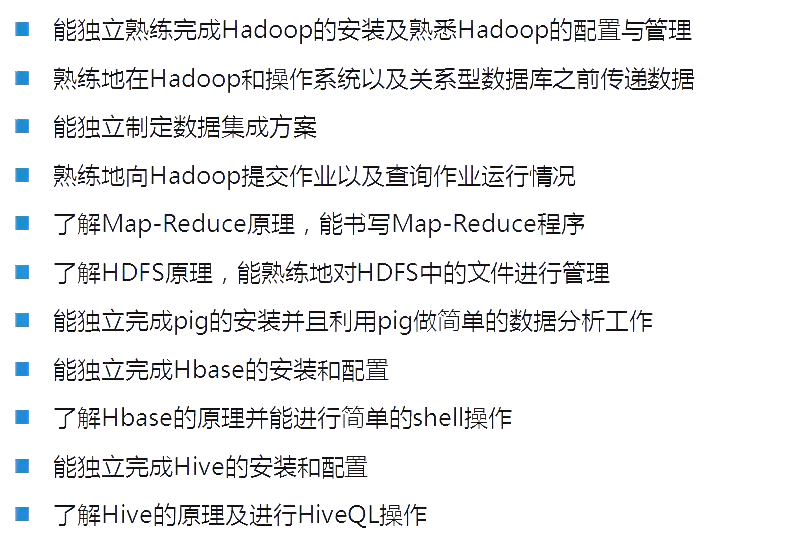
Hadoop笔记

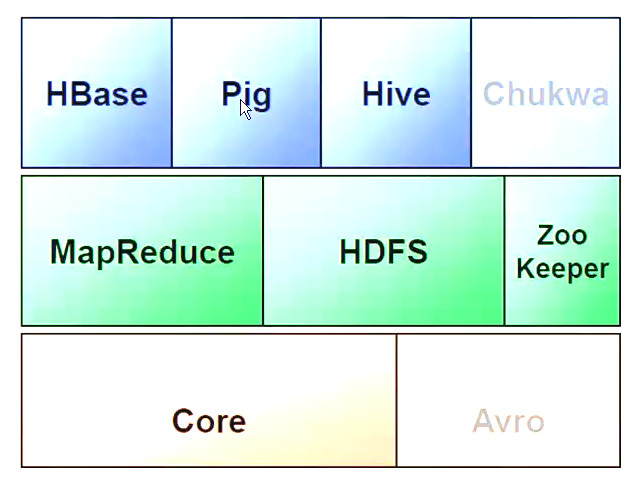
# Hadoop的技能



# Hadoop的特点

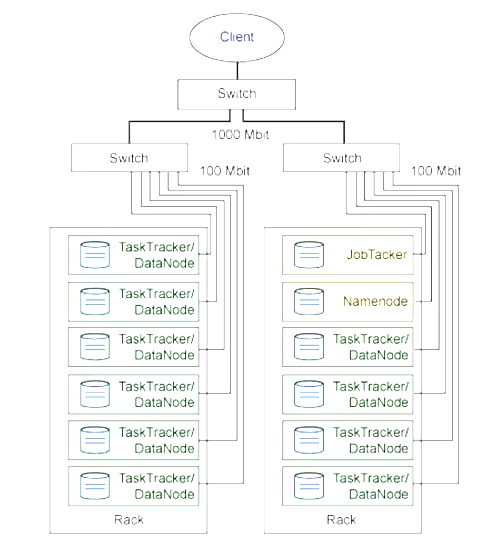
扩容能力、成本低、高效率、可靠性

# Hadoop家族



# Hadoop的架构

## Hadoop集群的物理分布

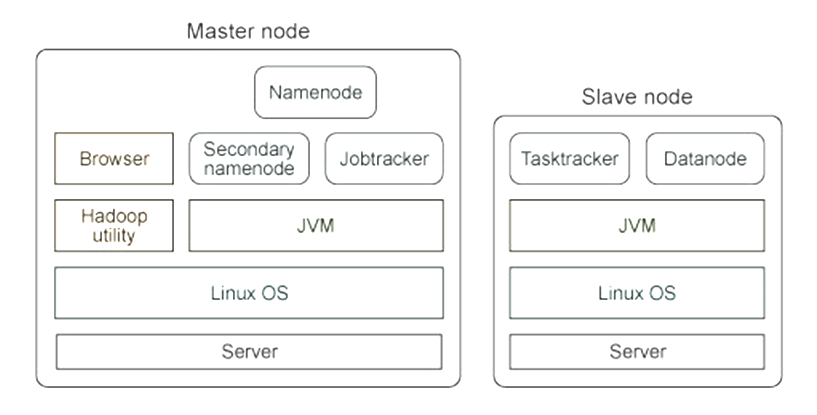


Master：NameNode、SecondaryNameNode、JobTracker

Slave：TaskTracker、DataNode

Master不是唯一的

## 单结点物理结构



# MapReduce的架构

主从结构

主节点，一个：JobTracker

从节点，多个：TaskTracker

## JobTracker



用于处理作业的后台程序

接收客户提交的计算任务

把任务分配给TaskTracker执行

决定有哪些文件参与处理，然后切割task并分配节点

监控tasktracker，重启失败的task

每个集群只有一个JobTracker，位于master节点

## TaskTracker

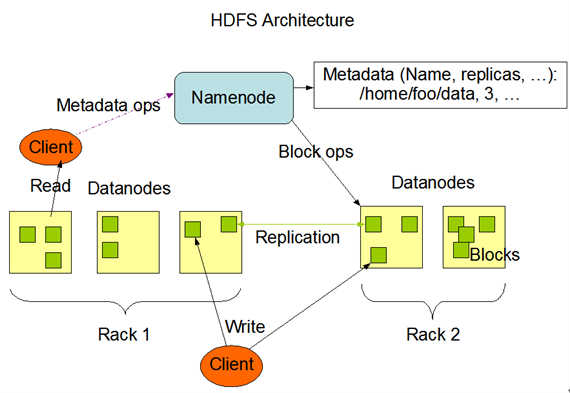
位于slave节点，与datanode结合

管理各自节点的task（JobTracker分配）

每个节点只有一个tasktracker，但一个tasktracker可以启动多个jvm，用于并行执行map或reduce任务

与jobtracker交互，执行JobTracker分配的计算任务

**# HDFS体系结构**



**HDFS的架构**

主从结构

**## NameNode**

HDFS的守护程序

本质是一个jetty服务器

对内存和IO进行集中管理

接收用户操作请求

提供名称查询服务，是一个jetty服务器。

保存的metadata信息（文件owership和permission、跟踪文件如何被分割成文件块，文件块又被哪些节点存储，即文件数据块列表）

NameNode是单点

**metadata**存储到**fsimage**



## DataNode:

保存块

HDFS数据块读写到本地文件系统

启动DN线程时向NN汇报block信息

向NN发送心跳保持联系

Block的副本放置策略：

第一个副本，放置在本地机架的节点上

第二个副本，第二个副本放在同一机架的另一个节点上

第三个副本，放置在不同机架的节点上

Block默认64MB，不够默认仍占用一个block的命名空间，但物理上并不会。

Block大小和副本数由client端上传文件时hdfs设置，其中副本数可以变更，block不可以上传后变更。

## 数据损坏

DN读取block的时候，会计算checksum

## SecondaryNameNode（SNN）

它不是NN的热备份

它可以作为冷备份：

将本地保存的fsimage导入

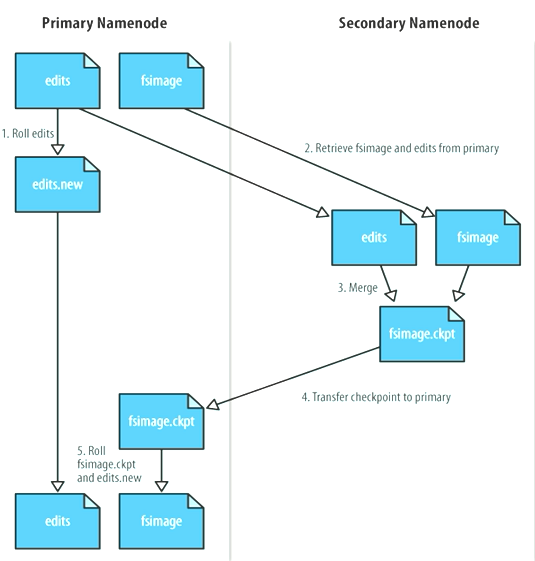
修改cluster所有DN的NameNode地址

修改所有client端NameNode地址

或者修改SNN IP为原NN IP

它的工作是帮助NN合并edits，减少NN启动时间

## SecondaryNameNode



简单描述一下具体流程：

步骤一：SSN在一个checkpoint时间点和NameNode进行通信，请求NameNode停止使用edits文件记录相关操作而是暂时将新的Write操作写到新的文件edit.new来。

步骤二：SSN通过HTTP GET的方式从NameNode中将fsimage和edits文件下载回来本地目录中。

步骤三：SSN中合并edits和fsimage。SSN将从NameNode中下载回来的fsimage加载到内存中，然后逐条 执行edits文件中的各个操作项，使得加载到内存中的fsimage中包含edits中的操作，这个过程就是所谓的合并了。

步骤四：在SSN中合并完fsimage和edits文件后，需要将新的fsimage回传到NameNode上，这个是通过HTTP POST方式进行的。

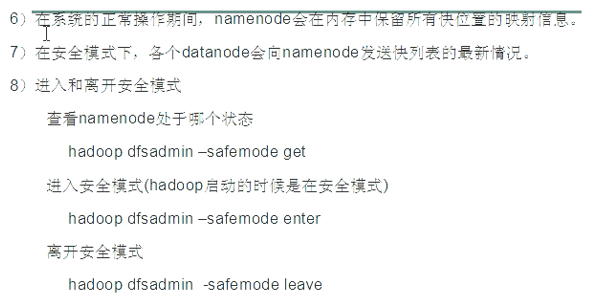
步骤五：NameNode将从SSN接收到的新的fsimage替换掉旧的fsimage。同时将edits.new文件转换为通常的edits文件，这样edits文件的大小就得到减少了。SSN整个合并以及和NameNode的交互过程到这里已经结束。

## HDFS权限

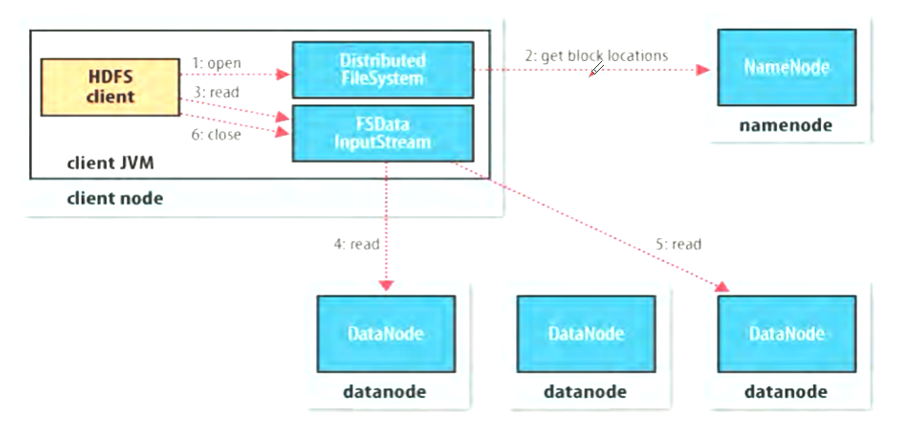
rwx权限

## 安全模式

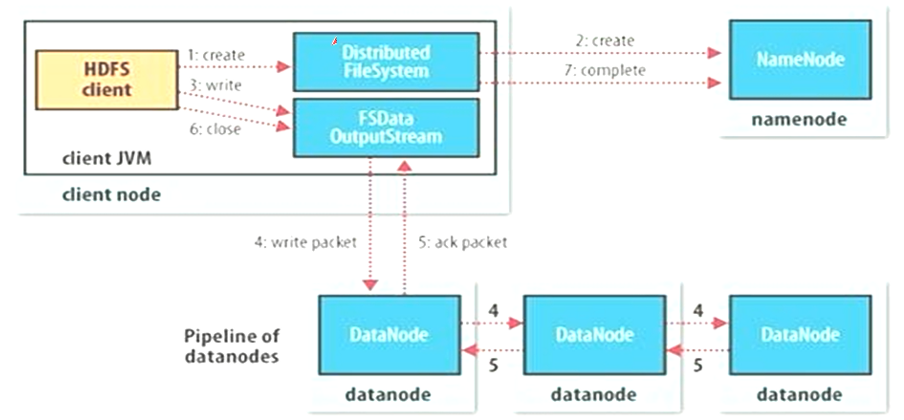




## HDFS文件读取



## HDFS文件写入



## HDFS开发常用命令

列出hdfs下的文件

hadoop fs –ls 目录

如列出HDFS目录下某个文档中的文件

hadoop fs –ls 目录（HDFS目录）

递归查看

hadoop fs –lsr 目录

创建文件夹

hadoop fs –mkdir /d1

上传文件

hadoop fs –put <linux-source> <hdfs-destination>

hadoop fs –put /root/a.txt /d1

注意：若/d1文件夹不存在，则/d1默认为文件名

同理hadoop fs –put /root/a.txt /d1/newa.txt 则被修改为文件名

将HDFS中的文件复制到本地系统中

hadoop fs –get <hdfs -souce> <linux-destination>

hadoop fs -get /d1/a.txt /root

删除HDFS下的文档

hadoop fs -rm /d1/a.txt

hadoop fs -rmr /d1 递归删除

查看HDFS下的某个文件

hadoop fs –text /d1/a.txt

管理与更新

hadoop dfsadmin -report

安全模式

退出 hadoop dfsadmin -safemode leave

进入 hadoop dfsadmin -safemode enter

拷贝

hadoop fs –cp <src><des>

拷贝多个文件时<des>需要是文件夹

移动

hadoop fs –mv <src><des>

其他命令

hadoop fs自行查看

hadoop fs –help

注意hdfs应使用绝对路径 如果是相对路径 默认是/user/<currentUser>

HDFS适用于一次写入多次查询，不支持并发，小文件不适合

# 安装和配置

## hadoop环境

虚拟机VMware Workstation10

ubuntu 14.04

jdk-6u24-linux-.bin

hadoop-1.1.2.tar.gz

## 伪分布式安装

注意：以下操作均使用root用户登录，不是su root

**关闭防火墙** （主机和虚拟机ufw disable）

关闭虚拟机防火墙

相关命令ufw disable|enable ufw status

**修改ip （桥接模式 静态IP）**

**编辑文件/etc/network/interfaces:  
sudo vi /etc/network/interfaces  
**

注意：

1、网卡设置生效sudo /etc/init.d/networking restart，如果无法生效改用sudo ifdown eth0 && sudo ifup eth0或者**ifconfig eth0 down/up**

~~2、注意：如果Ubuntu系统采用的是desktop版，由于desktop版安装了NetworkManager，修改完interfaces文档中的内容后，不会生效。需要先修改/etc/NetworkManager/NetworkManager.conf文档中的managed参数，使之为true，并重启系统， 然后在修改/etc/network/interfaces文件，设置静态IP。~~我的做法是删除NetworkManager然后设置vim /etc/resolv.conf和vim /etc/resolvconf/resolv.conf.d/base然后resolvconf -u

```

最后的配置

***auto lo***

***iface lo inet loopback***

***auto eth0***

***iface eth0 inet static***

***address 192.168.3.145***

***gateway 192.168.3.1***

***netmask 255.255.255.0***

***network 192.168.0.0***

***broadcast 192.168.0.255***

**```**

**修改hostname以及绑定**

1、ssh登录需要安装openssh

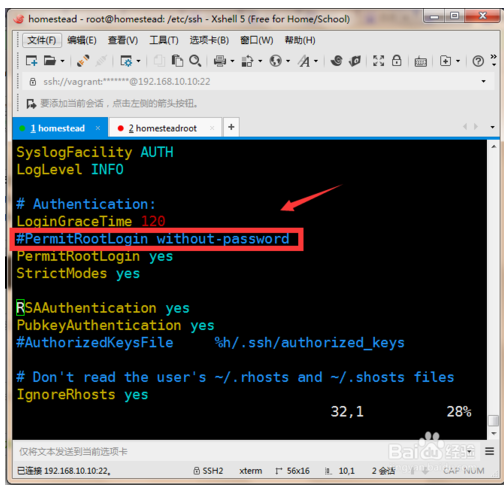
2、xshell普通用户直连 root用户拒绝

========解决方法==============

a、确认ssh服务已安装，通过普通用户连接成功；

b、确认ssh配置是否对root进行特殊设置，修改/etc/ssh/sshd\_config文件中

PermitRootLogin without-password将 without-password改为yes；



c、重启ssh服务/etc/init.d/ssh restart；

======================

3、解决修改主机名重启失效

========解决方法=============

a、centos里的/etc/sysconfig/network对应ubuntu里的/etc/network/interfaces文件，修改NETWORKING=yes HOSTNAME=hadoop 无效。（适用centos）

正确的方法：主机名存放在/etc/hostname（适用ubuntu）

b、hostname绑定

vim /etc/hosts

===============================

**设置ssh自动登录 （虚拟机ubuntu安装openssh）**

<1> 执行命令ssh-keygen –t rsa产生密钥，位于目录~/.ssh/

<2> cp id\_rsa.pub authorized\_keys

<3> 验证 ssh localhost

**安装jdk**

**安装hadoop**

<1> 配置HADOOP\_HOME环境变量

<2> 修改hadoop的4个配置文件 $HADOOP\_HOME/conf下

**hadoop-env.sh**

配置

```export JAVA\_HOME=/usr/local/jdk1.6.0\_24```

**core-site.xml**

配置

```

<configuration>

<property>

<name>fs.default.name</name>

<value>hdfs://hadoop:9000</value>

<description>改为自己的主机名</description>

</property>

<property>

<name>hadoop.tmp.dir</name>

<value>/usr/local/hadoop/tmp</value>

</property>

</configuration>

```

**hdfs-site.xml**

配置

```

<configuration>

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

<property>

<name>dfs.permissions</name>

<value>false</value>

</property>

</configuration>

```

**mapred-site.xml**

配置

```

<configuration>

<property>

<name>mapred.job.tracker</name>

<value>hadoop:9001</value>

<description>改为自己的主机名</description>

</property>

</configuration>

```

<3> Hadoop格式化

命令hadoop namenode -format

<4> 启动与验证

start-all.sh jps（5个守护进程）

NameNode界面: 浏览器查看hadoop:50070

Map/Reduce界面: 浏览器查看hadoop:50030

Windows/mac下查看 修改hosts

其他问题：

1、多次格式化hadoop错误？

解决方法：删除/usr/local/hadoop/tmp文件夹，重新格式化

2、解决$HADOOP\_HOME is deprecated

令$HADOOP\_HOME\_WARN\_SUPPRESS=0

3、从本地往hdfs文件系统上传文件

1）关闭防火墙

2）禁用selinux

编辑 "/etc/selinux/config"文件，设置"SELINUX=disabled"

更多问题参考http://www.linuxidc.com/Linux/2016-02/128149.htm

## **Vmware的三种网络方式**

Vmnet0 桥接

Vmnet1 主机模式

Vmnet8 NAT模式

Vmware虚拟机三种网络模式详解

*地址：http://blog.csdn.net/noob\_f/article/details/51099040*

桥接模式Vmnet0

什么是桥接模式？桥接模式就是将主机网卡与虚拟机虚拟的网卡利用虚拟网桥进行通信。在桥接的作用下，类似于把物理主机虚拟为一个交换机，所有桥接设置的虚拟机连接到这个交换机的一个接口上，物理主机也同样插在这个交换机当中，所以所有桥接下的网卡与网卡都是交换模式的，相互可以访问而不干扰。在桥接模式下，虚拟机ip地址需要与主机在同一个网段，如果需要联网，则网关与DNS需要与主机网卡一致。

适用： 桥接模式配置简单，但如果你的网络环境是ip资源很缺少或对ip管理比较严格的话，那桥接模式就不太适用了。

NAT地址转换模式（Vmnet8）

如果你的网络ip资源紧缺，但是你又希望你的虚拟机能够联网，这时候NAT模式是最好的选择。NAT模式借助虚拟NAT设备和虚拟DHCP服务器，使得虚拟机可以联网。

主机模式（VMnet1）

Host-Only模式其实就是NAT模式去除了虚拟NAT设备，然后使用VMware Network Adapter VMnet1虚拟网卡连接VMnet1虚拟交换机来与虚拟机通信的，Host-Only模式将虚拟机与外网隔开，使得虚拟机成为一个独立的系统，只与主机相互通讯。

###

**Vmware虚拟机-Ubuntu14.04**

**IP地址：192.168.3.117**

**账户saber密码1992**

**账户root密码root**

## MapReduce

1、MapReduce工作机制



2、调度机制

缺省为先入先出作业调度

支持公平调度器

支持容量调度器

3、任务执行优化机制

推测式执行：即如果jobtracker发现有拖后腿的任务，会再启动一个相同的备份任务，然后哪个先执行完就会kill掉另一个。因此在监控网页上经常能看到正常执行完的作业有被kill掉的任务。

推测式执行缺省打开，但如果是代码问题，并不能解决问题，而且会使集群变慢，通过在mapred-site.xml配置文件中设置mapred.map.tasks.speculative.execution和mapred.reduce.tasks.speculative.execution可为map或reduce任务开启或关闭推测式执行。

重用JVM，可以省去启动新的JVM消耗的时间和资源，在mapred-site.xml中设置mapred.job.reuse.jvm.num.tasks设置单个JVM上运行的最大任务数（1，>1和-1），-1表示没有限制，系统安全。

忽略模式，任务在读取数据失败2次后，会把数据位置告诉jobtracker，后者重新启动该任务并且在遇到所记录的坏数据时自动跳过。

4、错误处理机制：硬件故障

·硬件故障是指jobtracker故障或tasktracker故障

·jobtracker是单点，若发生故障目前hadoop还无法处理，所以选择牢固的硬件为jobtracker

·jobtracker通过心跳（周期为1分钟）了解tasktracker是否发生故障或负载过于严重

·jobtracker从任务节点中移除发生故障的tasktracker

·如果发生故障的节点在执行map任务并且尚未完成，jobtracker会要求其他节点重新执行此map任务

·如果故障节点在执行reduce任务并且尚未完成，jobtracker会要求其他节点继续执行尚未完成的reduce任务

错误处理机制：任务失败

·由于代码缺陷或进程崩溃引起任务失败

·JVM自动退出，向tasktracker父进程发送错误信息，错误信息被写入日志

·tasktracker监听程序若发现进程退出，或进程很久没有更新信息送回，将任务标记为失败

·标记失败任务后，任务计数器减一以便接受新任务，并通过心跳信号告诉jobtracker任务失败的信息。

·jobtracker获悉任务失败后，将把该任务重新放入调度队列，重新分配出去再执行

·如果一个任务失败超过4次（可以设置），将不会再被执行，同时作业也宣布失败

## Hadoop的生态

Pig

Zookeeper

Hbase

HiveSqoop

Avro

Chukwa

Cassandra

Pig：

类似于SQL的面向数据流的语言Pig Latin

Pig Latin可以完成排序，过滤，求和，聚组，关联和自定义函数

Pig 自动把Pig Latin映射为Map-Reduce作业上传集群运行

三种运行方式：Grunt Shell、脚本方式、嵌入式

Hbase

Google Bigtable的开源实现

列式数据库

可集群化

可使用shell、web、api等方式访问

适合高读写（insert）的场景

HQL查询语言

NoSQL

Hive

数据仓库工具

支持一种与SQL几乎完全相同的语言HiveQL，除了不支持更新、索引和事务，几乎SQL的其他特征都能支持

看成SQL到Map-Reduce的映射器

提供shell、JDBC/ODBC、Thrift、Web等接口

Zookeeper

Google Chubby的开源实现

通讯协调软件

用于协调分布式系统上的各种服务。例如确认消息是否准备到达，防止单点失效，处理负载均衡。

Zookeeper 分布式服务框架是 Apache Hadoop 的一个子项目，它主要是用来解决分布式应用中经常遇到的一些数据管理问题，如：统一命名服务、状态同步服务、集群管理、分布式应用配置项的管理等。

典型应用场景（配置文件的管理、集群管理、同步锁、Leader 选举、队列管理等）

Sqoop

用于在Hadoop和关系型数据库之间交换数据

通过JDBC接口连入关系型数据库

Avro

数据序列化工具

用于支持大批量数据交换的应用。支持二进制序列化方式，便捷、快速的处理大量数据。

Thrift接口

动态语言

Chukwa

架构在Hadoop之上的数据采集与分析框架

主要进行日志采集和分析

通过安装在收集节点的“代理”采集最原始的日志数据

代理将数据发给收集器

收集器定时将数据写入Hadoop集群

指定定时启动的Map-Reduce作业对数据进行加工处理和分析

Hadoop的基础管理中心（HICC）最终展示数据

Cassandra

NoSQL，分布式的Key-Value型数据库，由Facebook贡献

与Hbase类似，借鉴Google Bigtable的思想体系

只有顺序写，没有随机写的设计，满足高负荷情形的性能需求

**## Hbase**

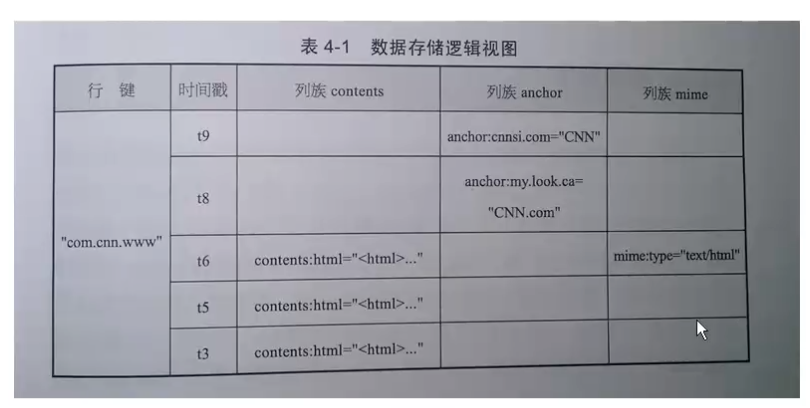
适合非结构化数据库的存储

Hbase逻辑模型

以表的形式存放数据

表由行与列组成，每个列属于某一个列族，由行与列确定的存储单元称为元素

每个元素保存了同一个数据的多个版本，由时间戳来标识区分



### 行键

行键是数据在表里的唯一标识，并作为检索记录的主键

访问表里的行只有三种方式

1、通过单个行键

2、给定行键的范围访问

3、全表扫描

行键可以是最大长度不超过64kb的任意字符串，并按照字典序存储

### 列族与列

列表示为为<列族>:<限定符>

Hbase在磁盘上按照列族存储数据

列族里的元素最好具有相同的读写方式

### 时间戳

Habse支持两种数据回收方式 1、每个数据单元只存储最新版本 2、指定长度

常见的客户端时间查询：某个时刻起的最新数据或给我全部版本的数据

元素由行键，列族：限定符，时间戳唯一确定

元素以字节码形式存放，没有类型之分

## Hbase物理模型

### Region和Region服务器

表在行方向上，按照行键划分若干的Region

每个表最初只有一个Region，当记录超过某个阈值时，开始分裂成两个Region

物理上所有数据存放在HDFS，由Region服务器提供Region的管理

一台物理节点只能跑一个HRegionServer

一个HRegionServer可以管理多个Region实例

一个Region实例包括Hlog日志和存放数据的Store

Hmaster为总控节点

Zookeeper负责调度

两张特殊的表

-ROOT-和.META.表

.META.表：记录了用户的Region信息，.META.可以有多个Region

-ROOT-:记录了.META.表的Region信息，-ROOT-只有一个Region

Zookeeper中记录了-ROOT-的location

### Memstore与storefile

一个region由多个store组成，每个store包含一个列族的所有数据

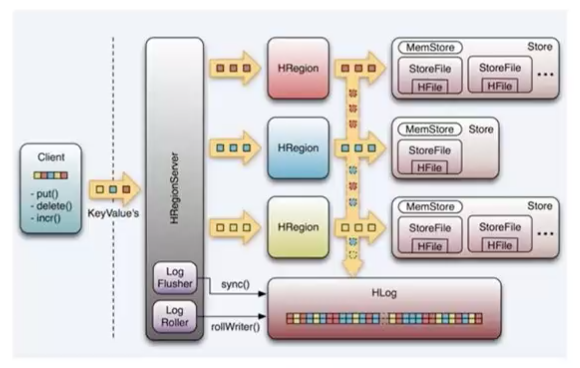
store包括位于内存的memstore和硬盘的storefile

写数据先进入memstore达到阈值HregionServer会启动flashcache进程写入storefile，每次写入形成一个单独的storefile

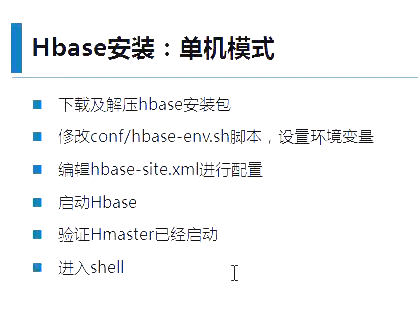
storefile的合并，进行版本合并和删除工作，形成更大的storefile

更大的storefile达到一定阈值后，会把当前region分割为两个，由Hmaster分配到相应的region服务器，实现负载均衡

客户端检索数据先在memstore查找，再找storefile



HBase安装：单机模式



Hadoop版本1.1.2

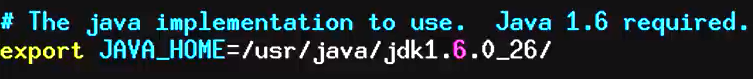
HBase版本0.90.5

## 单点模式

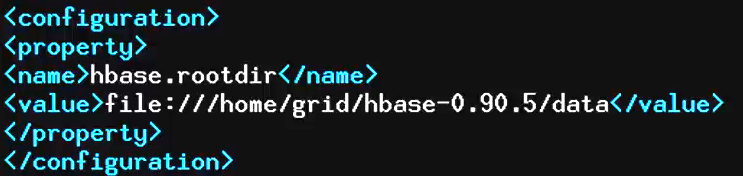
安装步骤：

1、修改**hbase-env.sh**

**JAVA\_HOME**



2、修改**hbase-site.xml**



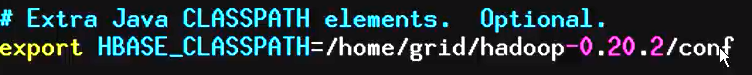
3、启动及验证

**start-hbase.sh**

## 伪分布式模式

1、修改**hbase-env.sh**

单点模式上继续添加HBASE\_CLASSPATH

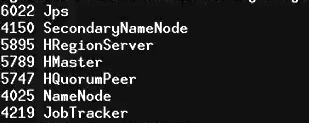


2、hbase-site.xml打开分布模式  


3、覆盖hadoop核心jar包

查看版本依赖

**http://hbase.apache.org/book.html#configuration**



**## 完全分布式**



1、hbase-env.sh



2、hbase-site.xml

```

<configuration>

<property>

<name>hbase.rootdir</name>

<value>hdfs://Master.Hadoop:9000/hbase</value>

</property>

<property>

<name>hbase.cluster.distributed</name>

<value>true</value>

</property>

<property>

<name>hbase.master</name>

<value>Master.Hadoop:60000</value>

</property>

<property>

<name>hbase.zookeeper.quorum</name>

<value>Master.Hadoop,Slave1.Hadoop,Slave2.Hadoop</value>

</property>

<property>

<name>hbase.tmp.dir</name>

<value>/usr/local/hbase/tmp</value>

</property>

</configuration>

```

3、配置regionservers文件

节点IP

**## HBase Shell**

<命令无需空格>

1、status

2、version

3、表相关

**create** ‘member’,’member\_id’,’address’,’info’

**list** 列出所有表

**describe** ‘member’ 查看具体表结构

**对列族的改变：alter、disable、enable**

e.g.:alter ‘member’,{NAME=>’member\_id’,METHOD=>’delete’}(变动前先置disable)

e.g.:disable ‘member’

e.g.:enable ‘member’

**drop**

e.g.:drop’member’ (删除表之前先置disabled)

**exists**

e.g.:exists‘member’

判断是否在线 is\_enabled

判断是否离线 is\_disabled

4、插入（更新数据）

put'member',’linekey1’,'info:age','24'

put'member',’linekey1’,'info:birthday','1987-06-17'

put'member',’linekey1’,'info:company','alibaba'

put'member',’linekey1’,'address:contry','china'

put'member',’linekey1’,'address:province','zhejiang'

put'member',’linekey1’,'address:city','hangzhou'

put'member','linekey2','info:birthday','1987-4-17'

put'member','linekey2','info:favorite','movie'

put'member','linekey2','info:company','alibaba'

put'member','linekey2','address:contry','china'

put'member','linekey2','address:province','guangdong'

put'member','linekey2','address:city','jieyang'

put'member','linekey2','address:town','xianqiao'

5、查询数据

get ‘member’,’linekey1’（查询指定行键的所有数据）

get ‘member’,’linekey1’,’info’（查询指定行键和列族的所有数据）

get ‘member’,’linekey1’,’info:age’（查询指定行键和列族的指定列的所有数据）

<注意：返回最新版本>

6、通过timestamp获取数据

get ‘member’,’linekey1’,{COLUMN=>’info:age’},TIMESTAMP=>1321586238965}

7、全表扫描

scan ‘member’

8、删除

delete ‘member’,’linekey1’,’info:age’(删除指定行键指定列族的指定列)

deleteall ‘member’, ‘linekey1’(删除整行)

9、统计

count ‘member’(查询一共多少行ps：针对于行键)

10、清空表

truncate ‘member’

实际过程是：diabled -> drop -> create

## 什么情况下使用HBase

成熟的数据分析主题，查询模式已经确立并且不轻易改变

传统的关系型数据库已经无法承受负荷，高速插入，大量读取

适合海量的，但同时是简单的操作。（例如key/value存储）

以当当最近图书浏览足迹为例

## 关系型数据库的困难

简单的事情只要上了量就会变成无比复杂的事情

OrderBy耗费很多性能

大量发生，但又无法分布式处理

顾客需要实时查看自己的足迹，因此不能使用缓存技巧

## HBase迎接挑战

基于行键的B+树查询异常快速，特别是最近的数据被放在内存的memstore里，没有IO开销。

分布式化解负荷

## 模式设计

行键：userid

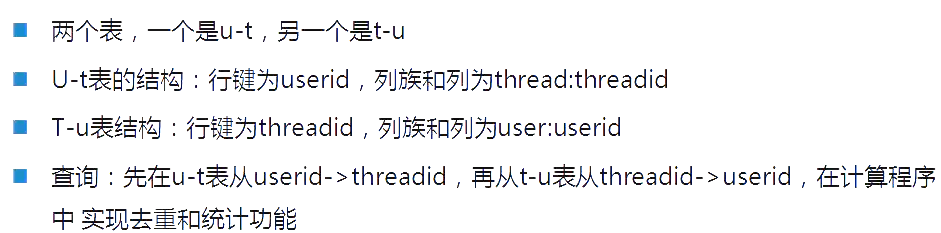
列族和列：book：bookid

为了充分利用分布式，可以用reverse key，hash等技巧

HBase的分布式是根据行键的分布范围来分布出去。为此需要分散key。

tag：推荐系统1000w行记录，mysql 9s。

表设计与查询实现



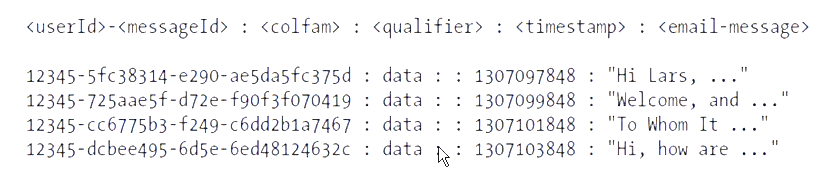
**## HBase辅助索引**



**##** **复合行键设计**

**<userId> - <messageId>**

**复合行键+范围查询**



起始12345终止123456

## 其他

Linux的awk脚本生成复制hadoop、密钥等文件

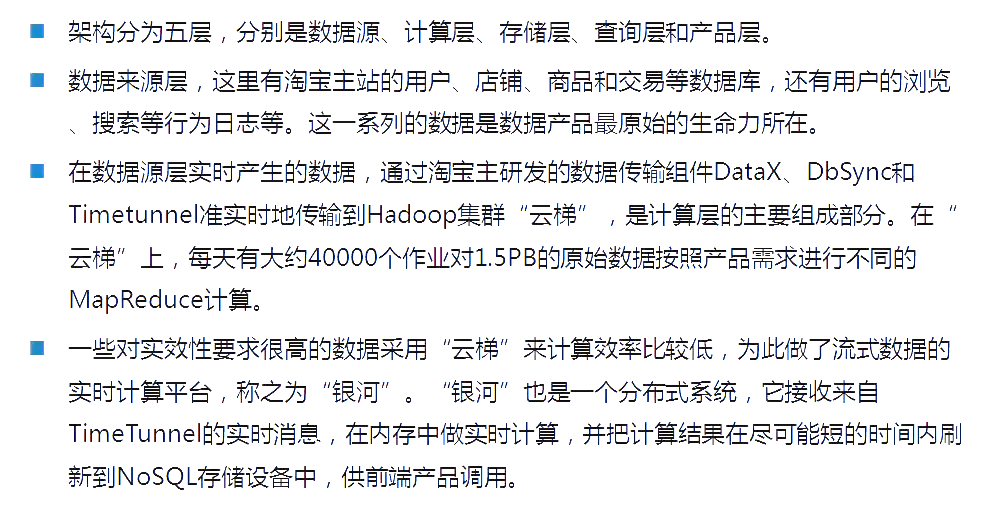
cat ./slave | awk ‘{print “scp -rp ./hadoop-1.1.2 root@”$1”:/home/saber”}’

## Hadoop@淘宝

1、架构

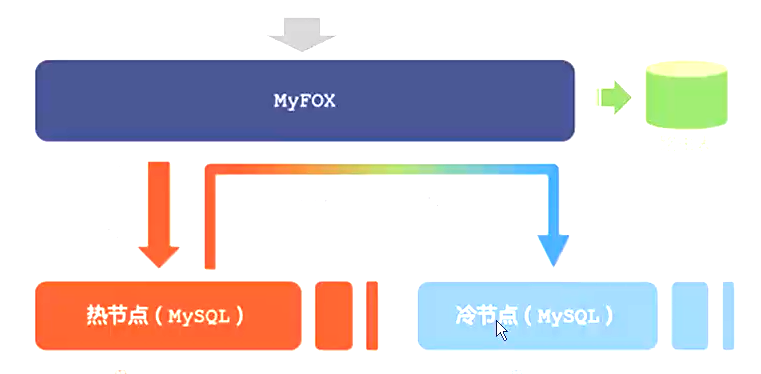


2、架构图介绍

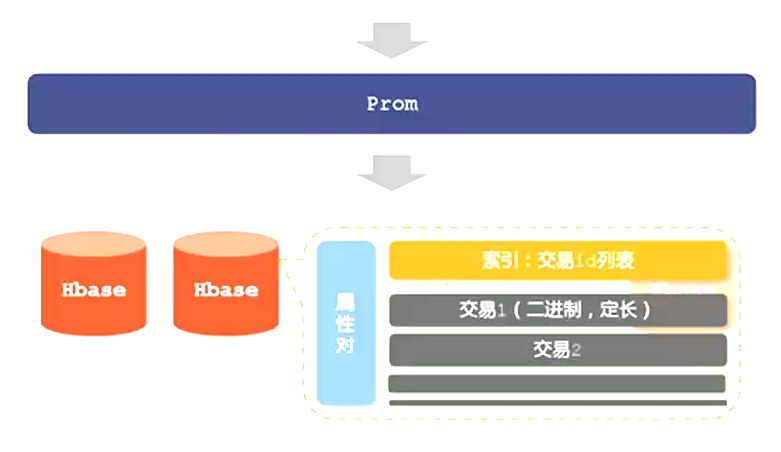


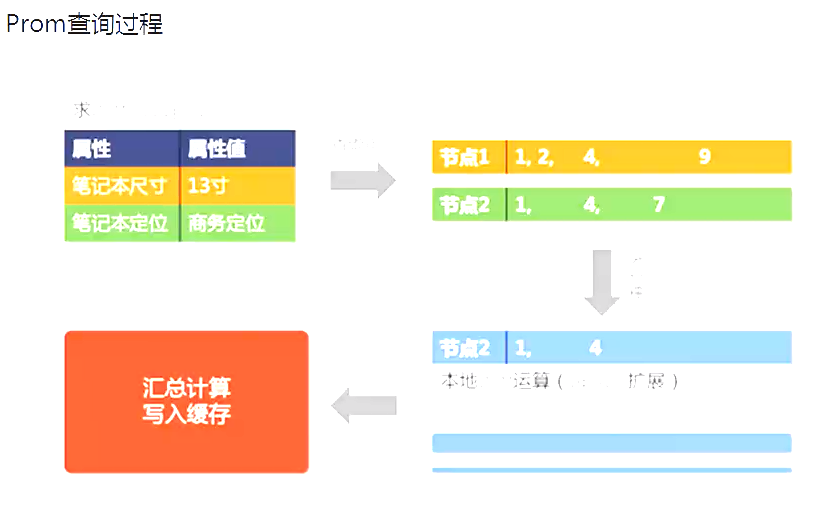
3、存储层MyFox（mysql集群）、Prom

MyFox节点架构



Prom的存储结构（Prometheus HBase集群）

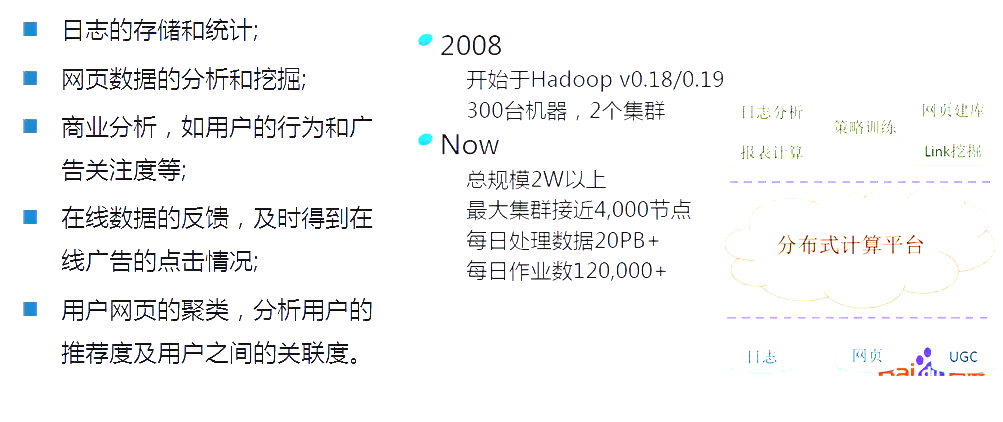




## Oceanbase

淘宝开发的分布式数据库，NoSQL

##Hadoop@Baidu

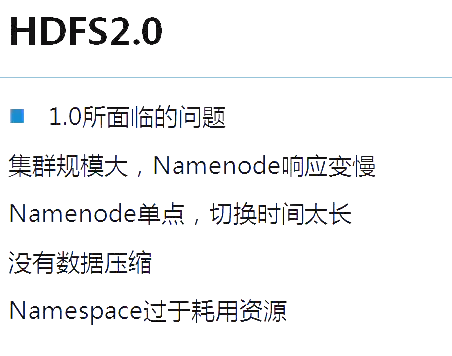


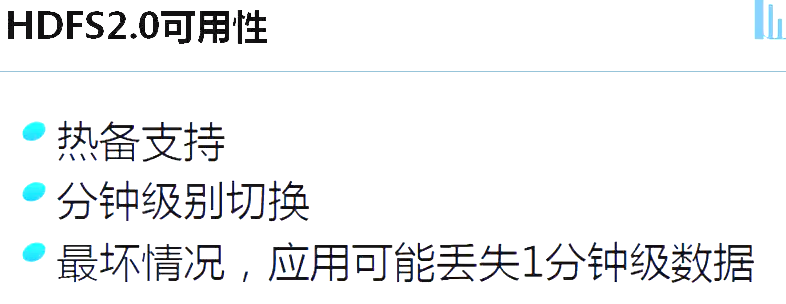
数据统计：2013年

### 百度Hadoop集群面临的挑战



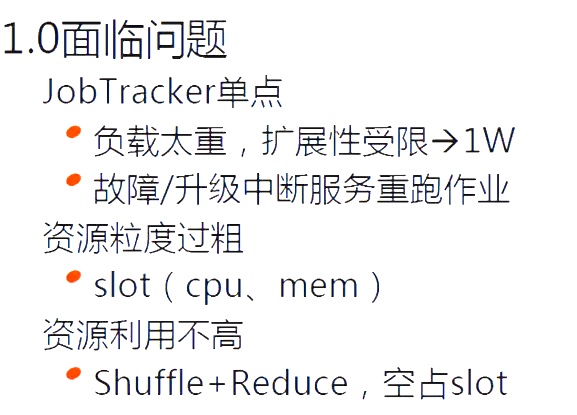
### 百度的HDFS2.0







### MapReduce2.0





### HBase系统架构及数据结构

http://www.open-open.com/lib/view/open1346821084631.html

**HLog(WAL log)**

WAL 意为Write ahead log(http://en.wikipedia.org/wiki/Write-ahead\_logging)，类似mysql中的binlog,用来 做灾难恢复只用，Hlog记录数据的所有变更,一旦数据修改，就可以从log中进行恢复。

每 个Region Server维护一个Hlog,而不是每个Region一个。这样不同region(来自不同table)的日志会混在一起，这样做的目的是不断追加单个文件相对于同时写多个文件而言，可以减少磁盘寻址次数，因此可以提高对table的写性能。带来的麻烦是，如果一台region server下线，为了恢复其上的region，需要将region server上的log进行拆分，然后分发到其它region server上进行恢复。

HLog 文件就是一个普通的Hadoop Sequence File，Sequence File 的Key是HLogKey对象，HLogKey中记录了写入数据的归属信息，除了table和region名字外，同时还包括 sequence number和timestamp，timestamp是”写入时间”，sequence number的起始值为0，或者是最近一次存入文件系统中sequence number。HLog Sequece File的Value是HBase的KeyValue对象，即对应HFile中的KeyValue，可参见上文描述。

**系统架构**

**Client**

1 包含访问hbase的接口，client维护着一些cache来加快对hbase的访问，比如regione的位置信息。

**Zookeeper**

1 保证任何时候，集群中只有一个master

2 存贮所有Region的寻址入口。（即-ROOT-表位于的RegionServer位置）

3 实时监控Region Server的状态，将Region server的上线和下线信息实时通知给Master

4 存储Hbase的schema,包括有哪些table，每个table有哪些column family

**Master**

1 为Region server分配region

2 负责region server的负载均衡

3 发现失效的region server并重新分配其上的region

4 GFS上的垃圾文件回收

5 处理schema更新请求

**Region Server**

1 Region server维护Master分配给它的region，处理对这些region的IO请求

2 Region server负责切分在运行过程中变得过大的region

可以看到，client访问hbase上数据的过程并不需要master参与（寻址访问zookeeper和region server，数据读写访问regione server），master仅仅维护者table和region的元数据信息，负载很低。

**读写过程**  
读过程

hbase使用MemStore和StoreFile存储对表的更新。

数据在更新时首先写入Log(WAL log)和内存(MemStore)中，MemStore中的数据是排序的，当MemStore累计到一定阈值时，就会创建一个新的MemStore，并 且将老的MemStore添加到flush队列，由单独的线程flush到磁盘上，成为一个StoreFile。于此同时，系统会在zookeeper中 记录一个redo point，表示这个时刻之前的变更已经持久化了。(minor compact)当系统出现意外时，可能导致内存(MemStore)中的数据丢失，此时使用Log(WAL log)来恢复checkpoint之后的数据。

前面提到过StoreFile是只读的，一旦创建后就不可以再修改。因此Hbase的更新其实是不断追加的操作。当一个Store中的StoreFile达到一定的阈值后，就会进行一次合并(major compact),将对同一个key的修改合并到一起，形成一个大的StoreFile，当StoreFile的大小达到一定阈值后，又会对StoreFile进行split，等分为两个StoreFile。

由于对表的更新是不断追加的，处理读请求时，需要访问Store中全部的StoreFile和MemStore，将他们的按照row key进行合并，由于StoreFile和MemStore都是经过排序的，并且StoreFile带有内存中索引，合并的过程还是比较快。

写过程

1 client向region server提交写请求

2 region server找到目标region

3 region检查数据是否与schema一致

4 如果客户端没有指定版本，则获取当前系统时间作为数据版本

5 将更新写入WAL log

6 将更新写入Memstore

7 判断Memstore的是否需要flush为Store文件。

-ROOT-表和.META.表结构详解

注意两点：

1. Client端在访问数据的过程中并没有涉及到Master节点，也就是说HBase日常的数据操作并不需要Master，不会造成Master的负担。
2. 并不是每次数据访问都要执行上面的整个流程，因为很多数据都会被Cache起来。

从存储结构和操作方法的角度来说， **-ROOT-** 、 **.META.** 与其他表没有任何区别。它们与众不同的地方是HBase用它们来存贮一个重要的系统信息：

* -ROOT-：记录.META.表的Region信息。
* .META.：记录用户表的Region信息。

其中-ROOT-表本身只会有一个region，这样保证了只需要三次跳转，就能定位到任意region。

3. 说明：

3.1

root region永远不会被split，保证了最需要三次跳转，就能定位到任意region 。

3.2

.META.表每行保存一个region的位置信息。

3.3

为了加快访问，.META.表的全部region都保存在内存中。

假设，.META.表的一行在内存中大约占用1KB。并且每个region限制为128MB。

那么上面的三层结构可以保存的region数目为：

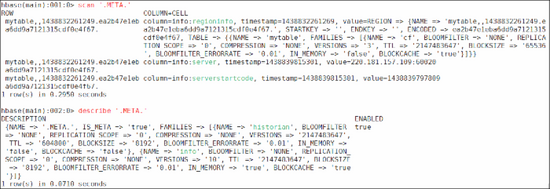
(128MB/1KB) \* (128MB/1KB) = = 2(34)个region。

3.4

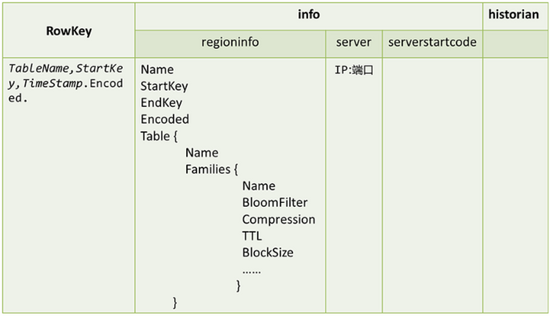
client会将查询过的位置信息保存缓存起来，缓存不会主动失效，因此如果client上的缓存全部失效，则需要进行6次网络来回，才能定位到正确的region(其中三次用来发现缓存失效，另外三次用来获取位置信息)。

## 一、META表结构

在 HBase Shell 里对 **.META.** 表进行 scan 和 describe ：



可以看出， **.META.** 表的结构如下：



**.META.** 表中每一行记录了一个Region的信息。

1) RowKey

RowKey就是Region Name，它的命名形式是 **TableName,StartKey,TimeStamp.Encoded.** 。

其中 Encoded 是 **TableName,StartKey,TimeStamp** 的md5值。

例如：

mytable,,1438832261249.ea2b47e1eba6dd9a7121315cdf0e4f67.

表名是 **mytable** ，StartKey为空，时间戳是 **1438832261249** ，前面三部分的md5是：

$ echo -n "mytable,,1438832261249" | md5sum # -n选项表示不输出换行符  
ea2b47e1eba6dd9a7121315cdf0e4f67 -

2) Column Family

**.META.** 表有两个Column Family： **info** 和 **historian** 。

其中 **info** 包含了三个Column：

* **regioninfo** ：region的详细信息，包括StartKey、EndKey以及Table信息等等。
* **server** ：管理该region的 RegionServer 的地址。
* **serverstartcode** ：RegionServer 开始托管该region的时间。

至于 **historian** ：

That was a family used to keep track of region operations like open,

close, compact, etc. It proved to be more troublesome than handy so we

disabled this feature until coming up with a better solution. The

family stayed for backward compatibility.

大致的意思是：这个Column Family是用来追踪一些region操作的，例如open、close、compact等。事实证明这非常的麻烦，所以在想出一个更好的解决方案之前我们禁用了此功能。这个列族会保持向后兼容。

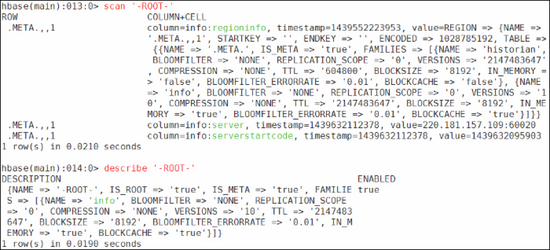
综上所述， **.META.** 表中保存了所有用户表的region信息，在进行数据访问时，它是必不可少的一个环节。当Region被拆分、合并或者重新分配的时候，都需要来修改这张表的内容 来保证访问数据时能够正确地定位region。

### 二、ROOT表结构

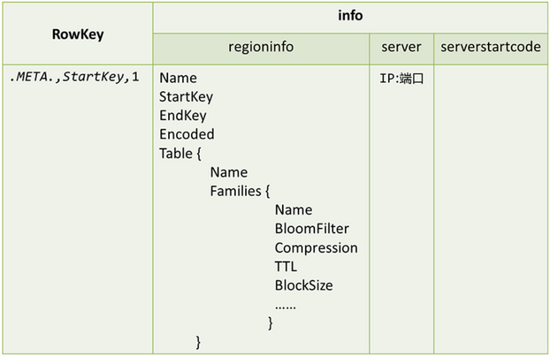
当用户表特别大时，用户表的region也会非常多。 **.META.** 表存储了这些region信息，也变得非常大，这时 **.META.** 自己也需要划分成多个Region，托管到多个RegionServer上。

这时就出现了一个问题： 当 **.META.** 被托管在多个RegionServer上，如何去定位 **.META.** 呢？ HBase的做法是用另外一个表来记录 **.META.** 的Region信息，就和 **.META.** 记录用户表的Region信息一样，这个表就是 **-ROOT-**表。

在 HBase Shell 里对 **-ROOT-** 表进行 scan 和 describe ：



**-ROOT-** 表的结构如下：



可以看出，除了没有 **historian** 列族之外， **-ROOT-** 表的结构与 **.META.** 表的结构是一样的。另外， **-ROOT-** 表的 RowKey 没有采用时间戳，也没有Encoded值，而是直接指定一个数字。

**-ROOT-** 表永远只有一个Region，也就只会存放在一台RegionServer上。—— 在进行数据访问时，需要知道管理 **-ROOT-** 表的RegionServer的地址。这个地址被存在 **ZooKeeper** 中。