奇虎360 HBASE二级索引的设计与实践

赵健博
系统部
2015/4/24

内容梗概

- 背景
- 设计
- 实践

背景

- 仅基于RK的索引问题：
  - 索引单一
  - 多维度（字段/列）查询困难
    - 多字段分别作为RK，写入多次
    - 组合字段作为RK，设计复杂，不灵活
  - 不经过索引的并行scan过滤，大量资源消耗，无实效性可言
- 多维度实时查询需求强烈：
  - 基于DNS 的网络行为特征分析
  - 基于病毒样本的网络行为特征分析

- 通用模式：
  - 将数据结构化存储（海量的数据，千亿级别）
  - 对多个列或者多列之间建立索引
  - 指定条件：
    - 单列等值、范围
    - 多列之间与、或
  - 获取结果：
    - 满足条件的记录
    - 满足条件的记录个数
    - 满足条件的记录按照某列，或者某几列的统计

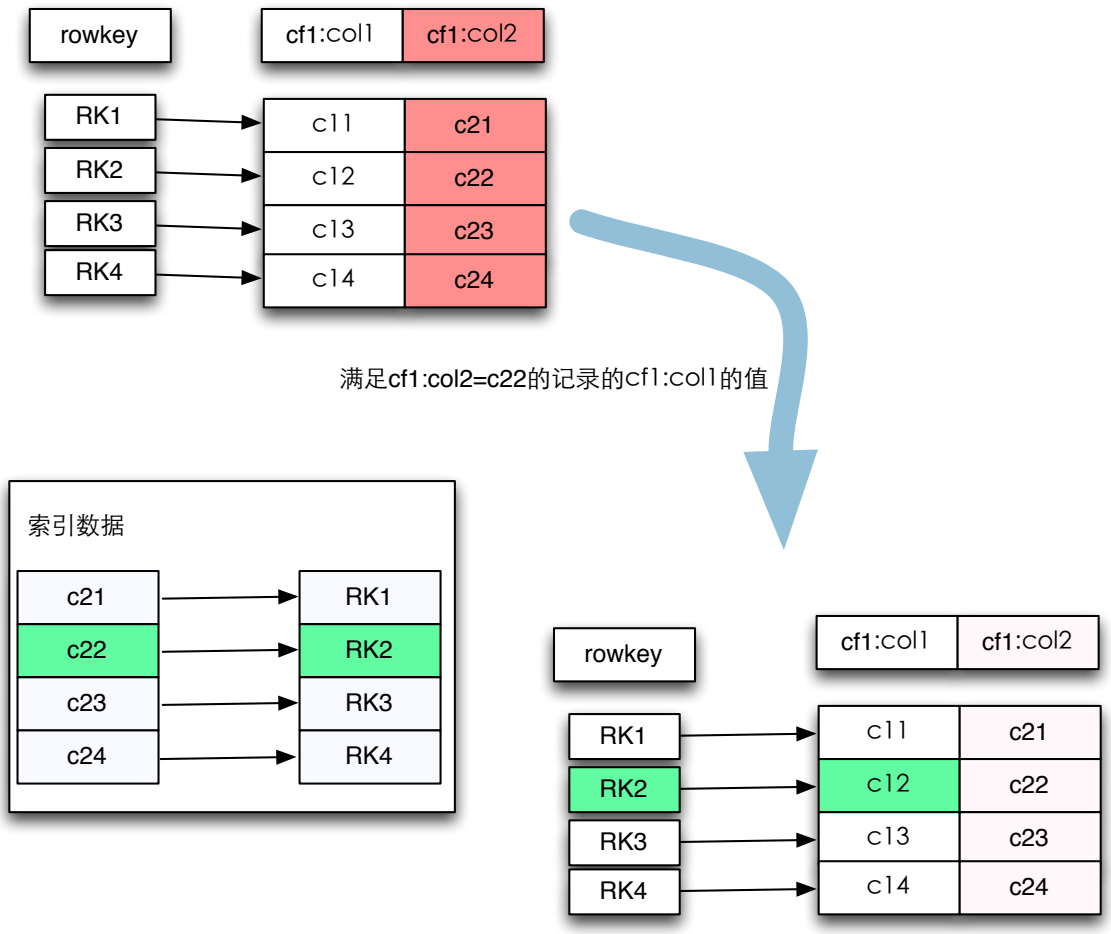
内容梗概

- 需求
- 设计
- 实践

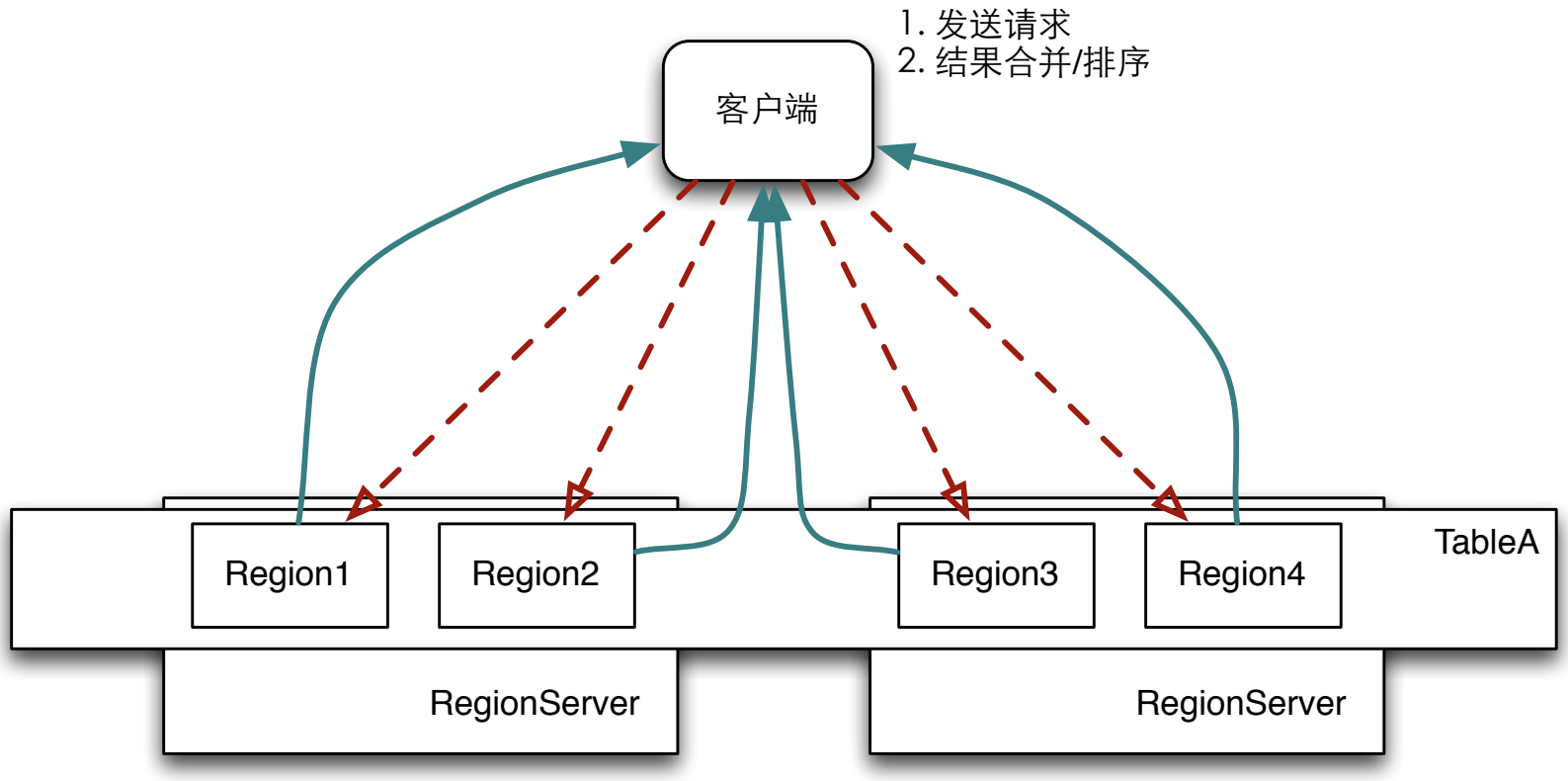
设计

- 总体设计
- 索引设计
- 索引类型
- 写路径
- 读路径
- 分裂
- 索引重建
- 优化
- 汇聚操作
- 模糊查询

总体设计

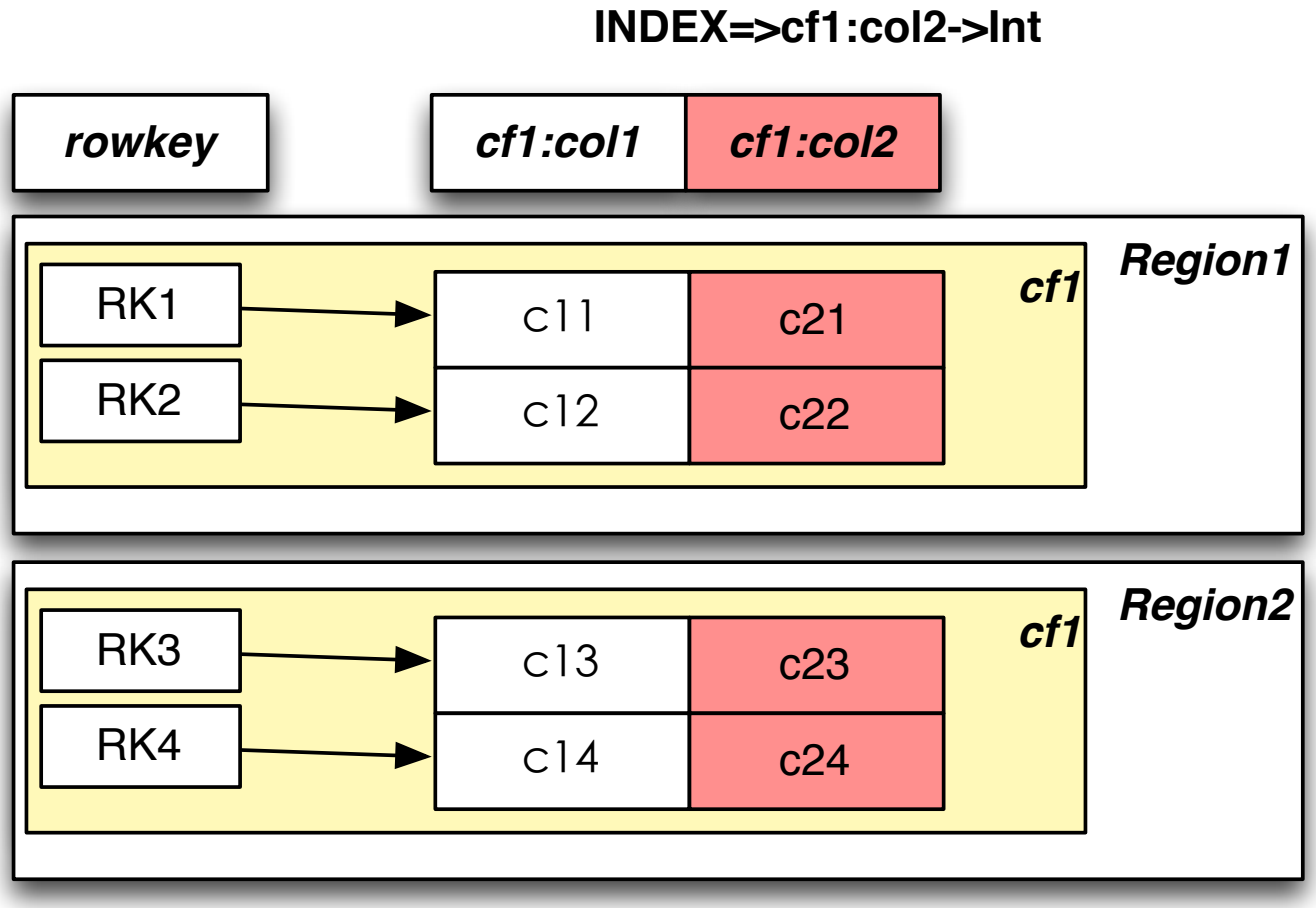

rowkey
cf1:col1
cf1:col2
RK1
RK2
RK3
RK4
c11
c12
c13
c14
c21
c22
c23
c24
满足cf1:col2=c22的记录的cf1:col1的值
索引数据
c21
c22
c23
c24
RK1
RK2
RK3
RK4
rowkey
cf1:col1
cf1:col2
RK1
RK2
RK3
RK4
c11
c12
c13
c14
c21
c22
c23
c24

总体设计

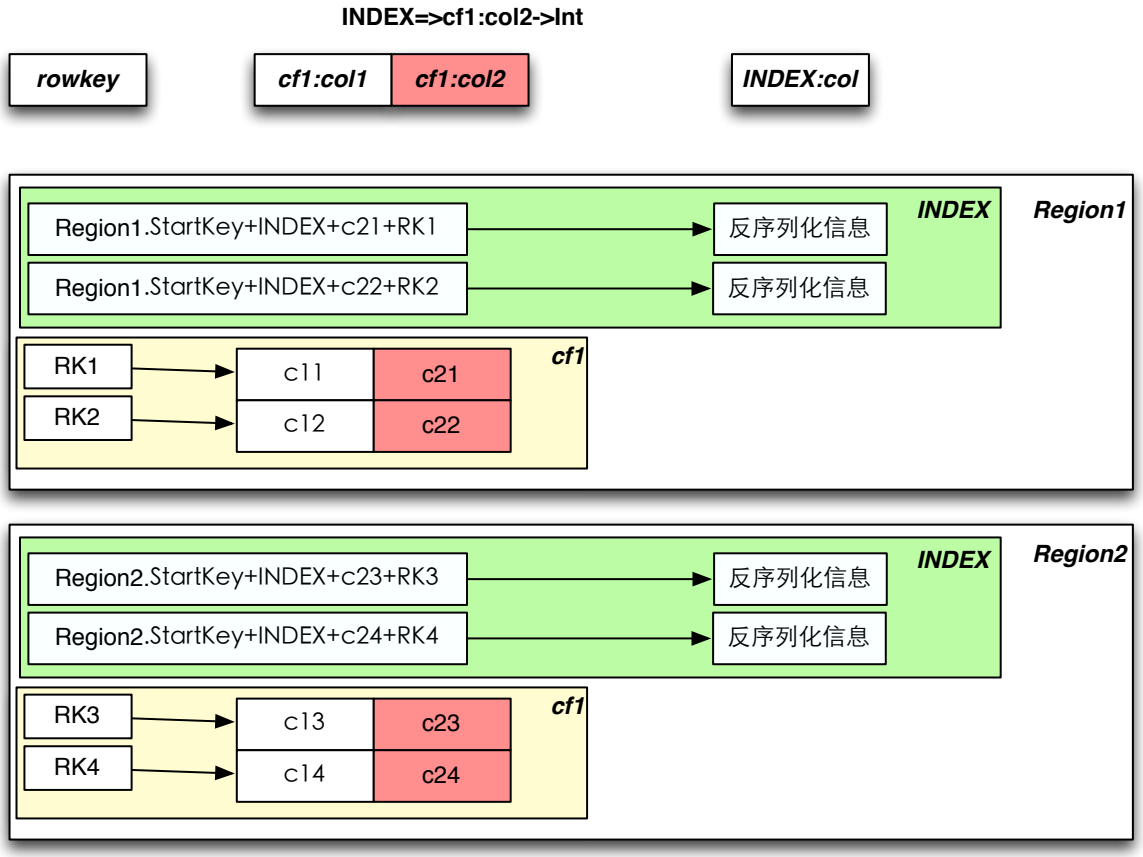


分布式、并发式数据查询与索引构建

索引设计

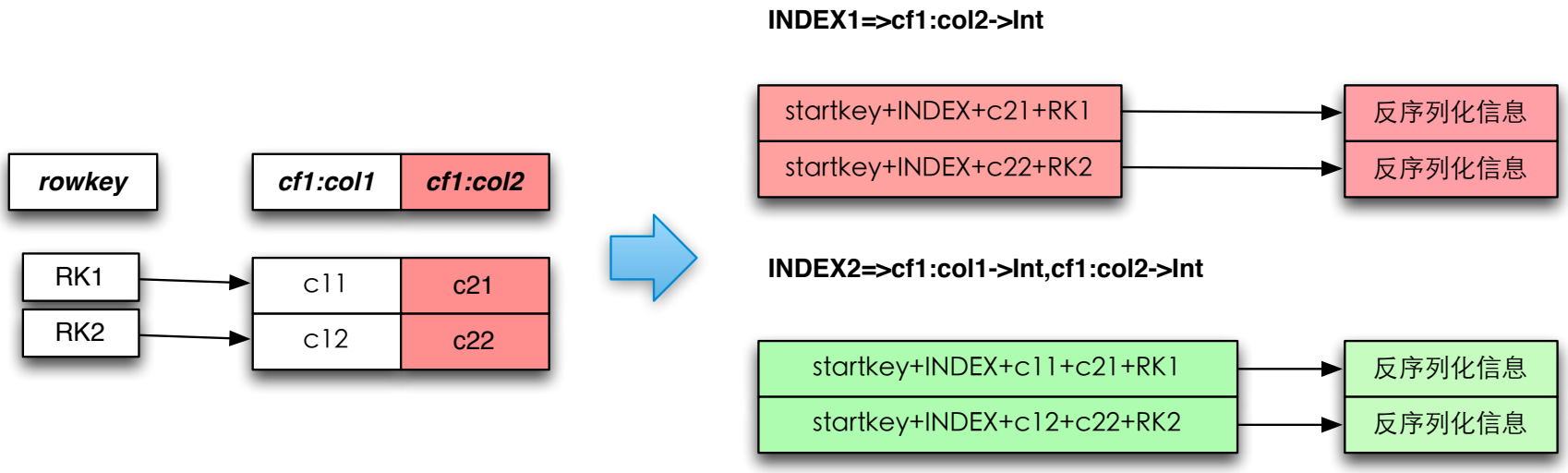

INDEX=>cf1:col2->Int
rowkey
cf1:col1
cf1:col2
Region1
cf1
RK1
RK2
c11
c21
c12
c22
Region2
cf1
RK3
RK4
c13
c23
c14
c24

索引设计



索引和数据在同一个Region内，但索引数据存储在单独INDEX family中

索引设计



反序列化信息：version+startkey.len+RK.len+value1.len+value2.len+...

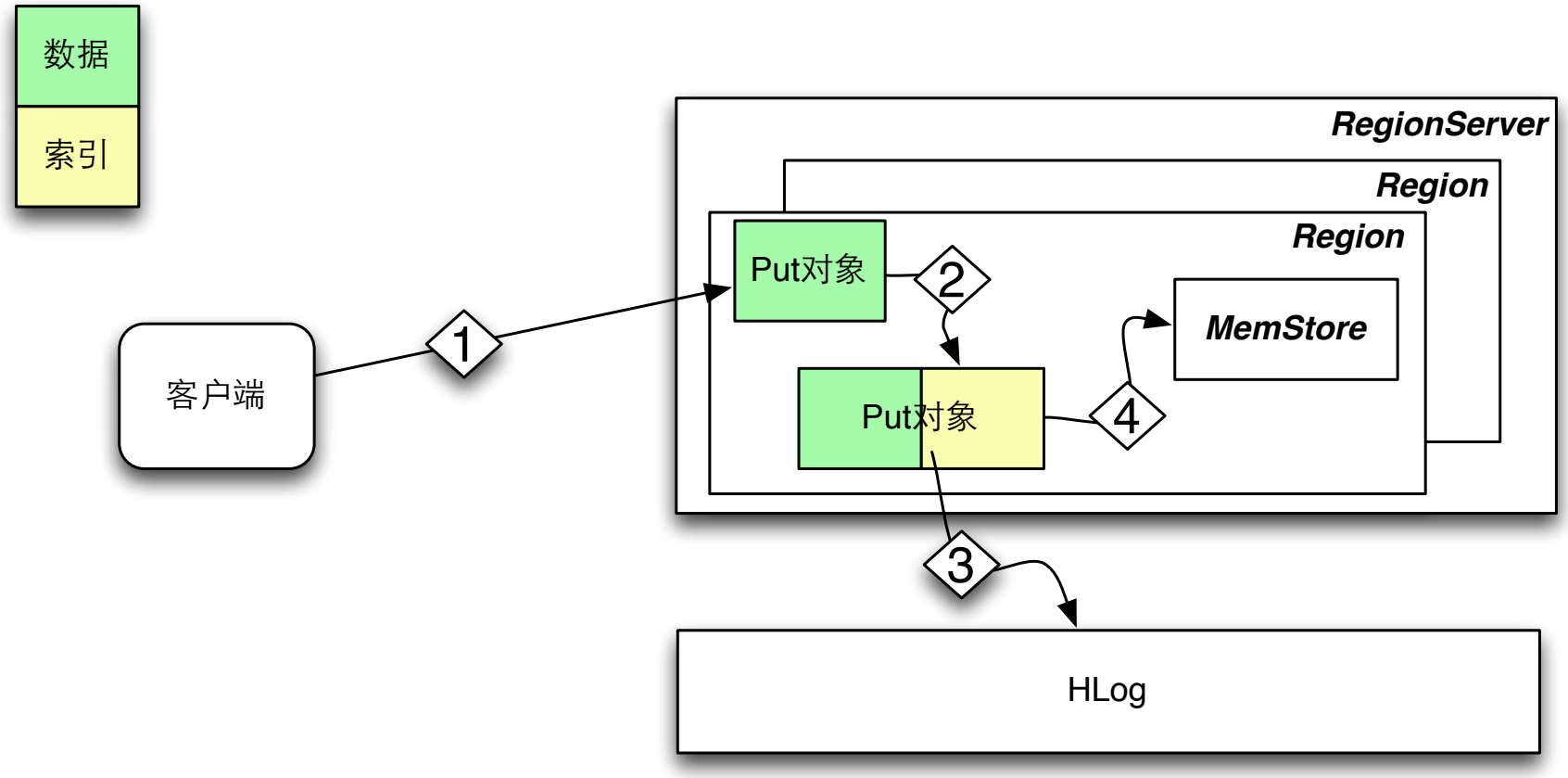
索引设计

- 索引的说明存储在表schema中
  - {NAME => 'whitelist_dsv', FAMILIES => [{NAME => 'INDEX', BLOOMFILTER => 'INDEXROW', COMPRESSION => 'LZO', VERSIONS => '1', INDEX=>{IDX_CERT=>value:cert_sha1# IDX_SIGN=>value:sign_corp#IDX_SHA1=>value:sha1} }, {NAME => 'value', BLOOMFILTER => 'ROW', VERSIONS => '1', COMPRESSION => 'LZO'}]}
- 索引说明可以动态变更

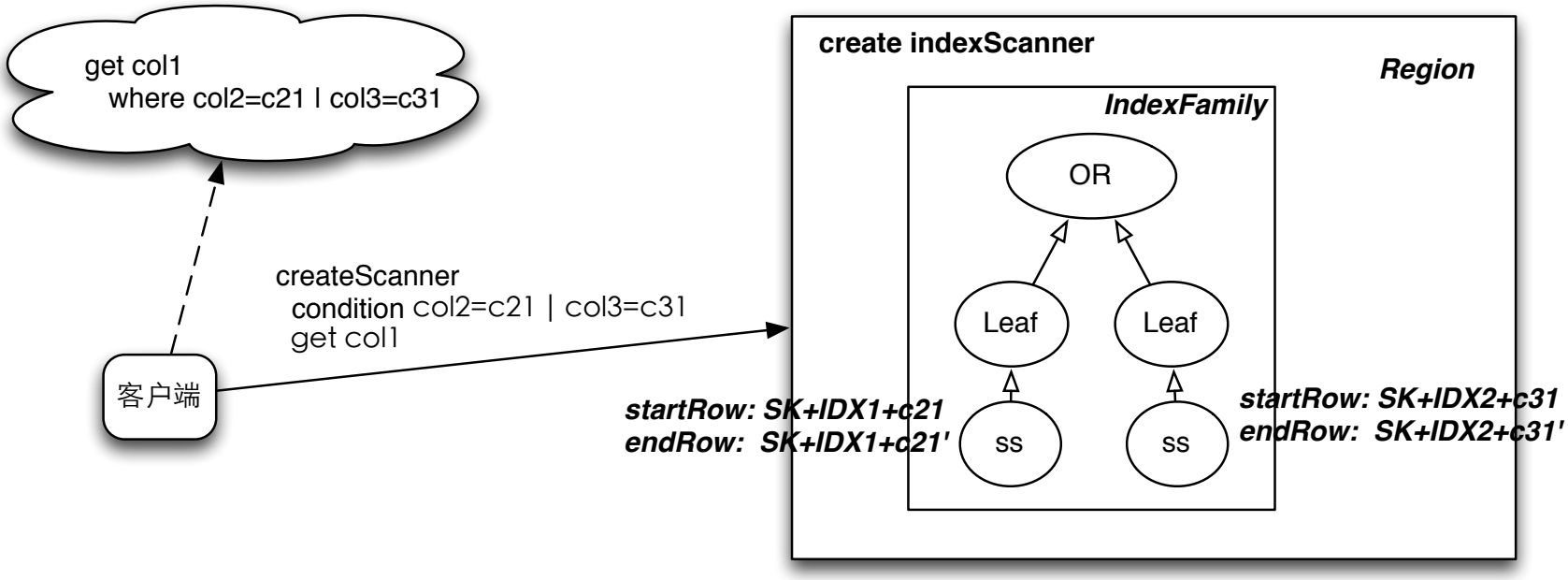
索引类型

索引类型	长度	备注
Char	2	等值，范围查询
Byte	1	等值，范围查询
Short	2	等值，范围查询
Int	4	等值，范围查询
Long	8	等值，范围查询
Float	4	等值，范围查询
Double	8	等值，范围查询
String	-	精确匹配，范围查询
Text	-	模糊查询
多列组合索引	-	索引优化

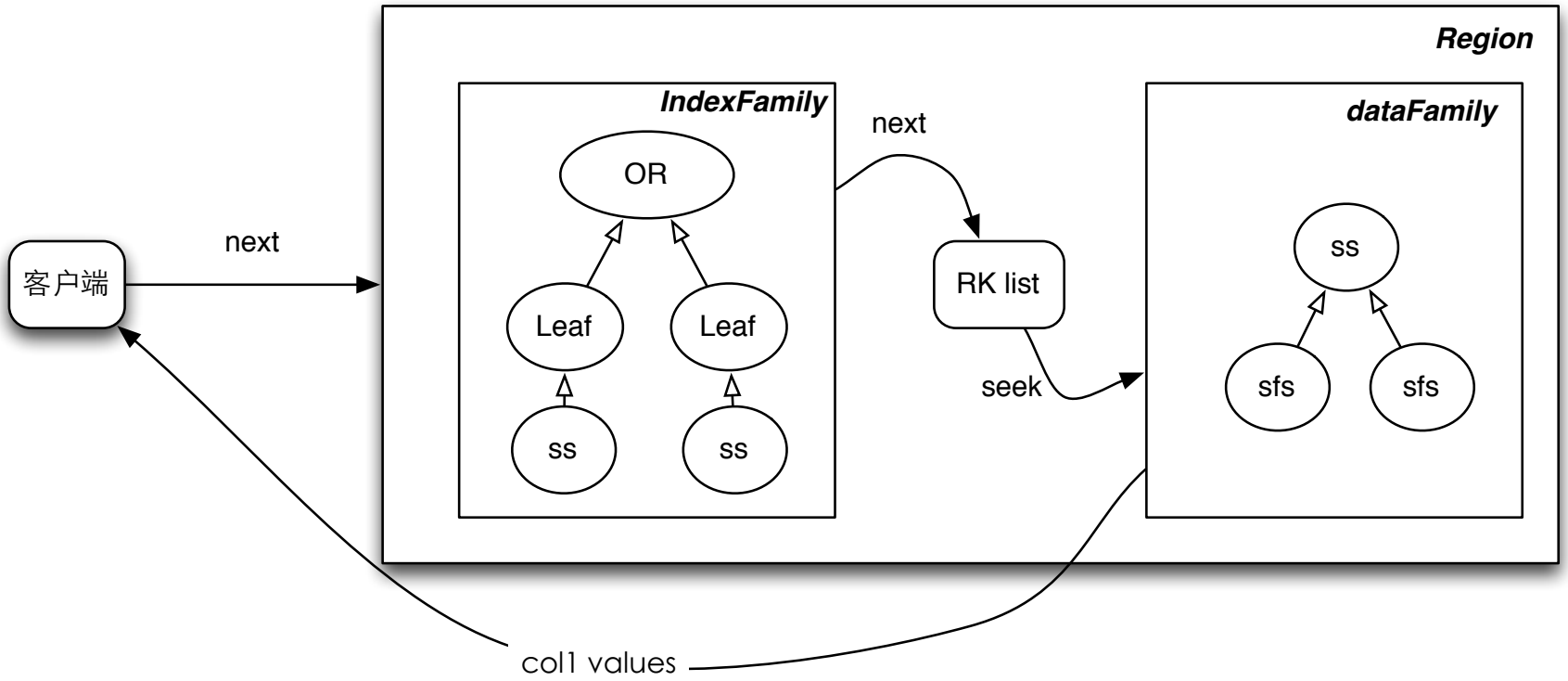
写路径


数据
索引
客户端
1
RegionServer
Region
Region
Put对象
2
MemStore
Put对象
4
3
HLog

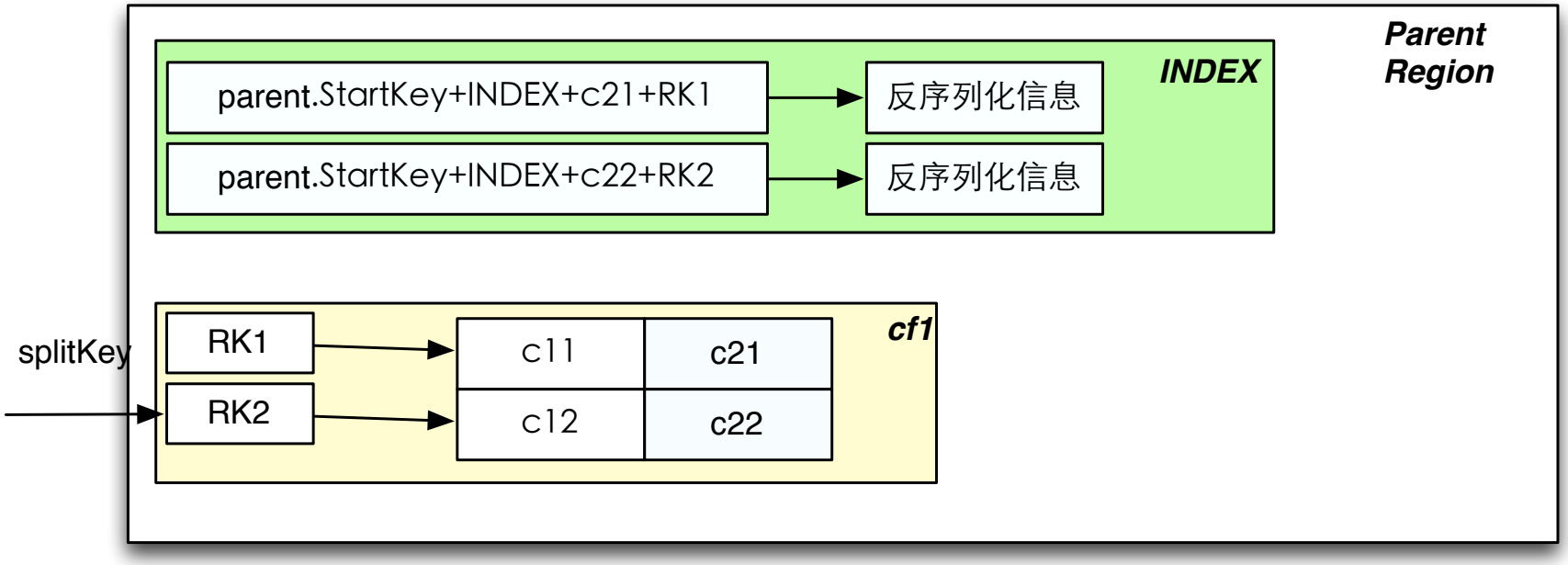
读路径


get col1
where col2=c21 | col3=c31
createScanner
condition col2=c21 | col3=c31
get col1
客户端
create indexScanner
Region
IndexFamily
OR
Leaf
Leaf
ss
ss
startRow: SK+IDX1+c21
endRow: SK+IDX1+c21'
startRow: SK+IDX2+c31
endRow: SK+IDX2+c31'

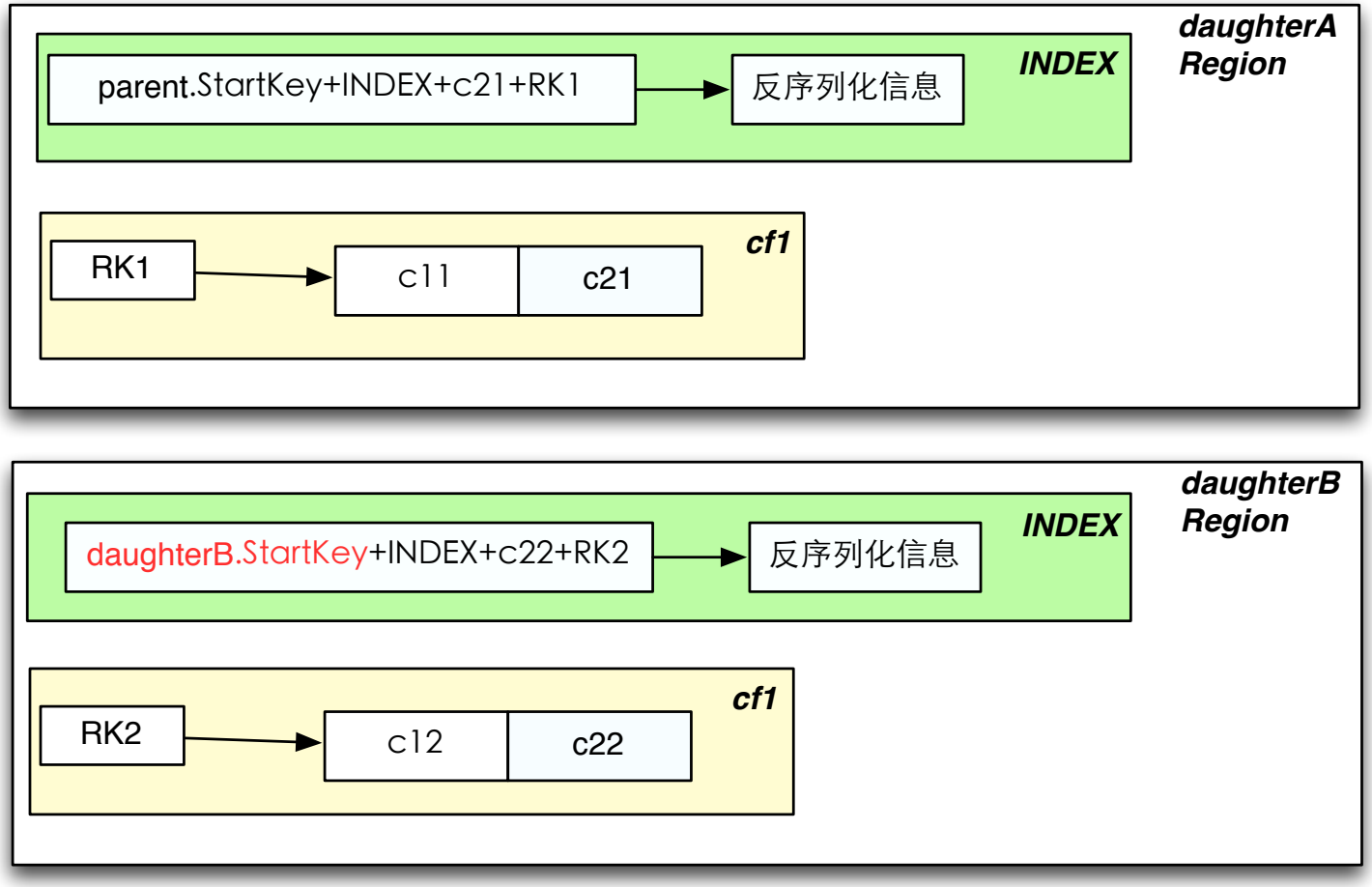
读路径


Region
IndexFamily
OR
Leaf
Leaf
ss
ss
next
RK list
seek
dataFamily
ss
sfs
sfs
客户端
next
col1 values

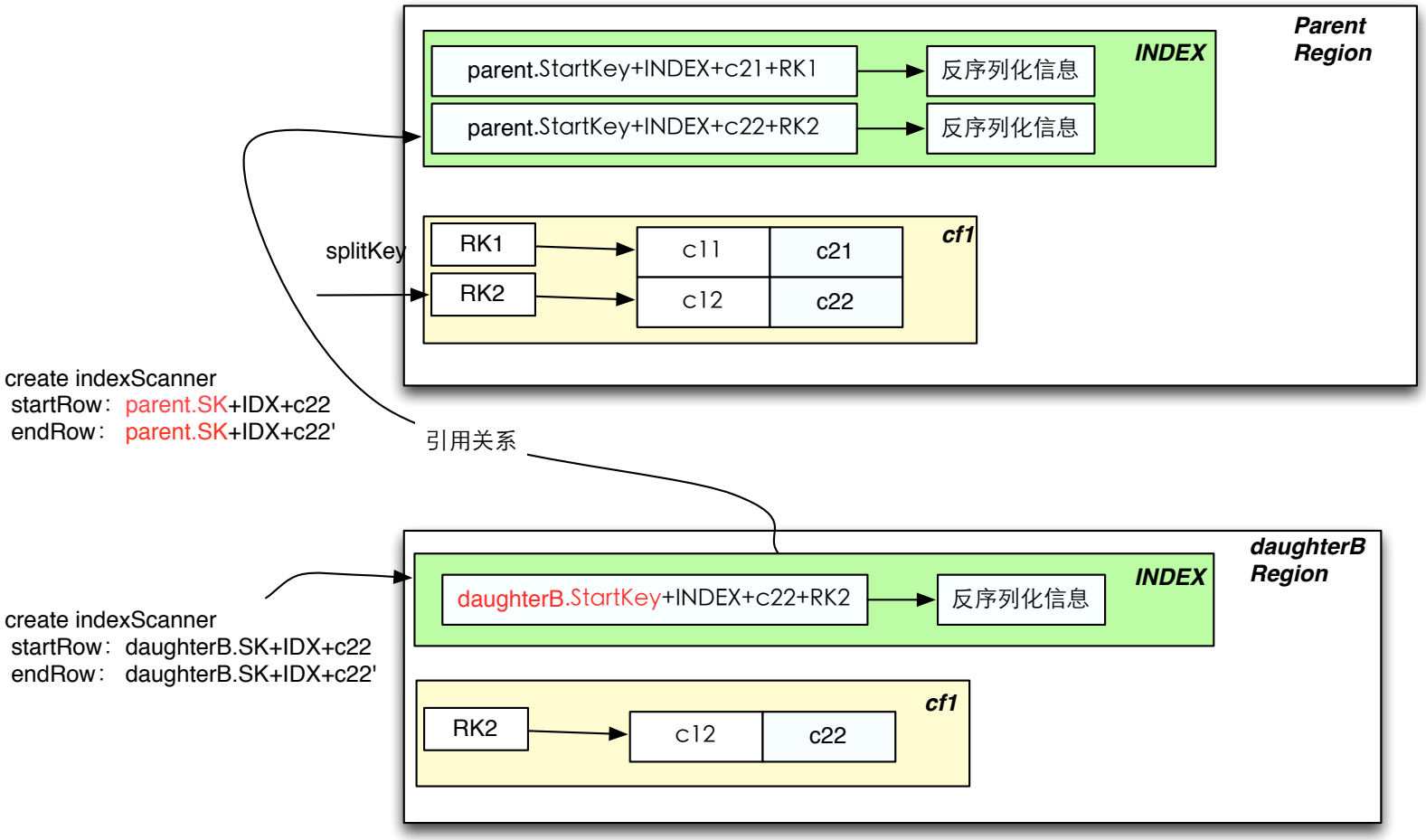
分裂


Parent Region
INDEX
parent.StartKey+INDEX+c21+RK1
反序列化信息
parent.StartKey+INDEX+c22+RK2
反序列化信息
splitKey
cf1
RK1
RK2
c11
c21
c12
c22

分裂


daughterA Region
INDEX
parent.StartKey+INDEX+c21+RK1
反序列化信息
cf1
RK1
c11
c21
daughterB Region
INDEX
daughterB.StartKey+INDEX+c22+RK2
反序列化信息
cf1
RK2
c12
c22

分裂


Parent Region
INDEX
parent.StartKey+INDEX+c21+RK1
反序列化信息
parent.StartKey+INDEX+c22+RK2
反序列化信息
cf1
splitKey
RK1
RK2
c11
c21
c12
c22
create indexScanner
startRow: parent.SK+IDX+c22
endRow: parent.SK+IDX+c22'
引用关系
daughterB Region
INDEX
daughterB.StartKey+INDEX+c22+RK2
反序列化信息
cf1
RK2
c12
c22
create indexScanner
startRow: daughterB.SK+IDX+c22
endRow: daughterB.SK+IDX+c22'

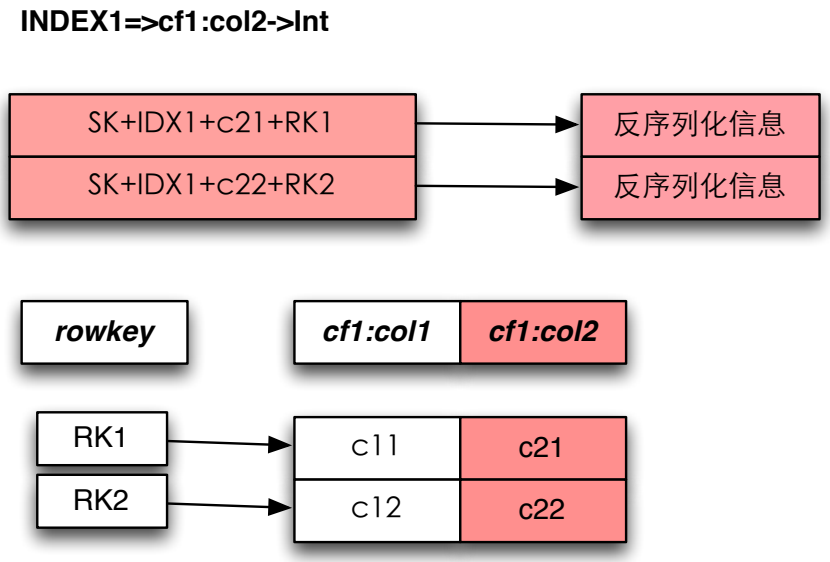
索引重建

- 索引会有垃圾：
  - 数据覆盖写操作，导致数据超过最大version。但由于value不同，导致索引产生垃圾
- 索引重建流程：
  - 重建前，记录当前所有索引文件
  - 扫面数据，重新生成索引
  - 删除掉之前记录的索引文件，重建完成
- 状态跟踪：
  - INDEXTABLE保存状态

优化

- 列和时间建立联合索引
- 常用组合查询建立联合索引
- 查询表达式的转换：
  - (A | B) & (C | D) ⇔ (A&C) | (A&D) | (B&C) | (B&D)
- New Bloomfilter type
- 多范围与操作查询优化
- 索引的带外数据（TODO）

New BloomFilter Type



Simple Case:

get cf:col1 where cf1:col2 = c22

1. 查询索引:
   scan (sr=SK+IDX1+c22,
   er=SK+IDX1+c22')
   result {RK2}

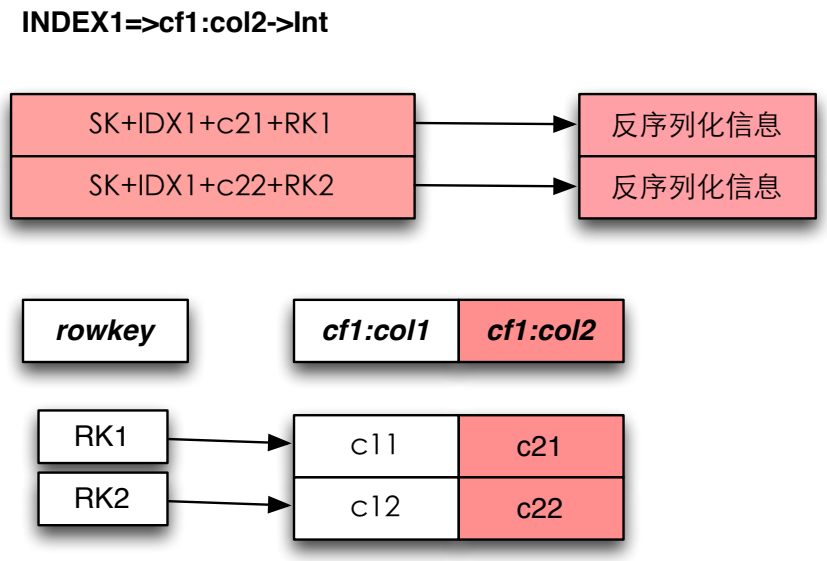
2. 获取数据:
   get 'RK2', 'cf1:col1'
   result {c12}

查询索引数据使用的是scan，bloomfilter失效！！！

New BloomFilter Type

- INDEXROW
- 针对定值查询优化
  - 单列索引定值
  - 组合索引定值
- 写入时，对索引数据的中的原RowKey截断，构建BF
  - StartKey+INDEX+value+原RowKey
  - 去除原RowKey后（黄色部分），构建BF数据
- 查询索引时，根据检索表达式定值部分匹配BF

New BloomFilter Type



Simple Case:

get cf:col1 where cf1:col2 = c22

1. 查询索引:
   scan (sr=SK+IDX1+c22,
   er=SK+IDX1+c22')
   result {RK2}

2. 获取数据:
   get 'RK2', 'cf1:col1'
   result {c12}

查询索引数据使用的是scan，bloomfilter失效！！！

数据写入时，根据如下数据构造BF数据:
SK+IDX1+c21
SK+IDX1+c22

查询索引时，根据如下信息匹配BF数据:
SK+IDX1+c22

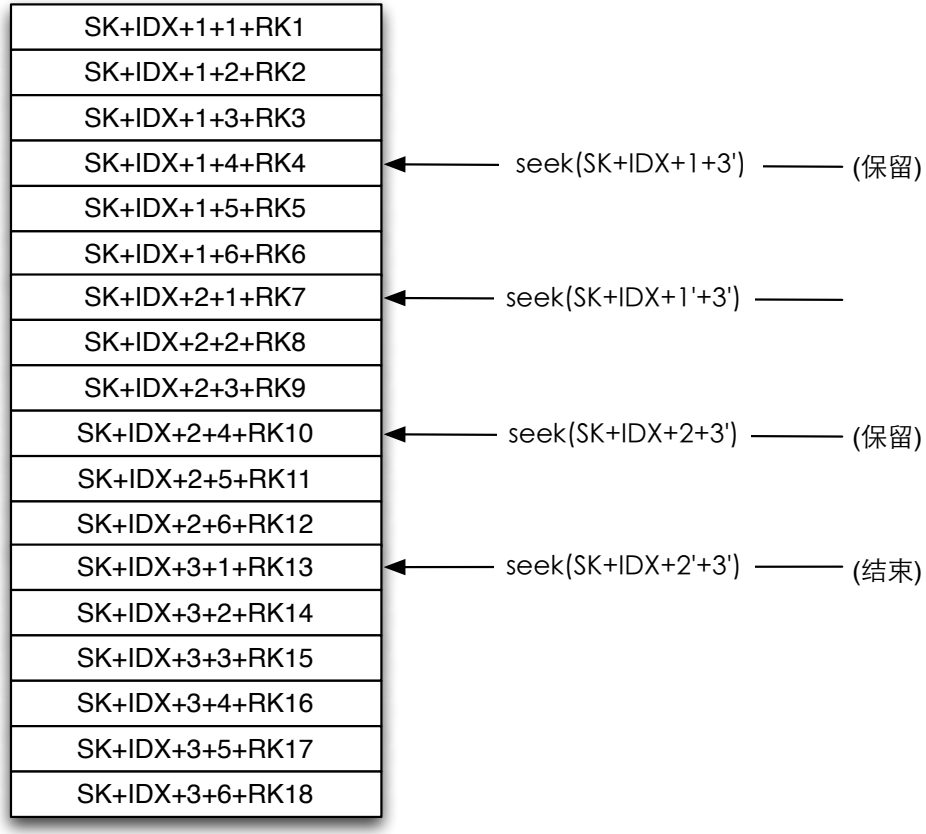
多范围与操作查询优化

- a’ < A < a’’ & b’ < B < b’’
- 传统过程：
  - 获取列A在(a’, a’’)区间的RK集合RKS1
  - 获取列B在(b’, b’’)区间的RK集合RKS2
  - 对RKS1和RKS2取交集
- 问题：
  - A和B集合任何一个过大，查询时间都很长
  - 内存风险
- 解决
  - A和B建立联合索引
  - 支持多范围查询，并优化

多范围与操作查询优化

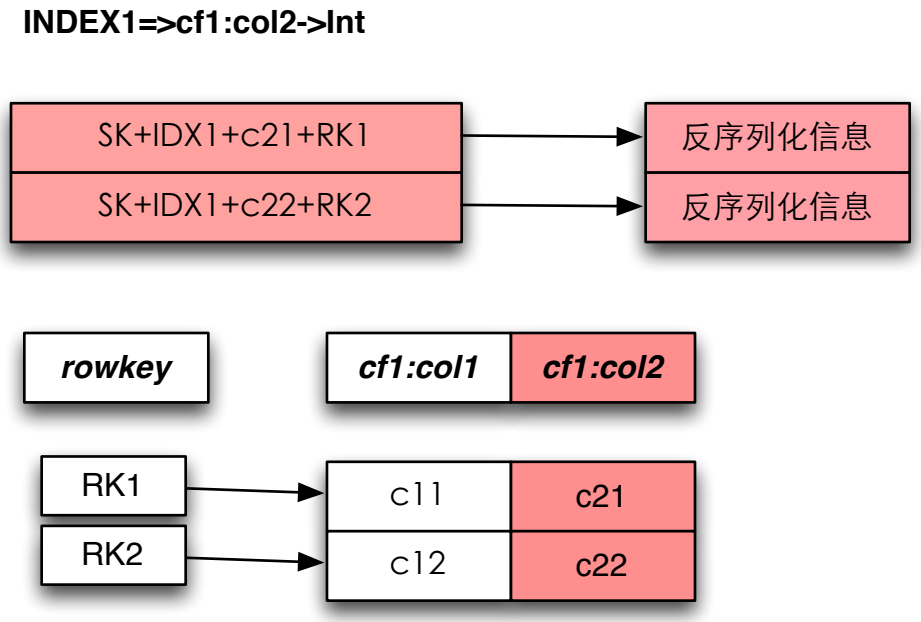
2个col建立索引，索引数据如下：

查询条件：1 <= col1 <3 & 3 < col2 < 5



索引的带外数据

- 检索过程能不能再快？



Simple Case:

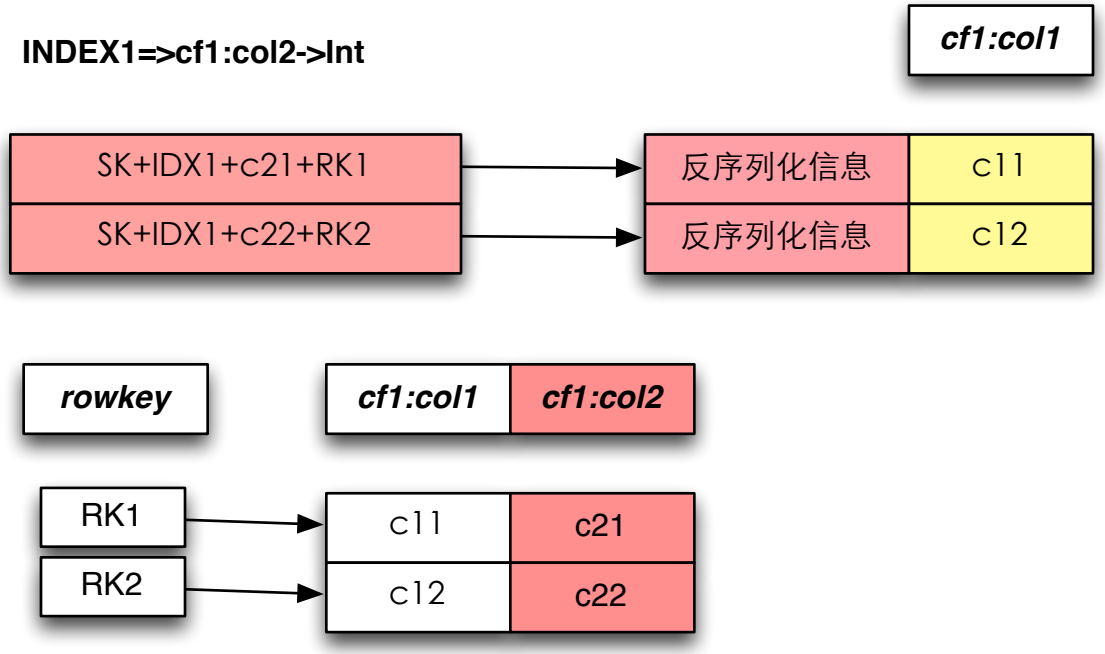
get cf:col1 where cf1:col2 = c22

1. 查询索引:
   scan (sr=SK+IDX1+c22,
   er=SK+IDX1+c22')
   result {RK2}

2. 获取数据:
   get 'RK2', 'cf1:col1'
   result {c12}

问题:
1. 一次查询需要两个步骤
2. 大量的seek操作，极具降低磁盘性能

索引的带外数据



Simple Case:

get cf:col1 where cf1:col2 = c22

查询索引与数据:

scan (sr=SK+IDX1+c22,
er=SK+IDX1+c22')
result {(RK2,c12)}

- 优点：
  - 一次scan，获取结果
  - 顺序IO，极高性能
- 缺点：
  - 更多的存储成本

汇聚操作

- 利用coprocessor框架实现
- 操作：
  - Count
    - 通过索引获取记录，统计获取的记录个数
    - 直接通过索引统计记录个数
  - groupBy
    - 通过索引获取记录，然后列计数统计
- 采样优化
  - 如果仅仅是估算大概数量，可以通过采样部分region的统计结果来估算整体的计数情况

模糊查询

- 模糊查询
  - 通过某列值中的某个term查询整行
  - 通过某列值中的子短语查询整行
  - 通过通用规则（*/?）查询整行
- 实现：
  - Lucene引擎引入到HBase
  - 写Text类型列时，建立RK与文档，列值分词与文档的映射。实现分词=>文档=>RK关系
  - 查询时，根据条件，通过Lucene引擎找到文档，然后从文档中取出RK，最后再根据RK获取整行

二级索引方案比较

	奇虎360	华为
单列，多列联合索引	YES	YES
多列之前与或查询	YES	YES
索引动态修改	YES	NO ?
索引重建	YES	NO ?
模糊查询	YES	NO
汇聚操作	YES	NO
单独索引表	NO	YES
Region分配策略修改	NO	YES
数据与索引一致性问题	NO	YES
多范围与操作优化	YES	NO

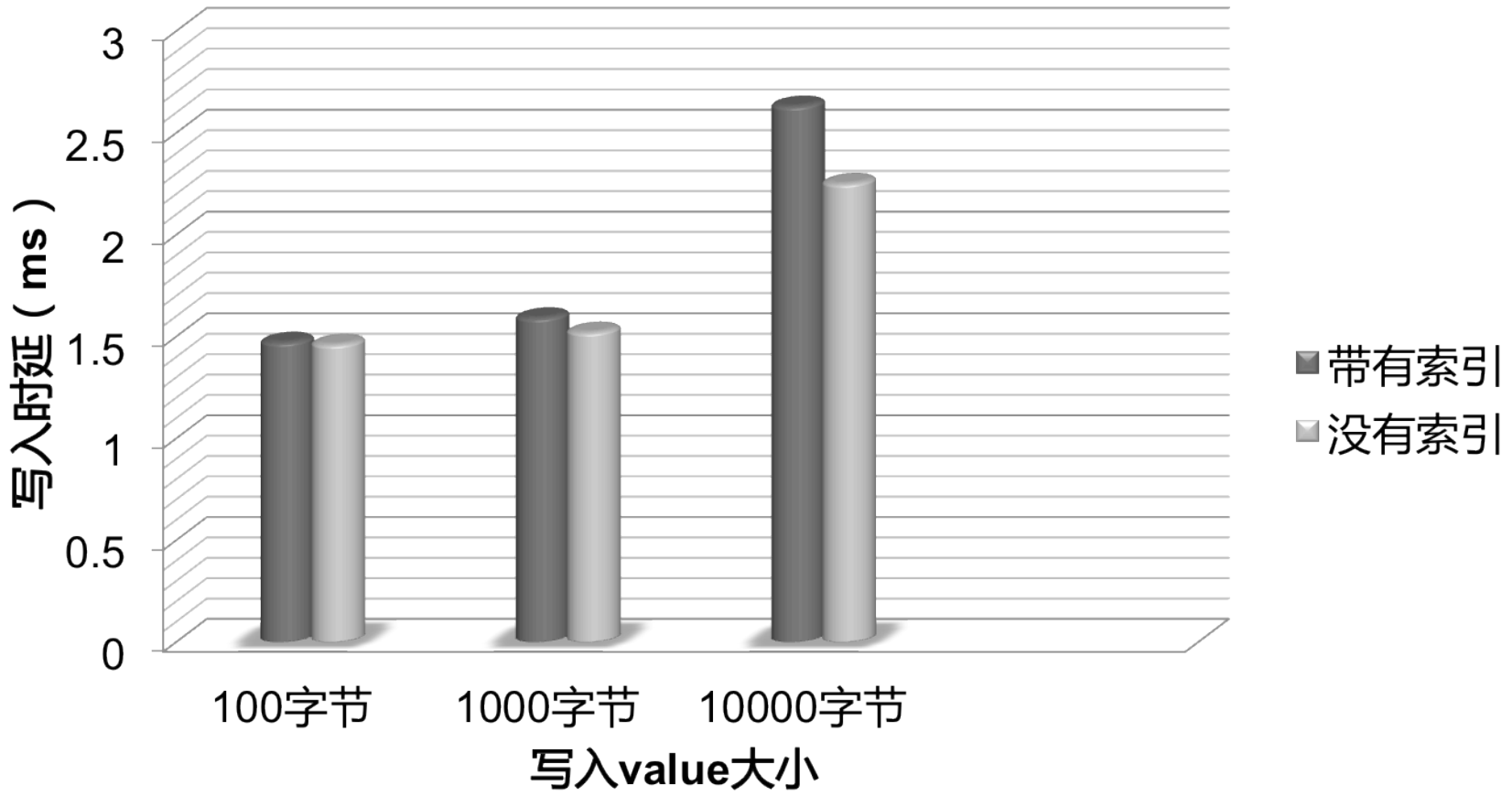
内容梗概

- 需求
- 设计
- 实践

- 表设计
  - 单列索引
  - 单列与时间的联合索引
  - 多列组合索引

- 查询表达式优化
  - (A | B) & (C | D) ⇔ (A&C) | (A&D) | (B&C) | (B&D)

单并发写入性能


3
2.5
2
1.5
1
0.5
0
写入时延（ms）
100字节
1000字节
10000字节
写入value大小
带有索引
没有索引

检索性能数据

70台机器 {CPU：2路6核，内存：64GB，12*4TB磁盘}

	CASE1	CASE2
列与索引列	19列 + 17个索引	10列 + 10个索引
数据规模	5000亿行（18万亿KV）	4000亿行（8万亿KV）
容量	500+TB	200+TB
高频查询	单列定值 (A|B) & (C|D)	单列定值
查询时延	平均5.5s	平均6s
返回条目数	5000	3000+
查询次数	70000+	2400+

360
www.360.cn

Thanks!