu.com:/opt/modules/cdh/

10.7、将 Hadoop 配置文件软连接到 HBase 的 conf 目录下

*** core-site.xml**
$ ln -s /opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/etc/hadoop/core-site.xml
/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/conf/core-site.xml
*** hdfs-site.xml**
$ ln -s /opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/etc/hadoop/hdfs-site.xml
/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/conf/hdfs-site.xml
（尖叫提示：不要忘记其他几台机器也要做此操作）

10.8、启动服务

$ bin/hbase-daemon.sh start master

$ bin/hbase-daemon.sh start regionserver
或者：
$ bin/start-hbase.sh
对应的停止命令：
$ bin/stop-hbase.sh

10.9、查看页面

启动成功后，可以通过主机名:60010 地址来访问 HBase 的管理页面
例如，http://hadoop-senior01.itguigu.com:60010

十二、HBase常用操作

12.1、进入 HBase 客户端命令操作界面

$ bin/hbase shell

12.2、查看帮助命令

hbase(main):001:0> help

12.3、查看当前数据库中有哪些表

hbase(main):002:0> list

12.4、创建一张表

hbase(main):003:0> create 'student','info'

12.5、向表中存储一些数据

hbase(main):004:0> put 'student','1001','info:name','Thomas'

hbase(main):005:0> put 'student','1001','info:sex','male'

hbase(main):006:0> put 'student','1001','info:age','18'

12.6、扫描查看存储的数据

hbase(main):007:0> scan 'student'

```
hbase(main):007:0>  scan 'student'
ROW                                   COLUMN+CELL
 1001                                 column=info:age, timestamp=1501435124679, value=18
 1001                                 column=info:name, timestamp=1501435094613, value=Thomas
 1001                                 column=info:sex, timestamp=1501435108942, value=male
1 row(s) in 0.0510 seconds

hbase(main):008:0>
```

或：查看某个 rowkey 范围内的数据

hbase(main):014:0> scan 'student',{STARTROW => '1001',STOPROW => '1007'}

12.7、查看表结构

hbase(main):009:0> describe ‘student’

```
hbase(main):008:0> describe 'student'
DESCRIPTION                                                                                                  ENABLED
 'student', {NAME => 'info', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEEP_DELETED_CELLS => 'false',  true
 DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', COMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCK
 SIZE => '65536', REPLICATION_SCOPE => '0'}
1 row(s) in 0.0630 seconds

hbase(main):009:0>
```

12.8、更新指定字段的数据

hbase(main):009:0> put 'student','1001','info:name','Nick'

hbase(main):010:0> put 'student','1001','info:age','100'
查看更新后的数据：

```
hbase(main):011:0> scan 'student'
ROW                                           COLUMN+CELL
 1001                                         column=info:age, timestamp=1501435208305, value=100
 1001                                         column=info:name, timestamp=1501435204604, value=Nick
 1001                                         column=info:sex, timestamp=1501435108942, value=male
1 row(s) in 0.0370 seconds
```

12.9、查看指定行的数据

hbase(main):012:0> get 'student','1001'

或：查看指定行指定列或列族的数据
hbase(main):013:0> get 'student','1001','info:name'

12.10、删除数据

12.10.1、删除某一个 rowKey 全部的数据

hbase(main):015:0> deleteall 'student','1001'

12.10.2、删除掉某个 rowKey 中某一列的数据

hbase(main):016:0> delete 'student','1001','info:sex'

12.11、清空表数据

hbase(main):017:0> truncate 'student'

```
hbase(main):017:0> truncate 'student'
Truncating 'student' table (it may take a while):
 - Disabling table...
 - Dropping table...
 - Creating table...
0 row(s) in 2.9740 seconds
```

12.12、删除表

首先需要先让该表为 disable 状态，使用命令：
hbase(main):018:0> disable 'student'

然后才能 drop 这个表，使用命令：
hbase(main):019:0> drop 'student'

(尖叫提示：如果直接 drop 表，会报错：Drop the named table. Table must first be disabled)

12.13、统计一张表有多少行数据

hbase(main):020:0> count 'student'

十三、HMaster的高可用

13.1、确保 HBase 集群已正常停止

$ bin/stop-hbase.sh

13.2、在 conf 目录下创建 backup-masters 文件

$ touch conf/backup-masters

13.3、在 backup-masters 文件中配置高可用 HMaster 节点

$ echo hadoop-senior02.itguigu.com > conf/backup-masters

13.4、将整个 conf 目录 scp 到其他节点

$ scp -r conf/ hadoop-senior02.itguigu.com:/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/
$ scp -r conf/ hadoop-senior03.itguigu.com:/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/

13.5、打开页面测试

http://hadoop-senior01.itguigu.com:60010

Region Servers

Base Stats | Memory | Requests | Storefiles | Compactions

ServerName	Start time	Requests Per Second	Num. Regions
hadoop-senior01.itguigu.com,60020,1501435980878	Mon Jul 31 01:33:00 CST 2017	0	1
hadoop-senior02.itguigu.com,60020,1501435977384	Mon Jul 31 01:32:57 CST 2017	0	1
hadoop-senior03.itguigu.com,60020,1501435977923	Mon Jul 31 01:32:57 CST 2017	0	1
Total:3		0	3

Backup Masters

ServerName	Port	Start Time
hadoop-senior02.itguigu.com	60000	Mon Jul 31 01:33:02 CST 2017
Total:1		

最后，可以尝试关闭第一台机器的 HMaster：
$ bin/hbase-daemon.sh stop master
然后查看第二台的 HMaster 是否会直接启用

十四、HBase 和 Hadoop 的集群类型

14.1、单机模式

主要用于开发工作，一台机器上运行所有的守护进程，或者一台机器运行多个虚拟机。一般用于评估和测试。

14.2、小型集群

20 台机器以内的集群，不同的机器运行不同的守护进程，适用于数据量和处理请求较少的小型生产环境。

14.3、中型集群

20 到 1000 台机器集群，3 到 5 个 zookeeper 节点，适用于成熟的生产环境。

14.4、大型集群

1000 台机器以上的集群，属于超大规模集群了，适用于大规模生产环境。

十五、集群配置举例

15.1、NameNode/HMaster 常见配置

内存：16~128G
CPU：2*（8~24）核处理器
硬盘：1TB-SATA 硬盘+1 个元数据备份盘（转速 7200R/MIN+）能使用固态更好。
网卡：2*1GB 网卡
为了有更好的性能，所有的元数据都缓存在内存中，因此内存需要拥有较快的速度和较好的质量。大内存意为着可以存储更多的文件，从而支持 NameNode 更大的命名空间。同时 NameNode 不需要很大的磁盘，小容量的磁盘就可以满足需求，元数据要存储加载到内存中，数据副本以及修改日志存储在磁盘上。

15.2、ResourceManager

可以运行在 NameNode 机器上，也可以运行在单独的机器上。硬件配置和 NameNode 一直，因为只是用于作业分发，因此不需要较大的磁盘和较强的运算能力。

15.3、DataNode、RegionServer

实际的数据存储于这些节点，因此这些节点需要较大的存储和较强的运算能力。较小的集群可以使用一般的磁盘，内存和 CPU，如果集群规模较大，可以考虑：
内存：16~128G
CPU：2*（8~24）核处理器
硬盘：2TB，转速 7200
网卡：2*1GB

十六、CDH配置

备选资源：
内存：64~512GB
硬盘：1TB~4TB
CPU：2*（8~24）核 CPU，主频 2~2.5GHZ
网卡：千、万兆以太网

16.1、CPU

工作负载核心，推荐 DataNode 配置为双 CPU 插槽，配置中等主频的 CPU，高端 CPU 太烧钱，所以我们可以增加数量。

16.2、电源

耐热性，稳定。

16.3、内存

需要足量的内存以保证不需要等待数据频繁的装载到内存中，因此 8~48G 内存比较合适，HBase 会使用大量的内存，将文件存放在内存中（如果开启了内存表的话），对于 HBase 集群，我们需要比单独的 Hadoop 集群更大的内存。如果 HBase 开启缓存，Hbase 会尝试将整张表缓存在内存中。

16.4、磁盘

不建议在某台机器上配置很大容量的磁盘，这样当这台机器出现问题，不容易将数据分散到其他机器节点中。必须不能低于 SATA 7200 转

16.5、网络

Hadoop 或者 HBase 在执行任务，读取数据和写入数据时，会在节点之间传输数据块，因此建议配置高速的网络和交换机。对于中小集群，1GB/s 的网络足矣。对于排序和 shuffle 这类操作，需要节点间传输大量数据，如果带宽不足，会导致一些节点连接超时，比如 RegionServer、Zookeeper。

十七、容量规划

运算公式：T = （S* R）* 1.25
尖叫提示：
S 表示存储数据量
R 表示副本数
T 表示整个集群需要的空间

十八、HBase 读写流程

18.1、HBase 读数据流程

HRegionServer 保存着 meta 表以及表数据，要访问表数据，首先 Client 先去访问 zookeeper，从 zookeeper 里面获取 meta 表所在的位置信息，即找到这个 meta 表在哪个 HRegionServer 上保存着。

接着 Client 通过刚才获取到的 HRegionServer 的 IP 来访问 Meta 表所在的 HRegionServer，从而读取到 Meta，进而获取到 Meta 表中存放的元数据。

Client 通过元数据中存储的信息，访问对应的 HRegionServer，然后扫描所在 HRegionServer 的 Memstore 和 Storefile 来查询数据。

最后 HRegionServer 把查询到的数据响应给 Client。

18.2、HBase 写数据流程

Client 也是先访问 zookeeper，找到 Meta 表，并获取 Meta 表元数据。

确定当前将要写入的数据所对应的 HRegion 和 HRegionServer 服务器。

Client 向该 HRegionServer 服务器发起写入数据请求，然后 HRegionServer 收到请求并响应。

Client 先把数据写入到 HLog，以防止数据丢失。

然后将数据写入到 Memstore。

如果 HLog 和 Memstore 均写入成功，则这条数据写入成功

如果 Memstore 达到阈值，会把 Memstore 中的数据 flush 到 Storefile 中。

当 Storefile 越来越多，会触发 Compact 合并操作，把过多的 Storefile 合并成一个大的 Storefile。

当 Storefile 越来越大，Region 也会越来越大，达到阈值后，会触发 Split 操作，将 Region 一分为二。

十九、HBase中的3个重要机制

19.1、flush 机制

当 MemStore 达到阈值，将 Memstore 中的数据 Flush 进 Storefile
涉及属性：
hbase.hregion.memstore.flush.size：134217728
即：128M 就是 Memstore 的默认阈值

hbase.regionserver.global.memstore.upperLimit：0.4
即：这个参数的作用是当单个 HRegion 内所有的 Memstore 大小总和超过指定值时，flush 该 HRegion 的所有 memstore。RegionServer 的 flush 是通过将请求添加一个队列，模拟生产消费模式来异步处理的。那这里就有一个问题，当队列来不及消费，产生大量积压请求时，可能会导致内存陡增，最坏的情况是触发 OOM。

hbase.regionserver.global.memstore.lowerLimit：0.38
即：当 MemStore 使用内存总量达到 hbase.regionserver.global.memstore.upperLimit 指定值时，将会有多个 MemStores flush 到文件中，MemStore flush 顺序是按照大小降序执行的，直到刷新到 MemStore 使用内存略小于 lowerLimit

19.2、compact 机制

把小的 Memstore 文件合并成大的 Storefile 文件。

19.3、split 机制

当 Region 达到阈值，会把过大的 Region 一分为二。

二十、HBaseAPI 的使用

20.1、解压 Maven 离线仓库到指定目录

$ tar -zxf /opt/softwares/hbase+hadoop_repository.tar.gz -C ~/.m2/

20.2、新建 Eclipse 的 Maven Project，添加 pom.xml 的 dependency 如下：

```
<dependency>
    <groupId>org.apache.hbase</groupId>
    <artifactId>hbase-server</artifactId>
    <version>0.98.6-hadoop2</version>
</dependency>

<dependency>
    <groupId>org.apache.hbase</groupId>
    <artifactId>hbase-client</artifactId>
    <version>0.98.6-hadoop2</version>
</dependency>
```

20.3、编写 HBaseAPI 代码

详见项目代码

二十一、文件格式的说明

21.1、tsv 格式的文件：字段之间以制表符\t 分割

21.2、csv 格式的文件：字段之间以逗号,分割

二十二、HBase的MapReduce的调用

22.1、查看 HBase 执行 MapReduce 所依赖的 Jar 包

执行命令：
$ bin/hbase mapredcp
出现如下内容：

```
/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/lib/hbase-common-0.98.6-cdh5.3.6.jar:
/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/lib/protobuf-java-2.5.0.jar:
/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/lib/hbase-client-0.98.6-cdh5.3.6.jar:
/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/lib/hbase-hadoop-compat-0.98.6-cdh5.3.6.jar:
/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/lib/hbase-protocol-0.98.6-cdh5.3.6.jar:
/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/lib/high-scale-lib-1.1.1.jar:
/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/lib/zookeeper-3.4.5-cdh5.3.6.jar:
/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/lib/guava-12.0.1.jar:
/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/lib/htrace-core-2.04.jar:
/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/lib/netty-3.6.6.Final.jar
```

22.2、执行环境变量导入

$ export HBASE_HOME=/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/
$ export HADOOP_HOME=/opt/modules/cdh/hadoop-2.5.0-cdh5.3.6
$ export HADOOP_CLASSPATH=`${HBASE_HOME}/bin/hbase mapredcp`

22.3、运行官方的 MapReduce 任务

22.3.1、案例一：统计 student 表中有多少行数据

* 执行代码
$ /opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/bin/yarn jar lib/hbase-server-0.98.6-cdh5.3.6.jar rowcounter student

22.3.2、案例二：使用 MapReduce 任务将数据从文件中导入到 HBase

Step1、创建一个 tsv 格式的文件
$ vi fruit.tsv，内容如下：

```
1001    Apple    Red
1002    Pear    Yellow
1003    Pineapple    Yellow
```

Step2、创建 HBase 表
$ bin/hbase shell
hbase(main):001:0> create 'fruit','info'

Step3、在 HDFS 中创建 input_fruit 文件夹并上传 fruit.tsv 文件

```
$ /opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/bin/hdfs dfs -mkdir /input_fruit/
$ /opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/bin/hdfs dfs -put fruit.tsv /input_fruit/
```

Step4、执行 MapReduce 到 HBase 的 fruit 表中

```
$ /opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/bin/yarn jar lib/hbase-server-0.98.6-
cdh5.3.6.jar
importtsv -Dimporttsv.columns=HBASE_ROW_KEY,info:name,info:color fruit \
hdfs://hadoop-senior01.itguigu.com:8020/input_fruit
```

Step5、使用 scan 命令查看导入后的数据即可

二十三、BulkLoad加载文件到HBase表

23.1、功能

将本地数据导入到 HBase 中

23.2、原理

BulkLoad 会将 tsv/csv 格式的文件编程 hfile 文件，然后再进行数据的导入，这样可以避免大量数据导入时造成的集群写入压力过大。

23.3、作用

* 减小 HBase 集群插入数据的压力
* 提高了 Job 运行的速度，降低了 Job 执行时间

23.4、案例

Step1、配置临时环境变量

```
$ export HBASE_HOME=/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/
$ export HADOOP_HOME=/opt/modules/cdh/hadoop-2.5.0-cdh5.3.6
$ export HADOOP_CLASSPATH=`${HBASE_HOME}/bin/hbase mapredcp`
```

Step2、创建一个新的 HBase 表

```
$ bin/hbase shell
```

hbase(main):001:0> create 'fruit_bulkload','info'

Step3、将 tsv/csv 文件转化为 HFile （别忘了要确保你的 fruit 格式的文件 fruit.tsv 在 input 目录下）

$ /opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/bin/yarn jar \

/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/lib/hbase-server-0.98.6-cdh5.3.6.jar importtsv \

-Dimporttsv.bulk.output=/output_file \

-Dimporttsv.columns=HBASE_ROW_KEY,info:name,info:color \

fruit hdfs://hadoop-senior01.itguigu.com:8020/input_fruit

Step4、把 HFile 导入到 HBase 表 fruit_bulkload

上一步完成之后，你会发现在 HDFS 的根目录下出现了一个 output_file 文件夹，里面存放的就是 HFile 文件，紧接着：把 HFile 导入到 HBase 表 fruit_bulkload
$ /opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/bin/yarn jar \

/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/lib/hbase-server-0.98.6-cdh5.3.6.jar \

completebulkload /output_file fruit_bulkload

Step5、查看使用 bulkLoad 方式导入的数据

hbase(main):001:0> scan ‘fruit_bulkload’

二十四、HBase自定义MapReduce

案例 1、HBase 表数据的转移

在 Hadoop 阶段，我们编写的 MR 任务分别进程了 Mapper 和 Reducer 两个类，而在 HBase 中我们需要继承的是 TableMapper 和 TableReducer 两个类。

目标：将 fruit 表中的一部分数据，通过 MR 迁入到 fruit_mr 表中

Step1、构建 ReadFruitMapper 类，用于读取 fruit 表中的数据

package com.z.hbase_mr;

```
import java.io.IOException;

import org.apache.hadoop.hbase.Cell;
import org.apache.hadoop.hbase.CellUtil;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.io.ImmutableBytesWritable;
import org.apache.hadoop.hbase.mapreduce.TableMapper;
import org.apache.hadoop.hbase.util.Bytes;

public class ReadFruitMapper extends TableMapper<ImmutableBytesWritable, Put> {

    @Override
    protected void map(ImmutableBytesWritable key, Result value, Context context)
    throws IOException, InterruptedException {
        //将 fruit 的 name 和 color 提取出来，相当于将每一行数据读取出来放入到 Put 对象中。
        Put put = new Put(key.get());
        //遍历添加 column 行
        for(Cell cell: value.rawCells()){
            //添加/克隆列族:info
            if("info".equals(Bytes.toString(CellUtil.cloneFamily(cell)))){
                //添加/克隆列：name
                if("name".equals(Bytes.toString(CellUtil.cloneQualifier(cell)))){
                    //将该列 cell 加入到 put 对象中
                    put.add(cell);
                    //添加/克隆列:color
                }else if("color".equals(Bytes.toString(CellUtil.cloneQualifier(cell)))){
                    //向该列 cell 加入到 put 对象中
                    put.add(cell);
                }
            }
        }
        //将从 fruit 读取到的每行数据写入到 context 中作为 map 的输出
        context.write(key, put);
    }

}
```

Step2、构建 WriteFruitMRReducer 类，用于将读取到的 fruit 表中的数据写入到 fruit_mr 表中

```
package com.z.hbase_mr;
```

```
import java.io.IOException;

import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.io.ImmutableBytesWritable;
import org.apache.hadoop.hbase.mapreduce.TableReducer;
import org.apache.hadoop.io.NullWritable;

public class WriteFruitMRReducer extends TableReducer<ImmutableBytesWritable, Put, NullWritable> {

    @Override
    protected void reduce(ImmutableBytesWritable key, Iterable<Put> values, Context context)
    throws IOException, InterruptedException {
        //读出来的每一行数据写入到 fruit_mr 表中
        for(Put put: values){
            context.write(NullWritable.get(), put);
        }
    }


}
```

Step3、构建 Fruit2FruitMRJob extends Configured implements Tool，用于组装运行 Job 任务

```
    //组装 Job
    public int run(String[] args) throws Exception {
        //得到 Configuration
        Configuration conf = this.getConf();
        //创建 Job 任务
        Job job = Job.getInstance(conf, this.getClass().getSimpleName());
        job.setJarByClass(Fruit2FruitMRJob.class);

        //配置 Job
        Scan scan = new Scan();
        scan.setCacheBlocks(false);
        scan.setCaching(500);

        //设置 Mapper，注意导入的是 mapreduce 包下的，不是 mapred 包下的，后者是老版本
        TableMapReduceUtil.initTableMapperJob(
        "fruit", //数据源的表名
        scan, //scan 扫描控制器
        ReadFruitMapper.class,//设置 Mapper 类
```

```
            ImmutableBytesWritable.class,//设置 Mapper 输出 key 类型
            Put.class,//设置 Mapper 输出 value 值类型
            job//设置给哪个 JOB
            );

            //设置 Reducer
            TableMapReduceUtil.initTableReducerJob("fruit_mr",      WriteFruitMRReducer.class,
job);

            //设置 Reduce 数量，最少 1 个
            job.setNumReduceTasks(1);

            boolean isSuccess = job.waitForCompletion(true);
            if(!isSuccess){
                  throw new IOException("Job running with error");
            }

            return isSuccess ? 0 : 1;
      }
```

Step4、主函数中调用运行该 Job 任务

```
public static void main( String[] args ) throws Exception{
      Configuration conf = HBaseConfiguration.create();
      int status = ToolRunner.run(conf, new Fruit2FruitMRJob(), args);
      System.exit(status);
}
```

案例 2、将文件中的数据导入到 HBase 数据表

其实该案例的思想和案例 1 没有太大不同，思路总体还是一样的，只不过这次 Mapper 不是从 HBase 的表里读取数据了，而是从 HDFS 上的文件中读取数据，所以 Mapper 可直接继承自 HDFS 的 Mapper。

Step1、构建 Mapper 用于读取 HDFS 中的文件数据

```
package com.z.hbase.mr2;

import java.io.IOException;

import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.io.ImmutableBytesWritable;
import org.apache.hadoop.hbase.util.Bytes;
```

```
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class ReadFruitFromHDFSMapper extends Mapper<LongWritable, Text, ImmutableBytesWritable, Put> {
    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        //从 HDFS 中读取的数据
        String lineValue = value.toString();
        //读取出来的每行数据使用\t 进行分割，存于 String 数组
        String[] values = lineValue.split("\t");

        //根据数据中值的含义取值
        String rowKey = values[0];
        String name = values[1];
        String color = values[2];

        //初始化 rowKey
        ImmutableBytesWritable rowKeyWritable = new ImmutableBytesWritable(Bytes.toBytes(rowKey));

        //初始化 put 对象
        Put put = new Put(Bytes.toBytes(rowKey));

        //参数分别:列族、列、值
        put.add(Bytes.toBytes("info"), Bytes.toBytes("name"), Bytes.toBytes(name));
        put.add(Bytes.toBytes("info"), Bytes.toBytes("color"), Bytes.toBytes(color));

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

Step2、构建 WriteFruitMRFromTxtReducer 类

```
package com.z.hbase.mr2;

import java.io.IOException;

import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.io.ImmutableBytesWritable;
import org.apache.hadoop.hbase.mapreduce.TableReducer;
import org.apache.hadoop.io.NullWritable;
```

```
public class WriteFruitMRFromTxtReducer extends TableReducer<ImmutableBytesWritable,
Put, NullWritable> {
    @Override
    protected void reduce(ImmutableBytesWritable key, Iterable<Put> values, Context
context) throws IOException, InterruptedException {
        //读出来的每一行数据写入到 fruit_hdfs 表中
        for(Put put: values){
            context.write(NullWritable.get(), put);
        }
    }

}
```

Step3、组装 Job

```
public int run(String[] args) throws Exception {
        //得到 Configuration
        Configuration conf = this.getConf();

        //创建 Job 任务
        Job job = Job.getInstance(conf, this.getClass().getSimpleName());
        job.setJarByClass(HDFS2HBaseDriver.class);
//      Path inPath = new Path("/input/fruit.txt");
        Path inPath = new Path("hdfs://hadoop-
senior01.itguigu.com:8020/input_fruit/fruit.tsv");
        FileInputFormat.addInputPath(job, inPath);

        //设置 Mapper
        job.setMapperClass(ReadFruitFromHDFSMapper.class);
        job.setMapOutputKeyClass(ImmutableBytesWritable.class);
        job.setMapOutputValueClass(Put.class);

        //设置 Reducer
        TableMapReduceUtil.initTableReducerJob("fruit_hdfs",
WriteFruitMRFromTxtReducer.class, job);

        //设置 Reduce 数量，最少 1 个
        job.setNumReduceTasks(1);

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

```
        return isSuccess ? 0 : 1;
    }
```

Step4、提交运行 Job

```
    public static void main(String[] args) throws Exception {
        Configuration conf = HBaseConfiguration.create();

        int status = ToolRunner.run(conf, new HDFS2HBaseDriver(), args);
        System.exit(status);
    }
```

二十五、HBase与Hive 的对比

25.1、Hive

25.1.1、数据仓库

Hive 的本质其实就相当于将 HDFS 中已经存储的文件在 Mysql 中做了一个双射关系，以方便使用 HQL 去管理查询。

25.1.2、用于数据分析、清洗

Hive 适用于离线的数据分析和清洗，延迟较高

25.1.3、基于 HDFS、MapReduce

Hive 存储的数据依旧在 DataNode 上，编写的 HQL 语句终将是转换为 MapReduce 代码执行。（不要钻不需要执行 MapReduce 代码的情况的牛角尖）

25.2、HBase

25.2.1、数据库

是一种面向列存储的非关系型数据库。

25.2.2、用于存储结构化和非结构话的数据

适用于单表非关系型数据的存储，不适合做关联查询，类似 JOIN 等操作。

25.2.3、基于 HDFS

数据持久化存储的体现形式是 Hfile，存放于 DataNode 中，被 ResionServer 以 region 的形式进行管理。

25.2.4、延迟较低，接入在线业务使用

面对大量的企业数据，HBase 可以直线单表大量数据的存储，同时提供了高效的数据访问速度。

总结：Hive 与 HBase

Hive 和 Hbase 是两种基于 Hadoop 的不同技术，Hive 是一种类 SQL 的引擎，并且运行 MapReduce 任务，Hbase 是一种在 Hadoop 之上的 NoSQL 的 Key/vale 数据库。这两种工具是可以同时使用的。就像用 Google 来搜索，用 FaceBook 进行社交一样，Hive 可以用来进行统计查询，HBase 可以用来进行实时查询，数据也可以从 Hive 写到 HBase，或者从 HBase 写回 Hive。

二十六、HBase与Hive交互操作

26.1、环境准备

因为我们后续可能会在操作 Hive 的同时对 HBase 也会产生影响，所以 Hive 需要持有操作 HBase 的 Jar，那么接下来拷贝 Hive 所依赖的 Jar 包（或者使用软连接的形式）。

```
$ export HBASE_HOME=/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/

$ export HIVE_HOME=/opt/modules/cdh/hive-0.13.1-cdh5.3.6/

$ ln -s $HBASE_HOME/lib/hbase-common-0.98.6-cdh5.3.6.jar  $HIVE_HOME/lib/hbase-common-0.98.6-cdh5.3.6.jar

$ ln -s $HBASE_HOME/lib/hbase-server-0.98.6-cdh5.3.6.jar $HIVE_HOME/lib/hbase-server-0.98.6-cdh5.3.6.jar

$ ln -s $HBASE_HOME/lib/hbase-client-0.98.6-cdh5.3.6.jar $HIVE_HOME/lib/hbase-client-0.98.6-cdh5.3.6.jar

$ ln -s $HBASE_HOME/lib/hbase-protocol-0.98.6-cdh5.3.6.jar $HIVE_HOME/lib/hbase-protocol-0.98.6-cdh5.3.6.jar
```

$ ln -s $HBASE_HOME/lib/hbase-it-0.98.6-cdh5.3.6.jar $HIVE_HOME/lib/hbase-it-0.98.6-cdh5.3.6.jar

$ ln -s $HBASE_HOME/lib/htrace-core-2.04.jar $HIVE_HOME/lib/htrace-core-2.04.jar

$ ln -s $HBASE_HOME/lib/hbase-hadoop2-compat-0.98.6-cdh5.3.6.jar $HIVE_HOME/lib/hbase-hadoop2-compat-0.98.6-cdh5.3.6.jar

$ ln -s $HBASE_HOME/lib/hbase-hadoop-compat-0.98.6-cdh5.3.6.jar $HIVE_HOME/lib/hbase-hadoop-compat-0.98.6-cdh5.3.6.jar

$ ln -s $HBASE_HOME/lib/high-scale-lib-1.1.1.jar $HIVE_HOME/lib/high-scale-lib-1.1.1.jar

同时在 **hive-site.xml** 中修改 zookeeper 的属性，如下：

```
<property>
  <name>hive.zookeeper.quorum</name>
  <value>hadoop-senior01.itguigu.com,hadoop-senior02.itguigu.com,hadoop-senior03.itguigu.com</value>
  <description>The list of ZooKeeper servers to talk to. This is only needed for read/write locks.</description>
</property>
```

26.2、案例 1：创建 Hive 表，关联 HBase 表，插入数据到 Hive 表的同时能够影响 HBase

Step1、在 Hive 中创建表同时关联 HBase

```
CREATE TABLE hive_hbase_emp_table(
empno int,
ename string,
job string,
mgr int,
hiredate string,
sal double,
comm double,
deptno int)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" =
":key,info:ename,info:job,info:mgr,info:hiredate,info:sal,info:comm,info:deptno")
TBLPROPERTIES ("hbase.table.name" = "hbase_emp_table");
```

（尖叫提示：完成之后，可以分别进入 Hive 和 HBase 查看，都生成了对应的表）

Step2、在 Hive 中创建临时中间表，用于 load 文件中的数据（注：不能将数据直接 load 进 Hive 所关联 HBase 的那张表中）。

```
CREATE TABLE emp(
```

empno int,
ename string,
job string,
mgr int,
hiredate string,
sal double,
comm double,
deptno int)
row format delimited fields terminated by '\t';

Step3、向 Hive 中间表中 load 数据

hive> load data local inpath '/home/admin/Desktop/emp.txt' into table emp;

Step4、通过 insert 命令将中间表中的数据导入到 Hive 关联 HBase 的那张表中

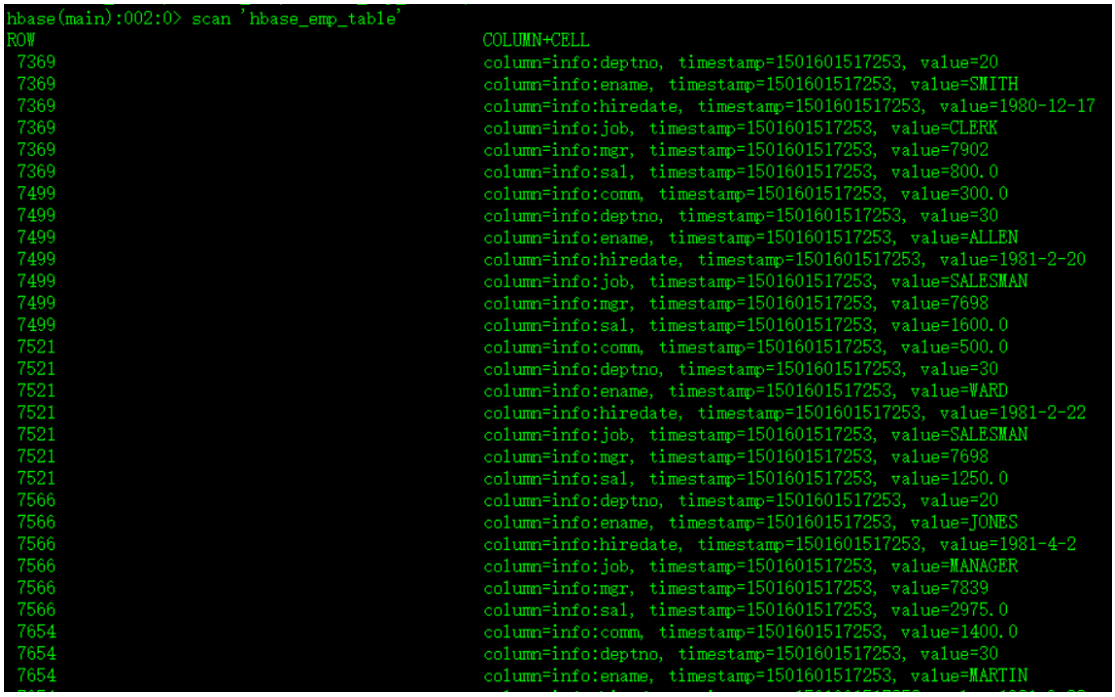
hive> insert into table hive_hbase_emp_table select * from emp;

Step5、测试，查看 Hive 以及关联的 HBase 表中是否已经成功的同步插入了数据

如图所示：
Hive 中：

```
hive (default)> select * from hive_hbase_emp_table;
OK
hive_hbase_emp_table.empno      hive_hbase_emp_table.ename      hive_hbase_emp_table.job        hi
ive_hbase_emp_table.comm        hive_hbase_emp_table.deptno
7369    SMITH   CLERK   7902    1980-12-17      800.0   NULL    20
7499    ALLEN   SALESMAN        7698    1981-2-20       1600.0  300.0   30
7521    WARD    SALESMAN        7698    1981-2-22       1250.0  500.0   30
7566    JONES   MANAGER 7839    1981-4-2        2975.0  NULL    20
7654    MARTIN  SALESMAN        7698    1981-9-28       1250.0  1400.0  30
7698    BLAKE   MANAGER 7839    1981-5-1        2850.0  NULL    30
7782    CLARK   MANAGER 7839    1981-6-9        2450.0  NULL    10
7788    SCOTT   ANALYST 7566    1987-4-19       3000.0  NULL    20
7839    KING    PRESIDENT       NULL    1981-11-17      5000.0  NULL    10
7844    TURNER  SALESMAN        7698    1981-9-8        1500.0  0.0     30
7876    ADAMS   CLERK   7788    1987-5-23       1100.0  NULL    20
7900    JAMES   CLERK   7698    1981-12-3       950.0   NULL    30
7902    FORD    ANALYST 7566    1981-12-3       3000.0  NULL    20
7934    MILLER  CLERK   7782    1982-1-23       1300.0  NULL    10
Time taken: 0.127 seconds, Fetched: 14 row(s)
hive (default)>
```

HBase 中：



26.3、案例 2：比如在 HBase 中已经存储了某一张表 hbase_emp_table，然后在 Hive 中创建一个外部表来关联 HBase 中的 hbase_emp_table 这张表，使之可以借助 Hive 来分析 HBase 这张表中的数据。

该案例 2 紧跟案例 1 的脚步，所以完成此案例前，请先完成案例 1。

Step1、在 Hive 中创建外部表

```
CREATE EXTERNAL TABLE relevance_hbase_emp(
empno int,
ename string,
job string,
mgr int,
hiredate string,
sal double,
comm double,
deptno int)
STORED BY
'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" =
":key,info:ename,info:job,info:mgr,info:hiredate,info:sal,info:comm,info:deptno")
TBLPROPERTIES ("hbase.table.name" = "hbase_emp_table");
```

Step2、关联后就可以使用 Hive 函数进行一些分析操作了

在此，我们查询一下所有数据试试看
hive (default)> select * from relevance_hbase_emp;

二十七、HBase 与 Sqoop 集成

回顾之前 Sqoop 案例：
1、 RDBMS 到 HDFS 的数据导入
2、 RDBMS 到 Hive 的数据导入
3、 Hive/HDFS 到 RDBMS 的数据导出
今天我们来讨论一下如何使用 Sqoop 将 RDBMS 中的数据导入到 HBase 当中。

27.1、案例：将 RDBMS 中的数据抽取到 HBase 中

Step1、配置 sqoop-env.sh 如下：

```
#Set path to where bin/hadoop is available
#export HADOOP_COMMON_HOME=
export HADOOP_COMMON_HOME=/opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/

#Set path to where hadoop-*-core.jar is available
#export HADOOP_MAPRED_HOME=
export HADOOP_MAPRED_HOME=/opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/

#set the path to where bin/hbase is available
#export HBASE_HOME=
export HBASE_HOME=/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/

#Set the path to where bin/hive is available
#export HIVE_HOME=
export HIVE_HOME=/opt/modules/cdh/hive-0.13.1-cdh5.3.6/

#Set the path for where zookeper config dir is
#export ZOOCFGDIR=
export ZOOCFGDIR=/opt/modules/cdh/zookeeper-3.4.5-cdh5.3.6/conf
export ZOOKEEPER_HOME=/opt/modules/cdh/zookeeper-3.4.5-cdh5.3.6/
```

Step2、在 Mysql 中创建一张数据库 library，一张表 book

CREATE DATABASE library;
CREATE TABLE book(
id int(4) PRIMARY KEY NOT NULL AUTO_INCREMENT,
name VARCHAR(255) NOT NULL,

price VARCHAR(255) NOT NULL);

Step3、向表中插入一些数据

INSERT INTO book(name, price) VALUES('Lie Sporting', '30');
INSERT INTO book (name, price) VALUES('Pride & Prejudice', '70');
INSERT INTO book (name, price) VALUES('Fall of Giants', '50');
完成后如图：

```
mysql> select  * from book;
+----+-------------------+-------+
| id | name              | price |
+----+-------------------+-------+
|  1 | Lie Sporting      | 30    |
|  2 | Pride & Prejudice | 70    |
|  3 | Fall of Giants    | 50    |
+----+-------------------+-------+
3 rows in set (0.00 sec)

mysql>
```

Step4、执行 Sqoop 导入数据的操作

```
$ bin/sqoop import \
--connect jdbc:mysql://hadoop-senior01.itguigu.com:3306/db_library \
--username root \
--password 123456 \
--table book \
--columns "id,name,price" \
--column-family "info" \
--hbase-create-table \
--hbase-row-key "id" \
--hbase-table "hbase_book" \
--num-mappers 1  \
--split-by id
```

Step5、在 HBase 中 scan 这张表得到如下内容

```
hbase(main):065:0> scan 'hbase_book'
ROW                                     COLUMN+CELL
 1                                      column=info:name, timestamp=1501607877641, value=Lie Sporting
 1                                      column=info:price, timestamp=1501607877641, value=30
 2                                      column=info:name, timestamp=1501607877641, value=Pride & Prejudice
 2                                      column=info:price, timestamp=1501607877641, value=70
 3                                      column=info:name, timestamp=1501607877641, value=Fall of Giants
 3                                      column=info:price, timestamp=1501607877641, value=50
3 row(s) in 0.0350 seconds

hbase(main):066:0>
```

（尖叫提示：在导入之前，HBase 中的表如果不存在则会自动创建）

二十八、HBase中计算存储数据的大小

28.1、固定大小 fixed size

即，当前表设计时，预算的字段名所占用的空间，比如：
RowKey：6 个字节
Value：4 个字节
TimeStamp：9 个字节
KeyType：1 个字节
CF：1 个字节
那么固定大小 fixed size = 6 + 4 + 9 + 1 + 1 = 21 字节
当然这只是一个近似估计。

28.2、可变大小 variable size

即，存入的数据的具体指可能会产生波动，比如全世界人类表，俄罗斯的人名比较长，从早上读到晚上都没有念完，中国人名短，这就造成了存入的数据可能是一个可变的大小。
那么总的数据量估算就是：total = fixed size + variable size

二十九、HBase Shell

29.1、status

例如：显示服务器状态
hbase> status ‘hadoop-senior01.itguigu.com’

29.2、whoami

显示 HBase 当前用户，例如：
hbase> whoami

29.3、list

显示当前所有的表

29.4、count

统计指定表的记录数，例如：
hbase> count 'hbase_book'

29.5、describe

展示表结构信息

29.6、exist

检查表是否存在，适用于表量特别多的情况

29.7、is_enabled、is_disabled

检查表是否启用或禁用

29.8、alter

该命令可以改变表和列族的模式，例如：
为当前表增加列族：
hbase> alter 'hbase_book', NAME => 'CF2', VERSIONS => 2
为当前表删除列族：
hbase> alter 'hbase_book', 'delete' => 'CF2'

29.9、disable

禁用一张表

29.10、drop

删除一张表，记得在删除表之前必须先禁用

29.11、delete

删除一行中一个单元格的值，例如：
hbase> delete ‘hbase_book’, ‘rowKey’, ‘CF:C’

29.11、truncate

禁用表-删除表-创建表

29.12、create

创建表，例如：
hbase> create ‘table’, ‘cf’

创建多个列族：
hbase> create 't1', {NAME => 'f1'}, {NAME => 'f2'}, {NAME => 'f3'}

29.13、更多后续拓展

三十、HBase节点的管理

30.1、服役（commissioning）

当启动 regionserver 时，regionserver 会向 Hmaster 注册并开始接收本地数据，开始的时候，新加入的节点不会有任何数据，平衡器开启的情况下，将会有新的 region 移动到开启的 RegionServer 上。如果启动和停止进程是使用 ssh 和 HBase 脚本，那么会将新添加的节点的主机名加入到 conf/regionservers 文件中。

30.2、退役（decommissioning）

顾名思义，就是从当前 HBase 集群节点中删除某个 RegionServer，这个过程分为如下几步：

Step1、使用以下命令停止负载平衡器

hbase> balance_switch false

Step2、在退役节点上停止 RegionServer

hbase> hbase-daemon.sh stop regionserver

Step3、RegionServer 一旦停止，会关闭维护的所有 region

Step4、Zookeeper 上的该 RegionServer 节点消失

Step5、Master 节点检测到该 RegionServer 下线

Step6、RegionServer 的 region 服务得到重新分配

该关闭方法比较传统，需要花费一定的时间，而且会造成部分 region 短暂的不可用。
我们有一种更骚气的关闭方法：

Step1、RegionServer 先卸载所管理的 region

$ bin/graceful_stop.sh <RegionServer-hostname>
例如：
$ bin/graceful_stop.sh hadoop-senior02.itguigu.com

Step2、自动平衡数据

Step3、和之前的 2~6 步是一样的

三十一、集群失效的可能性

集群机器上不同的 Java 版本可能会引起崩溃问题，不同版本的 Hadoop 和 HBase 也会引起兼容性问题，如下几种组件出现问题，也会导致集群失效。

30.1、磁盘

例如磁盘转速过慢，导致一些其他服务超时，或者磁盘损坏等。

30.2、操作系统

例如操作系统在升级补丁时，升级到了一个无法兼容当前集群的版本。

30.3、网络

网络震荡，阻塞等原因造成的集群服务连接超时。

30.4、内存

内存的损坏或者内存溢出，或者在执行一些具体任务时，由于编码问题造成的内存泄漏导致了内存溢出。

三十二、HBase数据的备份与恢复

HBase 中存储着大量的在线业务数据和离线业务数据，所以备份对于我们而言会非常重要。接下来讲解一下如何执行在线和离线备份。

32.1、离线备份

此方法时完全停止 HBase 服务后，使用 distcp 命令运行 MapReduce 任务进行备份，将数据

备份到另一个地方，可以是同一个集群，也可以是专用的备份集群。如果你的集群是线上集群，不能下线停止服务，则此方法不可取，请阅读在线备份方法。

32.1.1、同一集群的备份

即，把数据转移到当前集群的其他目录下，可以使用命令：
$ bin/hadoop distcp \
hdfs://hadoop-senior01.itguigu.com:8020/hbase \
hdfs://hadoop-senior01.itguigu.com:8020/HbaseBackup/backup20170803

32.1.2、备份数据到另外一个集群

即，把数据复制转移到另一个集群的存储空间中，使用如下命令：
$ bin/hadoop distcp \
hdfs://hadoop-senior01.itguigu.com:8020/hbase \
hdfs://z01:8020/HbaseBackup/backup20170803
这种备份方式其实更适用于建立专门的备份集群。
尖叫提示：复制时，一定要开启 Yarn 服务。

32.1.3、离线恢复

与备份方法一样，将数据整个移动回来即可。

32.1、在线备份

即，在 HBase 不离线的情况下，使用快照来备份数据。

三十三、HBase的过滤器

这里所说的是布隆过滤器，就是当需要在扫描过程中过滤掉大部分数据的时候使用，它可以减少内部数据的查找，从而加快扫描速度。这些数据存储在 Hfiles 的元数据区，一旦写入就不会再改变。布隆过滤器通过折叠的方式减少占用的空间，当分配 region 给 RegionServer 时，Hfile 被打开，布隆过滤器载入内存。

33.1、主要功能：提高随机读性能

33.2、参考文章：http://blog.csdn.net/opensure/article/details/46453681，

http://blog.csdn.net/dadoneo/article/details/6847481

三十四、HBase的协处理

34.1、简介

协处理器（coprocessor），每个 RegionServer 都会有一个协处理器进程，所有的 region 都包含对协处理器实现的引用。可以通过 RegionServer 的类路径加载，也可以通过 HDFS 类加载器加载。协处理器的设计是为了方便开发者向 HBase 中添加额外的功能，而不是给普通使用者使用。它可以用来做服务器端的操作，比如 region 分裂、合并以及客户端的创建，读取，删除，更新等操作，实现用户自定义功能。

34.2、协处理器类型

34.2.1、Coprocessor

提供 region 生命周期的管理，比如打开，关闭，分裂，合并等等。

34.2.2、RegionServer

提供对表修稿操作的监控，比如 get，put，scan，delete

34.2.3、Endpoint

提供在 region 端执行任意函数的功能，比如 RegionServer 上的列聚集函数。

三十五、HBase 版本确界

35.1、版本是下界

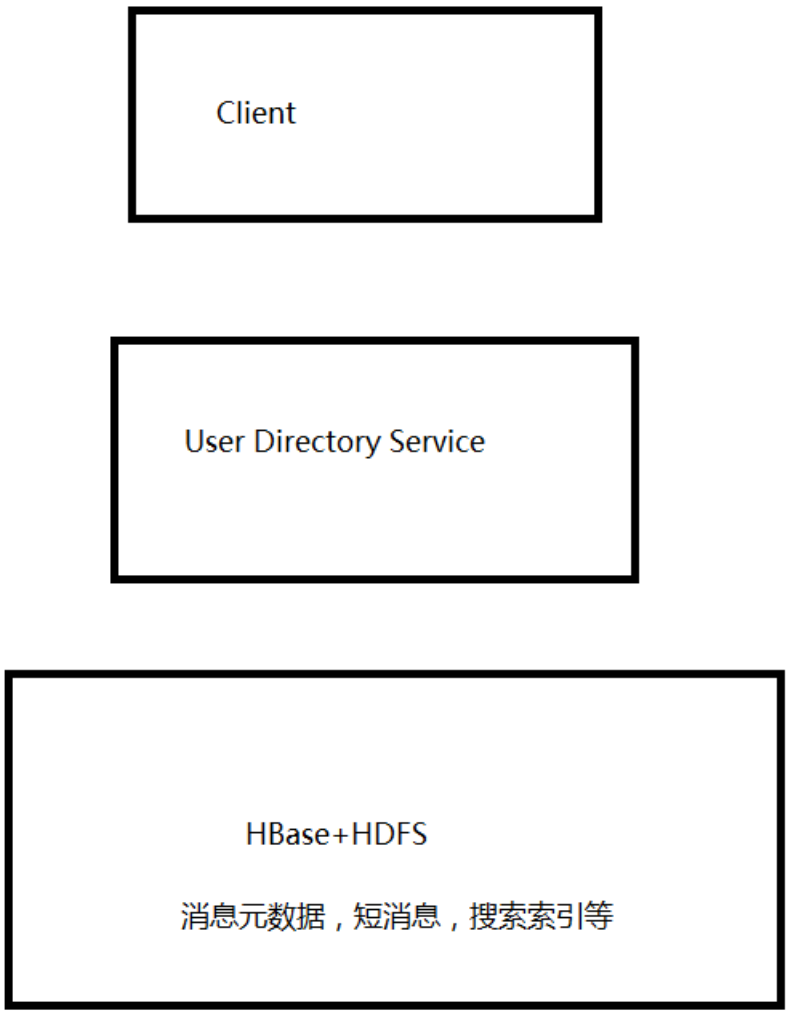
默认的版本下界是 0，即禁用。row 版本使用的最小数目是与生存时间（TTL Time To Live）相结合的，并且我们根据实际需求可以有 0 或更多的版本，使用 0，即只有 1 个版本的值写入 cell。

35.2、版本的上界

默认的版本上界是 3，也就是一个 row 保留 3 个副本（基于时间戳的插入）。该值不要设计的过大，一般的业务不会超过 100。如果 cell 中存储的数据版本号超过了 3 个，再次插入数据时，最新的值会将最老的值覆盖。

三十六、HBase在目前行业前景中的应用

36.1、FaceBook



36.1.1、为何选择

* 提高读写吞吐量
* 相比于其他数据库有更高的随机读取性能
* 水平可伸缩性
* 自动故障转移
* 超强的一致性
* 超强的容错性

36.1.2、在 HBase 中存储什么

* 即时消息
* 搜索数据

36.1.3、流程控制

客户端请求用户目录服务获取用户详细信息，得到用户详细信息之后，客户端发送请求给应用服务器，其可以是一个运行特定应用程序的 Tomcat 服务器，使用定制服务的应用程序可以通过 HBase 索引来搜索用户相关的单元格数据，单元格可以存放写入的消息、追加或者检索的数据。

36.1.4、规模

每天：
* 消息量：发送和接收的消息数超过 60 亿
* 将近 1000 亿条数据的读写
* 高峰期每秒 150 万左右操作
* 整体读取数据占有约 55%，写入占有 45%
* 超过 2PB 的数据，涉及冗余共 6PB 数据
* 数据使用 LZO 压缩
* 数据每月大概增长 300 千兆字节。

三十七、一些优化技巧

37.1、通用优化

37.1.1、NameNode 的元数据备份使用 SSD

37.1.2、定时备份 NameNode 上的元数据，每小时或者每天备份，如果数据极其重要，可以 5~10 分钟备份一次。备份可以通过定时任务复制元数据目录即可。

37.1.3、为 NameNode 指定多个元数据目录，使用 dfs.name.dir 或者 dfs.namenode.name.dir 指定。一个指定本地磁盘，一个指定网络磁盘。这样可以提供元数据的冗余和健壮性，以免发生故障。

37.1.4、设置 dfs.namenode.name.dir.restore 为 true，允许尝试恢复之前失败的 dfs.namenode.name.dir 目录，在创建 checkpoint 时做此尝试，如果设置了多个磁盘，建议允许。

37.1.5、NameNode 节点必须配置为 RAID1（镜像盘）结构。

37.1.6、补充：什么是 Raid0、Raid0+1、Raid1、Raid5

Standalone
最普遍的单磁盘储存方式。

Cluster
集群储存是通过将数据分布到集群中各节点的存储方式,提供单一的使用接口与界面,使用户可以方便地对所有数据进行统一使用与管理。

Hot swap
用户可以再不关闭系统,不切断电源的情况下取出和更换硬盘,提高系统的恢复能力、拓展性和灵活性。

Raid0
Raid0 是所有 raid 中存储性能最强的阵列形式。其工作原理就是在多个磁盘上分散存取连续的数据,这样,当需要存取数据是多个磁盘可以并排执行,每个磁盘执行属于它自己的那部分数据请求,显著提高磁盘整体存取性能。但是不具备容错能力,适用于低成本、低可靠性的台式系统。

Raid1
又称镜像盘,把一个磁盘的数据镜像到另一个磁盘上,采用镜像容错来提高可靠性,具有 raid 中最高的数据冗余能力。存数据时会将数据同时写入镜像盘内,读取数据则只从工作盘读出。发生故障时,系统将从镜像盘读取数据,然后再恢复工作盘正确数据。这种阵列方式可靠性极

高,但是其容量会减去一半。广泛用于数据要求极严的应用场合,如商业金融、档案管理等领域。只允许一颗硬盘出故障。

Raid0+1

将 Raid0 和 Raid1 技术结合在一起,兼顾两者的优势。在数据得到保障的同时,还能提供较强的存储性能。不过至少要求 4 个或以上的硬盘，但也只允许一个磁盘出错。是一种三高技术。

Raid5

Raid5 可以看成是 Raid0+1 的低成本方案。采用循环偶校验独立存取的阵列方式。将数据和相对应的奇偶校验信息分布存储到组成 RAID5 的各个磁盘上。当其中一个磁盘数据发生损坏后,利用剩下的磁盘和相应的奇偶校验信息 重新恢复/生成丢失的数据而不影响数据的可用性。至少需要 3 个或以上的硬盘。适用于大数据量的操作。成本稍高、储存新强、可靠性强的阵列方式。

RAID 还有其他方式，请自行查阅。

37.1.7、保持 NameNode 日志目录有足够的空间，这些日志有助于帮助你发现问题。

37.1.8、因为 Hadoop 是 IO 密集型框架，所以尽量提升存储的速度和吞吐量（类似位宽）。

37.2、Linux 优化

37.2.1、开启文件系统的预读缓存可以提高读取速度

$ sudo blockdev --setra 32768 /dev/sda
（尖叫提示：ra 是 readahead 的缩写）

37.2.2、关闭进程睡眠池

$ sudo sysctl -w vm.swappiness=0

37.2.3、调整 ulimit 上限，默认值为比较小的数字

$ ulimit -n 查看允许最大进程数
$ ulimit -u 查看允许打开最大文件数
修改：

```
$ sudo vi /etc/security/limits.conf 修改打开文件数限制

末尾添加：
```

```
*               soft    nofile          1024000

*               hard    nofile          1024000

Hive            -       nofile          1024000

hive            -       nproc           1024000

$ sudo vi /etc/security/limits.d/20-nproc.conf 修改用户打开进程数限制

修改为：

#*          soft    nproc     4096

#root       soft    nproc     unlimited

*          soft    nproc     40960

root       soft    nproc     unlimited
```

37.2.4、开启集群的时间同步 NTP，请参看之前文档

37.2.5、更新系统补丁（尖叫提示：更新补丁前，请先测试新版本补丁对集群节点的兼容性）

37.3、HDFS 优化（hdfs-site.xml）

37.3.1、保证 RPC 调用会有较多的线程数

属性：dfs.namenode.handler.count
解释：该属性是 NameNode 服务默认线程数，的默认值是 10，根据机器的可用内存可以调整为 50~100

属性：dfs.datanode.handler.count
解释：该属性默认值为 10，是 DataNode 的处理线程数，如果 HDFS 客户端程序读写请求比较多，可以调高到 15~20，设置的值越大，内存消耗越多，不要调整的过高，一般业务中，5~10 即可。

37.3.2、副本数的调整

属性：dfs.replication
解释：如果数据量巨大，且不是非常之重要，可以调整为 2~3，如果数据非常之重要，可以调整为 3~5。

37.3.3.、文件块大小的调整

属性：dfs.blocksize
解释：块大小定义，该属性应该根据存储的大量的单个文件大小来设置，如果大量的单个文件都小于 100M，建议设置成 64M 块大小，对于大于 100M 或者达到 GB 的这种情况，建议设置成 256M，一般设置范围波动在 64M~256M 之间。

37.4、MapReduce 优化（mapred-site.xml）

37.4.1、Job 任务服务线程数调整

mapreduce.jobtracker.handler.count
该属性是 Job 任务线程数，默认值是 10，根据机器的可用内存可以调整为 50~100

37.4.2、Http 服务器工作线程数

属性：mapreduce.tasktracker.http.threads
解释：定义 HTTP 服务器工作线程数，默认值为 40，对于大集群可以调整到 80~100

37.4.3、文件排序合并优化

属性：mapreduce.task.io.sort.factor
解释：文件排序时同时合并的数据流的数量，这也定义了同时打开文件的个数，默认值为 10，如果调高该参数，可以明显减少磁盘 IO，即减少文件读取的次数。

37.4.5、设置任务并发

属性：mapreduce.map.speculative
解释：该属性可以设置任务是否可以并发执行，如果任务多而小，该属性设置为 true 可以明显加快任务执行效率，但是对于延迟非常高的任务，建议改为 false，这就类似于迅雷下载。

37.4.6、MR 输出数据的压缩

属性：mapreduce.map.output.compress、mapreduce.output.fileoutputformat.compress
解释：对于大集群而言，建议设置 Map-Reduce 的输出为压缩的数据，而对于小集群，则不

需要。

37.4.7、优化 Mapper 和 Reducer 的个数

属性：
mapreduce.tasktracker.map.tasks.maximum
mapreduce.tasktracker.reduce.tasks.maximum
解释：以上两个属性分别为一个单独的 Job 任务可以同时运行的 Map 和 Reduce 的数量。设置上面两个参数时，需要考虑 CPU 核数、磁盘和内存容量。假设一个 8 核的 CPU，业务内容非常消耗 CPU，那么可以设置 map 数量为 4，如果该业务不是特别消耗 CPU 类型的，那么可以设置 map 数量为 40，reduce 数量为 20。这些参数的值修改完成之后，一定要观察是否有较长等待的任务，如果有的话，可以减少数量以加快任务执行，如果设置一个很大的值，会引起大量的上下文切换，以及内存与磁盘之间的数据交换，这里没有标准的配置数值，需要根据业务和硬件配置以及经验来做出选择。

在同一时刻，不要同时运行太多的 MapReduce，这样会消耗过多的内存，任务会执行的非常缓慢，我们需要根据 CPU 核数，内存容量设置一个 MR 任务并发的最大值，使固定数据量的任务完全加载到内存中，避免频繁的内存和磁盘数据交换，从而降低磁盘 IO，提高性能。
大概配比：

CPU CORE	MEM（GB）	Map	Reduce
1	1	1	1
1	5	1	1
4	5	1~4	2
16	32	16	8
16	64	16	8
24	64	24	12
24	128	24	12

大概估算公式：
map = 2 + ⅔cpu_core
reduce = 2 + ⅓cpu_core

37.5、HBase 优化

37.5.1、在 HDFS 的文件中追加内容

不是不允许追加内容么？没错，请看背景故事：

File Appends in HDFS

July 17, 2009 | By Tom White | 2 Comments
Categories: General | Hadoop | HDFS

There is some confusion about the state of the file append operation in HDFS. It was in, now it's out. Why was it removed, and when will it be reinstated? This post looks at some of the history behind HDFS capability for supporting file appends.

Background

Early versions of HDFS had no support for an append operation. Once a file was closed, it was immutable and could only be changed by writing a new copy with a different filename. This style of file access actually fits very nicely with MapReduce, where you write the output of a data processing job to a set of new files; this is much more efficient than manipulating the input files that are already in place.

A file didn't exist until it had been successfully closed (by calling `FSDataOutputStream`'s `close()` method). If the client failed before it closed the file, or if the `close()` method failed by throwing an exception, then (to other clients at least), it was as if the file had never been written. The only way to recover the file was to rewrite it from the beginning. MapReduce worked well with this behavior, since it would simply rerun the task that had failed from the beginning.

First Steps Toward Append

It was not until the 0.15.0 release of Hadoop that open files were visible in the filesystem namespace (HADOOP-1708). Until that point, they magically appeared after they had been written and closed. At the same time, the contents of files could be read by other clients as they were being written, although only the last fully-written block was visible (see HADOOP-89). This made it possible to gauge the progress of a file that was being written, albeit in a crude manner. Additionally, tools such as `hadoop fs -tail` (and its web UI equivalent) were introduced and allowed users to view the contents of a file as it was being written, block by block.

属性：dfs.support.append
文件：hdfs-site.xml、hbase-site.xml
解释：开启 HDFS 追加同步，可以优秀的配合 HBase 的数据同步和持久化。默认值为 true。

37.5.2、优化 DataNode 允许的最大文件打开数

属性：dfs.datanode.max.transfer.threads
文件：hdfs-site.xml
解释：HBase 一般都会同一时间操作大量的文件，根据集群的数量和规模以及数据动作，设置为 4096 或者更高。默认值：4096

37.5.3、优化延迟高的数据操作的等待时间

属性：dfs.image.transfer.timeout
文件：hdfs-site.xml
解释：如果对于某一次数据操作来讲，延迟非常高，socket 需要等待更长的时间，建议把该值设置为更大的值（默认 60000 毫秒），以确保 socket 不会被 timeout 掉。

37.5.4、优化数据的写入效率

属性：
mapreduce.map.output.compress
mapreduce.map.output.compress.codec
文件：mapred-site.xml
解释：开启这两个数据可以大大提高文件的写入效率，减少写入时间。第一个属性值修改为 true，第二个属性值修改为：org.apache.hadoop.io.compress.GzipCodec

37.5.5、优化 DataNode 存储

属性：dfs.datanode.failed.volumes.tolerated
文件：hdfs-site.xml
解释：默认为 0，意思是当 DataNode 中有一个磁盘出现故障，则会认为该 DataNode shutdown 了。如果修改为 1，则一个磁盘出现故障时，数据会被复制到其他正常的 DataNode 上，当前的 DataNode 继续工作。

37.5.6、设置 RPC 监听数量

属性：hbase.regionserver.handler.count
文件：hbase-site.xml
解释：默认值为 30，用于指定 RPC 监听的数量，可以根据客户端的请求数进行调整，读写请求较多时，增加此值。

37.5.7、优化 HStore 文件大小

属性：hbase.hregion.max.filesize
文件：hbase-site.xml
解释：默认值 10737418240（10GB），如果需要运行 HBase 的 MR 任务，可以减小此值，因为一个 region 对应一个 map 任务，如果单个 region 过大，会导致 map 任务执行时间过长。该值的意思就是，如果 HFile 的大小达到这个数值，则这个 region 会被切分为两个 Hfile。

37.5.8、优化 hbase 客户端缓存

属性：hbase.client.write.buffer
文件：hbase-site.xml
解释：用于指定 HBase 客户端缓存，增大该值可以减少 RPC 调用次数，但是会消耗更多内存，反之则反之。一般我们需要设定一定的缓存大小，以达到减少 RPC 次数的目的。

37.5.9、指定 scan.next 扫描 HBase 所获取的行数

属性：hbase.client.scanner.caching
文件：hbase-site.xml
解释：用于指定 scan.next 方法获取的默认行数，值越大，消耗内存越大。

37.6、内存优化

HBase 操作过程中需要大量的内存开销，毕竟 Table 是可以缓存在内存中的，一般会分配整个可用内存的 70%给 HBase 的 Java 堆。但是不建议分配非常大的堆内存，因为 GC 过程持续太久会导致 RegionServer 处于长期不可用状态，一般 16~48G 内存就可以了，如果因为框架占用内存过高导致系统内存不足，框架一样会被系统服务拖死。

37.7、JVM 优化

涉及文件：hbase-env.sh

37.7.1、并行 GC

参数：-XX:+UseParallelGC
解释：开启并行 GC

37.7.2、同时处理垃圾回收的线程数

参数：-XX:ParallelGCThreads=cpu_core – 1
解释：该属性设置了同时处理垃圾回收的线程数。

37.7.3、禁用手动 GC

参数：-XX:DisableExplicitGC
解释：防止开发人员手动调用 GC

37.8、Zookeeper 优化

37.8.1、优化 Zookeeper 会话超时时间

参数：zookeeper.session.timeout
文件：hbase-site.xml
解释：In hbase-site.xml, set zookeeper.session.timeout to 30 seconds or less to bound failure detection (20-30 seconds is a good start).该值会直接关系到 master 发现服务器宕机的最大周期，默认值为 30 秒，如果该值过小，会在 HBase 在写入大量数据发生而 GC 时，导致 RegionServer 短暂的不可用，从而没有向 ZK 发送心跳包，最终导致认为从节点 shutdown。一般 20 台左右的集群需要配置 5 台 zookeeper。

三十八、HBase表类型的设计

38.1、短宽

这种设计一般适用于：
* 有大量的列
* 有很少的行

38.2、高瘦

这种设计一般适用于：
* 有很少的列
* 有大量的行

38.3、短宽-高瘦的对比

38.3.1、短宽

* 使用列名进行查询不会跳过行或者存储文件
* 更好的原子性
* 不如高瘦设计的可扩展性

38.3.2、高瘦

* 如果使用 ID 进行查询，会跳过行
* 不利于原子性
* 更好的扩展

三十九、HBase的预分区

39.1、为何要预分区？

* 增加数据读写效率
* 负载均衡，防止数据倾斜
* 方便集群容灾调度 region
* 优化 Map 数量

39.2、如何预分区？

每一个 region 维护着 startRow 与 endRowKey，如果加入的数据符合某个 region 维护的 rowKey 范围，则该数据交给这个 region 维护。

39.3、如何设定预分区？

39.3.1、手动指定预分区

create 'staff','info','partition1',SPLITS => ['1000','2000','3000','4000']
完成后如图：

Table Regions

Name	Region Server	Start Key	End Key	Requests
table1,,1495642218397.ab492f425a1ca6422073cb3ce19f7d7e.	z01:60020		1000	0
table1,1000,1495642218397.f6b5c80ed2214379f2b1b91b8515a935.	z03:60020	1000	2000	0
table1,2000,1495642218398.06cedace5adf2a4bd41e7e09cff8f9c3.	z01:60020	2000	3000	0
table1,3000,1495642218398.243f41c7f91b5fc6b71034a6ebe34f70.	z02:60020	3000	4000	0
table1,4000,1495642218398.fb95de71f9dfc7900986bee39ed1fd7a.	z02:60020	4000		0

39.3.2、使用 16 进制算法生成预分区

create 'staff2','info','partition2',{NUMREGIONS => 15, SPLITALGO => 'HexStringSplit'}
完成后如图：

Table Regions

Name	Region Server	Start Key	End Key	Requests
table3,,1495642786733.5368b42666fde7fb0b642daf3e1b3b2f.	z02:60020		11111111	0
table3,11111111,1495642786734.c45f2b8f700c7e6e6b35403da2c84249.	z01:60020	11111111	22222222	0
table3,22222222,1495642786734.3edd9c3cbf2efe64f147871764f380fb.	z03:60020	22222222	33333333	0
table3,33333333,1495642786734.91dab7e69f030638662e6c42de2d20b4.	z01:60020	33333333	44444444	0
table3,44444444,1495642786734.d6ce0f595f199d3d2a285d5fe5ec34bd.	z01:60020	44444444	55555555	0
table3,55555555,1495642786734.574da98eb76c0ef5a948859457bcdff0.	z02:60020	55555555	66666666	0
table3,66666666,1495642786734.b1d92cbe1bf5c06795984f456b2cd692.	z01:60020	66666666	77777777	0
table3,77777777,1495642786734.b3717a3950653ba8905862be766de600.	z02:60020	77777777	88888888	0
table3,88888888,1495642786734.90f42a9bec7a4909b9b1f4eb7783d6cc.	z03:60020	88888888	99999999	0
table3,99999999,1495642786734.7912adf2cba00a4dd28a3c479321b596.	z03:60020	99999999	aaaaaaaa	0
table3,aaaaaaaa,1495642786734.5af37085a70baea33ebededcd28219e1a.	z01:60020	aaaaaaaa	bbbbbbbb	0
table3,bbbbbbbb,1495642786734.09284191e02a938512bfe8d3e588d3ee.	z02:60020	bbbbbbbb	cccccccc	0
table3,cccccccc,1495642786734.c0f8b0d20fe5d0d7861e3c49db6cbe16.	z03:60020	cccccccc	dddddddd	0
table3,dddddddd,1495642786734.93600a2dc08821fd01f2ca00965bbaf8.	z03:60020	dddddddd	eeeeeeee	0
table3,eeeeeeee,1495642786734.4f4e56a23fd51df42b33f3dc5ba880a1.	z02:60020	eeeeeeee		0

39.3.3、分区规则创建于文件中

创建 splits.txt 文件内容如下：

```
aaaa
bbbb
cccc
dddd
```

然后执行：

create 'table2','partition2',SPLITS_FILE => 'splits.txt'，成功后如图：

Table Regions

Name	Region Server	Start Key	End Key	Requests
table2,,1495642727718.5ebcf21bfb7fe838288af5046d79d79d.	z03:60020		aaaa	0
table2,aaaa,1495642727718.7aa4c6aa1c84beb78971dc214ca853d4.	z03:60020	aaaa	bbbb	0
table2,bbbb,1495642727718.de32a1e5e21f4f73932eabd7572d9cdf.	z02:60020	bbbb	cccc	0
table2,cccc,1495642727719.ec6ea106d8b05a2575dc1e3f20391d23.	z02:60020	cccc	dddd	0
table2,dddd,1495642727719.8109c3415b405595ecc77020b4c22558.	z01:60020	dddd		0

39.3.4、使用 JavaAPI 创建预分区

Java 代码如下：

```
//自定义算法，产生一系列Hash散列值存储在二维数组中
byte[][] splitKeys = 某个散列值函数
//创建HBaseAdmin实例
HBaseAdmin hAdmin = new HBaseAdmin(HBaseConfiguration.create());
//创建HTableDescriptor实例
HTableDescriptor tableDesc = new HTableDescriptor(tableName);
//通过HTableDescriptor实例和散列值二维数组创建带有预分区的HBase表
hAdmin.createTable(tableDesc, splitKeys);
```

```
//自定义算法，产生一系列 Hash 散列值存储在二维数组中
byte[][] splitKeys = 某个散列值函数
//创建 HBaseAdmin 实例
HBaseAdmin hAdmin = new HBaseAdmin(HBaseConfiguration.create());
//创建 HTableDescriptor 实例
HTableDescriptor tableDesc = new HTableDescriptor(tableName);
//通过 HTableDescriptor 实例和散列值二维数组创建带有预分区的 HBase 表
hAdmin.createTable(tableDesc, splitKeys);
```

四十、HBase的rowKey设计技巧

40.1、设计宗旨与目标

主要目的就是针对特定的业务模型，按照 rowKey 进行预分区设计，使之后面加入的数据能够尽可能的分散于不同的 rowKey 中。比如复合 RowKey。

40.2、设计方式案例

40.2.1、案例一：生成随机数、hash、散列值

比如：

原本 rowKey 为 1001 的，MD5 后变成：b8c37e33defde51cf91e1e03e51657da

原本 rowKey 为 3001 的，MD5 后变成：908c9a564a86426585b29f5335b619bc

原本 rowKey 为 5001 的，MD5 后变成：03b264c595403666634ac75d828439bc

在做此操作之前，一般我们会选择从数据集中抽取样本，来决定什么样的 rowKey 来 Hash 后作为每个分区的临界值。

40.2.2、案例二：字符串反转

比如：

20170524000001 转成 10000042507102

20170524000002 转成 20000042507102

这样也可以在一定程度上散列逐步 put 进来的数据。

40.2.3、案例三：字符串拼接

比如：

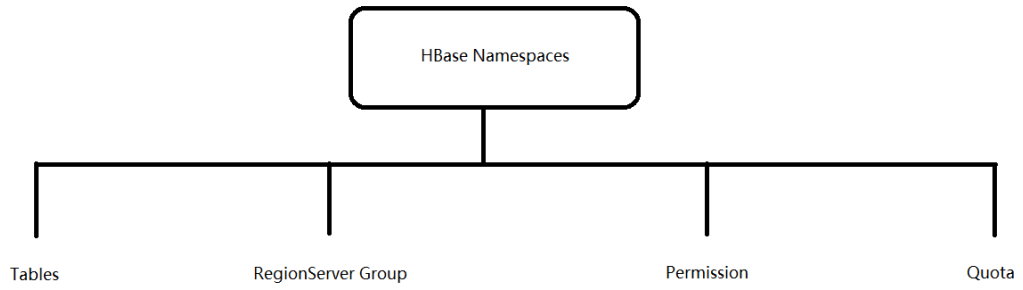
20170524000001_a12e

20170524000001_93i7

四十一、设计案例

41.1、牵扯概念：命名空间

41.1.1、命名空间结构图



41.1.2、组成部分

-Table：表，所有的表都是命名空间的成员，即表必属于某个命名空间，如果没有指定，则在 default 默认的命名空间中。
-RegionServer group：一个命名空间包含了默认的 RegionServer Group。
-Permission：权限，命名空间能够让我们来定义访问控制列表 ACL（Access Control List）。例如，创建表，读取表，删除，更新等等操作。
-Quota：限额，可以强制一个命名空间可包含的 region 的数量

41.1.3、命名空间命令

-创建命名空间
例如：

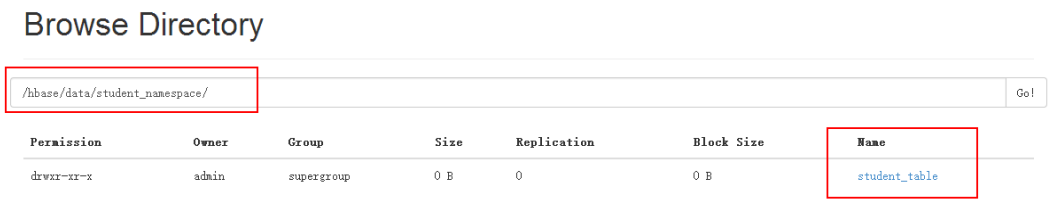
```
hbase(main):002:0> create_namespace 'student_namespace'
```

-创建表时指定命名空间
例如：

```
hbase(main):004:0> create 'student_namespace:student_table','student_info'
```

41.1.4、观察 HDFS 中的目录结构的变化

如图所示：



41.2、微博项目

41.2.1、需求分析

-微博内容的浏览，数据库表设计
-用户社交体现：关注用户，取关用户
-拉取关注的人的微博内容

41.2.2、步骤拆解

创建命名空间

创建微博内容表

方法	creatTableeContent
Table Name	weibo:content
RowKey	用户 ID_时间戳
ColumnFamily	info
ColumnLabel	标题　　内容　图片
Version	1 个版本

创建用户关系表

方法	createTableRelations
Table Name	weibo:relations
RowKey	用户 ID
ColumnFamily	attends、fans
ColumnLabel	关注用户 ID，粉丝用户 ID
ColumnValue	用户 ID
Version	1 个版本

创建用户微博内容接收邮件表

方法	createTableReceiveContentEmails
Table Name	weibo:receive_content_email
RowKey	用户 ID
ColumnFamily	info
ColumnLabel	用户 ID
ColumnValue	取微博内容的 RowKey
Version	1000

发布微博内容

a、 微博内容表中添加 1 条数据
b、 微博收件箱表对所有粉丝用户添加数据

添加关注用户

a、 在微博用户关系表中，对当前主动操作的用户添加新关注的好友
b、 在微博用户关系表中，对被关注的用户添加新的粉丝
c、 微博收件箱表中添加所关注的用户发布的微博

移除（取关）用户

a、在微博用户关系表中，对当前主动操作的用户移除取关的好友(attends)
b、在微博用户关系表中，对被取关的用户移除粉丝
c、微博收件箱中删除取关的用户发布的微博

获取关注的人的微博内容

a、从微博收件箱中获取所关注的用户的微博 RowKey
b、根据获取的 RowKey，得到微博内容

41.2.3、代码实现

见代码。

End、HUE工具集成

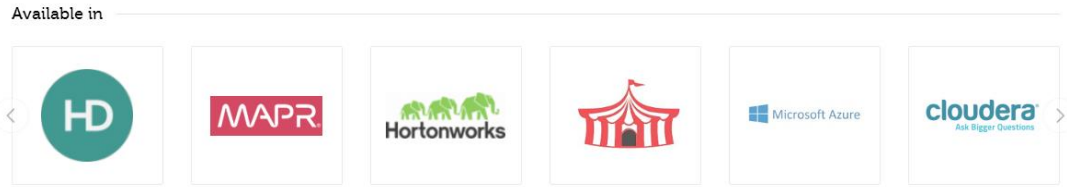
End.1、HUE 简介

End.1.1、来源

HUE=HadoopUser Experience，看这名字就知道怎么回事了吧，没错，直白来说就是 Hadoop 用户体验，是一个开源的 Apache Hadoop UI 系统，由 Cloudera Desktop 演化而来，最后 Cloudera 公司将其贡献给 Apache 基金会的 Hadoop 社区，它是基于 Python Web 框架 Django 实现的。通过使用 HUE 我们可以在浏览器端的 Web 控制台上与 Hadoop 集群进行交互来分析处理数据。

End.1.2、官网及使用者

官网网站：http://gethue.com/
使用的公司：



End.2、安装 HUE

End.2.1、帮助文档

http://archive.cloudera.com/cdh5/cdh/5/hue-3.7.0-cdh5.3.0/manual.html

End.2.2、准备环境依赖

打开官方网站可以找到对应的部署 HUE 所需的各种依赖列表，如下图所示：

2.1.1. Hue Dependencies

Hue employs some Python modules which use native code and requires certain development libraries be installed on your system. To install from the tarball, you must have the following installed:

Table 1. Required Dependencies

Redhat	Ubuntu 10.04	Ubuntu 12.04/14.04
gcc	gcc	gcc
g++	g++	g++
libxml2-devel	libxml2-dev	libxml2-dev
libxslt-devel	libxslt-dev	libxslt-dev
cyrus-sasl-devel	libsasl2-dev	libsasl2-dev
cyrus-sasl-gssapi	libsasl2-modules-gssapi-mit	libsasl2-modules-gssapi-mit
mysql-devel	libmysqlclient-dev	libmysqlclient-dev
python-devel	python-dev	python-dev
python-setuptools	python-setuptools	python-setuptools
python-simplejson	python-simplejson	python-simplejson
sqlite-devel	libsqlite3-dev	libsqlite3-dev
ant	ant	ant
libsasl2-dev	cyrus-sasl-devel	libsasl2-dev
libsasl2-modules-gssapi-mit	cyrus-sasl-gssapi	libsasl2-modules-gssapi-mit
libkrb5-dev	krb5-devel	libkrb5-dev
libtidy-0.99-0	libtidy	libtidy-0.99-0 (For unit tests only)
mvn	mvn (From maven2 package or tarball)	mvn (From maven2/maven3 package or tarball)
openldap-dev / libldap2-dev	openldap-devel	libldap2-dev

如上图所示，这部分内容是告诉你，安装编译 Hue 需要依赖哪些 Linux 安装包，你只需要使用 yum 命令一次安装就可以了，在此给大家整理好该命令（注意使用 root 权限安装）：

yum -y install ant asciidoc cyrus-sasl-devel cyrus-sasl-gssapi gcc gcc-c++ krb5-devel libtidy libxml2-devel libxslt-devel openldap-devel python-devel sqlite-devel openssl-devel mysql-devel gmp-devel
（尖叫提示：使用 yum 安装这些包的同时，也会自动安装 openJDK 的依赖，所以，请自行删除安装后的 openJDK，忘记的同学请参考 Linux 基础）
（查询：# rpm -qa | grep java）
（删除：# rpm -e --nodeps xxxxxxx-java-xxxx.rpm）

End.2.3、解压 HUE

$ tar -zxf /opt/softwares/hue-3.7.0-cdh5.3.6.tar.gz -C /opt/modules/cdh/

End.2.4、编译 HUE

到 hue 安装目录下，执行 make apps

$ make apps
尖叫提示：使用普通用户编译

大概等个几分钟之后，就编译成功了。

End.2.5、配置 HUE

修改 Hue.ini 文件

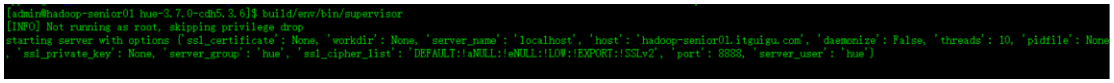
文件位置：/opt/modules/cdh/hue-3.7.0-cdh5.3.6/desktop/conf/hue.ini
其中的 secret_key 请参照官方网站配置：

修改内容参照如下：

```
[desktop]

  # Set this to a random string, the longer the better.
  # This is used for secure hashing in the session store.

  secret_key=jFE93j;2[290-eiw.KEiwN2s3['d;/.q[eIW^y#e=+Iei*@Mn<qW5o

  # Webserver listens on this address and port
  http_host=hadoop-senior01.itguigu.com
  http_port=8888

  # Time zone name
  time_zone=Asia/Shanghai

  # Enable or disable Django debug mode.
  django_debug_mode=false

  # Enable or disable backtrace for server error
  http_500_debug_mode=false

  # Enable or disable memory profiling.
  ## memory_profiler=false
```

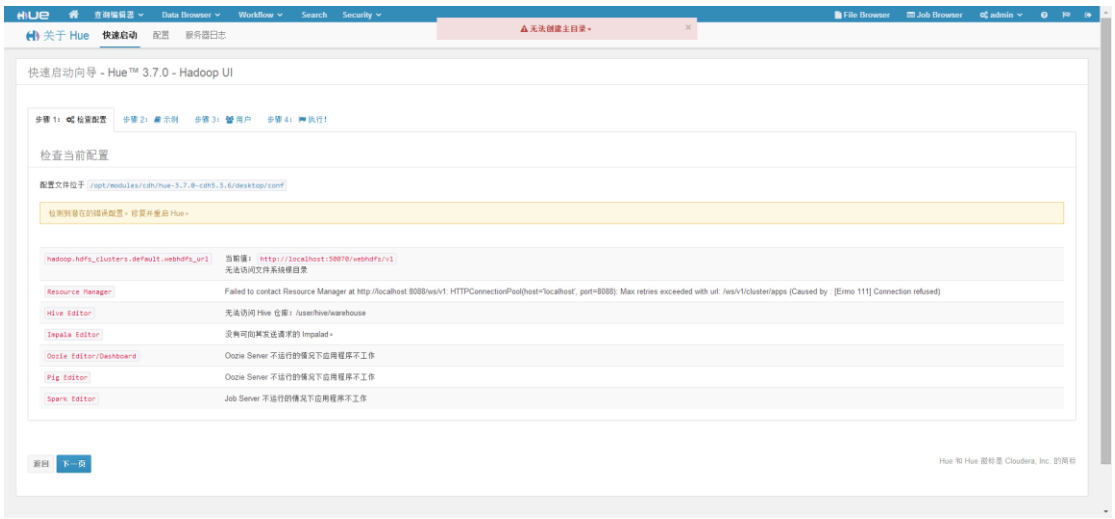
End.2.6、启动 HUE

完成之后呢，保存退出，我们来使用命令启动 Hue
$ build/env/bin/supervisor，出现如下界面表示启动成功：



接下来使用浏览器来查看 hue 界面：
http://hadoop-senior01.itguigu.com:8888，接着我们就看到如下界面：

这句话是在提示你，第一次使用本工具，需要创建一个用户及密码，且会成为 hue 的超级用户凭证，在此呢，我设置为 admin 用户名，密码随意，那就 123456 吧，然后呢就可以见到如下界面了：



End.3、HUE 与 HDFS

End.3.1、梳理集群环境

hadoop-senior01.itguigu.com	hadoop-senior02.itguigu.com	hadoop-senior03.itguigu.com
NameNode	ResourceManager	
DataNode	DataNode	DataNode
NodeManager	NodeManager	NodeManager
JobHistoryServer		
Zookeeper	Zookeeper	Zookeeper
HMaster	HMaster	
RegionServer	RegionServer	RegionServer

End.3.2、配置 hdfs-site.xml

添加
属性：dfs.webhdfs.enabled
属性值：true
解释：Enable WebHDFS (REST API) in Namenodes and Datanodes.

End.3.3、配置 core-site.xml

添加
属性：hadoop.proxyuser.hue.hosts
变更为：hadoop.proxyuser.admin.hosts
属性值：*
解释：代理的用户

属性：hadoop.proxyuser.hue.groups
变更为：hadoop.proxyuser.admin.groups
属性值：*
解释：代理的用户组

如果你的 Hadoop 配置了高可用，则必须通过 httpfs 来访问，需要添加如下属性，反则则不必须。（如果 HUE 服务与 Hadoop 服务不在同一节点，则必须配置）

属性：hadoop.proxyuser.hue.hosts
变更为：hadoop.proxyuser.httpfs.hosts
属性值：*

属性：hadoop.proxyuser.hue.groups
变更为：hadoop.proxyuser.httpfs.groups
属性值：*

End.3.4、httpfs-site.xml

添加
属性：httpfs.proxyuser.hue.hosts
属性值：*

属性：httpfs.proxyuser.hue.groups
属性值：*

解释：以上两个属性主要用于 HUE 服务与 Hadoop 服务不在同一台节点上所必须的配置。
尖叫提示：
* 如果没有配置 NameNode 的 HA，HUE 可以用 WebHDFS 来管理 HDFS
* 如果配置了 NameNodeHA，则 HUE 只可用 HttpFS 来管理 HDFS

End.3.5、scp 同步配置

```
$ scp -r etc/ hadoop-senior02.itguigu.com:/opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/
$ scp -r etc/ hadoop-senior03.itguigu.com:/opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/
```

End.3.6、启动 httpfs 服务

```
$ /opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/sbin/httpfs.sh start &
```

End.3.7、配置 hue.ini

找到[hadoop]标签

```
[[[default]]]
  # Enter the filesystem uri
  fs_defaultfs=hdfs://hadoop-senior01.itguigu.com:8020
  # fs_defaultfs=hdfs://mycluster

  # NameNode logical name.
  # 如果开启了高可用，需要配置如下
  ## logical_name=mycluster

  # Use WebHdfs/HttpFs as the communication mechanism.
  # Domain should be the NameNode or HttpFs host.
  # Default port is 14000 for HttpFs.
  ## webhdfs_url=http://localhost:50070/webhdfs/v1
  webhdfs_url=http://hadoop-senior01.itguigu.com:14000/webhdfs/v1
  # Change this if your HDFS cluster is Kerberos-secured
  ## security_enabled=false

  # Default umask for file and directory creation, specified in an octal value.
  ## umask=022

  # Directory of the Hadoop configuration
  ## hadoop_conf_dir=$HADOOP_CONF_DIR when set or '/etc/hadoop/conf'
  hadoop_conf_dir=/opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/etc/hadoop
  hadoop_hdfs_home=/opt/modules/cdh/hadoop-2.5.0-cdh5.3.6
  hadoop_bin=/opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/bin
```

End3.8、测试

$ build/env/bin/supervisor
打开 HUE 的页面，进行 HDFS 管理。
尖叫提示：
如果提示错误根目录应该归属于 hdfs，请修改 python 变量，位置如下：
/opt/modules/cdh/hue-3.7.0-cdh5.3.6/desktop/libs/hadoop/src/hadoop/fs/webhdfs.py
修改其中的变量值为：
DEFAULT_HDFS_SUPERUSER = 'admin'
然后重启 HUE 服务即可。
尖叫提示：
启动 HUE 服务时，请先 kill 掉之前的 HUE 服务，如果提示地址被占用，请使用如下命令查看占用 8888 端口的进程并 kill 掉：
$ netstat -tunlp | grep 8888

End.4、HUE 与 YARN

End.4.1、配置 hue.ini

找到[[yarn_clusters]]标签，修改配置如下图所示：

```
[[yarn_clusters]]

  [[[default]]]
    # Enter the host on which you are running the ResourceManager
    ## resourcemanager_host=localhost
    resourcemanager_host=hadoop-senior02.itguigu.com

    # The port where the ResourceManager IPC listens on
    ## resourcemanager_port=8032
    resourcemanager_port=8032

    # Whether to submit jobs to this cluster
    submit_to=True

    # Resource Manager logical name (required for HA)
    ## logical_name=

    # Change this if your YARN cluster is Kerberos-secured
    ## security_enabled=false

    # URL of the ResourceManager API
    ## resourcemanager_api_url=http://localhost:8088
    resourcemanager_api_url=http://hadoop-senior02.itguigu.com:8088

    # URL of the ProxyServer API
    ## proxy_api_url=http://localhost:8088
    proxy_api_url=http://hadoop-senior02.itguigu.com:8088

    # URL of the HistoryServer API
    ## history_server_api_url=http://localhost:19888
    history_server_api_url=http://hadoop-senior01.itguigu.com:19888
```

End.4.2、重启 HUE 测试查看

$ build/env/bin/supervisor

End.5、HUE 与 Hive

End.5.1、修改 Hive 配置文件 hive-site.xml

HUE 与 hive 集成需要 hive 开启 HiveServer2 服务

属性：hive.server2.thrift.port
属性值：10000

属性：hive.server2.thrift.bind.host
属性值：hadoop-senior01.itguigu.com

属性：hive.server2.long.polling.timeout
属性值：5000

属性：hive.metastore.uris
属性值：thrift://hadoop-senior01.itguigu.com:9083

End.5.2、启动 Hive

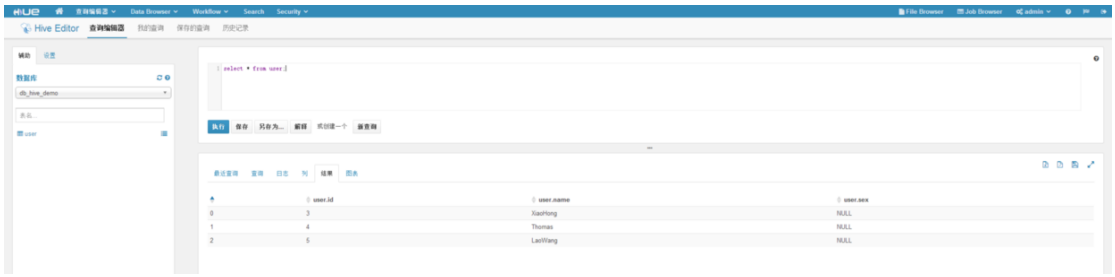
$ bin/hive --service metastore &
$ bin/hive --service hiveserver2 &
尖叫提示：如果设置了 uris，在今后使用 Hive 时，那么必须启动如上两个命令，否则 Hive 无法正常启动。

End.5.3、配置 hue.ini

找到[beeswax]属性标签，配置如图：

```
[beeswax]

  # Host where HiveServer2 is running.
  # If Kerberos security is enabled, use fully-qualified domain name (FQDN).
  ## hive_server_host=localhost
  hive_server_host=hadoop-senior01.itguigu.com

  # Port where HiveServer2 Thrift server runs on.
  ## hive_server_port=10000
  hive_server_port=10000

  # Hive configuration directory, where hive-site.xml is located
  ## hive_conf_dir=/etc/hive/conf
  hive_conf_dir=/opt/modules/cdh/hive-0.13.1-cdh5.3.6/conf
```

End.5.4、重启 hue 进行 Hive 测试



来到这样的界面进行简单的查询即可测试

$ build/env/bin/supervisor

End.6、HUE 与 Mysql

End.6.1、配置 hue.ini

找到[[[mysql]]]标签，并删掉标签注释，配置如下：

```
# mysql, oracle, or postgresql configuration.
[[[mysql]]]
  # Name to show in the UI.
  ## nice_name="My SQL DB"
  nice_name="My SQL DB"

  # For MySQL and PostgreSQL, name is the name of the database.
  # For Oracle, Name is instance of the Oracle server. For express edition
  # this is 'xe' by default.
  ## name=mysqldb

  # Database backend to use. This can be:
  # 1. mysql
  # 2. postgresql
  # 3. oracle
  ## engine=mysql
  engine=mysql

  # IP or hostname of the database to connect to.
  ## host=localhost
  host=hadoop-senior01.itguigu.com

  # Port the database server is listening to. Defaults are:
  # 1. MySQL: 3306
  # 2. PostgreSQL: 5432
  # 3. Oracle Express Edition: 1521
  ## port=3306
  port=3306

  # Username to authenticate with when connecting to the database.
  ## user=example
  user=root

  # Password matching the username to authenticate with when
  # connecting to the database.
  ## password=example
  password=123456
```

End.6.2、重启 hue.ini 测试

启动后即可测试是否成功连接 Mysql 服务，并且测试是否可以看到数据
$ build/env/bin/supervisor

End.7、HUE 与 Oozie

End.7.1、配置 hue.ini

找到[liboozie]标签以及[oozie]标签配置如下
[liboozie]：

```
[liboozie]
  # The URL where the Oozie service runs on. This is required in order for
  # users to submit jobs. Empty value disables the config check.
  ## oozie_url=http://localhost:11000/oozie
  oozie_url=http://hadoop-senior01.itguigu.com:11000/oozie

  # Requires FQDN in oozie_url if enabled
  ## security_enabled=false

  # Location on HDFS where the workflows/coordinator are deployed when submitted.
  ## remote_deployement_dir=/user/hue/oozie/deployments
  remote_deployement_dir=/user/admin/oozie-apps
```

[oozie]：

```
[oozie]
  # Location on local FS where the examples are stored.
  ## local_data_dir=..../examples
  local_data_dir=/opt/modules/cdh/oozie-4.0.0-cdh5.3.6/examples

  # Location on local FS where the data for the examples is stored.
  ## sample_data_dir=...thirdparty/sample_data
  sample_data_dir=/opt/modules/cdh/oozie-4.0.0-cdh5.3.6/oozie-apps

  # Location on HDFS where the oozie examples and workflows are stored.
  ## remote_data_dir=/user/hue/oozie/workspaces
  remote_data_dir=/user/admin/oozie-apps

  # Maximum of Oozie workflows or coodinators to retrieve in one API call.
  ## oozie_jobs_count=100

  # Use Cron format for defining the frequency of a Coordinator instead of the old frequency number/unit.
  ## enable_cron_scheduling=true
  enable_cron_scheduling=true
```

End.7.2、启动 Oozie 相关服务

$ bin/oozied.sh start

End.7.3、重启 HUE 测试查看 Oozie

$ build/env/bin/supervisor
尖叫提示：如果提示无法关联 oozie 的 share/lib，请使用 hdfs 命令创建该目录即可：
$ bin/hdfs dfs -mkdir -p /user/oozie/share/lib

End.8、HUE 与 HBase

End.8.1、修改 hue.ini 配置

找到[hbase]标签，修改内容如图：

```
[hbase]
  # Comma-separated list of HBase Thrift servers for clusters in the format of '(name|host:port)'.
  # Use full hostname with security.
  ## hbase_clusters=(Cluster|localhost:9090)
  hbase_clusters=(Cluster|hadoop-senior01.itguigu.com:9090)

  # HBase configuration directory, where hbase-site.xml is located.
  ## hbase_conf_dir=/etc/hbase/conf
  hbase_conf_dir=/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/conf

  # Hard limit of rows or columns per row fetched before truncating.
  ## truncate_limit = 500

  # 'buffered' is the default of the HBase Thrift Server and supports security.
  # 'framed' can be used to chunk up responses,
  # which is useful when used in conjunction with the nonblocking server in Thrift.
  ## thrift_transport=buffered
```

End.8.2、启动 HBase 的 thrift 服务

$ bin/hbase-daemon.sh start thrift

End.8.3、重启 HUE 进行测试

$ build/env/bin/supervisor

End.9、HUE与Zookeeper

End.9.1、配置 hue.ini

找到[zookeeper]标签，配置如下：

```
[zookeeper]

  [[clusters]]

    [[[default]]]
      # Zookeeper ensemble. Comma separated list of Host/Port.
      # e.g. localhost:2181,localhost:2182,localhost:2183
      ## host_ports=localhost:2181
      host_ports=hadoop-senior01.itguigu.com:2181,hadoop-senior02.itguigu.com:2181,hadoop-senior03.itguigu.com:2181

      # The URL of the REST contrib service (required for znode browsing)
      ## rest_url=http://localhost:9998
```

End.9.2、重启 HUE 查看即可

$ build/env/bin/supervisor

End.10、HUE与Sqoop2

End.10.1、如何配置

尖叫提示：HUE 只支持 Sqoop2 的集成，不支持 Sqoop1，在此不再演示。
http://archive.cloudera.com/cdh5/cdh/5/hue-3.7.0-cdh5.3.0/user-guide/sqoop.html

End.11、总结

在此我们总结一下集成 HUE 时，我们开启的后台服务项

End.11.1、Hadoop

$ /opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/sbin/httpfs.sh start &

End.11.2、Hive

$ /opt/modules/cdh/hive-0.13.1-cdh5.3.6/bin/hive --service metastore &
$ /opt/modules/cdh/hive-0.13.1-cdh5.3.6/bin/hive --service hiveserver2 &

End.11.3、HBase

$ /opt/modules/cdh/hbase-0.98.6-cdh5.3.6/bin/hbase-daemon.sh start thrift &

End.11.4、Oozie

$ /opt/modules/cdh/oozie-4.0.0-cdh5.3.6/bin/oozied.sh start &
为了方便，我们把这些服务加在群起脚本中，如图所示：

```
echo "================================正在第1节点开启httpfs服务============================"
ssh admin@hadoop-senior01.itguigu.com '/opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/sbin/httpfs.sh start &'

echo "================================正在第1节点开启metastore与hiveserver2服务============="
ssh admin@hadoop-senior01.itguigu.com '/opt/modules/cdh/hive-0.13.1-cdh5.3.6/bin/hive --service metastore &'
ssh admin@hadoop-senior01.itguigu.com '/opt/modules/cdh/hive-0.13.1-cdh5.3.6/bin/hive --service hiveserver2 &'

echo "================================正在第1节点开启HBase thrift服务======================="
ssh admin@hadoop-senior01.itguigu.com '/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/bin/hbase-daemon.sh start thrift &'

echo "================================正在第1节点开启Oozie服务============================="
ssh admin@hadoop-senior01.itguigu.com '/opt/modules/cdh/oozie-4.0.0-cdh5.3.6/bin/oozied.sh start &'

echo "================================正在第1节点开启HUE服务==============================="
ssh admin@hadoop-senior01.itguigu.com '/opt/modules/cdh/hue-3.7.0-cdh5.3.6/build/env/bin/supervisor &'
```

完整脚本如下：

```
#!/bin/bash
echo "==========================================正在开启集群服务
========================================="

echo "==========================================正在开启 Zookeeper 节点
==================================="
for i in admin@hadoop-senior01.itguigu.com admin@hadoop-senior02.itguigu.com
admin@hadoop-senior03.itguigu.com
do
        ssh $i '/opt/modules/cdh/zookeeper-3.4.5-cdh5.3.6/bin/zkServer.sh start'
done

echo "==========================================正在开启 NameNode 节点
==================================="
ssh admin@hadoop-senior01.itguigu.com '/opt/modules/cdh/hadoop-2.5.0-
cdh5.3.6/sbin/hadoop-daemon.sh start namenode'

echo "==========================================正在开启 DataNode 节点
==================================="
for i in admin@hadoop-senior01.itguigu.com admin@hadoop-senior02.itguigu.com
admin@hadoop-senior03.itguigu.com
```

```
do
      ssh $i '/opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/sbin/hadoop-daemon.sh start
datanode'
done

echo "=================================正在开启
SeondaryNameNode 节点========================="
ssh admin@hadoop-senior03.itguigu.com '/opt/modules/cdh/hadoop-2.5.0-
cdh5.3.6/sbin/hadoop-daemon.sh start secondarynamenode'

echo "=================================正在开启 ResourceManager
节点========================="
ssh admin@hadoop-senior02.itguigu.com '/opt/modules/cdh/hadoop-2.5.0-
cdh5.3.6/sbin/yarn-daemon.sh start resourcemanager'

echo "=================================正在开启 NodeManager 节
点============================="
for i in admin@hadoop-senior01.itguigu.com admin@hadoop-senior02.itguigu.com
admin@hadoop-senior03.itguigu.com
do
          ssh $i '/opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/sbin/yarn-daemon.sh start
nodemanager'
done

echo "=================================正在开启 JobHistoryServer
节点========================="
ssh admin@hadoop-senior01.itguigu.com '/opt/modules/cdh/hadoop-2.5.0-
cdh5.3.6/sbin/mr-jobhistory-daemon.sh start historyserver'

echo "=================================正在开启 HBase 节点
================================="
ssh admin@hadoop-senior01.itguigu.com '/opt/modules/cdh/hbase-0.98.6-
cdh5.3.6/bin/start-hbase.sh'


echo "=================================正在第 1 节点开启 httpfs 服
务============================="
ssh admin@hadoop-senior01.itguigu.com '/opt/modules/cdh/hadoop-2.5.0-
cdh5.3.6/sbin/httpfs.sh start &'

echo "=================================正在第 1 节点开启
metastore 与 hiveserver2 服务============="
ssh admin@hadoop-senior01.itguigu.com '/opt/modules/cdh/hive-0.13.1-
cdh5.3.6/bin/hive --service metastore &'
```

```
ssh admin@hadoop-senior01.itguigu.com '/opt/modules/cdh/hive-0.13.1-
cdh5.3.6/bin/hive --service hiveserver2 &'

echo "==================================正在第 1 节点开启 HBase
thrift 服务========================"
ssh admin@hadoop-senior01.itguigu.com '/opt/modules/cdh/hbase-0.98.6-
cdh5.3.6/bin/hbase-daemon.sh start thrift &'

echo "==================================正在第 1 节点开启 Oozie 服
务=============================="
ssh admin@hadoop-senior01.itguigu.com '/opt/modules/cdh/oozie-4.0.0-
cdh5.3.6/bin/oozied.sh start &'

echo "==================================正在第 1 节点开启 HUE 服务
================================"
ssh admin@hadoop-senior01.itguigu.com '/opt/modules/cdh/hue-3.7.0-
cdh5.3.6/build/env/bin/supervisor &'
```