尚硅谷大数据技术之HBase基础

（作者：尽际）

官网：<http://www.atguigu.com>

# 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

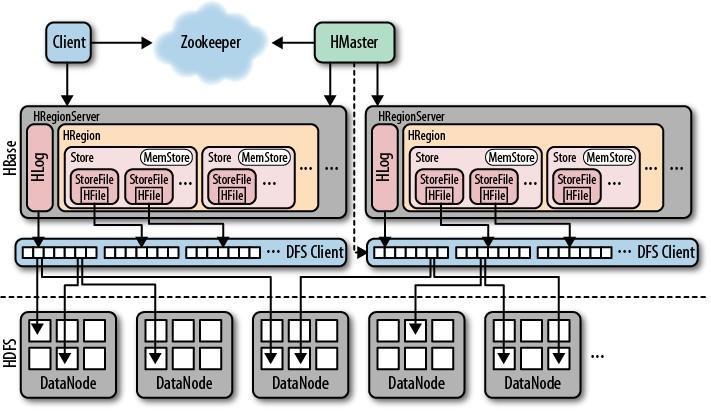
\* 2010年HBase成为Apache顶级项目

\* 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C:\Users\61661\AppData\Local\Temp\1501410678(1).png

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HRegion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。

### 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

C:\Users\61661\AppData\Local\Temp\1501432507(1).png

C:\Users\61661\AppData\Local\Temp\1501432485(1).png

### 10.4.2、hbase-site.xml



### 10.4.3、regionservers



## 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

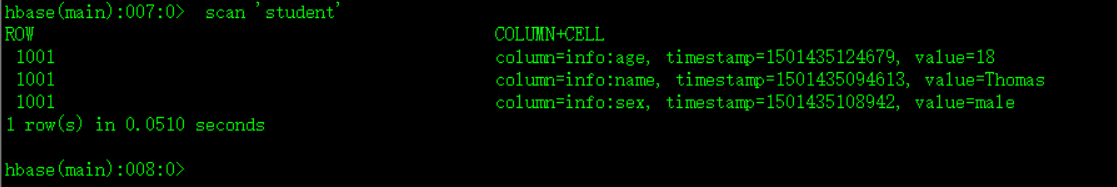
hbase(main):004:0> put 'student','1001','info:name','Thomas'

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'

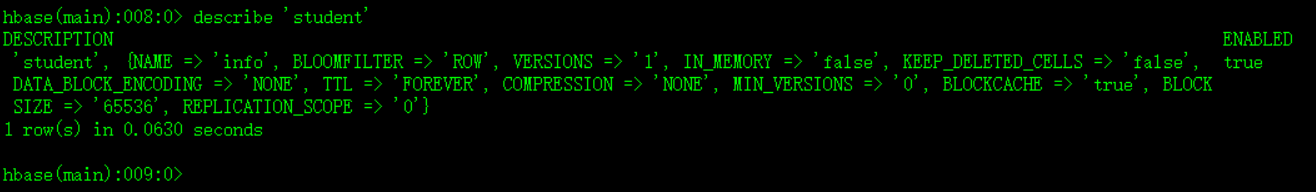


或：查看某个rowkey范围内的数据

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

## 12.7、查看表结构

hbase(main):009:0> describe ‘student’

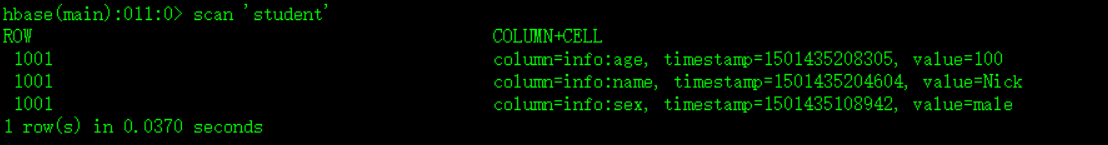


## 12.8、更新指定字段的数据

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：



## 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

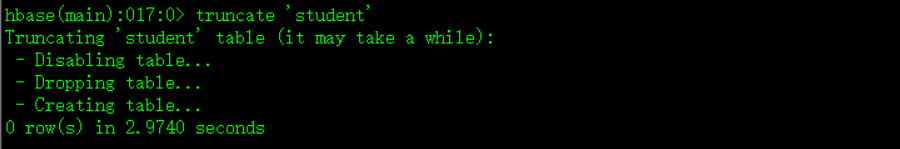
hbase(main):015:0> deleteall 'student','1001'

### 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'



## 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

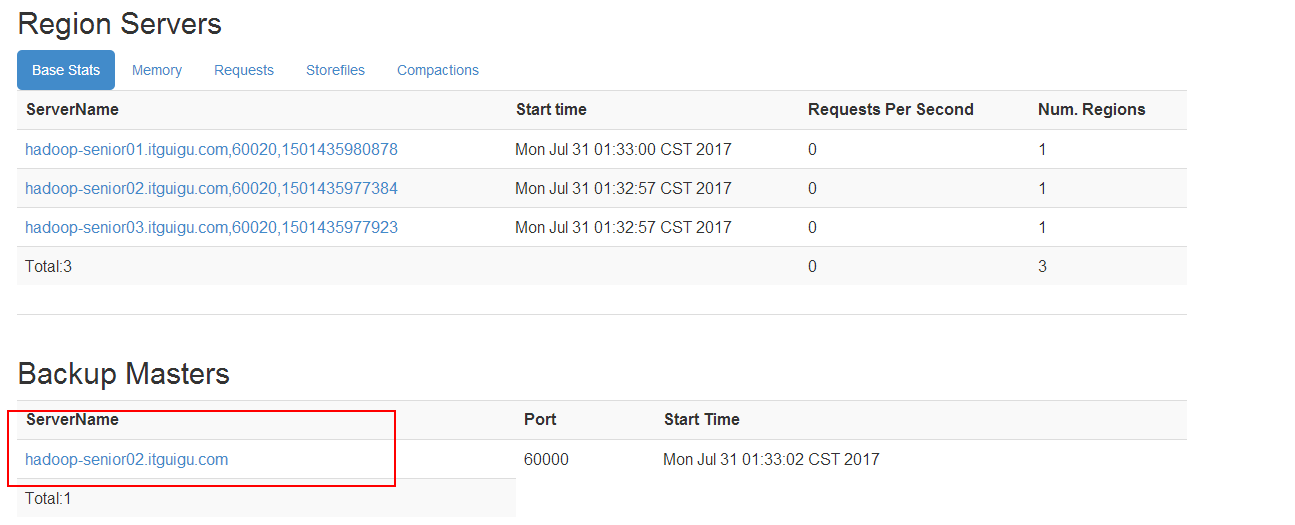
## 13.4、将整个conf目录scp到其他节点

$ 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>



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

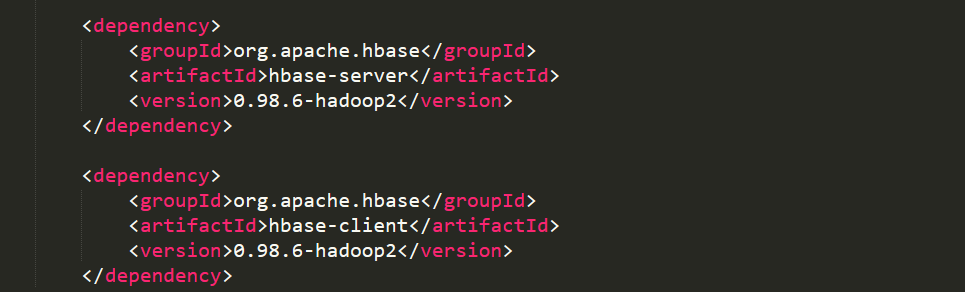
当Region达到阈值，会把过大的Region一分为二。

# 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：



## 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：



## 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

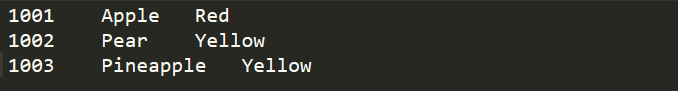
\* **执行代码**

$ /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：



**Step2、创建HBase表**

$ bin/hbase shellhbase(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);

}