数字转型 架构演进

SACC

2019 中国系统架构师大会

SYSTEM ARCHITECT CONFERENCE CHINA 2019

IT168.com ChinaUnix.net ITPUB

2019年10月31-11月2日 | 北京海淀永泰福朋喜来登酒店

Pharos-可插拔的HBase索引组件

王磊
光大银行 企业架构师

王 磊

光大银行 企业架构师，曾任职于IBM全球咨询服务部从事技术咨询工作，具有十余年数据领域研发及咨询经验。目前负责全行数据领域系统的日常架构设计、评审及内部研发等工作，对分布式数据库、Hadoop等基础架构研究有浓厚兴趣， Pharos架构设计和主要开发者。

个人公众号：金融数士

Bank 中国光大银行
共享阳光 · 创新生活

目 录

数字转型
架构演进

研究背景

现存产品与解决方案

1. 原生Filter，性能较差，多数场景下无法实际使用
2. HBase + Solr(ES)，架构复杂度高，性能优势不明显，维护成本高
3. Phoenix，整体方案较重，社区活跃度低
4. HBase On Cloud，不适合安全性要求较高的场景

我们的需求

1. 非侵入性
2. 高性能
3. 通用性
4. 架构简单
5. 支持事务一致性

Pharos

产品命名

来自英文单词 pharos ， 世界上第一座灯塔

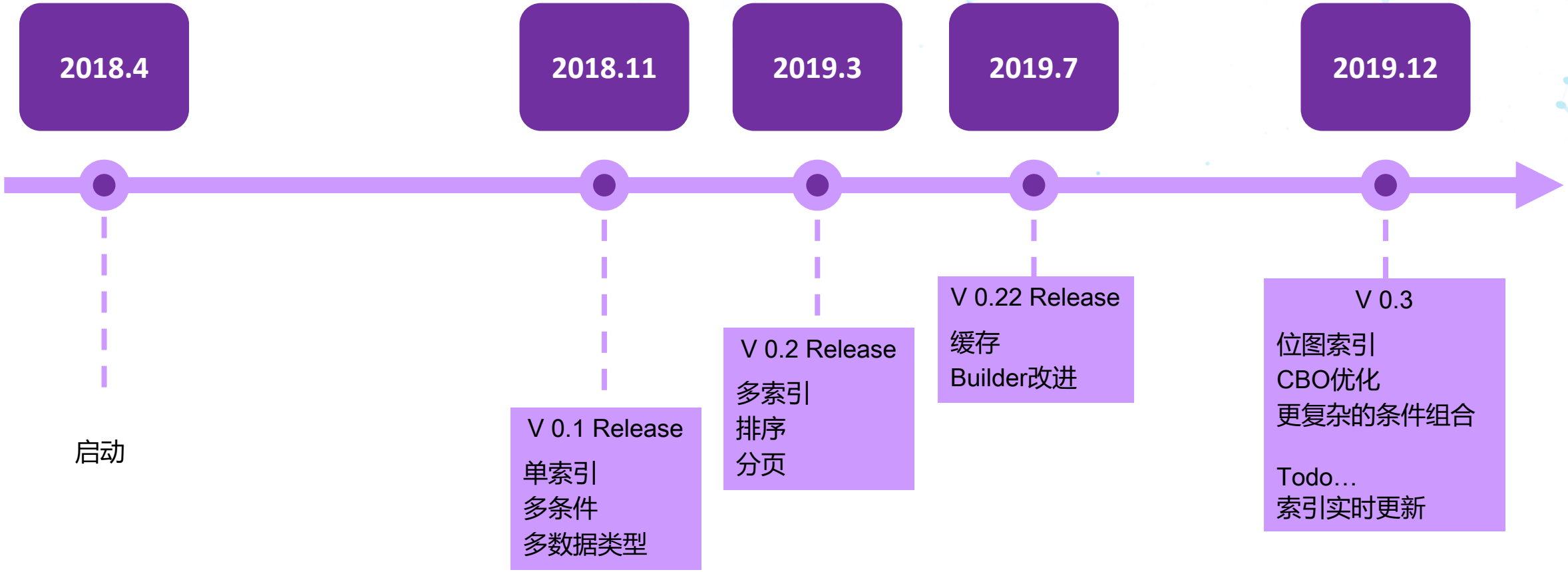
适用业务场景

- 只读场景，T+1数据加载
- 多读少写场景（实验版本）

设计原则

- 非侵入
- 架构简洁

研发过程


2018.4
2018.11
2019.3
2019.7
2019.12
启动
V 0.1 Release
单索引
多条件
多数据类型
V 0.2 Release
多索引
排序
分页
V 0.22 Release
缓存
Builder改进
V 0.3
位图索引
CBO优化
更复杂的条件组合
Todo…
索引实时更新

PharosV0.22 Features（HBase1.2.6/CDH5.8.3-HBase1.2.0）

现存产品与解决方案

1. 单索引（单列、多列复合），多索引
2. 分页，排序
3. 多条件查询，包括等于、大于、小于
4. 与或逻辑运算
5. 多种数据类型，包括Char、Data、Double等
6. 简单函数，例如count
7. 批量创建/更新索引

组件

Client

API

协调器

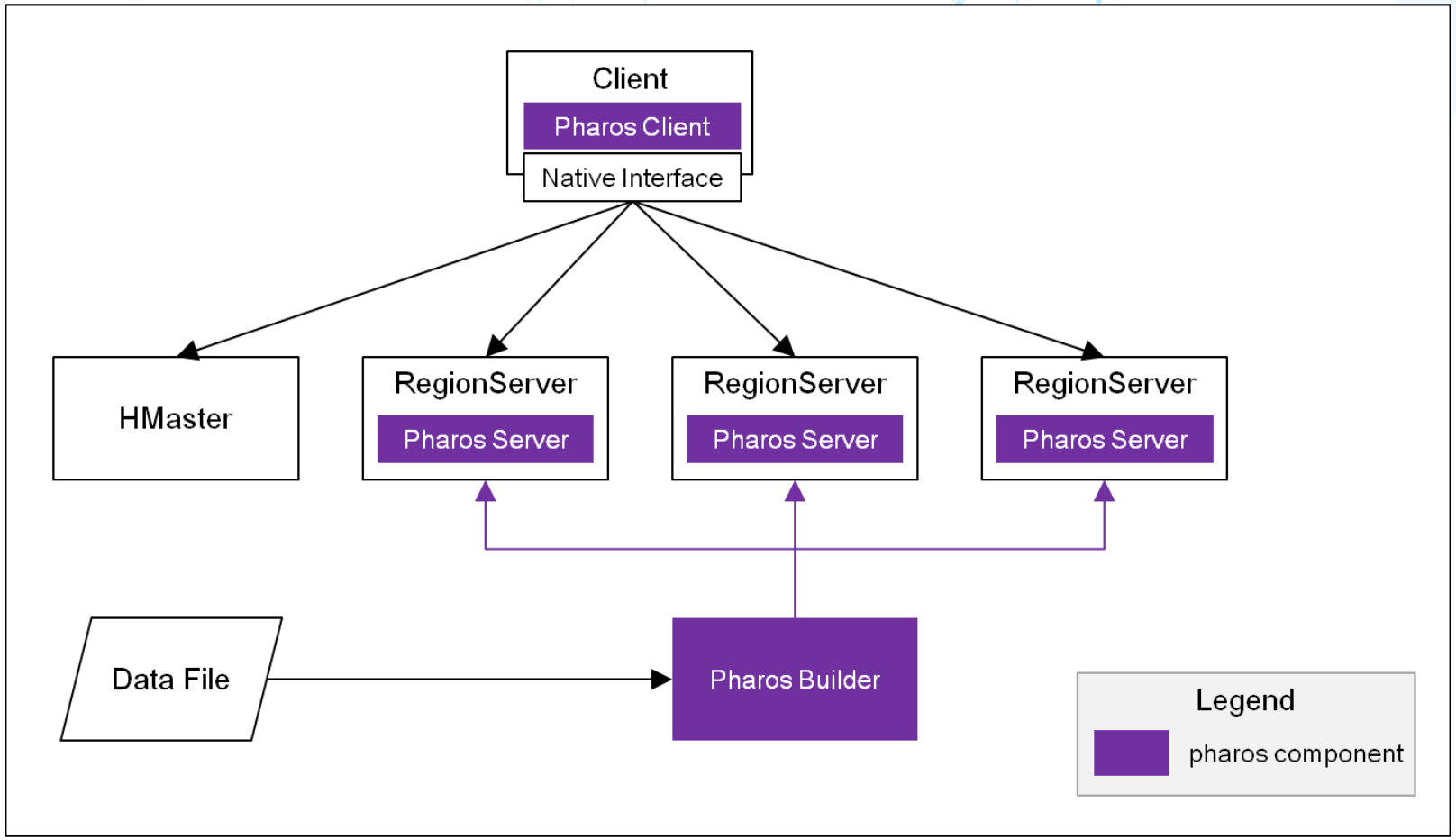
Server

协处理器Coprocessor

主要处理逻辑

Builder

索引创建/更新



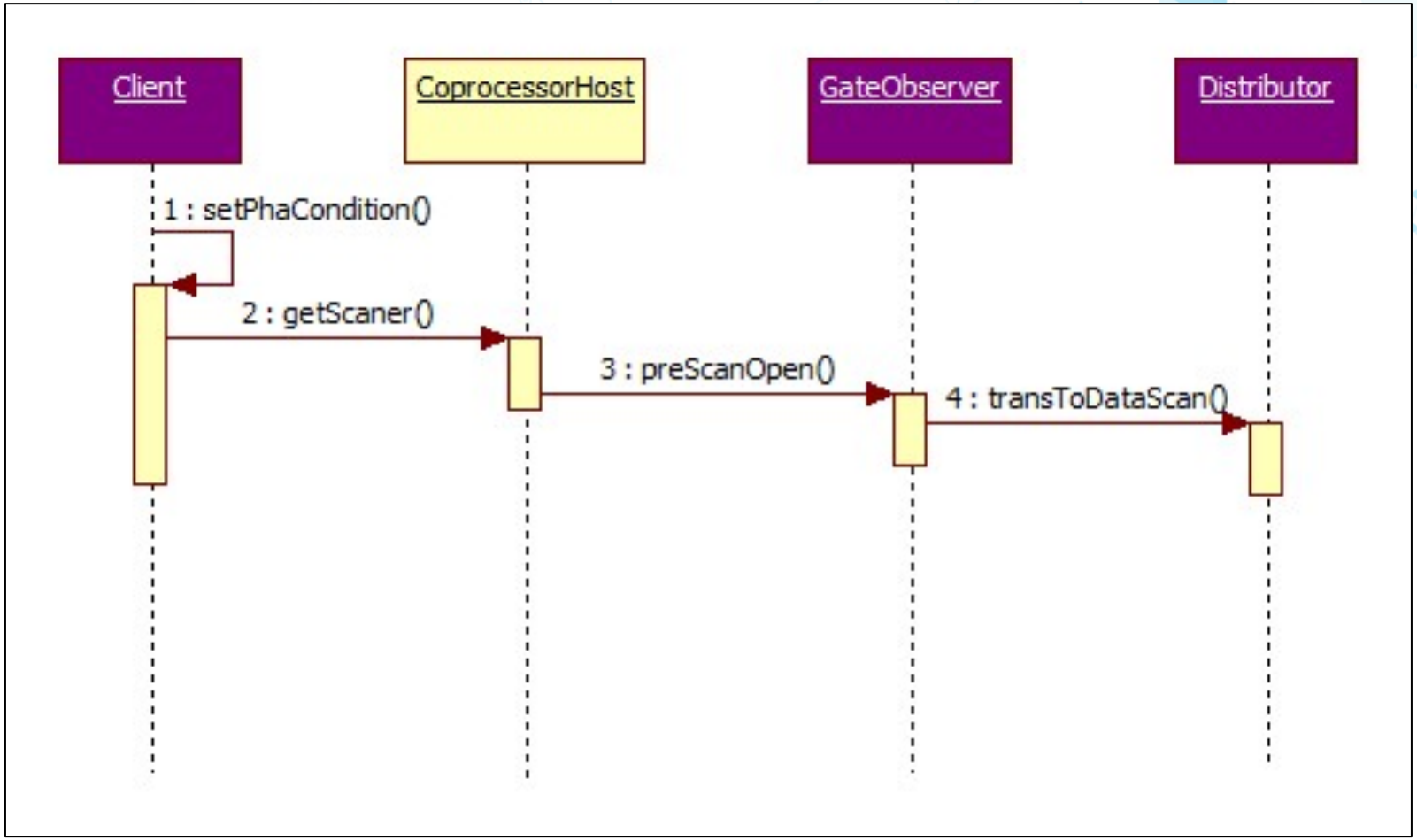
Coprocessor

Client

1. 定义查询条件
2. 置为Scan对象属性

Server

1. 拦截并解析scan
2. 扫描索引信息
3. 使用索引信息，重置Filter



Client Code Example

- 保持HBase原有代码风格
- 避免复杂的SQL解析

```
String colName = "JIOYRQ";
String col2Name = "CPZNXH";

ScanPha s = new ScanPha(); //create scan object

Condition cond = ConditionFactory.createCond(col2Name,ValueCompareConsts.EQUAL , "2");
Condition cond2 = ConditionFactory.createCond(colName,ValueCompareConsts.GREATER , "20180901");

cond.and(cond2);   //connect multiple conditions

s.setPhaCondition(cond);  //set condition
s.setPhaLimit(35);        //set page size
s.setSorted(colName);     //sorted by the column

HTablePhaProxy proxy = new HTablePhaProxy(table);
List<Result> list = proxy.getScanner(s);
```

目 录

全局索引 VS 分区索引

全局索引 Global Index

优势

支持唯一索引

缺陷

索引的创建与维护都会出发分布式事务，性能损耗较大

查询过程跨节点多次交互，带来性能损耗

分区索引 Partition Index

优势

索引与数据同分布，查询操作可以下推到每个节点，从而获得更好的性能

避免分布式事务

缺陷

不支持唯一索引或其他全局性控制

Pharos

存储策略

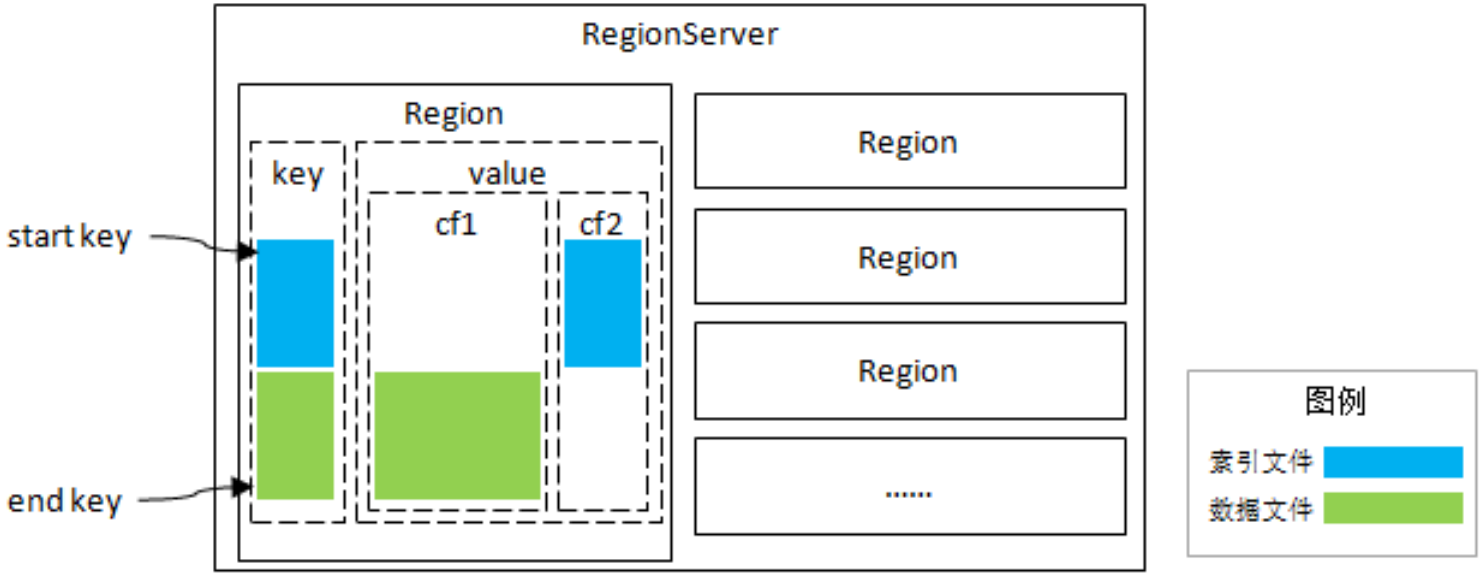
独立索引表存储

Region是HBase的最小调度单位，要保证数据与索引的同分布，必须干预balancer，侵入性较高

影子列族存储

通过控制索引rowkey的生成策略，保证数据与索引存储在同一Region不同列族(column family)中，侵入性较低

Pharos



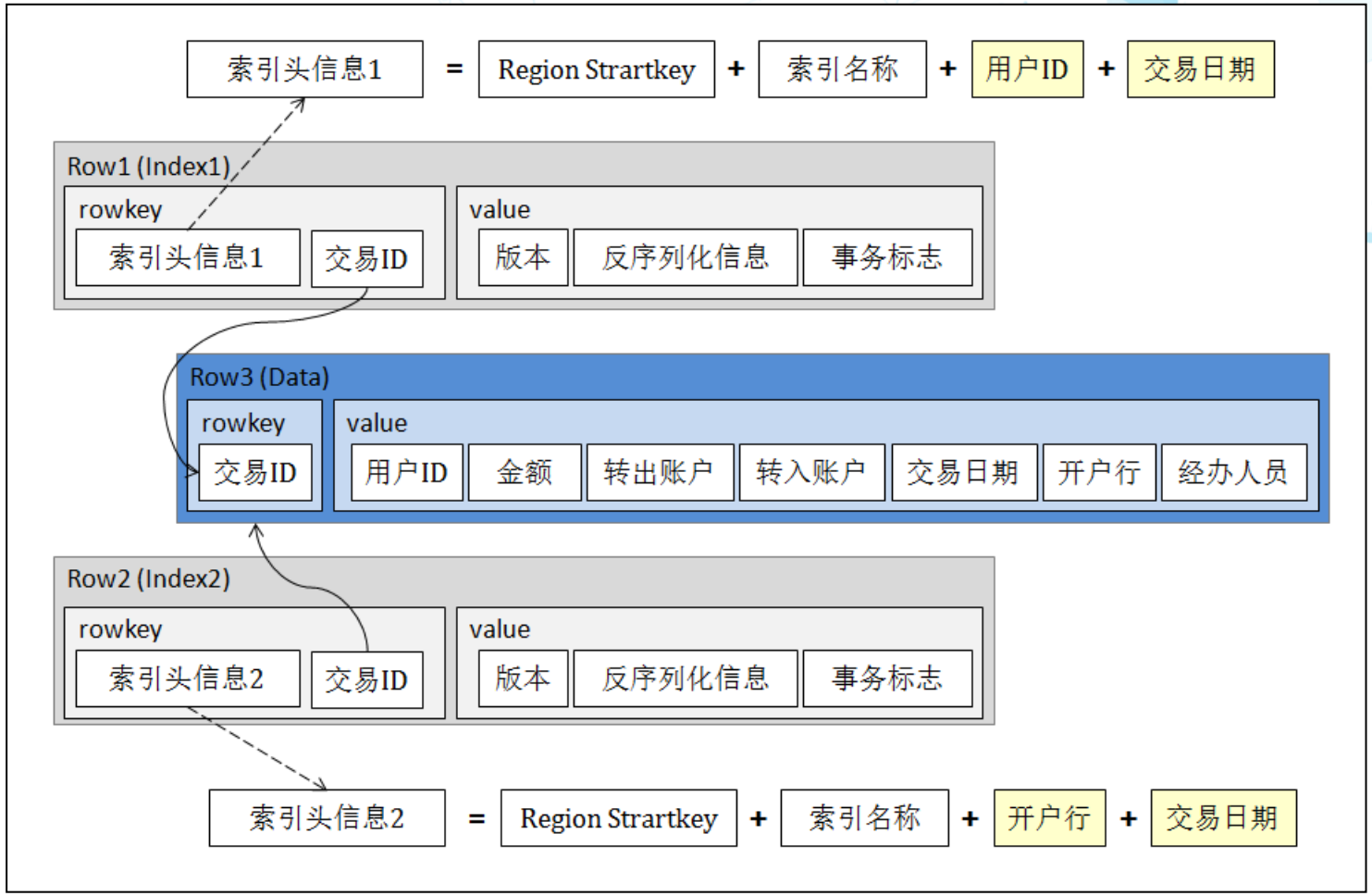
索引数据结构

Key 设计

1. 起始部分Region StartKey，保证索引与数据同分布
2. 索引名称/序号
3. 被索引列值
4. 数据行rowkey

Value设计

1. 版本号
2. 反序列化信息
3. 事务标志



排序机制

客户端作为协调器，以保持架构简单

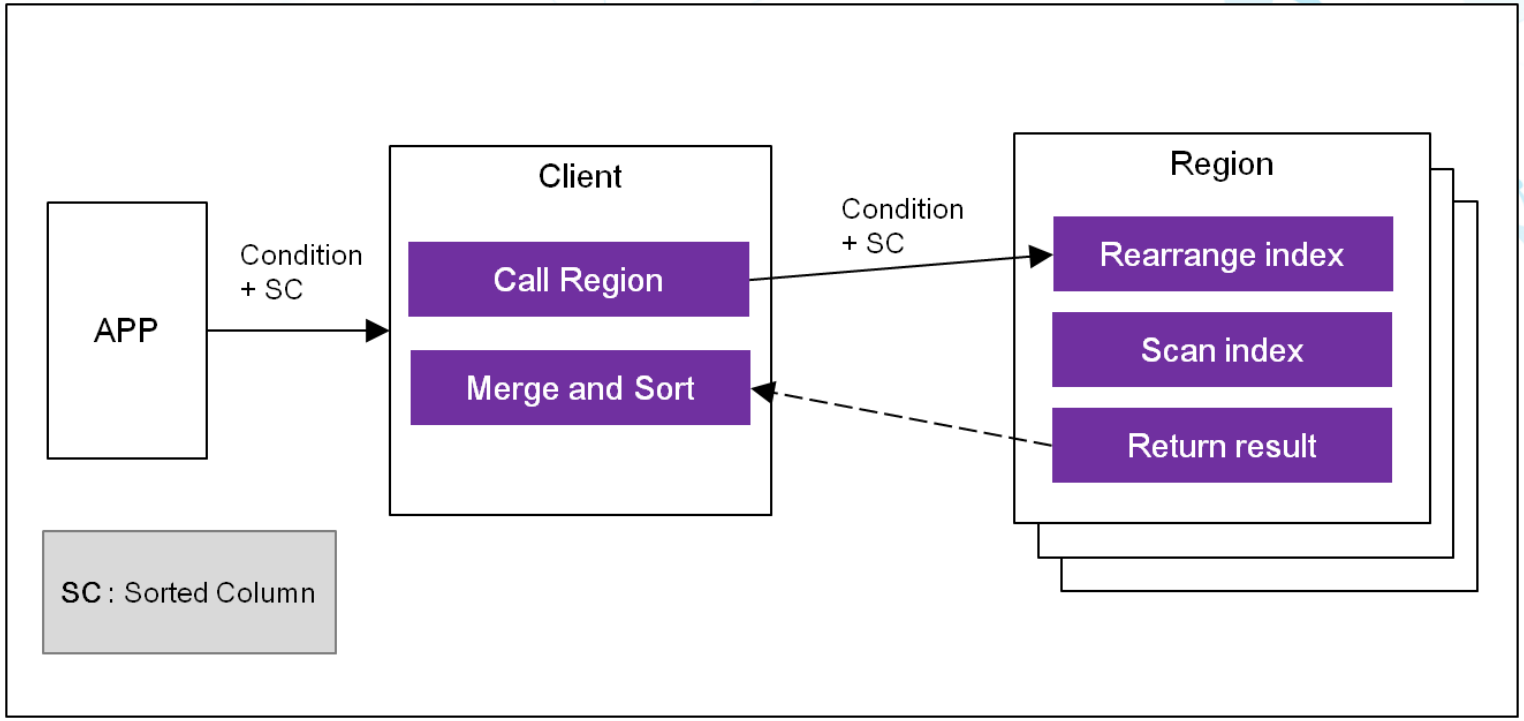
两阶段排序

1. Region 端排序

基于索引的天然有序性

2. Client端排序

合并多个Region结果，如果数据倾斜严重，则要再次查询



分页机制

必要性

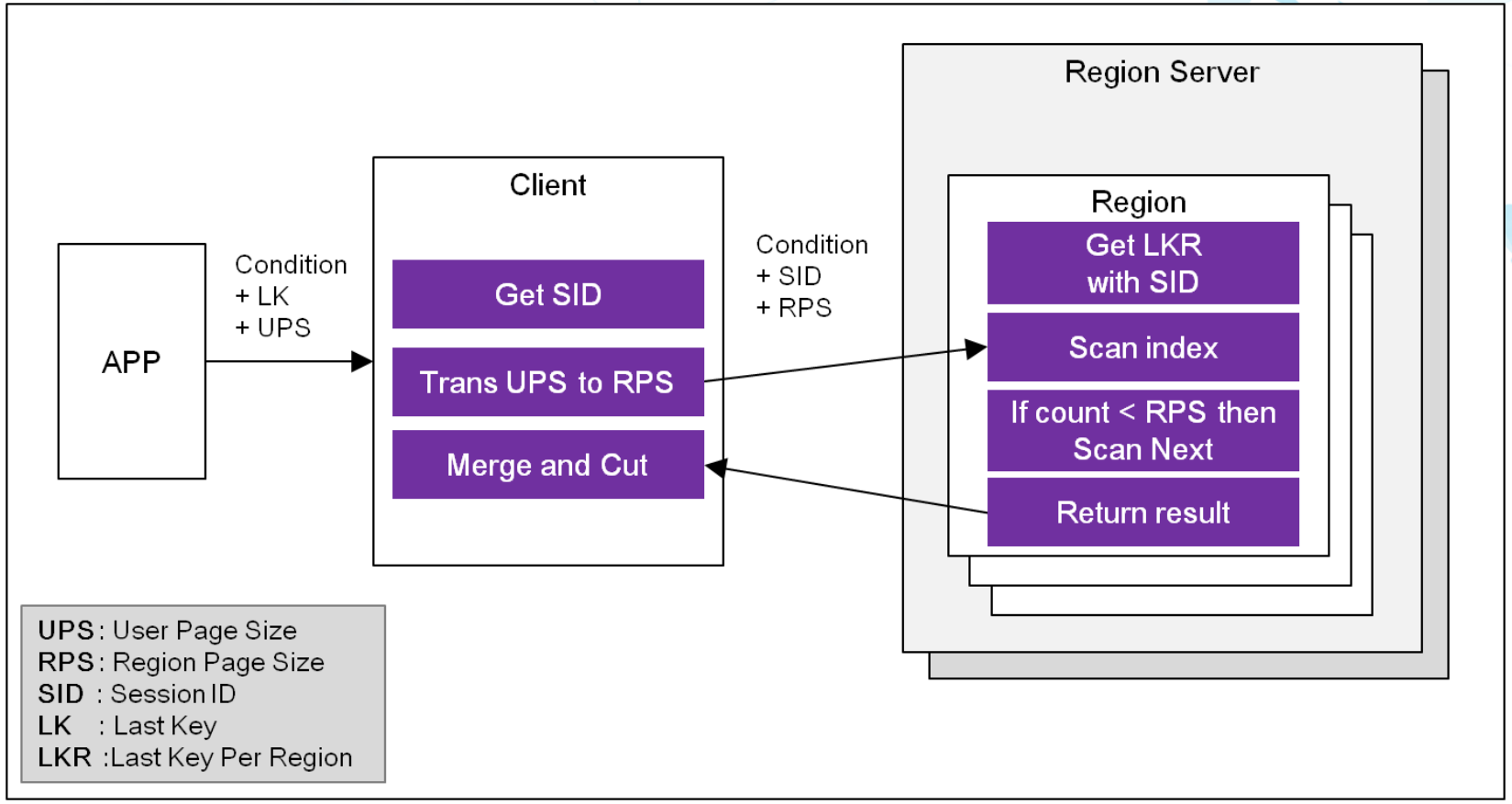
海量数据下，匹配索引无法全部加入内存

策略

增加全局session，映射多个Region的断点；

在Region端缓存断点；

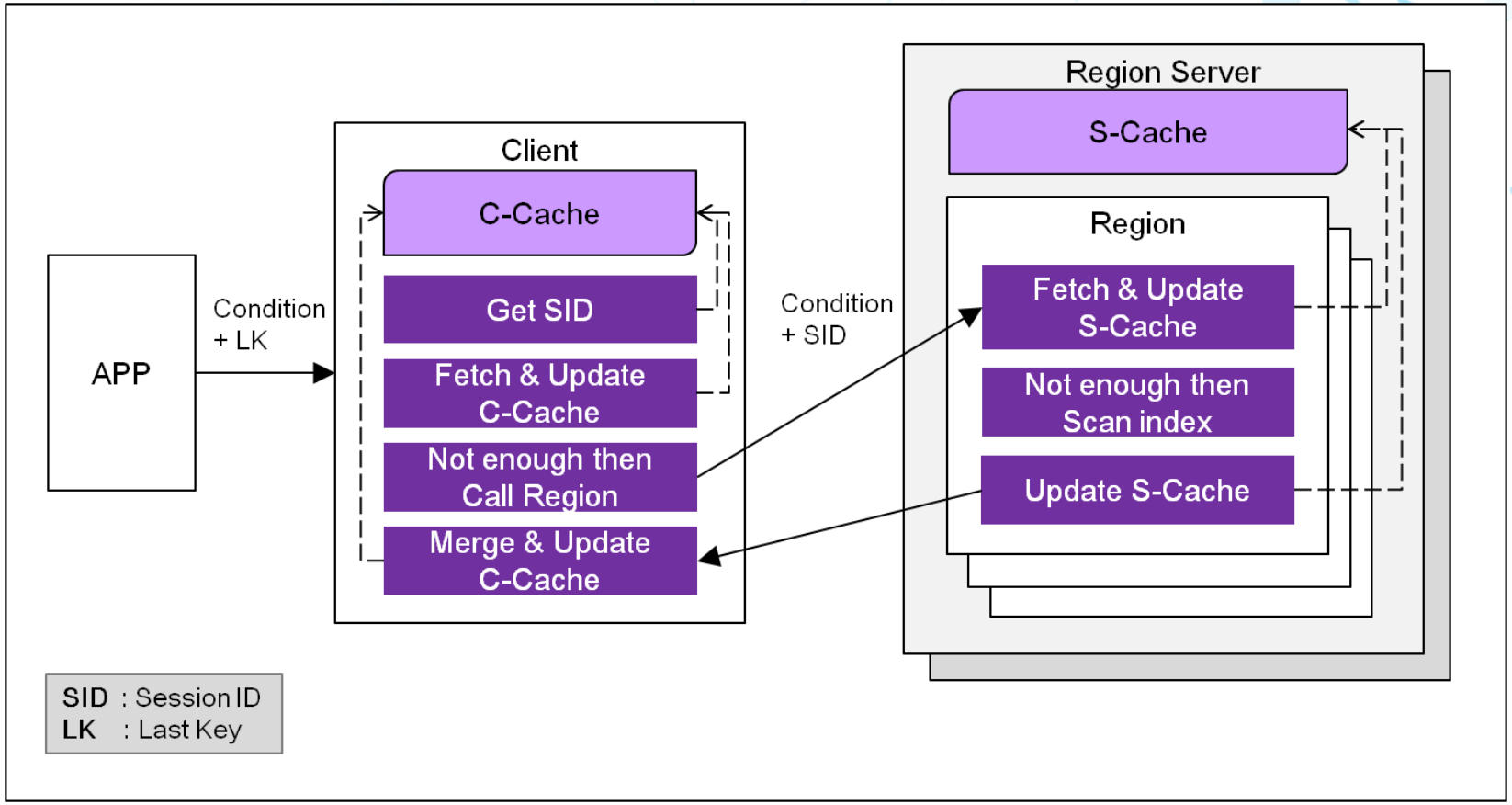
Client合并查询结果，如果不满足分页大小则再次发起查询。



缓存机制

使用本地缓存从而保持架构简洁

Client端缓存 + Server端缓存

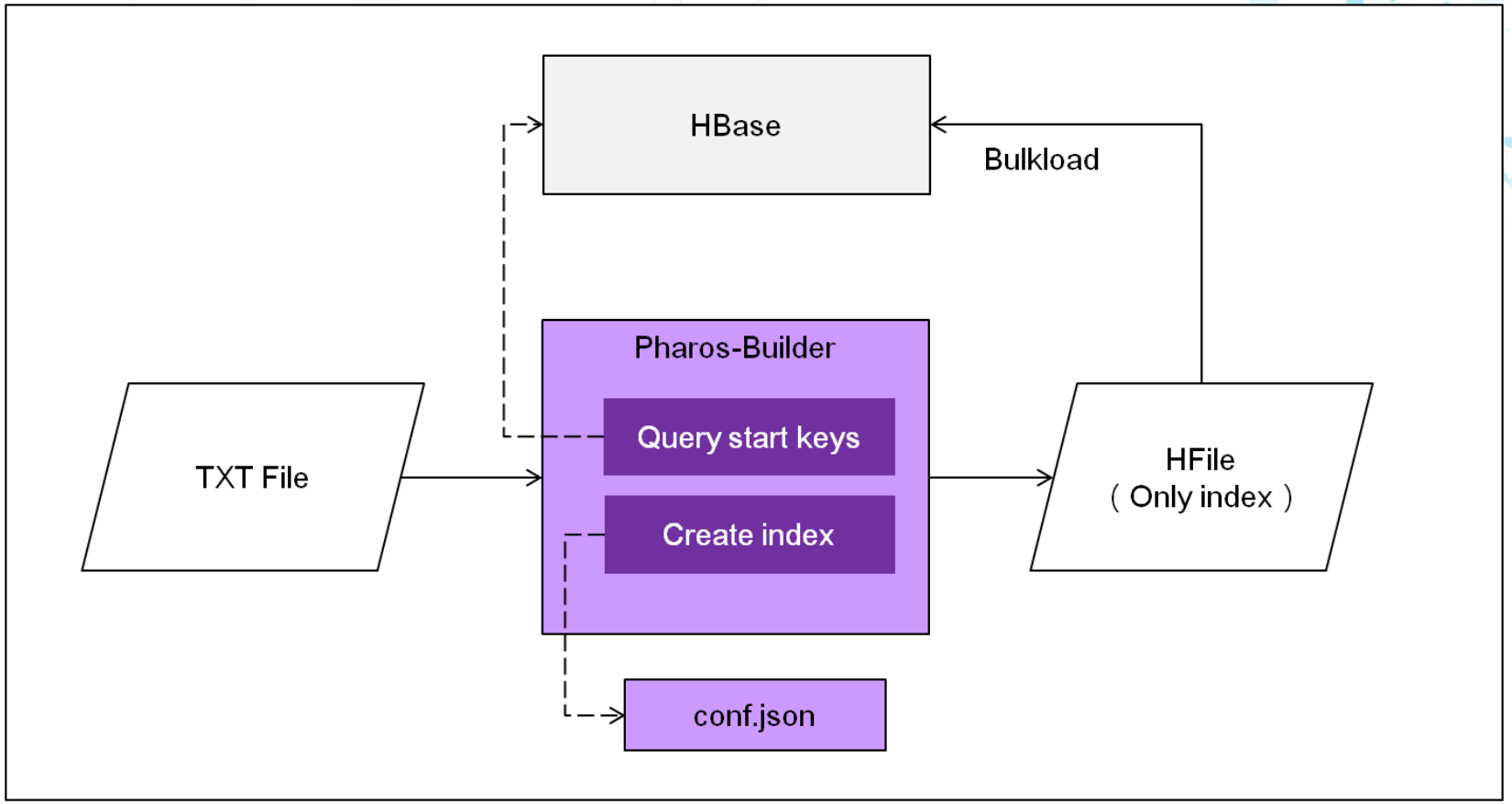


索引创建

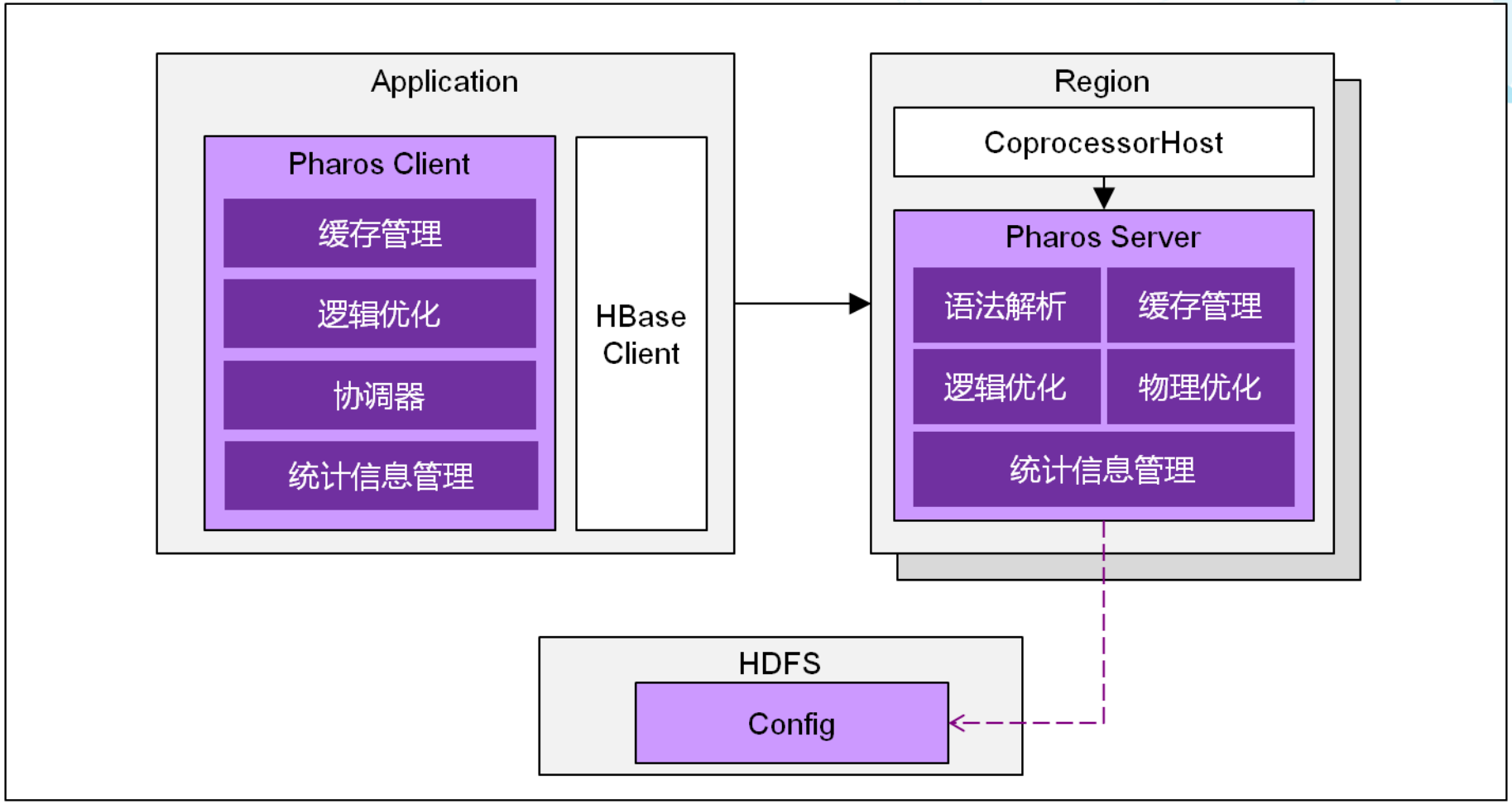
避免Region Split破坏数据索引同分布

由于应用中不同的rowkey设计策略，Bulkload也可能导致Split发生

因此，选择在数据加载完成后，Region稳定后，再加载索引。



整体功能架构


Application
Pharos Client
缓存管理
逻辑优化
协调器
统计信息管理
HBase Client
Region
CoprocessorHost
Pharos Server
语法解析
缓存管理
逻辑优化
物理优化
统计信息管理
HDFS
Config

目 录

事务一致性原理

灵感来自 Google Percolator

两阶段事务的变形

Write阶段仅完成主信息的更新；读阶段（或异步进程）完成从信息的更新。

示例： Bob -> Joe $7

key	bal:data	bal:lock	bal:write
Bob	*6:* *5:* $10	*6:* *5:*	*6:* data @ 5 *5:*
Joe	*6:* *5:* $2	*6:* *5:*	*6:* data @ 5 *5:*

1. Initial state: Joe's account contains $2 dollars, Bob's $10.

Bob	*7:*$3 *6:* *5:* $10	*7:* **I am primary** *6:* *5:*	*7:* *6:* data @ 5 *5:*
Joe	*6:* *5:* $2	*6:* *5:*	*6:* data @ 5 *5:*

2. The transfer transaction begins by locking Bob's account balance by writing the lock column. This lock is the primary for the transaction. The transaction also writes data at its start timestamp, 7.

Bob	*7:* $3 *6:* *5:* $10	*7:* I am primary *6:* *5:*	*7:* *6:* data @ 5 *5:*
Joe	***7:* $9** *6:* *5:* $2	***7:* primary @ Bob.bal** *6:* *5:*	*7:* *6:* data @ 5 *5:*

3. The transaction now locks Joe's account and writes Joe's new balance (again, at the start timestamp). The lock is a secondary for the transaction and contains a reference to the primary lock (stored in row "Bob," column "bal"); in case this lock is stranded due to a crash, a transaction that wishes to clean up the lock needs the location of the primary to synchronize the cleanup.

Bob	*8:* *7:* $3 *6:* *5:* $10	*8:* ***7:*** *6:* *5:*	*8:* **data @ 7** *7:* *6:* data @ 5 *5:*
Joe	*7:* $9 *6:* *5:* $2	*7:* primary @ Bob.bal *6:* *5:*	*7:* *6:*data @ 5 *5:*

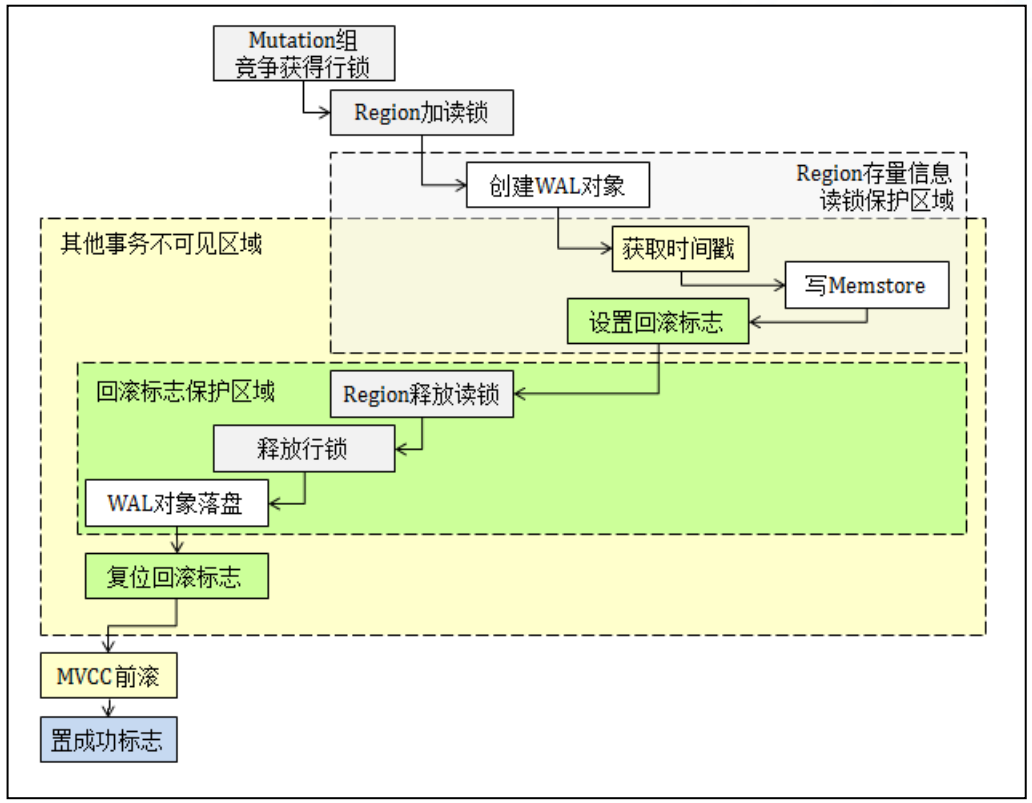
4. The transaction has now reached the commit point: it erases the primary lock and replaces it with a write record at a new timestamp (called the commit timestamp): 8. The write record contains a pointer to the timestamp where the data is stored. Future readers of the column "bal" in row "Bob" will now see the value $3.

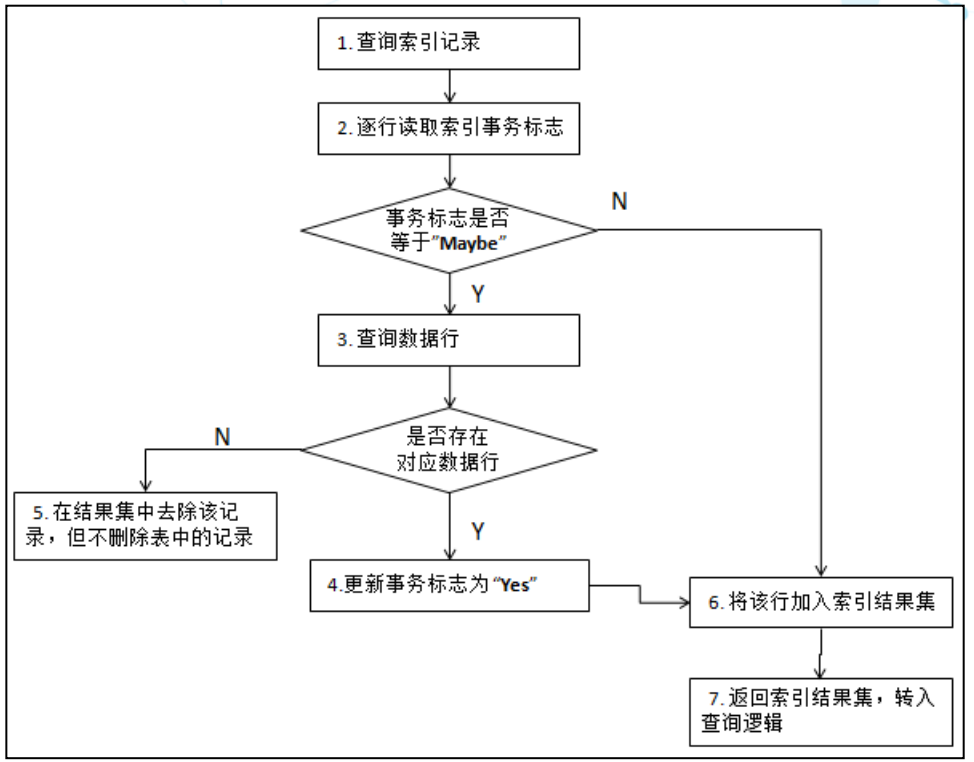
Step 4 -> Transaction complete

Bob	*8:* *7:* $3 *6:* *5:* $10	*8:* *7:* *6:* *5:*	*8:* data @ 7 *7:* *6:* data @ 5 *5:*
Joe	*8:* *7:* $9 *6:* *5:*$2	*8:* ***7:*** *6:* *5:*	***8:* data @ 7** *7:* *6:* data @ 5 *5:*

5. The transaction completes by adding write records and deleting locks at the secondary cells. In this case, there is only one secondary: Joe.

索引实时更新机制


Mutation组
竞争获得行锁
Region加读锁
创建WAL对象
Region存量信息
读锁保护区域
其他事务不可见区域
获取时间戳
写Memstore
设置回滚标志
回滚标志保护区域
Region释放读锁
释放行锁
WAL对象落盘
复位回滚标志
MVCC前滚
置成功标志


1. 查询索引记录
2. 逐行读取索引事务标志
事务标志是否
等于"Maybe"
N
Y
3. 查询数据行
是否存在
对应数据行
N
Y
5. 在结果集中去除该记录，但不删除表中的记录
4.更新事务标志为 "Yes"
6. 将该行加入索引结果集
7. 返回索引结果集，转入查询逻辑

其他特性

位图索引

CBO 优化

与SQL 引擎的集成（Presto ？）

Thanks

公众号：金融数士