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发布了一些工具，有两个支持导入和导出的mapreduce作业

$ hadoop jar $HBASE\_HOME/hbase-0.91.0-SNAPSHOT.jar export

例如：$ hadoop jar $HBASE\_HOME/HBase-0.91.0-SNAPSHOT.jar export \

testable /user/larsgeorge/backup-testtable 就会启动mapreduce作业，并显示进度

$ hadoop jar $HBASE\_HOME/HBase-0.91.0-SNAPSHOT.jar import \

testable /user/larsgeorge/backup-testtable

优化：

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]}

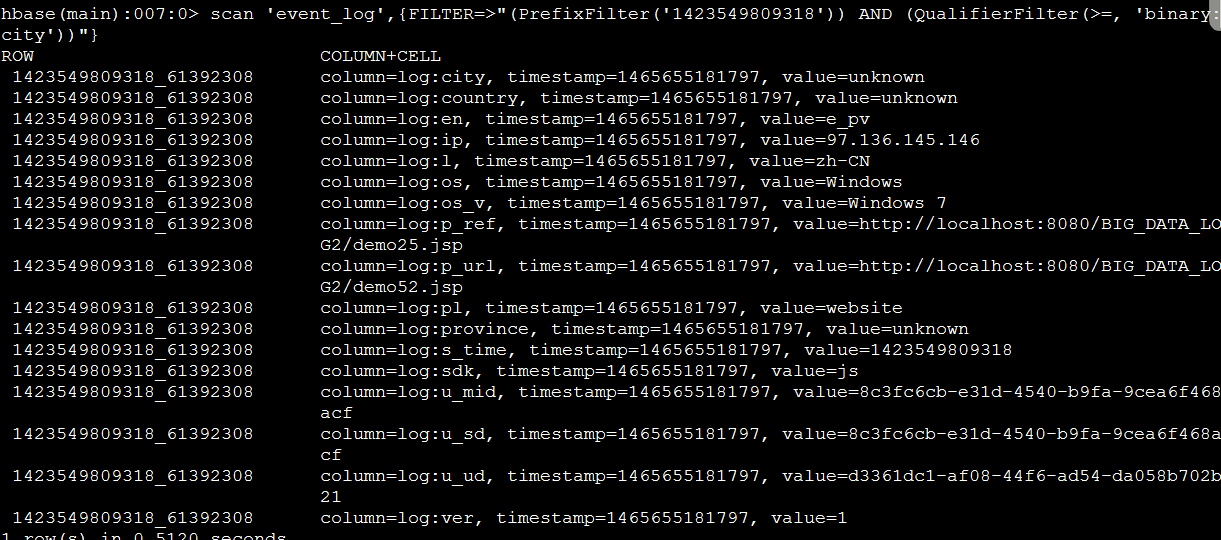
scan ‘qy’,{FILTER=>”PrefixFilter(‘001’)”} rowkey前缀过滤

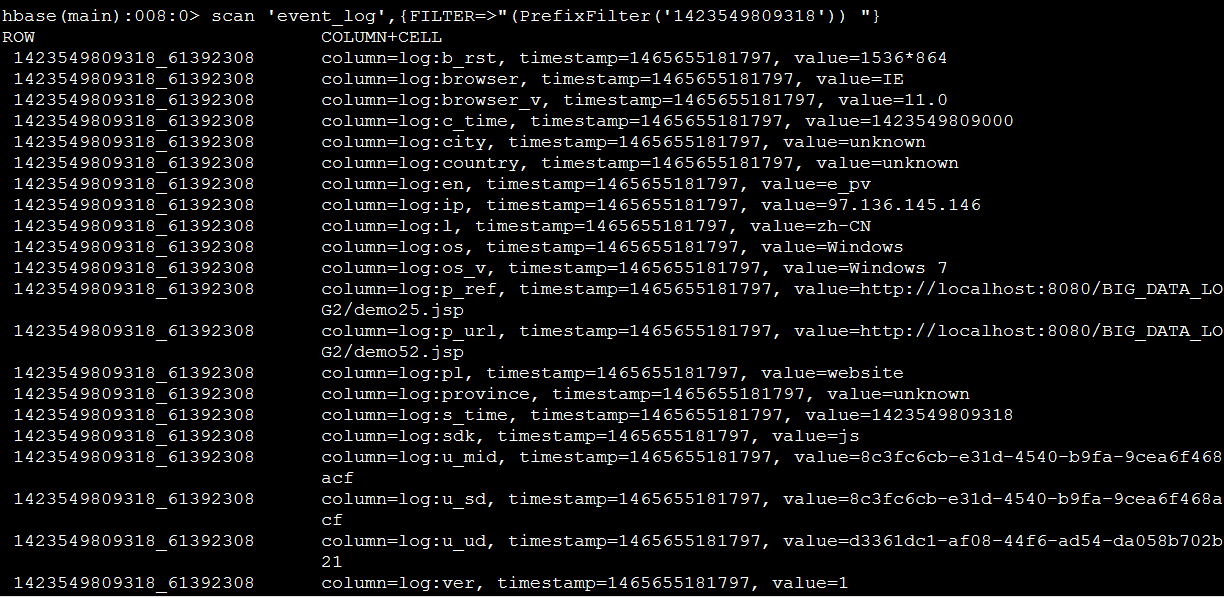
scan ‘qy’,{FILTER=>”PrefixFilter(‘t’) AND QualifierFilter(>=,’binary:b’)”}

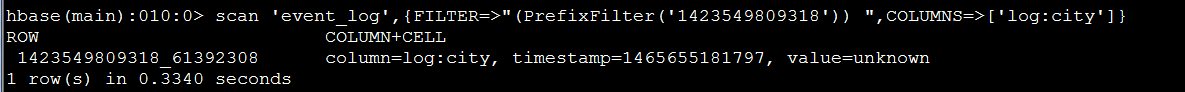
scan ‘qy’,{FILTER=>”TimestampsFilter(1448069941270,1548069941230)” }

shell脚本中date+%s获取当前时间戳

hbase web localhost:60010







list\_namespace 新建命名空间，类似database

create\_namespace 'ai\_ns'

describe\_namespace 'ai\_ns'

drop\_namespace '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

**[html]** [view plain](https://blog.csdn.net/opensure/article/details/46470969" \o "view plain) [copy](https://blog.csdn.net/opensure/article/details/46470969" \o "copy)

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'

0 row(s) in 2.0390seconds

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

0 row(s) in 0.0560seconds

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'}]}

1 row(s) in 0.0230seconds

该列族已经删除，我们继续将表enable

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

0 row(s) in 2.0420seconds

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:

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。

**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'

# encodeRegionName指的regioName后面的编码，ServerName指的是master-status的Region Servers列表

# 示例

hbase(main)>move '4343995a58be8e5bbc739af1e91cd72d', 'db-41.xxx.xxx.org,60020,1390274516739'

**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 'r1'

#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， RowKey实际上是数据表的一级索引（Primary Index），由于HBase本身没有二级索引（Secondary Index）机制，基于索引检索数据只能单纯地依靠RowKey，为了能支持多条件查询，开发者需要将所有可能作为查询条件的字段一一拼接到RowKey中，这是HBase开发中极为常见的做法，但是无论怎样设计，单一RowKey固有的局限性决定了它不可能有效地支持多条件查询。

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。

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

(2)添加aggregation hbase> alter 'mytable', METHOD => 'table\_att','coprocessor'=>'|org.apache.Hadoop.hbase.coprocessor.AggregateImplementation||'

(3)重启指定表 hbase> enable 'mytable'