1. Hbase减少节点：./bin/hbase-daemon.sh stop regionserver

若关闭节点时负载均衡还在运行，则在负载均衡和master恢复刚才下线的region服务器之间会产生竞争，若要避免，可设置 hbase>balance\_switch false

重新启用hbase>balance\_switch true

Region服务器会先将其管理的所有region关闭，再把自己的进程停止，region服务器在zookeeper中对应的临时节点会过期，master会注意到regionserver停止，并将其按崩溃处理，并将region重新分配到其他机器上

或采用让服务器逐渐减少负载并停止服务的方法，graceful\_stop.sh脚本

$ ./bin/graceful\_stop.sh hostname

1. 滚动重启

$ for I in `cat conf/regionservers | sort` ; do ./bin/graceful\_stop.sh \

--restart –reload –debug $i ; done &> /tmp/log.txt &

可以tail该文件查看进度

1. 新增服务器

修改配置文件，regionservers中添加机器ip，通过master节点运行start-hbase.sh即可。

1. 导入导出数据

Hbase登陆master节点，执行hbase-daemon.sh stop master,切换master

**一、需要在hbase集群停掉的情况下迁移**

步骤：(1)执行hadoop distcp -f filelist "hdfs://new cluster ip:9000/hbasetest"

(2)在new cluster执行./hbase org.jruby.Main add\_table.rb /hbase/table20111222，将拷贝来的表加入到.MEAT.中（会出现region的数目不一致的问题，这个需要重启hase才能解决）

说明：(1)filelist为hdfs上的列表文件，内容如下：

/hbase/table20111222

/hbase/table20120131

(2)如果两个集群的hadoop版本不一致，需要在new cluster上执行hadoop distcp，否则会出现读写异常；

**二、集群运行情况下迁移**

**1：export,imort工具**

步骤：(1)在old cluster上执行：./hbase org.apache.hadoop.hbase.mapreduce.Export test hdfs://new cluster ip:9000/zhuangyang/test

(2)在new cluster上执行：./hbase org.apache.hadoop.hbase.mapreduce.Import test hdfs://new cluster ip:9000/zhuangyang/test

说明：(1)一定要写全路径，不能写相对路劲;

(2)在import前，需要将表事先在new cluster中创建好.

**2、CopyTable：可以在本集群中拷贝一张表，也可以将表拷贝到其他的集群中。**

命令：./hbase org.apache.hadoop.hbase.mapreduce.CopyTable --peer.adr=new cluster ip:2181:/hbase zy\_test

说明：(1)拷贝完成，不需要重启机器，在new cluster中就可以看到该表;

(2)稳定性还需要考虑，测试过程中遇到一个问题，拷贝过程中始终都停留在这里Lookedup root region location, 查看日志没有什么错误和相关的信息。

**3、Replication：这个是动态的备份（可以理解为实时备份）**

步骤：(1)在old cluster将需要迁移的表属性进行修改：

disable 'your\_table'

alter 'your\_table', {NAME => 'family\_name', REPLICATION\_SCOPE => '1'}

enable 'your\_table'

(2)打开new cluster集群的replication，修改hbase-site.xml

<property>

<name>hbase.replication</name>

<value>true</value>

</property>

(3)添加peer，在new cluster的hbase shell中执行：add\_peer '1','old cluster ip:2181:/hbase'，启动replication，执行start\_replication

说明：需要两个集群的hadoop版本一致，否则出现读写错误

**拷贝文件distcp,put,get方式可以使用下面命令生成meta信息，已测可行**

hbase org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFiles -Dhbase.mapreduce.bulkload.max.hfiles.perRegion.perFamily=1024 hdfs://tianxi-ha/test/hbaseout/aa812523baecf01abf46f14410df8d83 'test:tb'

修复.META.表：hbase hbck -fixMeta

查看该表的meta数据： hbase(main):001:0> scan 'hbase:meta'

重新分配数据到各RegionServer：hbase hbck -fixAssignments

## META.数据丢失恢复

进入HBase shell客户端执行如下命令修改.META. Region所在的RegionServer

put '.META.', 'Region名','info:server','160-172-0-13:26003'

执行./hbase hbck -repair

Snapshot方式

hbase> snapshot 'src\_table', 'snapshot\_src\_table' #查看创建的快照，可用list\_snapshots命令 hbase> list\_snapshots #如果快照创建有问题，可以先删除，用delete\_snapshot命令 hbase >delete\_snapshot 'snapshot\_src\_table'

创建完快照后在/hbase根目录会产生一个目录：

/hbase/.hbase-snapshot/snapshot\_src\_table #子目录下有如下几个文件 /hbase/.hbase-snapshot/snapshot\_src\_table/.snapshotinfo /hbase/.hbase-snapshot/snapshot\_src\_table/data.manifest

在上面创建好快照后，使用ExportSnapshot命令进行数据迁移，ExportSnapshot也是HDFS层的操作，本质还是利用MR进行迁移，这个过程主要涉及IO操作并消耗网络带宽，在迁移时要指定下map数和带宽，不然容易造成机房其它业务问题，如果是单独的MR集群，可以在MR集群上使用如下命令：

hbase org.apache.hadoop.hbase.snapshot.ExportSnapshot \

-snapshot snapshot\_src\_table \

-copy-from hdfs://src-hbase-root-dir/hbase \

-copy-to hdfs://dst-hbase-root-dir/hbase \

-mappers 20 \

-bandwidth 20

snapshot只是对元数据信息克隆，不拷贝实际数据文件

当发现spit/compact等操作时，HBase会将原表发生变化的HFile拷贝到/hbase/.archive目录

hbase> clone\_snapshot 'snapshot\_src\_table' , 'new\_table\_name'

这个命令也是不涉及实际数据文件的拷贝，所以执行起来很快，那拷贝的是什么呢，与上面提到的引用文件不同，它所生成的是linkfile，这个文件不包含任何内容，和上面引用文件一样的是，在发生compact等操作时，会将原文件copy到/hbase/.archive目录。

Major\_compact ‘new\_table\_name’

1. 垃圾回收优化

并发模式失败（可能内存不足，或者碎片过多）

第一种失败的模式，是简单的并发模式失败。最好的一个例子：假设有一个8GB堆，已经使用了7GB。当CMS的收集开始第一阶段，它欢快的隆隆的做 着并发标记。与此同时，有更多的数据被分配到老年代。如果老年代增长的速度太快，在CMS完成第一阶段标记工作之前就填满了全部老年代。这时候因为没有自 由空间，CMS就无法工作！CMS必须放弃并行工作，并回落到停止世界（stop-the-world）单线程复制收集算法。此算法开始搬迁堆，检查所有 活动对象，并释放了所有死角。长时间的停顿后，该程序可能会继续。

 但我们可以很容易的通过调整JVM参数避免并发模式失败：我们只需要鼓励CMS提前开始工作！设置 -XX：CMSInitiatingOccupancyFraction = N，其中N是堆在开始收集过程中的百分比。HBase仔细的计算了内存使用，以保持其只使用60％的堆空间，所以我们通常将此值设置为大约70。（译者 注：同时也可以考虑Old区的30%要比Young区大，这样即使Young区在CMS之前全部搬迁到Old区也不会把Old区填满）

一个实验：测量碎片

 我们将运行一个实验证实这一假设。第一步是收集一些有关堆的碎片信息。在探查OpenJDK源代码后，我发现鲜为人知的参数的 -XX：PrintFLSStatistics = 1（译者注：JDK1.6也支持该选项），结合其他详细GC日志记录选项时，会导致CMS之前和之后每打印有关其自由空间的统计信息。特别是，我们关心的 指标是：

* free space - 老年代的空闲内存的总数
* NUM块 - 非连续的内存空闲块总数
* 最大块大小 - 空闲块的最大大小

我启用了这个选项，启动了一个集群，然后运行了Yahoo Cloud Serving Benchmark（YCSB）的三个独立的压力测试：

只写：每行10列，每列100个字节，1亿个row key。

只读（有缓存替换）：随机读取1亿不同的行键的数据，使数据不能完全存储在适LRU缓存。

只读（无缓存替换）：随机读取1万不同的行键的数据，使数据完全符合LRU缓存。

每个压力测试将运行至少一个小时，这样我们可以收集GC行为数据。这个实验的目标是首先要验证我们的假说，暂停是由碎片引起的，第二，以确定造成这些问题的主要原因是读取路径（包括LRU缓存）还是写路径（包

1. 本地memstore分配：0.9版本引入机制缓解region服务器内存碎片问题，这个问题主要是memstore的扰动造成的（不断创建和释放内存空间），本地Memstore分配缓冲区（Memstore-Local Allocation Buffers,MSLAB）

Memstore刷新到磁盘，会在老生代的堆上产生孔洞，由于碎片过多导致没有连续的空间分配，JRE会退回到使应用程序停止垃圾回收器，会导致其重写整个堆空间并压缩剩余的可用对象，减少压缩回收的关键是减少碎片，MSLAB就是为此设计的，关键在于只允许从堆中留下固定大小的孔洞，之后调用相同大小的新对象将会重新使用这些孔洞，就不需要应用程序停止压缩回收了。

1. 压缩
2. 优化拆分和合并
3. 负载均衡

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

**scanner规范：**全表扫描一般不会用，数据量大的时候会死人的。。  
TIMERANGE，   
FILTER，   
LIMIT，   
STARTROW（start row），   
STOPROW（stop row），   
ROWPREFIXFILTER（row prefix filter,行前缀）   
TIMESTAMP，   
MAXLENGTH，   
or COLUMNS，   
CACHE，   
or RAW，   
VERSIONS

scan ‘qy’,{COLUMNS=>[‘name’,’foo’],LIMIT=>1}

scan ‘qy’,{TIMERANGE=>[1448045892646,1448045892647]}

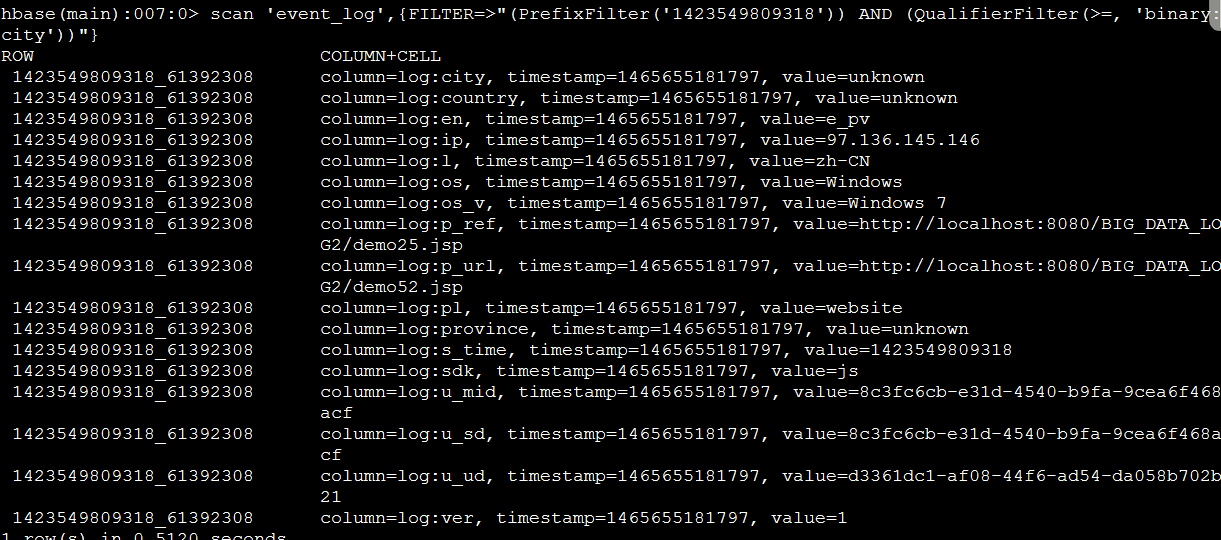
scan ‘qy’,{FILTER=>”PrefixFilter(‘001’)”} rowkey前缀过滤

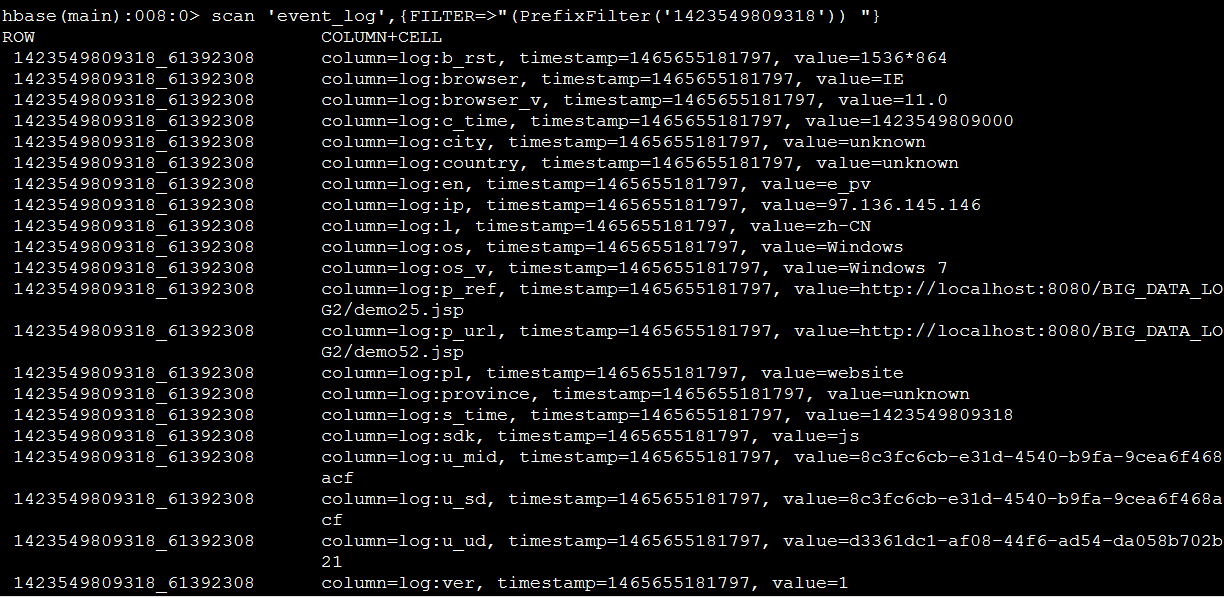
scan ‘qy’,{FILTER=>”PrefixFilter(‘t’) AND QualifierFilter(>=,’binary:b’)”}

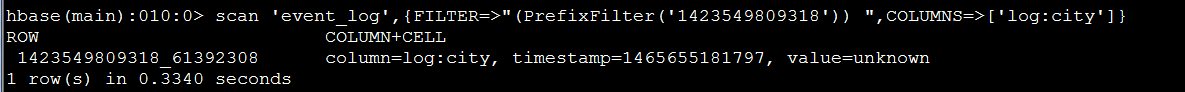
scan ‘qy’,{FILTER=>”TimestampsFilter(1448069941270,1548069941230)” }

shell脚本中date+%s获取当前时间戳

hbase web localhost:60010







查看元数据信息:

scan 'hbase:meta'

list\_namespace 新建命名空间，类似database

create\_namespace 'ai\_ns'

describe\_namespace 'ai\_ns'

drop\_namespace 'ai\_ns'

list\_namespace\_tables 'ai\_ns'

授权tenant-A用户对ai\_ns下的写权限

hbase>grant 'tenant-A' 'W' '@ai\_ns'

回收tenant-A用户对ai\_ns的所有权限

1. hbase>revoke 'tenant-A''@ai\_ns'

****在HBase中启用授权机制****

hbase-site.xml

1. **<property>**
2. **<name>**hbase.security.authorization**</name>**
3. **<value>**true**</value>**
4. **</property>**
5. **<property>**
6. **<name>**hbase.coprocessor.master.classes**</name>**
7. **<value>**org.apache.hadoop.hbase.security.access.AccessController**</value>**
8. **</property>**
9. **<property>**
10. **<name>**hbase.coprocessor.region.classes**</name>**
11. **<value>**org.apache.hadoop.hbase.security.token.TokenProvider,org.apache.hadoop.hbase.security.access.AccessController**</value>**
12. **</property>**

配置完成后需要重启HBase集群

#!/bin/sh

exec $HBASE\_HOME/bin/hbase shell <<EOF

create 'test', {NAME => 't', VERSIONS => 1}

EOF

**我们之前建了3个列族，但是发现member\_id这个列族是多余的，因为他就是主键，所以我们要将其删除。**

hbase(main):003:0>alter 'member',{NAME=>'member\_id',METHOD=>'delete'}

ERROR: Table memberis enabled. Disable it first before altering.

报错，删除列族的时候必须先将表给disable掉。

hbase(main):004:0>disable 'member'

hbase(main):005:0>alter'member',NAME=>'member\_id',METHOD=>'delete'

hbase(main):006:0>describe 'member'

{NAME => 'member', FAMILIES => [{NAME=> 'address', BLOOMFILTER => 'NONE', REPLICATION\_SCOPE => '0',false,VERSIONS => '3', COMPRESSION => 'NONE',TTL => '2147483647', BLOCKSIZE => '65536', IN\_MEMORY => 'false', BLOCKCACHE => 'true'}, {NAME =>'info', BLOOMFILTER => 'NONE', REPLICATION\_SCOPE => '0', VERSIONS => '3', COMPRESSION => 'NONE', TTL=> '2147483647', BLOCKSIZE => '65536', IN\_MEMORY => 'false',BLOCKCACHE => 'true'}]}

该列族已经删除，我们继续将表enable

hbase(main):008:0> enable 'member'

查询表是否存在

hbase(main):021:0>exists 'member'

Table member doesexist

判断表是否disable

hbase(main):032:0>is\_disabled 'member'

false

删除整行

hbase(main):001:0>deleteall 'member','xiaofeng'

0 row(s) in 0.3990seconds

查询表中有多少行：

hbase(main):019:0>count 'member'

2 row(s) in 0.0160seconds

**每个regionserver节点可以自由启动或停止，可以不随hbase整体一起。**

停止后regionserver上的数据会被移到其他regionserver上，不影响hbase的使用

重启regionserver

bin/graceful\_stop.sh --restart --reload --debug nodename

启动regionserver

/bin/hbase-daemon.sh start regionserver RegionServer

2.bin/hbase shell,这个就是常用的shell工具，运维常用的DDL和DML都会通过此进行，其具体实现（对hbase的调用）是用ruby写的

3.bin/hbase hbck, 运维常用工具，检查集群的数据一致性状态，其执行是直接调用org.apache.hadoop.hbase.util.HBaseFsck中的main函数

4.bin/hbase hlog, log分析工具，其执行是直接调用org.apache.hadoop.hbase.regionserver.wal.HLogPrettyPrinter中的main函数

5.bin/hbase hfile， hfile分析工具，其执行是直接调用org.apache.hadoop.hbase.io.hfile.HFile中的main函数

6.bin/hbase zkcli,查看/管理ZK的shell工具，很实用，经常用，比如你可以通过（get /IP/master）其得知当前的active master,可以通过（get /IP/root-region-server）得知当前root region所在的server，你也可以在测试中通过（delete /IP/rs/dwxx.yy.taobao），模拟regionserver与ZK断开连接，

其执行则是调用了org.apache.zookeeper.ZooKeeperMain的main函数

7.bin/hbase classpath 打印classpath

8.bin/hbase version 打印hbase版本信息

**Hbase运行脚本**

首先，编写一个文本文件firsthbaseshell.txt：

status 查看状态

version 查看版本

list

list 'abc.\*' #显示abc开头的表

describe 'table\_name'

scan 'ns1:t1',{COLUMNS=>['c1','c2'],LIMIT=>10,STARTROW=>'xyz'}

**create 'test', 'col', SPLITS => ['user1', 'user2','user3','user4','user5','user6','user7','user8','user9',]**

create 'test', 'cf'

list 'test'

put 'test', 'row1', 'cf:a', 'value1'

scan 'test'

get 'test', 'row1'

disable 'test'

enable 'test'

Exit

hbase(main):024:0>status

3 servers, 0 dead,1.0000 average load

hbase(main):025:0>version

0.90.4, r1150278,Sun Jul 24 15:53:29 PDT 2011

2、在HBase shell中运行这个脚本：

利用命令：hbase shell firstbaseshell.txt:

谁的值=sku188

scan 'test1', FILTER=>"ValueFilter(=,'binary:sku188')"

ROW COLUMN+CELL user1|ts2 column=sf:c1, timestamp=1409122354918, value=sku188

谁的值包含88

scan 'test1', FILTER=>"ValueFilter(=,'substring:88')"

ROW COLUMN+CELL

user1|ts2 column=sf:c1, timestamp=1409122354918, value=sku188 user2|ts5 column=sf:c2, timestamp=1409122355030, value=sku288

通过搜索进来的(column为s)值包含123或者222的用户

scan 'test1', FILTER=>"ColumnPrefixFilter('s') AND ( ValueFilter(=,'substring:123') OR ValueFilter(=,'substring:222') )"

ROW COLUMN+CELL

user1|ts3 column=sf:s1, timestamp=1409122354954, value=sku123

user2|ts6 column=sf:s1, timestamp=1409122355970, value=sku222

rowkey为user1开头的

scan 'test1', FILTER => "PrefixFilter ('user1')"

ROW COLUMN+CELL

user1|ts1 column=sf:c1, timestamp=1409122354868, value=sku1

user1|ts2 column=sf:c1, timestamp=1409122354918, value=sku188

user1|ts3 column=sf:s1, timestamp=1409122354954, value=sku123

从user1|ts2开始,找到所有的到rowkey以user2开头

scan 'test1', {STARTROW=>'user1|ts2', STOPROW=>'user2'}

查询rowkey里面包含ts的

import org.apache.hadoop.hbase.filter.CompareFilter

import org.apache.hadoop.hbase.filter.SubstringComparator

import org.apache.hadoop.hbase.filter.RowFilter

scan 'test1', {FILTER => RowFilter.new(CompareFilter::CompareOp.valueOf('EQUAL'), SubstringComparator.new('ts'))} ROW COLUMN+CELL

user1|ts1 column=sf:c1, timestamp=1409122354868, value=sku1

user1|ts2 column=sf:c1, timestamp=1409122354918, value=sku188

user1|ts3 column=sf:s1, timestamp=1409122354954, value=sku123

user2|ts4 column=sf:c1, timestamp=1409122354998, value=sku2

user2|ts5 column=sf:c2, timestamp=1409122355030, value=sku288

user2|ts6 column=sf:s1, timestamp=1409122355970, value=sku222

HBASE按单个Rowkey检索的效率是很高的，耗时在1毫秒以下，每秒钟可获取1000~2000条记录，不过非key列的查询很慢。

**HBase的RowKey设计**

**Rowkey长度原则：**Rowkey是一个二进制码流，Rowkey的长度被很多开发者建议说设计在10~100个字节，不过建议是越短越好，不要超过16个字节。

1）数据的持久化文件HFile中是按照KeyValue存储的，如果Rowkey过长比如100个字节，1000万列数据光Rowkey就要占用100\*1000万=10亿个字节，将近1G数据，这会极大影响HFile的存储效率；

2）MemStore将缓存部分数据到内存，如果Rowkey字段过长内存的有效利用率会降低，系统将无法缓存更多的数据，这会降低检索效率。因此Rowkey的字节长度越短越好。

3）目前操作系统是都是64位系统，内存8字节对齐。控制在16个字节，8字节的整数倍利用操作系统的最佳特性。

**Rowkey散列原则：**如果Rowkey是按时间戳的方式递增，不要将时间放在二进制码的前面，建议将Rowkey的高位作为散列字段，由程序循环生成，低位放时间字段，这样将提高数据均衡分布在每个Regionserver实现负载均衡的几率。如果没有散列字段，首字段直接是时间信息将产生所有新数据都在一个 RegionServer上堆积的热点现象，这样在做数据检索的时候负载将会集中在个别RegionServer，降低查询效率。

**Rowkey唯一原则**：必须在设计上保证其唯一性。

HBase按指定的条件获取一批记录时，使用的就是scan方法。 scan方法有以下特点：

（1）scan可以通过setCaching与setBatch方法提高速度（以空间换时间）；

（2）scan可以通过setStartRow与setEndRow来限定范围。范围越小，性能越高。

通过巧妙的RowKey设计使我们批量获取记录集合中的元素挨在一起（应该在同一个Region下），可以在遍历结果时获得很好的性能。

（3）scan可以通过setFilter方法添加过滤器，这也是分页、多条件查询的基础。

在满足长度、散列、唯一原则后，我们需要考虑如何通过巧妙设计RowKey以利用scan方法的范围功能，使得获取一批记录的查询速度能提高。

RowFilter是用来对rowkey进行过滤的,比较符如下:

|  |  |  |
| --- | --- | --- |
| Operator | Description | |
| LESS | 小于 | |
| LESS\_OR\_EQUAL | 小于等于 | |
| EQUAL | 等于 | |
| NOT\_EQUAL | 不等于 | |
| GREATER\_OR\_EQUAL | 大于等于 | |
| GREATER | 大于 | |
| NO\_OP | 排除所有 | |
| Comparator | | Description | |
| BinaryComparator | | 使用Bytes.compareTo()比较 | |
| BinaryPrefixComparator | | 和BinaryComparator差不多，从前面开始比较 | |
| NullComparator | | Does not compare against an actual value but whether a given one is null, or not null. | |
| BitComparator | | Performs a bitwise comparison, providing a BitwiseOp class with AND, OR, and XOR operators. | |
| RegexStringComparator | | 正则表达式 | |
| SubstringComparator | | 把数据当成字符串，用contains()来判断 | |

**集群数据倾斜**

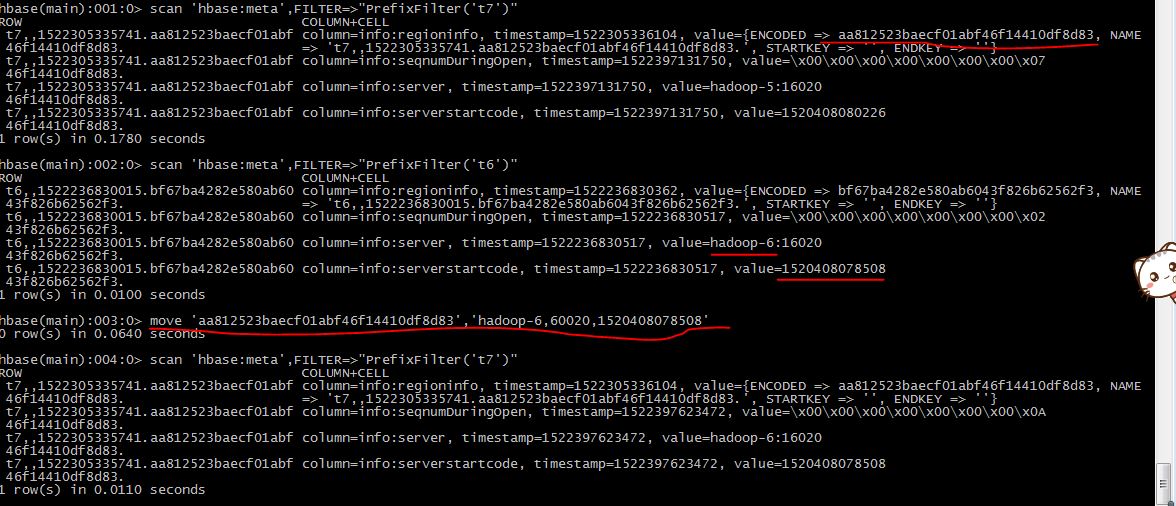
在集群中为了得到更好的并行性，我们希望有好的load blance，让每个节点提供的请求都是均衡的，我们也不希望，region不要经常split，因为split会使server有一段时间的停顿，如何能做到呢？

**随机散列与预分区二者结合起来，是比较完美的。**预分区一开始就预建好了一部分region，这些region都维护着自己的start-end keys，在配合上随机散列，写数据能均衡的命中这些预建的region，就能解决上面的那些缺点，大大提供性能。

**Region管理**

1）移动region

# 语法：move 'encodeRegionName', 'ServerName'



**手动触发major compaction**

#语法：

#Compact all regions in a table:

#hbase> major\_compact 't1'

#Compact an entire region:

#hbase> major\_compact 'regionId'见上图

#Compact a single column family within a region:

#hbase> major\_compact 'r1', 'c1'

#Compact a single column family within a table:

#hbase> major\_compact 't1', 'c1'

**手动split**

# 语法：split 'regionName', 'splitKey'

在逻辑上，HBase的表数据按RowKey进行字典排序

通常来说，RowKey只能针对条件中含有其首字段的查询给予令人满意的性能支持，在查询其他字段时，表现就差强人意了，在极端情况下某些字段的查询性能可能会退化为全表扫描的水平，这是因为**字段在RowKey中的地位是不等价的，它们在RowKey中的排位决定了它们被检索时的性能表现，排序越靠前的字段在查询中越具有优势，特别是首位字段具有特别的先发优势**，如果查询中包含首位字段，检索时就可以通过首位字段的值确定RowKey的前缀部分，从而大幅度地收窄检索区间，如果不包含则只能在全体数据的RowKey上逐一查找，由此可以想见两者在性能上的差距。

**Hbase作为列族数据库最经常被人诟病的特性包括**：无法轻易建立“二级索引”，难以执行求和、计数、排序等操作。

比如，在旧版本的(<0.92)Hbase中，统计数据表的总行数，需要使用Counter方法，执行一次MapReduce Job才能得到。

虽然HBase在数据存储层中集成了MapReduce，能够有效用于数据表的分布式计算。然而在很多情况下，做一些简单的相加或者聚合计算的时候，如果直接将计算过程放置在server端能够减少通讯开销，从而获得很好的性能提升。

于是，HBase在0.92之后引入了协处理器(coprocessors)，

**实现一些激动人心的新特性：能够轻易建立二次索引、复杂过滤器(谓词下推)以及访问控制等。**

CoprocessorRowcounter已经在实践中统计了我们大部分表的行数，正确性也经过了验证，存在的问题主要是资源占用率过高，

不宜用于在线集群。其次效率仍需提高，即使能够对region进行并发操作，大表的统计仍需要分钟级甚至是小时级的耗时。

实践也发现每次rpc调用时间都比正常操作长很多，所以务必将hbase.rpc.timeout设置为Integer.MAX\_VALUE，

才能保证程序不会因为rpc超时而退出。

实现hbase表的行数统计工作：

1.count命令

最直接的方式是在hbase shell中执行count的命令可以统计行数。

[html] view plain copy

hbase> count ‘t1′

hbase> count ‘t1′, INTERVAL => 100000

hbase> count ‘t1′, CACHE => 1000

hbase> count ‘t1′, INTERVAL => 10, CACHE => 1000

其中，INTERVAL为统计的行数间隔，默认为1000，CACHE为统计的数据缓存。这种方式效率很低，如果表行数很大的话不建议采用这种方式。

2. 调用Mapreduce

[plain] view plain copy

$HBASE\_HOME/bin/hbase org.apache.hadoop.hbase.mapreduce.RowCounter ‘tablename’

这种方式效率比上一种要搞很多，调用的hbase jar中自带的统计行数的类。

**4.1启用协处理器 Aggregation(Enable Coprocessor Aggregation)**

我们有两个方法：

1.启动全局aggregation，能过操纵所有的表上的数据。通过修改hbase-site.xml这个文件来实现，只需要添加如下代码：

<property>

<name>hbase.coprocessor.user.region.classes</name>

<value>org.apache.hadoop.hbase.coprocessor.AggregateImplementation</value>

</property>

2.启用表aggregation，只对特定的表生效。通过HBase Shell 来实现。**不好使**

(1)disable指定表。hbase> disable 'mytable'

(2)添加aggregation hbase>

alter 't7', METHOD => 'table\_att','coprocessor'=>'|org.apache.Hadoop.hbase.coprocessor.AggregateImplementation||'

(3)重启指定表 hbase> enable 'mytable'

**步骤1：通过如下方法创建表**

hbase(main):001:0> create 'coprocessor\_table','F'

**步骤2：通过alter命令将协处理器加载到表中**

alter 'testOb' , METHOD =>'table\_att','coprocessor'=>'hdfs://tianxi-ha/test/pagerank/original-test-1.0-SNAPSHOT.jar|hbase.coprocessor.ObserverDemo|1001'

其中：'coprocessor'=>'jar文件在hdfs上的绝对路径|协处理器主类|优先级|协处理器参数。上述协处理器并没有参数，所以未给出参数，对于协处理器的优先级不在此做讨论。

**步骤3：检查协处理器的加载**

Desc 'testOb'

**步骤4:输入数据**

put ''testOb'','row1','F:A',123

delete ''testOb'','row1','F:B'

# IMG_256**一、HBase Architecture**

# **二、HBase架构中的客户端Client**

客户端有以下几点作用：

       1. 整个HBase集群的访问入口；

       2. 使用HBase RPC机制与HMaster和HRegionServer进行通信；

       3. 使用HMaster进行通信进行管理类操作；

       4. 与HRegionServer进行数据读写类操作；

       5. 包含访问HBase的接口，并维护cache来加快对HBase的访问。

# **三、协调服务组件Zookeeper**

Zookeeper的作用如下：

       1. 保证任何时候，集群中只有一个HMaster；

       2. 存储所有的HRegion的寻址入口；

       3. 实时监控HRegionServer的上线和下线信息，并实时通知给HMaster；

       4. 存储HBase的schema和table元数据；

       5. Zookeeper Quorum存储-ROOT-表地址、HMaster地址。

# **四、主节点HMaster**

HMaster的主要功能如下：

       1. HMaster没有单节点问题，HBase中可以启动多个HMaster，通过Zookeeper的Master Election机制保证总有一个Master在运行，主要负责Table和Region的管理工作。

       如何启动多个HMaster？

       通过hbase-daemons.sh启动，步骤如下：1）在hbase/conf目录下编辑backup-masters；2）编辑内容为自己的主机名；3）保存后，执行如下命令：bin/hbase-daemons.sh start master-backup。

       2. 管理用户对表的增删改查操作；

       3. 管理HRegionServer的负载均衡，调整Region分布（在命令行里面有一个tools，tools这个分组命令其实全部都是Master做的事情）；

       4. Region Split后，负责新Region的分布；

       5. 在HRegionServer停机后，负责失效HRegionServer上Region迁移工作。

# **五、Region节点HRegionServer**

HRegionServer的功能如下：

       1. 维护HRegion，处理HRegion的IO请求，向HDFS文件系统中读写数据；

       2. 负责切分运行过程中变得过大的HRegion；

       3. Client访问HBase上数据的过程并不需要Master参与（寻址访问zookeeper和HRegionServer，数据读写访问HRegionServer），HMaster仅仅维护着table和Region的元数据信息，负载很低