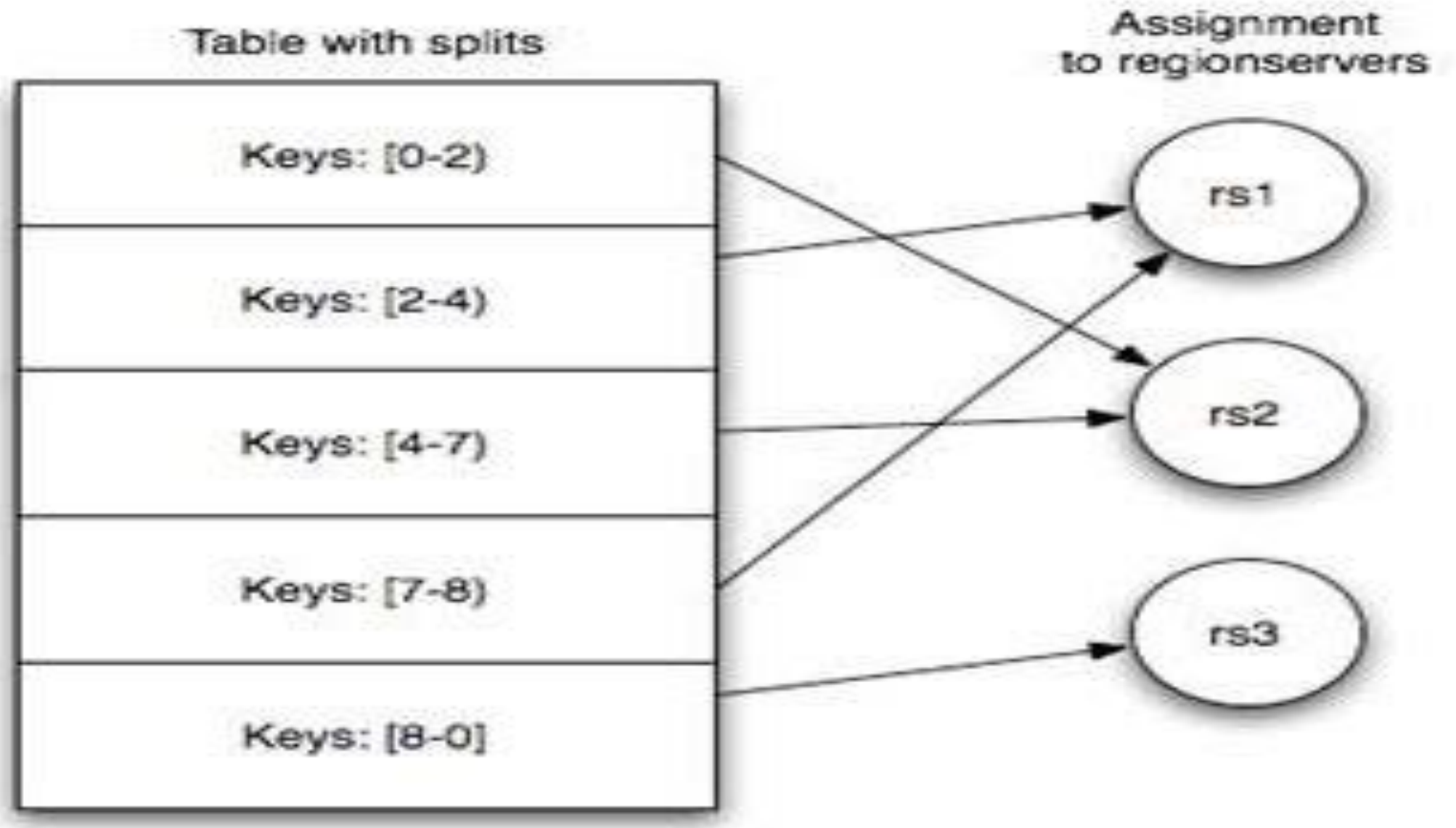
大讲台
dajiangtai.com

HBase

主讲人：杨老师

目录

Hbase 热点问题？



Hbase 热点问题？

- 什么是热点

HBase中的行是按照rowkey的字典顺序排序的，这种设计优化了scan操作，可以将相关的行以及会被一起读取的行存取在临近位置，便于scan。然而糟糕的rowkey设计是热点的源头。热点发生在大量的client直接访问集群的一个或极少数个节点（访问可能是读，写或者其他操作）。大量访问会使热点region所在的单个机器超出自身承受能力，引起性能下降甚至region不可用，这也会影响同一个RegionServer上的其他region，由于主机无法服务其他region的请求。设计良好的数据访问模式以使集群被充分，均衡的利用。

- 产生原因
  - Hbase 创建表默认只有一个分区
  - Rowkey 设计不合理
- 解决方案
  - Hbase 创建表时指定分区
  - 合理设计Rowkey

Hbase 预分区

- Shell createTable 并预分区

```
hbase(main):002:0> create 'split01','cf1',SPLITS=>['1000000','2000000','3000000']
0 row(s) in 0.7880 seconds


=> Hbase::Table - split01
```

```
hbase(main):003:0> create 'split02','cf1',SPLITS_FILE=>'/usr/java/split.txt'
0 row(s) in 1.2420 seconds


=> Hbase::Table - split02
```

```
100000000000000
200000000000000
300000000000000
400000000000000
500000000000000
~
~
```

Hbase 预分区

- javaAPI createTable并预分区---直接根据描述创建表

```
/**
 * Creates a new table. Synchronous operation.
 *
 * @param desc table descriptor for table
 * @throws IllegalArgumentException if the table name is reserved
 * @throws MasterNotRunningException if master is not running
 * @throws org.apache.hadoop.hbase.TableExistsException if table already exists (If concurrent
 * threads, the table may have been created between test-for-existence and attempt-at-creation).
 * @throws IOException if a remote or network exception occurs
 */
void createTable(HTableDescriptor desc) throws IOException;
```

Hbase 预分区

- javaAPI createTable并预分区---根据描述和region个数以及startKey、endKey自动分配

```
/**
   * Creates a new table with the specified number of regions.  The start key specified will become
   * the end key of the first region of the table, and the end key specified will become the start
   * key of the last region of the table (the first region has a null start key and the last region
   * has a null end key). BigInteger math will be used to divide the key range specified into enough
   * segments to make the required number of total regions. Synchronous operation.
   *
   * @param desc table descriptor for table
   * @param startKey beginning of key range
   * @param endKey end of key range
   * @param numRegions the total number of regions to create
   * @throws IllegalArgumentException if the table name is reserved
   * @throws MasterNotRunningException if master is not running
   * @throws org.apache.hadoop.hbase.TableExistsException if table already exists (If concurrent
   * threads, the table may have been created between test-for-existence and attempt-at-creation).
   * @throws IOException
   */
  void createTable(HTableDescriptor desc, byte[] startKey, byte[] endKey, int numRegions)
      throws IOException;
```

Hbase 预分区

- javaAPI createTable并预分区---根据表的描述和自定义的分区设置创建表（同步）

```
byte[][] splitKeys = new byte[][] { Bytes.toBytes("100000"),
                Bytes.toBytes("200000"), Bytes.toBytes("400000"),
                Bytes.toBytes("500000") };
```

```
/**
   * Creates a new table with an initial set of empty regions defined by the specified split keys.
   * The total number of regions created will be the number of split keys plus one. Synchronous
   * operation. Note : Avoid passing empty split key.
   *
   * @param desc table descriptor for table
   * @param splitKeys array of split keys for the initial regions of the table
   * @throws IllegalArgumentException if the table name is reserved, if the split keys are repeated
   * and if the split key has empty byte array.
   * @throws MasterNotRunningException if master is not running
   * @throws org.apache.hadoop.hbase.TableExistsException if table already exists (If concurrent
   * threads, the table may have been created between test-for-existence and attempt-at-creation).
   * @throws IOException
   */
  void createTable(final HTableDescriptor desc, byte[][] splitKeys) throws IOException;
```

Hbase 预分区

- javaAPI createTable并预分区---根据表的描述和自定义的分区设置创建表（异步）

```
/**
   * Creates a new table but does not block and wait for it to come online. Asynchronous operation.
   * To check if the table exists, use {@link #isTableAvailable} -- it is not safe to create an
   * HTable instance to this table before it is available. Note : Avoid passing empty split key.
   *
   * @param desc table descriptor for table
   * @throws IllegalArgumentException Bad table name, if the split keys are repeated and if the
   * split key has empty byte array.
   * @throws MasterNotRunningException if master is not running
   * @throws org.apache.hadoop.hbase.TableExistsException if table already exists (If concurrent
   * threads, the table may have been created between test-for-existence and attempt-at-creation).
   * @throws IOException
   */
  void createTableAsync(final HTableDescriptor desc, final byte[][] splitKeys) throws IOException;
```

Hbase Rowkey 设计原则

- rowkey长度原则

rowkey是一个二进制码流，可以是任意字符串，最大长度64kb，实际应用中一般为10-100bytes，以byte[]形式保存，一般设计成定长。

建议越短越好，不要超过16个字节，原因如下：

1）数据的持久化文件HFile中是按照KeyValue存储的，如果rowkey过长，比如超过100字节，1000w行数据，光rowkey就要占用100*1000w=10亿个字节，将近1G数据，这样会极大影响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，内存的有效利用率就会降低，系统不能缓存更多的数据，这样会降低检索效率。

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

Hbase Rowkey 设计原则

- rowkey散列原则

如果rowkey按照时间戳的方式递增，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，由程序随机生成，低位放时间字段，这样将提高数据均衡分布在每个RegionServer，以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，所有的数据都会集中在一个RegionServer上，这样在数据检索的时候负载会集中在个别的RegionServer上，造成热点问题，会降低查询效率。

Hbase Rowkey 设计原则

- rowkey唯一原则

必须在设计上保证其唯一性，rowkey是按照字典顺序排序存储的，因此，设计rowkey的时候，要充分利用这个排序的特点，将经常读取的数据存储到一块，将最近可能会被访问的数据放到一块。

Hbase 常见避免热点问题方法

- 加盐

这里的加盐不是化学中的加盐，而是在rowkey 的前面增加随机数。具体就是给 rowkey 分配一个随机前缀 以使得它和之前排序不同。分配的前缀种类数量应该和你想使数据分散到不同的 region 的数量一致。 如果你有一些 热点 rowkey 反复出现在其他分布均匀的 rwokey 中，加盐是很有用的。考虑下面的例子：它将写请求分散到多个 RegionServers，但是对读造成了一些负面影响。

Hbase 常见避免热点问题方法

- 加盐

假如你有下列 rowkey，你表中每一个 region 对应字母表中每一个字母。 以 'a' 开头是同一个 region, 'b'开头的是同一个region。在表中，所有以 'f'开头的都在同一个 region， 它们的 rowkey 像下面这样：

foo0001

foo0002

foo0003

foo0004

Hbase 常见避免热点问题方法

- 加盐

现在，假如你需要将上面这个 region 分散到 4个 region。你可以用4个不同的盐：'a', 'b', 'c', 'd'.

在这个方案下，每一个字母前缀都会在不同的 region 中。加盐之后，你有了下面的 rowkey:

a-foo0003

b-foo0001

c-foo0004

d-foo0002

Hbase 常见避免热点问题方法

- 加盐

所以，你可以向4个不同的 region 写，理论上说，如果所有人都向同一个region 写的话，你将拥有之前4倍的吞吐量。

现在，如果再增加一行，它将随机分配a,b,c,d中的一个作为前缀，并以一个现有行作为尾部结束：

a-foo0003

b-foo0001

c-foo0003

c-foo0004

d-foo0002

因为分配是随机的，所以如果你想要以字典序取回数据，你需要做更多工作。加盐这种方式增加了写时的吞吐量，但是当读时有了额外代价。

Hbase 常见避免热点问题方法

- 哈希

除了加盐，你也可以使用哈希，哈希会使同一行永远用同一个前缀加盐。哈希也可以使负载分散到整个集群，但是读却是可以预测的。使用确定的哈希可以让客户端重构完成的 rowkey，使用Get操作获取正常的获取某一行数据。

像在加盐方法中给出的那个例子，你可以使用某种哈希方法使得 foo0003 这样的 rowkey 的前缀永远是 ‘a',然后，为了取得某一行，你可以通过哈希获得相应的 rowkey. 你也可以优化哈希方法，使得某些rowkey 永远在同一个 region.

Hbase 常见避免热点问题方法

- 反转

第三种防止热点的方法时反转固定长度或者数字格式的rowkey。这样可以使得rowkey中经常改变的部分（最没有意义的部分）放在前面。这样可以有效的随机rowkey，但是牺牲了rowkey的有序性。

反转rowkey的例子

以手机号为rowkey，可以将手机号反转后的字符串作为rowkey，这样就避免了以手机号那样比较固定开头导致热点问题

Hbase 常见避免热点问题方法

- 时间戳反转

一个常见的数据处理问题是快速获取数据的最近版本，使用反转的时间戳作为rowkey的一部分对这个问题十分有用，可以用Long.Max_Value - timestamp追加到key的末尾，例如[key][reverse_timestamp],[key]的最新值可以通过scan [key]获得[key]的第一条记录，因为HBase中rowkey是有序的，第一条记录是最后录入的数据。

比如需要保存一个用户的操作记录，按照操作时间倒序排序，在设计rowkey的时候，可以这样设计[userId反转][Long.Max_Value - timestamp]，在查询用户的所有操作记录数据的时候，直接指定反转后的userId，startRow是[userId反转][000000000000],stopRow是[userId反转][Long.Max_Value - timestamp]如果需要查询某段时间的操作记录，startRow是[user反转][Long.Max_Value - 起始时间]，stopRow是[userId反转][Long.Max_Value - 结束时间]

Hbase 总结

- 尽量减少行和列的大小

在HBase中，value永远和它的key一起传输的。当具体的值在系统间传输时，它的rowkey，列名，时间戳也会一起传输。如果你的rowkey和列名很大，甚至可以和具体的值相比较，那么你将会遇到一些有趣的问题。HBase storefiles中的索引（有助于随机访问）最终占据了HBase分配的大量内存，因为具体的值和它的key很大。可以增加block大小使得storefiles索引再更大的时间间隔增加，或者修改表的模式以减小rowkey和列名的大小。压缩也有助于更大的索引。

- 列族尽可能越短越好，最好是一个字符（f）
- 冗长的属性名虽然可读性好，但是更短的属性名存储在HBase中会更好

Hbase 连续查询的Rowkey设计

- 连续查询特点
  - 要使用scan方式查询
  - 相同的数据均匀的分布在相同的region上
- 表的要求
  - 表的块要大。块越大，那么块存放的数据就越多，利于连续查询

Rowkey=id+类型+时间

1）直接以id进行分区，id相同，所在region相同，实现相同id数据同一个region连续查询。

00001003xxxxxxxx

10000002xxxxxxxx

20000001xxxxxxxx

Hbase 随机查询的Rowkey设计

- 随机查询特点
  - 不需要连续查询一段范围内的数据
  - 一般不需要scan方式查询
  - 数据均匀的分布在不同的region上
- 表的要求
  - 表的块要小。块越小，那么索引就越多。

Hbase 随机查询的Rowkey设计

- Rowkey随机设计方式一：Hash 散列

0,1,2,3,4

byte prefix = (byte)(Long.hashCode(timestamp)%5) //5代表regionserver个数

byte[] rowkey = Bytes.add(Bytes.toBytes(prefix),Bytes.toBytes(uid+timestamp+编号))

0uid+timestamp+1

1uid+timestamp+2

2uid+timestamp+3

3uid+timestamp+4

4uid+timestamp+5

0uid+timestamp+6

1uid+timestamp+7

Hbase 随机查询的Rowkey设计

- Rowkey随机设计方式二：MD5随机散列

byte[] rowkey = 前三位MD5(timestamp)+uid+timestamp+编号

Rowkey被随机分布到一组region服务器上

Hbase 表设计

在HBase中表可以设计为窄表和宽表两种形式。

所谓窄表：列少行多。

所谓宽表：列多行少。

Hbase 窄表设计

- 用户社区发帖案例

RowKey（uid+时间+cid）	列簇		
	Uid	Content	time
10001-201703151200-1	10001	李克强就朝鲜问题答记者问	201703151200
10001-201703151210-2	10001	李克强回应中美的底线是什么	201703151210
10001-201703151230-3	10001	朴槿惠入狱的可能性有多大	201703151230
10002-201703151230-4	10002	李克强谈全球化	201703151230
10003-201703151240-4	10003	李克强谈人民币汇率	201703151240

Hbase 宽表设计

- 用户社区发帖案例

RowKey	列簇			
	Uid	2017031512-Content	2017031513-Content	2017031514-Content
myrowkey-1	10001	李克强就朝鲜问题答记者问	李克强回应中美的底线是什么	朴槿惠入狱的可能性有多大

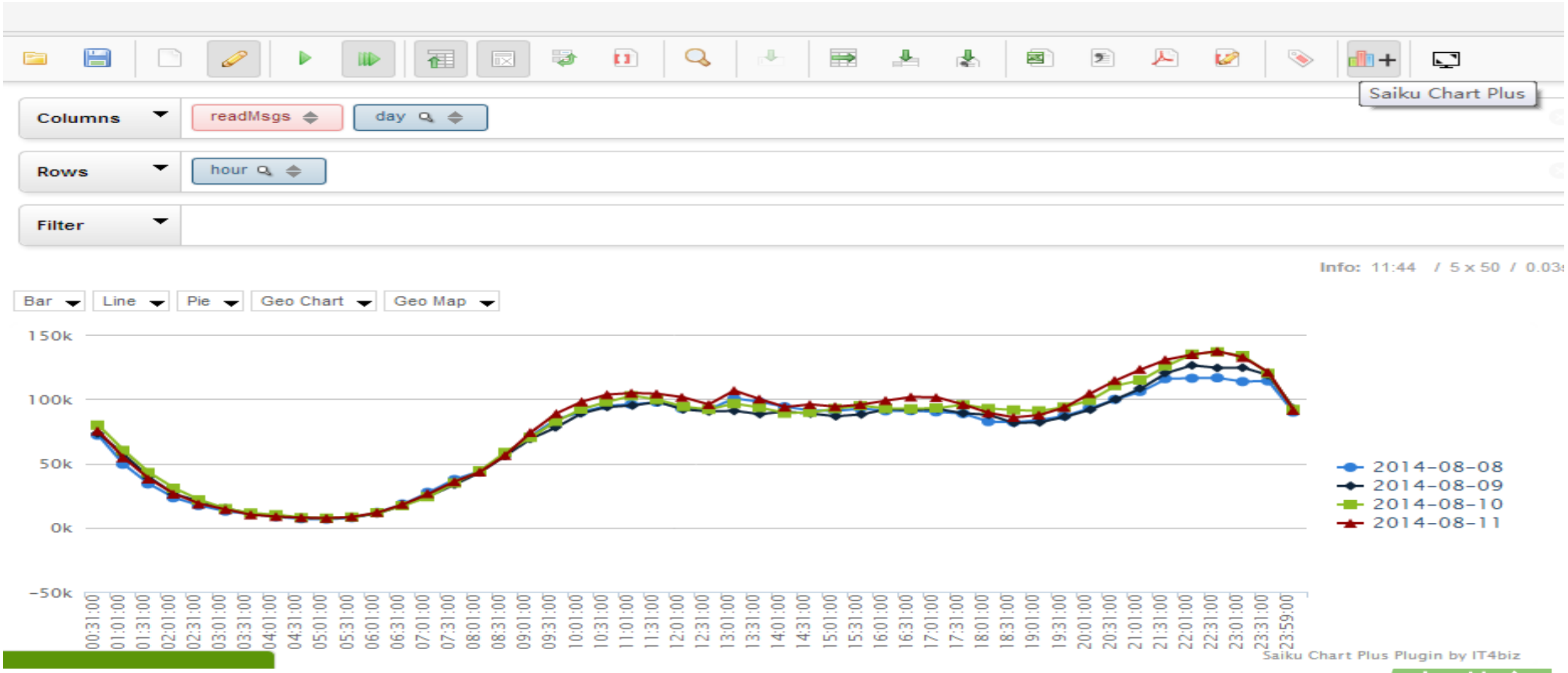
Hbase 宽表设计

- 社交网站私信收发案例

RowKey（id+type+时间）	列簇					
	01	02	03	04	05	06
001-1-20170315						
001-1-20170316						
001-1-20170317						
001-1-20170318						

Hbase 宽表表设计

- 社交网站私信收发案例



Hbase 负载均衡

- 全局计划
- 随机分配计划
- 批量启动分配计划

Hbase 全局计划

全局计划是最常用的负载均衡，它贯穿在整个集群的平稳运行期内，负载均衡以特定时间间隔（hbase.balancer.period默认是5分钟）执行一次，重新分配整个集群的Region。一般情况下不需要修改该值，当集群写入操作非常频繁时，可以将该值调小（比如3分钟）；当集群写入操作非常稀疏时，可以将该值调大（比如10分钟），这样能够有效地、合理地利用系统资源。

vi hbase-site.xml

hbase.balancer.period

Master执行region balancer的间隔。

默认: 300000

Hbase 全局计划

以下情况不执行全局计划：

- 负载均衡开关balanceSwitch关闭
- Hmaster未完成初始化
- 有正在处理的Dead RegionServer
- RegionServer 上的平均Region 数量小于等于1

Hbase 全局计划执行流程—估算

- 总Region个数totalRegion=17+4+29=50
- 每个Region平均承载Region=50/3=16.7
- 最小Region min=numRegions/numRegionServers=16
- 最大Region max=numRegions%numRegionServers=0 ? min : min+1=17

Hbase 全局计划执行流程—查找

- RegionServer上的Region数量从大到小排序，计算该RegionServer上需要转移的

  RegionToMove=Math.min(regionCount-max,regions.size())

  //RegionToMove=Math.min(29-17=12,17)=12

  预留12个作为转移计划，此时并没有真正转移

Hbase 全局计划执行流程—排序

- RegionServer 上的Region数量从小到大排序，如果找一个负载比min还小的，就把12个region分配给最小的。

Hbase 随机分配计划

适用场景：用于新加入的RegionServer，随机分配Region，即使分配的不合理，后续交给全局计划（默认每隔5分钟处理一次）。

Hbase 批量启动分配计划

顾名思义：批量启动分配计划应用于集群启动时，决定Region分配到哪台机器，

org.apache.hadoop.hbase.master.DefaultLoadBalancer 类中包含两种批量启动分配计划：

- retainAssignment：留存分配。尝试使用MATA中的分配信息，有分配信息的按照原 有的分配信息分配Region，剩下的Region随机分配。
- roundRobinAssignment：循环分配。

Hbase 通过shell控制负载均衡

在Hbase shell 中，使用balance_switch 和 balancer 手动控制集群的负载均衡。当维护或者重启一个RegionServer时，会关闭负载均衡，使得Region在RegionServer上的分布不均，此时需要手动开启负载均衡。

- balance_switch 用于开启负载均衡功能
- Balancer用于调用负载均衡类，实现负载均衡，输出结果为true表示“负载均衡”已经被成功触发，并在后台执行。

```
Hbase(mian)>balance_switch true /false
Hbase(mian)> balancer
```

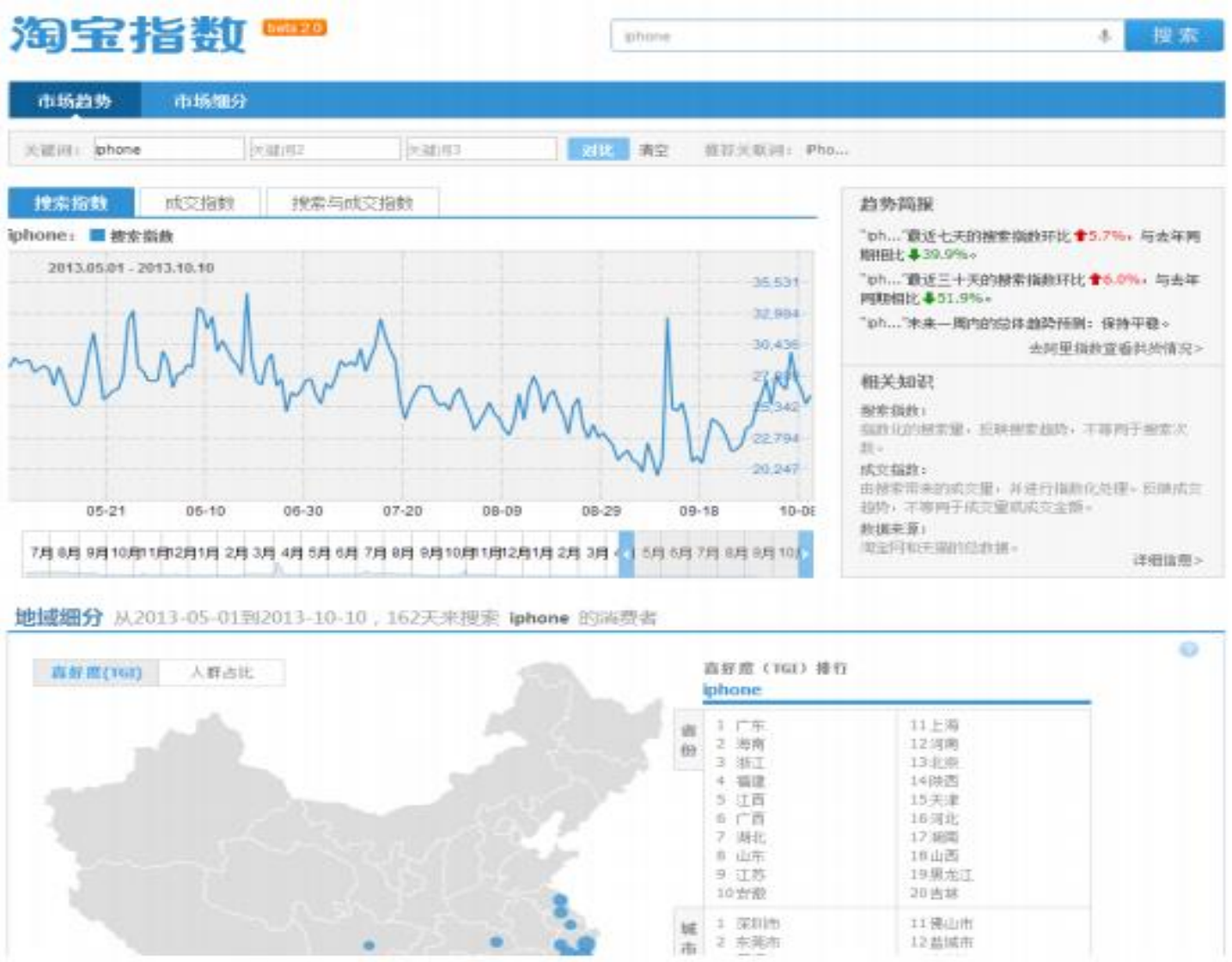
05 应用案例

何时使用HBase

- 需对数据进行随机读操作或者随机写操作；
- 大数据上高并发操作，比如每秒对PB级数据进行上千次操作；
- 读写访问均是非常简单的操作

- storing large amounts of data (100s of TBs)
- need high write throughput
- need efficient random access (key lookups) within large data sets
- need to scale gracefully with data
- for structured and semi-structured data
- don't need full RDMS capabilities (cross row/cross table transactions, joins, etc.)

HBase在淘宝的应用



HBase在淘宝的应用

- 交易历史记录查询系统
  - 百亿行数据表，千亿级二级索引表
  - 每天千万行更新
  - 查询场景简单，检索条件较少
  - 关系型数据库所带来的问题
  - 基于userId + time + id rowkey设计
  - 成本考虑

某移动移动上网日志查询系统

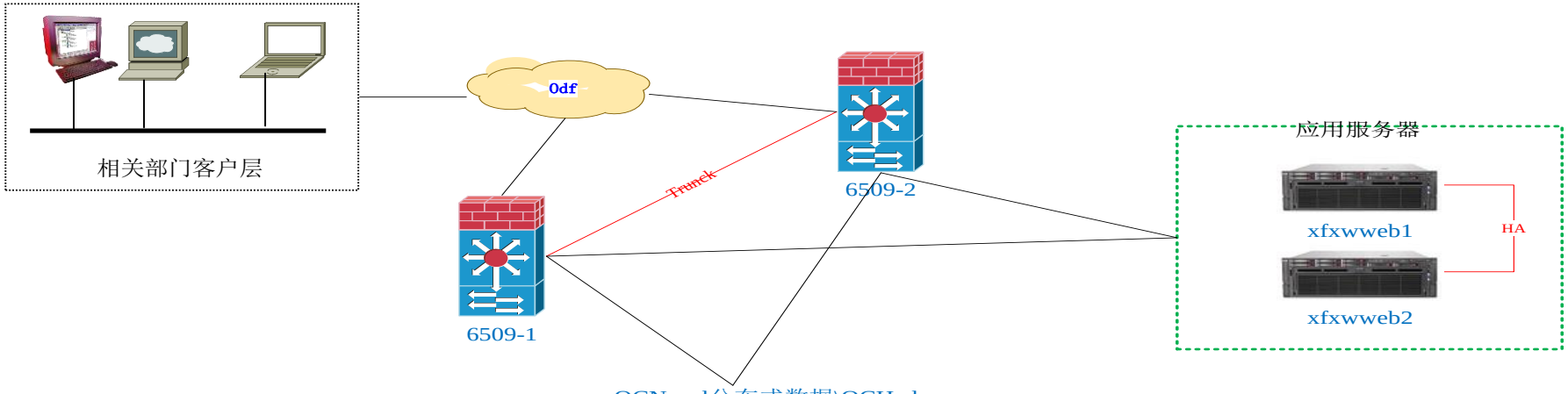
建设目标

- 实现客户上网消费账单透明化，即可针对客户投诉或流量质疑等诸多相关数据流量问题实现流量账单明细级查询

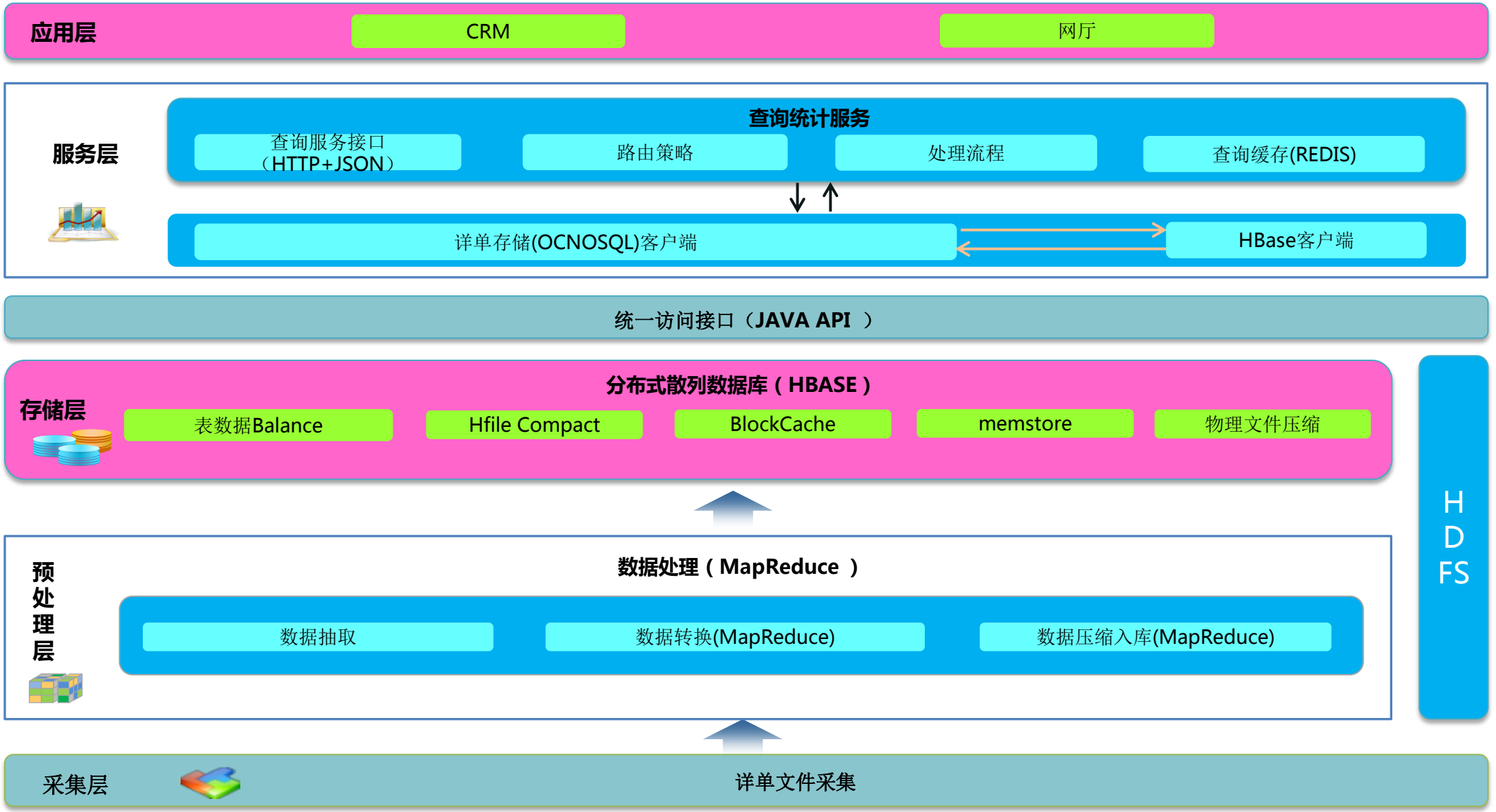
集群介绍

- 集群规模：2台namenode，25台datanode，2台web查询服务
- 存储周期：3 + 1，3个月历史数据，1个月实时数据
- 每月存储量：45T GPRS上网清单日志
- 查询响应时间：平均在300ms左右

集群部署


相关部门客户层
Odf
6509-2
Trunck
6509-1
应用服务器
xfxwweb1
HA
xfxwweb2
OCNosql分布式数据\OCHadoop
NameNode/JobTracker
xfxwmager1
xfxwmager2
DataNode\TaskTracker\RegionServer
xfxwdnod1
xfxwdnod2
xfxwdnod3
xfxwdnod4
xfxwdnod5
xfxwdnod6
xfxwdnod7
xfxwdnod8
xfxwdnod9
xfxwdnod10
Journalnode
xfxwdnod11
xfxwdnod12
xfxwdnod13
xfxwdnod14
xfxwdnod15
Zookeeper
xfxwdnod16
xfxwdnod17
xfxwdnod18
xfxwdnod19
xfxwdnod20
Master
xfxwdnod21
xfxwdnod22
xfxwdnod23
xfxwdnod24
xfxwdnod25

系统架构


应用层
CRM
网厅
服务层
查询统计服务
查询服务接口（HTTP+JSON）
路由策略
处理流程
查询缓存(REDIS)
详单存储(OCNOSQL)客户端
HBase客户端
统一访问接口（JAVA API ）
存储层
分布式散列数据库（HBASE）
表数据Balance
Hfile Compact
BlockCache
memstore
物理文件压缩
HDFS
预处理层
数据处理（MapReduce ）
数据抽取
数据转换(MapReduce)
数据压缩入库(MapReduce)
采集层
详单文件采集

HBase 表设计

Rowkey设计：
MD5(手机号码)取前三位 + 手机号码 + 开始时间（到秒） + 唯一标识，
如：f9a18601134210201406242345 31f059b84aba2e6095，唯一标识是对整条记录取16位md5值，防止rowkey重复

建表原则：
每月定期建表，表名为GPRS_yyyyMM，如：GPRS_201407

预建分区：
提前对表进行分区，防止region分裂，北京移动生产环境每张表800个分区

建表方式：
通过crontab定期建未来三个月表，包括hbase表、phoenix表

压缩方式：
采用SNAPPY + PefixTree的压缩方式，HFile文件压缩采用SNAPPY，
dataBlock压缩采用PrefixTree，压缩比大概为1:1.6

HBase 表字段

序号	字段名称	字段编码	字段类型
1	用户帐号	PHONE_NO	VARCHAR
2	终端型号id	TERM_MODEL_ID	NUMBER
3	位置区号	LAC	Number
4	小区识别码	CI	Number
5	业务编号	BUSI_ID	Number
6	业务入口编号	APP_ID	Number
7	业务类别编号	BUSI_TYPE_ID	Number
8	开始时间	START_TIME	VARCHAR
9	上行流量	UP_FLOW	Number
10	下行流量	DOWN_FLOW	Number
11	网站名称编号	SITE_NAME_ID	Number
12	网站频道编号	SITE_CHANNEL_ID	Number
13	计费id	CHARGING_ID	Number
14	信令面GGSN的IP	C_GGSN_IP	VARCHAR
15	一级域名	EX_TL_DOMAIN	VARCHAR
16	完整域名	EX_COMP_DOMAIN	VARCHAR
17	访问URL	ACCE_URL	VARCHAR
18	流量和	FLOW	Number
19	预留字段1	REV_01	VARCHAR

SQL查询

集成Apache Phoenix

- **Apache Phoenix：**在OCNoSQL1.6版本引入Phoenix， Phoenix查询引擎会将SQL解析成一系列HBase Scan，并编排执行以生成标准的JDBC结果集。Phoenix直接使用HBase API、协同处理器、自定义过滤器。
- **通过JDBC访问HBase：**

```
//查询18601134210手机号码2014年6月24号上网日志明细清单
sql = "select * from GPRS_201406 where id>= ‘f9a18601134210 20140624’ and id <= ‘f9a18601134210 20140624’ ";
Class.forName("org.apache.phoenix.jdbc.PhoenixDriver");
Connection conn = DriverManager.getConnection("jdbc:phoenix:ubuntu:2181");
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
while(rs.next) {
        …
}
```

大讲台
dajiangtai.com

THANKS

黄丽老师：3354223855

小夏老师：972628726