

分布式NewSQL数据库Hyperbase

范颖捷 | 2018年7月

目录 > CONTENTS

- 1 Hyperbase简介
- 2 Hyperbase原理
- 3 Hyperbase安装与配置
- 4 Hyperbase基本用法
- 5 Hyperbase全局索引
- 6 Hyperbase数据快速入库



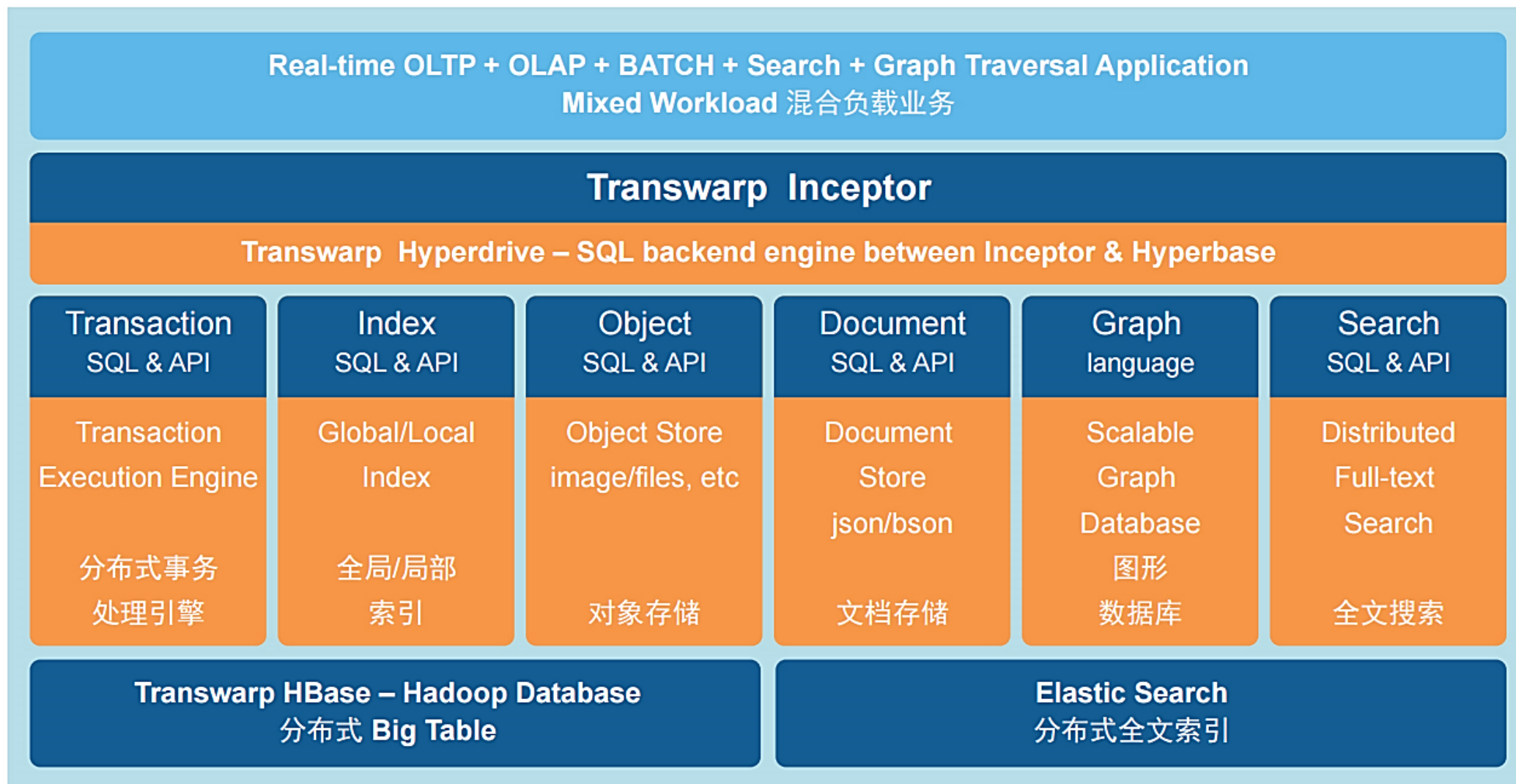
1 chapter

Hyperbase简介

- ✓ 什么是Hyperbase
- ✓ 适用场景
- ✓ 访问方式

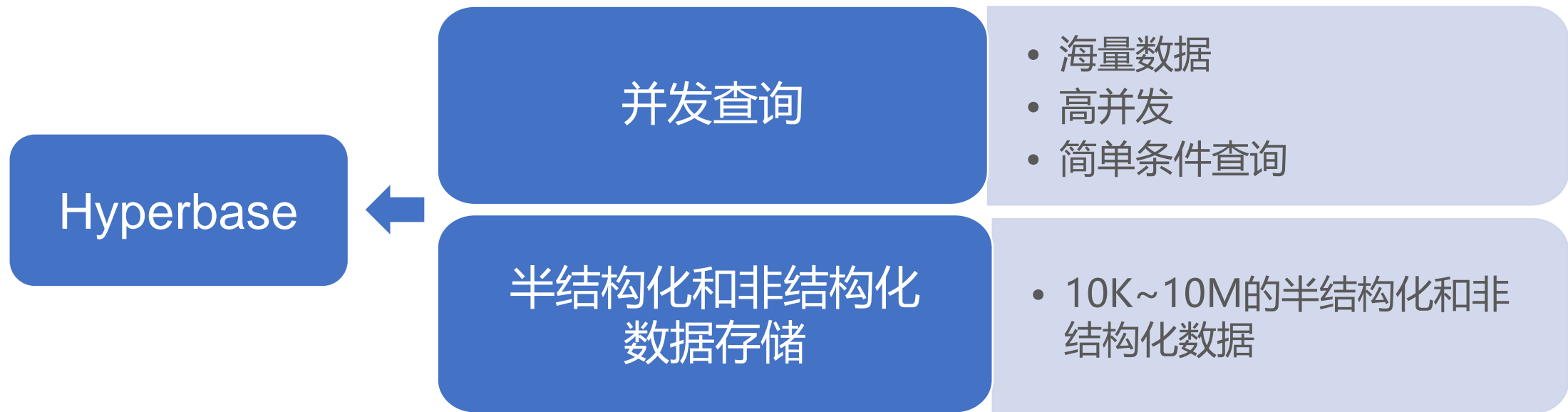
➤ 概念

- 高可靠、高性能、可伸缩、实时读写、面向列的分布式NewSQL数据库
- 基于HBase实现
 - HBase是Hadoop Database, Google Bigtable的开源实现
- NewSQL数据库
 - 既具有NoSQL数据库的海量数据存储管理能力, 又继承了关系数据库的SQL特性
- 列式存储
 - 按列族存储, 对Schema限制很少, 可以自由添加列
 - 适合存储海量的半结构化数据
- Key-Value数据库
 - 按Key的字典序顺序存储
 - 主要通过Key实现数据的增删改查, 以及扫库操作
- 采用HDFS为文件存储系统




➤ 特点

- 海量数据存储
 - 主要存储半结构化、非结构化的稀疏数据
 - PB以上数据规模
- 线性扩展
 - 容量和处理能力随节点数量线性增长
- 高并发
- 高可用
- 数据实时随机读写
- 数据强一致性







2 chapter

Hyperbase原理

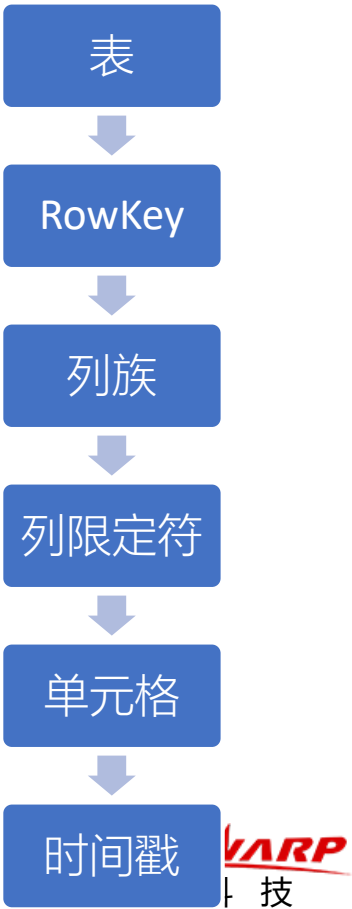
- ✓ 数据模型
- ✓ 系统架构
- ✓ 数据存储
- ✓ 读写过程

➤ 示例：表结构

• 四维表：RowKey | 列族 | 列限定符 | 时间戳； 二维表：RowKey | 列

RowKey	BasicInfo			CourseInfo		timestamp
	name	age	telephone	course	score	
rk001	zhangsan					t1
		18				t4
			1380100001			t9
			1590939995			t5
				math		t2
					90	t3
rk002	lisi					t6
		20				t7
			1391129396			t8
				english		t10
					85	t11

数据读取流程



➤ 命名空间 (Namespace)

- 命名空间是对表的逻辑分组，类似于关系数据库中的Database
- 利用命名空间，在多租户场景下可做到更好的资源和数据隔离

➤ 表 (Table)

- Hyperbase以“表”为单位组织数据
- 表由多行组成

➤ 行 (Row)

- 表以“行”为单位组织数据
- 行由一个RowKey和多个列族组成
- RowKey
 - 行的主键，用于唯一地标识和定位行
 - 各行按RowKey的字典序排列
 - 大小64K

➤ 列族 (Column Family)

- 表中数据按“列族”分组，每一行由若干列族组成，每个列族下可包含多个列
- 列族必须在建表时必须明确定义，如：create ‘student’, ‘basicinfo’, ‘courseinfo’
- 每一行的列族都相同，但列族不一定要包含数据
- 物理上，同一列族的数据存储在一起
- 权限控制、存储以及调优都在列族层面进行

➤ 列与列限定符 (Column & Column Qualifier)

- 一个列族可包含多个列，且各列族可包含不同的列
- 列由列族和列限定符唯一指定，列名由列族名和列限定符组成，以“:”分隔
 - 例如：basicinfo列族包含basicinfo:name列、basicinfo:age列和basicinfo:telephone列，其中name、age和telephone是列限定符
- 新列可按需动态加入列族

➤ 单元格 (Cell)

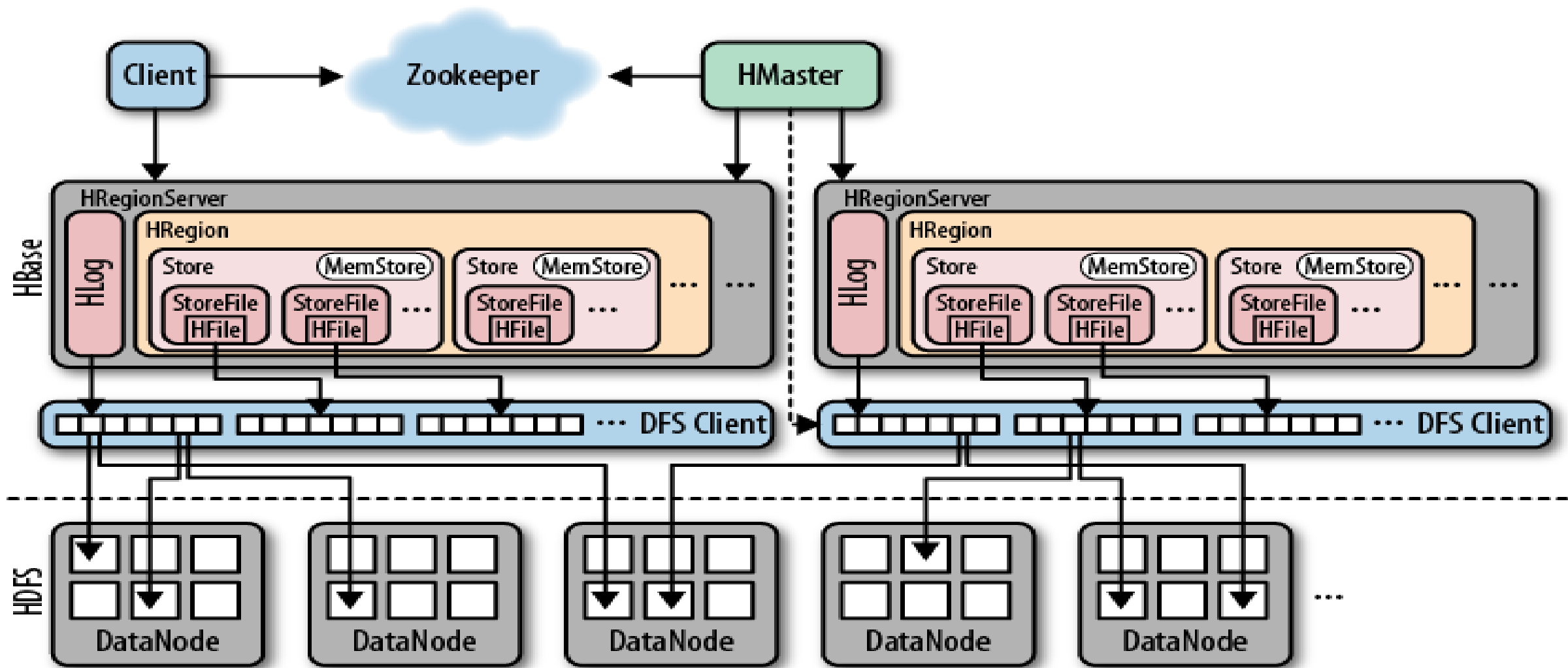
- 单元格由RowKey、列族、列限定符唯一定位
- 单元格中存放若干<value, version>
- 单元格的值是有版本的，版本号为时间戳
- 单元格的值由{rowkey, column(=<column family>+<column qualifier>), version}来唯一定位
- 单元格的数据类型为字节数组byte[]

➤ 时间戳 (Timestamp)

- 单元格的值利用时间戳来标识版本，时间戳具有唯一性
- 单元格内不同版本的值按时间倒序排列，最新的数据排在最前面
- 时间戳可以在数据写入时由系统自动赋值（精确到毫秒的当前系统时间），也可以由客户显式赋值，为了避免版本冲突，必须生成具有唯一性的时间戳
- 类型是 64位整型

➤ 表的特点

- 数据规模大
 - 单表可容纳数十亿行，上百万列
- 无模式
 - 每行可以有任意多的列，列可以动态增加，不同行可以有不同的列，列的类型没有限制
- 面向列族
 - 面向列族的存储和权限控制，支持列族独立查询
- 稀疏
 - 值为空的列不占存储空间，表可以非常稀疏
- 数据多版本
 - 单元格的值可以有多个版本，利用时间戳来标识版本
- 数据无类型
 - 所有数据以字节数组形式存储



➤ HMaster (Master)

- 管理元数据
- 管理表的创建、删除和修改
- 为HRegionServer分配Region
 - HRegionServer宕机后，重新分配其上的Region
- 负责HRegionServer的负载均衡
- 不处理Client的数据读写请求

➤ HRegionServer (Slave)

- 系统运行过程中动态添加、删除HRegionServer
- 处理Client的数据读写请求
- 管理Region Split（分裂）
- 管理StoreFile Compaction（合并）

➤ Zookeeper

- 实现HMaster高可用
- 监控HRegionServer的上下线信息，并通知HMaster
- 存储元数据的寻址入口
- 存储所有Region的寻址入口

➤ Client

- 通过接口访问Hyperbase
- 为了加快数据访问速度，将元数据、Region位置等信息缓存在Client Cache中

➤ Region

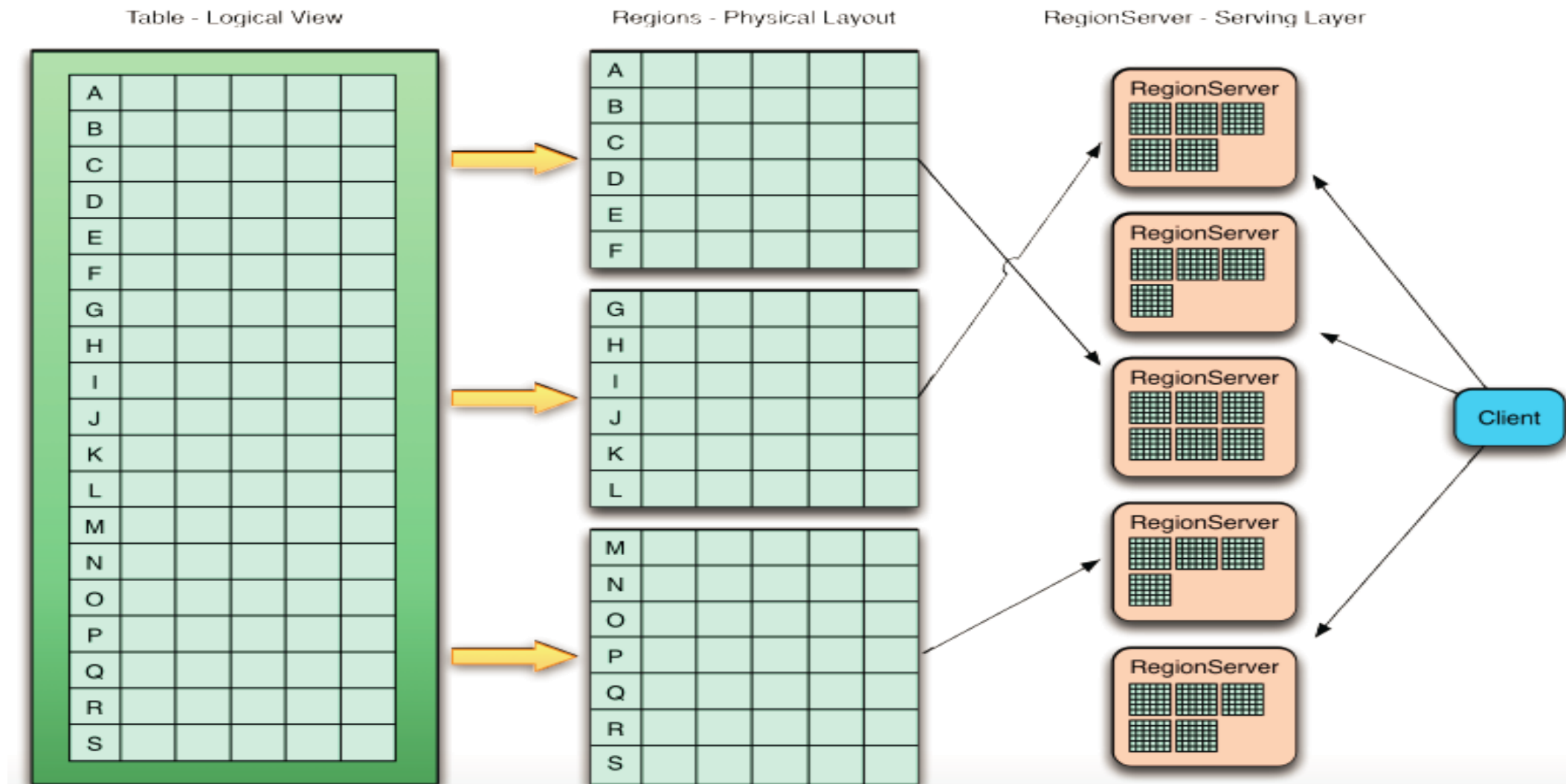
- 分布式存储和负载的最小单元
- 系统将表水平划分（按行）为多个Region，每个Region保存表的一段连续数据
- 默认每张表开始只有一个Region，随着数据不断写入，Region不断增大，当Region大小超过阈值时，当前Region会分裂成两个子Region
- 随着Region的不断增多，HMaster会将部分Region迁移到其他HRegionServer中，实现负载均衡
- 表通常被保存在多个HRegionServer的多个Region中

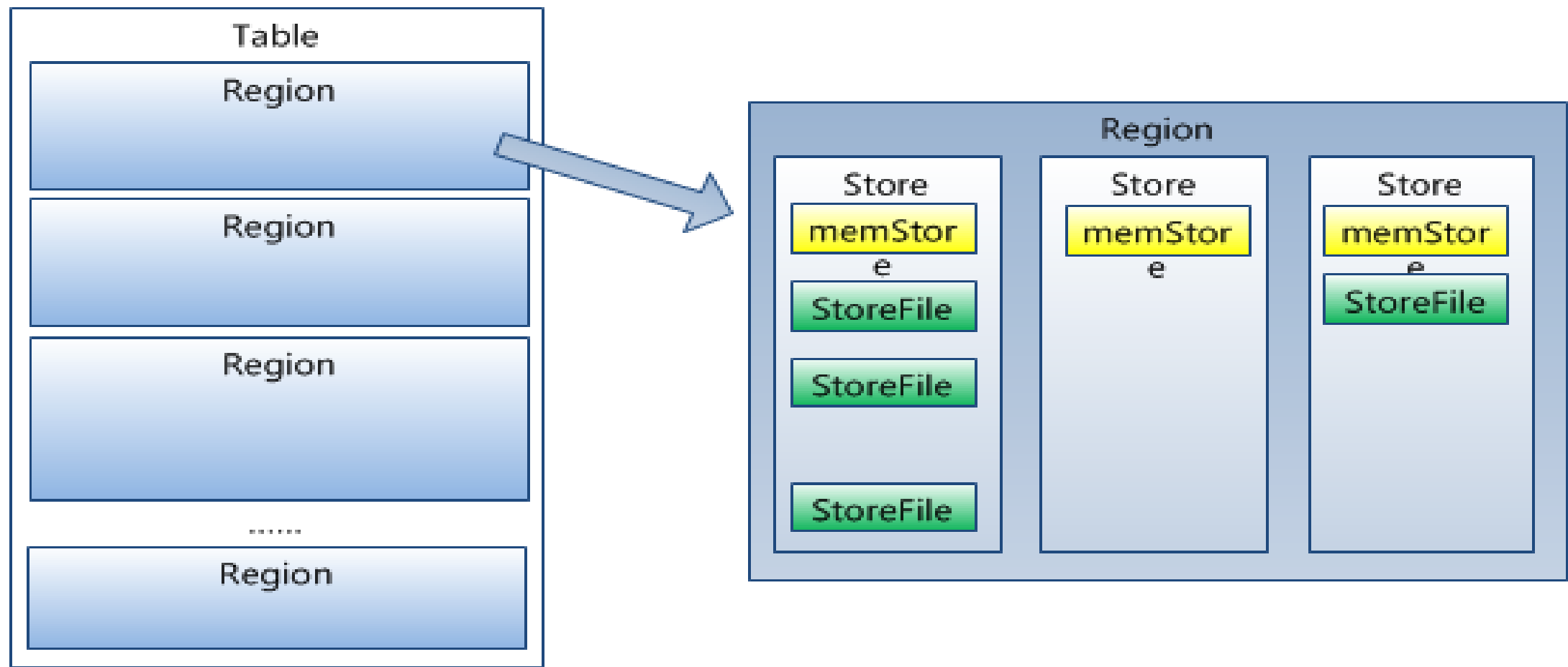
➤ Store

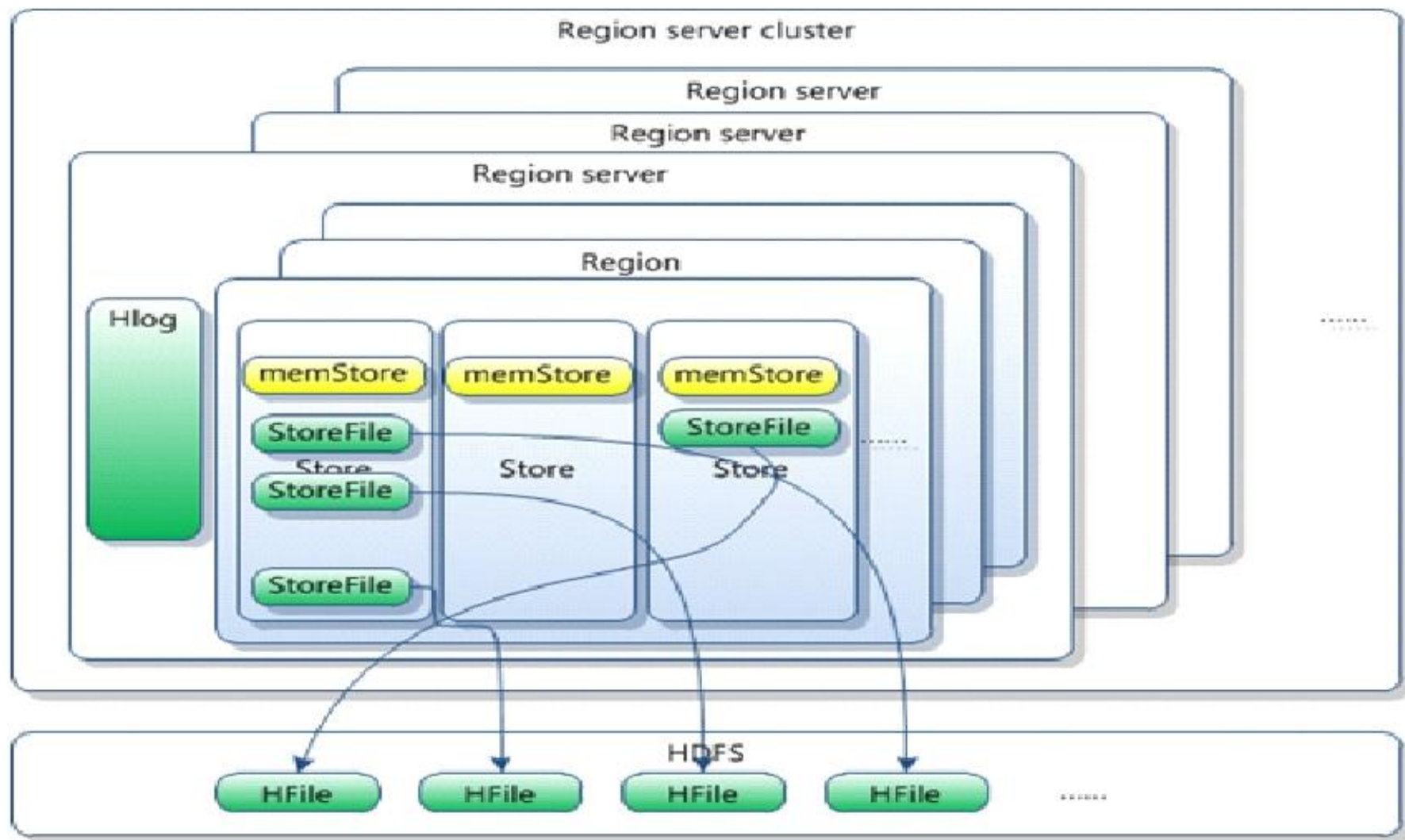
- 一个Region由多个Store组成，每个Store存储一个列族
- Store由内存中的MemStore和磁盘中的若干StoreFile组成
- Region是分布式存储的最小单元，而Store是存储落盘的最小单元

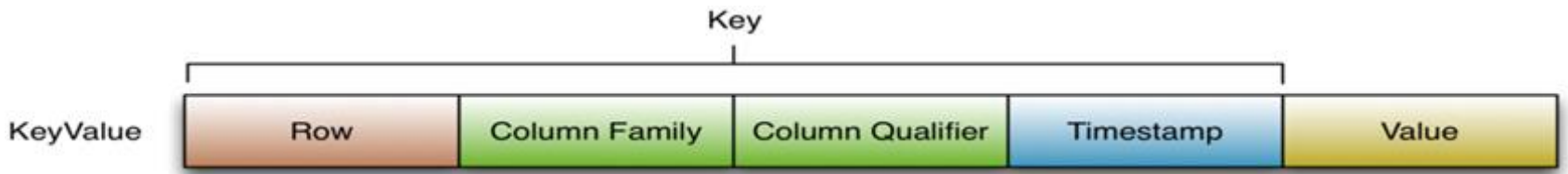
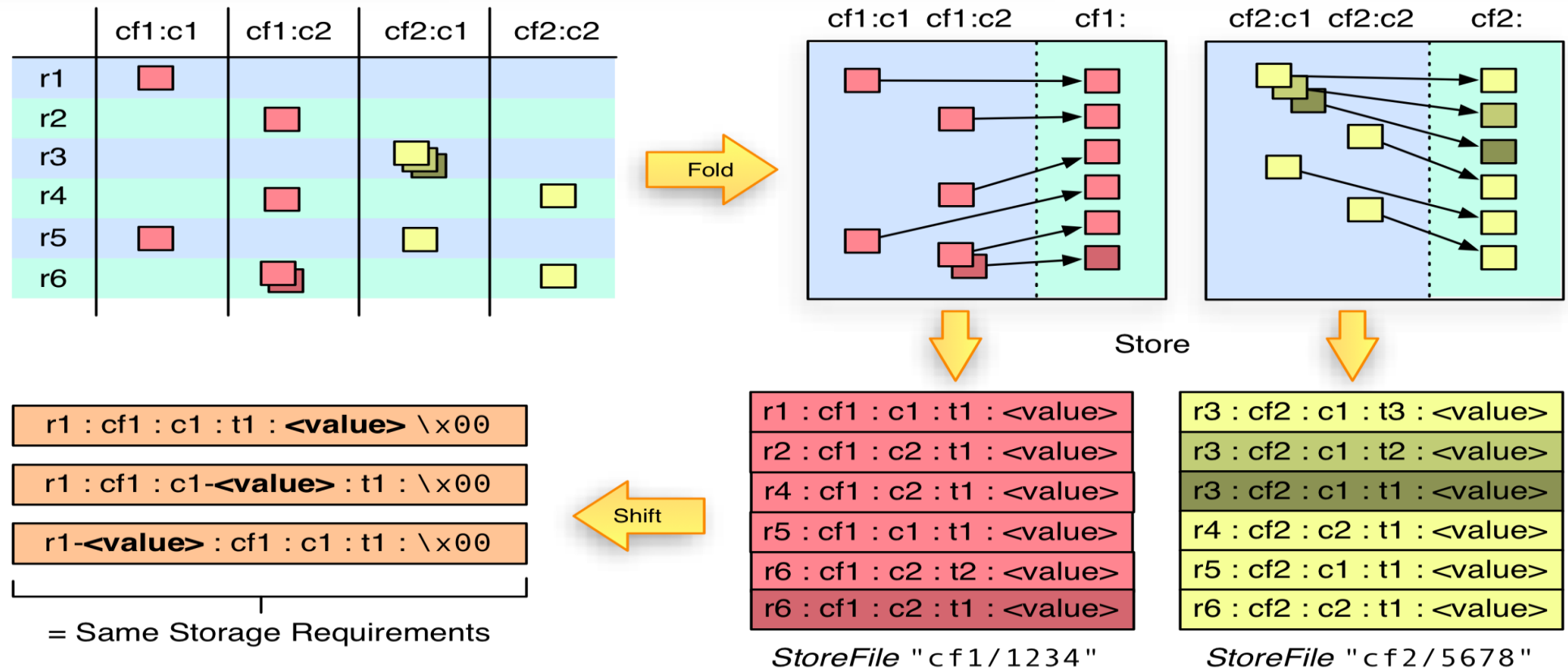
➤ MemStore与StoreFile

- MemStore是Store的内存缓冲区，数据读写都先访问MemStore
- StoreFile是MemStore的磁盘溢写文件，在HDFS中被称为HFile
- 写数据时，先写MemStore，当数据量超过阈值时，HRegionServer会将MemStore中的数据溢写磁盘，每次溢写都生成一个独立的StoreFile（HFile）
- 当Store中的StoreFile数量超过阈值时，HRegionServer会将若干小StoreFile合并为一个大的StoreFile
- 当Region中最大Store的大小超过阈值时，HRegionServer会将其等分为两个子Region
- Client读取数据时，先找MemStore，再找StoreFile









RowKey升序
TimeStamp降序

➤ 写过程

- (1) Client访问ZK，获取meta表所在Region的位置信息，并将该信息写入Client Cache
- (2) Client读取meta表，根据Namespace、表名和RowKey，获取将要写入Region的位置信息，并将meta表写入Client Cache
- (3) Client向HRegionServer发出写请求，HRegionServer先将操作和数据写入HLog（预写日志，Write Ahead Log，WAL），再将数据写入MemStore，并保持有序
- (4) 当MemStore的数据量超过阈值时，将数据溢写磁盘，生成一个StoreFile文件
- (5) 当StoreFile的数量超过阈值时，将若干小StoreFile合并（Compact）为一个大StoreFile
- (6) 当Region中最大Store的大小超过阈值时，Region分裂（Split），等分成两个子Region

➤ 读过程

- (1) 获取将要读取Region的位置信息（见写过程的第1、2步）
- (2) Client向HRegionServer发出读请求
- (3) HRegionServer先从MemStore读取数据，如未找到，再从StoreFile中读取

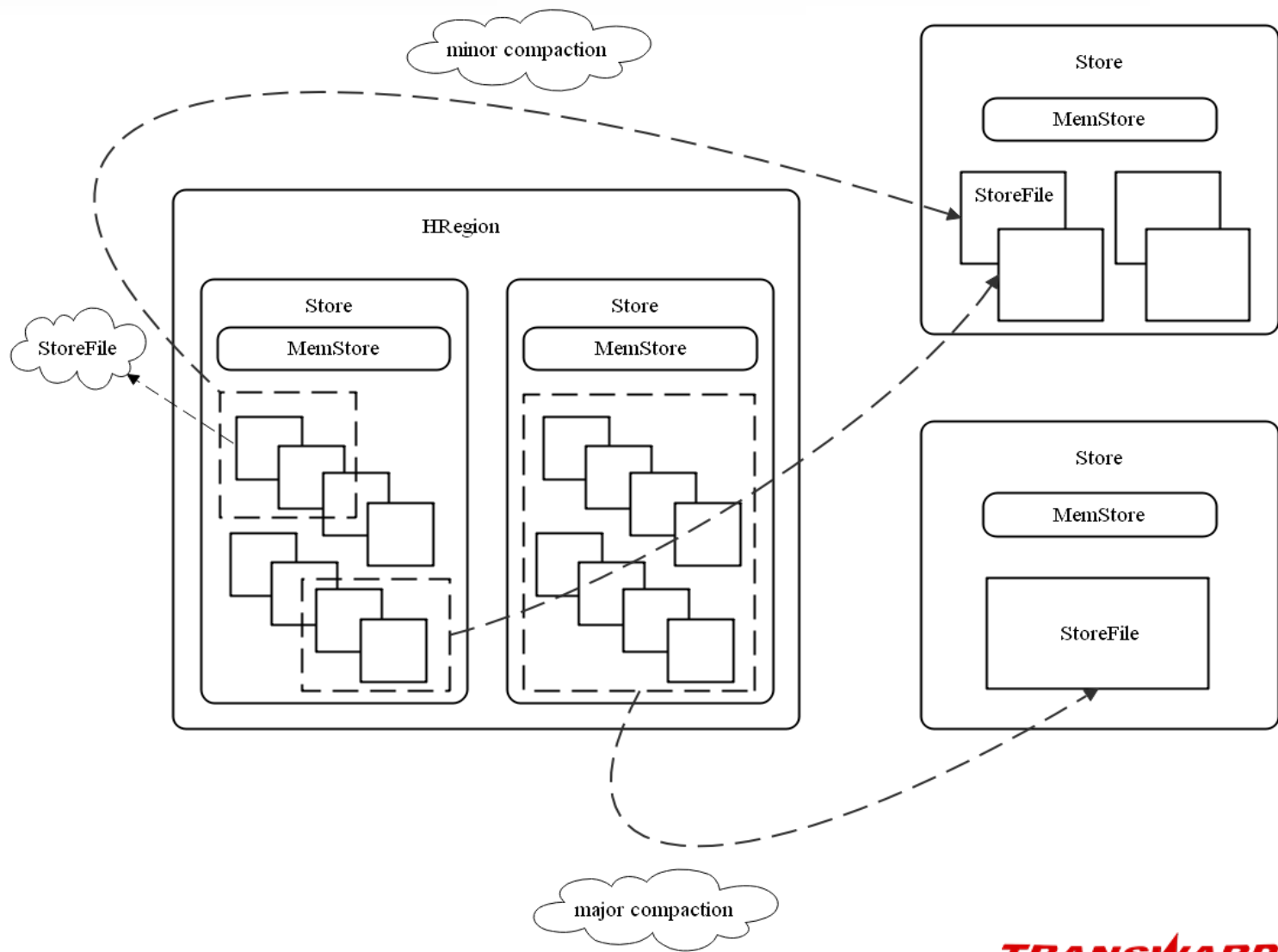
➤ StoreFile Compaction

- 含义：将Store中的全部或部分StoreFile合并为一个StoreFile的过程
- 目的：减少StoreFile数量，提升数据读取效率
- 触发
 - 时机：①Memstore溢写；②后台线程周期性检查；③手动，且触发后不能停止
 - 条件：当Store中的StoreFile数量超过阈值时，触发StoreFile Compaction
- Minor Compaction
 - 选取Store下的部分StoreFile（小的或相邻的），将它们合并为一个StoreFile
 - 不删除过期版本、delete marker数据
 - 速度较快，IO和带宽开销相对较低
- Major Compaction
 - 将Store下的所有StoreFile合并为一个StoreFile
 - 删除过期版本、delete marker数据

➤ StoreFile Compaction

• Major Compaction

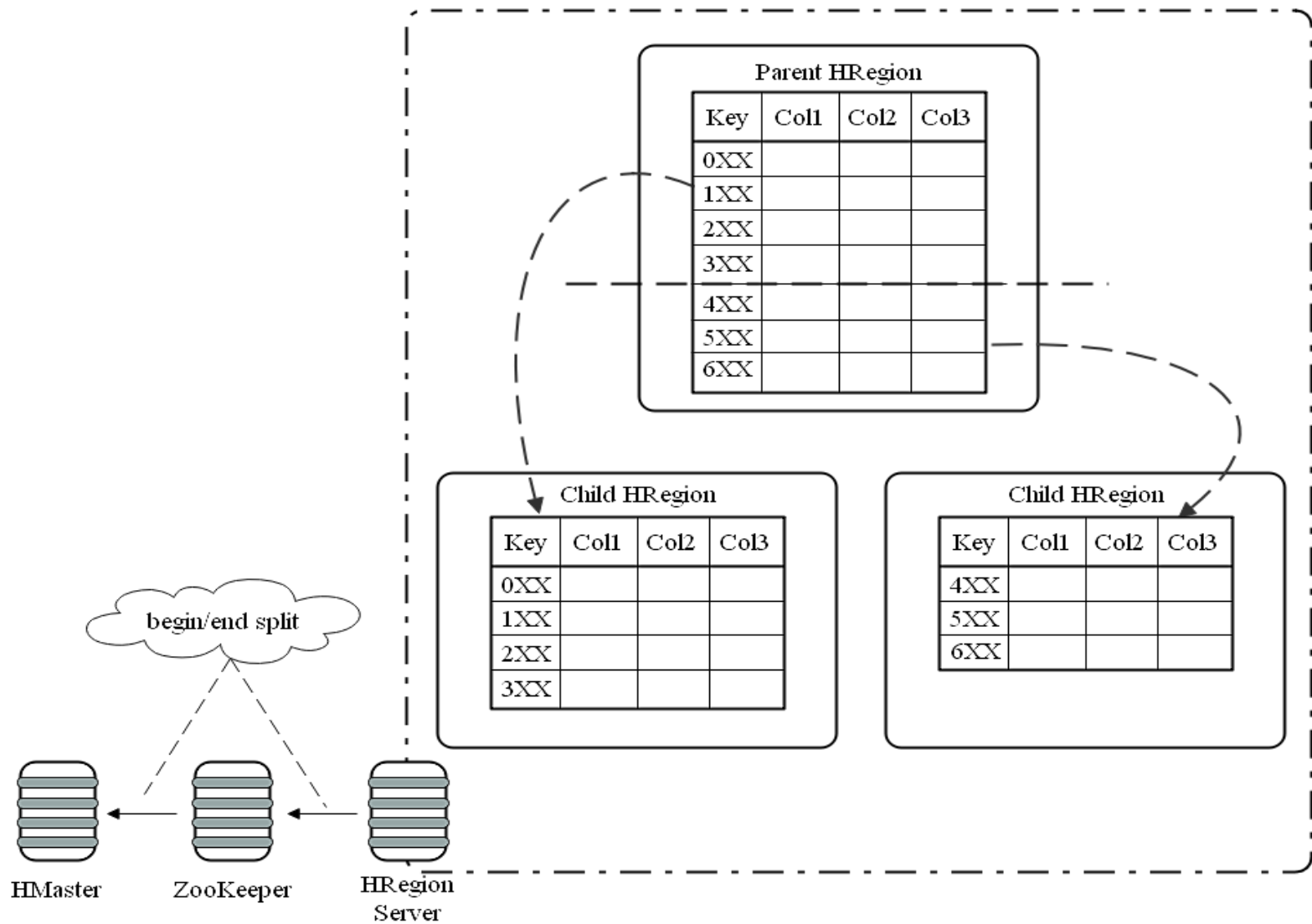
- 通常手动触发，选择集群比较空闲的时间操作
- 速度很慢，IO和带宽开销非常高



➤ Region Split


- 含义：根据一定的触发条件和分裂策略，将Region划分为两个子Region的过程
- 目的：实现数据访问的负载均衡
- 方法：利用Middle Key将当前Region划分为两个等分的子Region
- 触发
 - 时机：①Memstore溢写；②StoreFile Compaction；③手动
 - 条件：当Region中最大Store的大小超过阈值时，触发Region Split
- Split会产生大量的I/O操作
- Split开始前和Split完成后，HRegionServer都会通知HMaster
- Split完成后，HRegionServer会更新meta表

➤ Region Split



➤ HLog

- 含义：以WAL（Write Ahead Log，预写日志）方式写数据时产生的日志文件
- 目的：HRegionServer意外宕机时的数据恢复
- 先写HLog，再写MemStore，最后写StoreFile
- 每个HRegionServer维护一个HLog
- 定期删除HLog过期数据
- 数据恢复过程
 - HMaster通过ZooKeeper自动感知HRegionServer宕机
 - HMaster处理HLog，针对不同的Region拆分HLog
 - HMaster重新分配失效的Region
 - 通过HLog将尚未持久化的数据重新写入MemStore，然后溢写到StoreFile



3 chapter

Hyperbase安装与配置

- ✓ 系统安装
- ✓ 系统配置

1.选择服务	2.分配角色	3.配置服务	4.配置安全	5.服务总览	6.安装
选择想要添加的服务，可能存在某些由于依赖服务未添加而无法添加的服务，请先添加其依赖的服务。					
<input type="checkbox"/> ZOOKEEPER ZooKeeper用于协调同步其他服务	<input type="checkbox"/> HDFS HDFS是Hadoop应用的基本存储系统	<input type="checkbox"/> YARN YARN是资源管理框架			
<input type="checkbox"/> SEARCH Search是一个分布式的搜索分析引擎	<input checked="" type="checkbox"/> HYPERBASE Hyperbase是实时在线事务处理引擎	<input type="checkbox"/> TXSQL TxSQL是一个分布式的关系型数据库			
<input type="checkbox"/> NOTIFICATION 通知组件是一个提供相关组件中产生的消息的存储，查询，记录组件的工作状态的服务	<input type="checkbox"/> INCEPTOR Inceptor是基于内存的交互式SQL分析引擎	<input type="checkbox"/> TRANSPORTER TDT是数据整合工具			
<input type="checkbox"/> TRANSPEDIA Transpedia是TDH产品的文档检索服务	<input type="checkbox"/> OOZIE Oozie是Hadoop的一个工作流调度系统	<input type="checkbox"/> SQOOP Sqoop 用于在Hadoop和其他结构化数据存储中传送数据			
<input type="checkbox"/> SOPHON Sophon是一个深度学习交互式分析工具	<input type="checkbox"/> SLIPSTREAM Slipstream是基于Spark的实时流处理引擎	<input type="checkbox"/> DISCOVER Discover是基于内存的数据挖掘引擎			
<input type="checkbox"/> KAFKA Kafka是一个分布式的消息发布订阅系统	<input type="checkbox"/> WORKFLOW Workflow是图形化的工作流设计、调试、调度和分析的服务平台	<input type="checkbox"/> RUBIK Rubik是OLAP Cube的可视化设计工具			
<input type="checkbox"/> HUE Hue是Apache Hadoop中用来分析数据的用户界面	<input type="checkbox"/> GOVERNOR Governor是元数据管理工具				

1.选择服务

2.分配角色

3.配置服务

4.配置安全

5.服务总览

6.安装

本页面为选中的服务分配角色，默认已按推荐方法分配，如果想要修改分配，先点选左侧菜单的一个角色，再在中间面板上选中/取消从而为该角色分配节点

▼ Hyperbase2

Hyperbase Master

Region Server

Hyperbase Thrift

搜索主机...

✓

tdh-12

/default-rack

✓

tdh-13

/default-rack

✓

tdh-14

/default-rack

全选

全不选

依赖ZOOKEEPER

ZooKeeper1

依赖HDFS

HDFS1

依赖YARN

YARN1

依赖SEARCH

Search1

ARP
技

1.选择服务

2.分配角色

3.配置服务

4.配置安全

5.服务总览

6.安装

配置选中的服务

Hyperbase2

属性

基础参数

自定义参数

基础参数	值
hbase.rootdir	<input type="text" value="hdfs://nameservice1/hyperbase2"/>
hregion.index.path	<input type="text" value="hdfs://nameservice1/hyperbase2_hregionindex"/>
master.port	<input type="text" value="60000"/>
master.info.port	<input type="text" value="60010"/>
master.jmx.port	<input type="text" value="10101"/>
regionserver.port	<input type="text" value="60020"/>
regionserver.info.port	<input type="text" value="60030"/>
regionserver.jmx.port	<input type="text" value="10102"/>
zookeeper.session.timeout	<input type="text" value="180000"/>

>

Hyperbase1

●

HEALTHY

▶

■

✕

更多操作 ▾

🏠 主页 > Hyperbase1


可在本页编辑服务的配置。修改或者增加配置项后请点击“保存更改”按钮，更改的配置项才被保存。

🔍 搜索配置项...

全部配置 ▾

+ 增加自定义参数

配置项	配置类型	配置文件	值	描述
dfs.support.append	预定义		true	🔄 恢复推荐值
hbase.abort.disconnected.batchmutate	预定义		true	🔄 恢复推荐值
hbase.assignment.timeout.management	预定义		true	🔄 恢复推荐值
hbase.balancer.period	预定义		300000	🔄 恢复推荐值
hbase.client.meta.operation.timeout	预定义		60000	🔄 恢复推荐值
hbase.client.operation.timeout	预定义		60000	🔄 恢复推荐值
hbase.client.scanner.caching	预定义		200	🔄 恢复推荐值
hbase.client.scanner.timeout.period	预定义		60000	🔄 恢复推荐值
hbase.cluster.distributed	预定义		true	🔄 恢复推荐值
hbase.coprocessor.master.classes	预定义		org.apache.hadoop.hbase.master.TransactionMasterObserver	🔄 恢复推荐值
hbase.coprocessor.region.classes	预定义		org.apache.hadoop.hbase.transaction.cp.TransactionServiceImpl,org.apache.hadoop.hbase.transaction.cp.TransactionScannerObserver,org.apache.hadoop.hbase.regionserver.TransactionRegionObserver,org.apache.hadoop.hyperbase.coprocessor.LocalIndexerCoprocessor,org.apache.hadoop.hyperbase.secondaryindex.coprocessor...	🔄 恢复推荐值



4 chapter

Hyperbase基本用法

- ✓ HBase Shell命令
- ✓ Inceptor SQL

➤ 创建表

```
/* 创建表，cf1、cf2为列族名 */  
create 'table_name', 'cf1', 'cf2' ...
```

➤ 写入数据

```
/* 插入或更新数据，cf为列族名，cq为列名，value为要写入的数据 */  
put 'table_name', 'row_key', 'cf:cq', 'value'[, timestamp]
```

➤ 读取数据

```
/* 读取单行数据 */  
get 'table_name', 'row_key'  
/* 读取列族数据 */  
get 'table_name', 'row_key', 'cf'  
/* 读取列数据 */  
get 'table_name', 'row_key', 'cf:cq'
```

➤ 删除数据

```
/* 删除列 */  
delete 'table_name', 'row_key', 'cf: cq',[, timestamp]  
/* 删除行 */  
deleteall 'table_name', 'row_key'
```

➤ 全表扫描

```
/* 查看表的全部数据 */  
scan 'table_name'
```

➤ 计算表的行数

```
count 'table_name'
```

➤ Hyperdrive表

- Hyperdrive表是为了实现SQL与HBase的对接而专门设计的Inceptor表
- Hyperdrive表是HBase表的二维映射表，映射表的列对应于原表的列（Column Family:Column Qualifier），映射表中单元格的值对应于原表中单元格的最新值
- Inceptor在Hyperbase中的作用
 - SQL引擎：SQL解析、优化和执行，SQL与HBase之间的桥梁
 - 数据仓库：管理Hyperdrive表，以及与HBase表的映射关系，表的Schema存储在Inceptor Metastore中

➤ 创建Hyperdrive内表

- 在HBase中创建一张表，同时在Inceptor中创建对应的映射表（Hyperdrive内表）

```
CREATE TABLE <tableName>  
  (<key> <data_type>, <column1> <data_type>, <column2> <data_type>, ...)  
  STORED AS HYPERDRIVE;
```

➤ 创建Hyperdrive外表

- 在Inceptor中建映射表（Hyperdrive外表），与已存在的HBase表建立映射关系

```
CREATE EXTERNAL TABLE <tableName>  
  (<key> <data_type>, <column1> <data_type>, <column2> <data_type>, ...)  
  STORED BY 'io.transwarp.hyperdrive.HyperdriveStorageHandler'  
  WITH SERDEPROPERTIES('hbase.columns.mapping' = ':key, <f:q1>, <f:q2>, ...') ①  
  TBLPROPERTIES ("hbase.table.name" = "<hbase_table>"); ②
```

① 定义外表与HBase表的映射关系，:key为固定写法，其他项为 <columnfamily>:<column_qualifier>的自定义组合

② 定义外表对应的HBase表，且<hbase_table>必须是HBase中已存在的表

➤ 删除Hyperdrive表

- 对于内表，删除HBase表（含数据），以及Inceptor中的Hyperdrive表（含Metastore中的元数据）
- 对于外表，只删除Inceptor中的Hyperdrive表（含Metastore中的元数据），不删除HBase表

```
DROP TABLE <tableName>;
```

➤ 插入数据

```
/* 向Hyperdrive表中插入数据 */  
INSERT INTO TABLE <tableName> [(<column1>, <column2>, ...)]  
VALUES (<value1>, <value2>, ...);
```

➤ 查询数据

```
/* 通过索引查询Hyperdrive表 */  
SELECT /*+USE_INDEX(<table_alias> USING <index_name>)* / ①  
FROM <tableName> <table_alias> ②  
WHERE <filter_conditions>; ③
```

① 指定表的别名和索引名

② 必须为表起别名

③ 过滤条件中至少包含所使用索引中的一个列

```
/* 不通过索引查询Hyperdrive表 */  
SELECT /*+USE_INDEX(<table_alias> USING NOT_USE_INDEX)* /  
FROM <tableName> <table_alias>  
WHERE <filter_conditions>;
```




5 chapter

Hyperbase全局索引

- ✓ 索引原理
- ✓ Index DDL

- 原表（一级索引）
 - 各行数据按Rowkey的字典序排列
 - 虽然对Rowkey的查询效率很高，但无法实现基于其他列的条件查询，如： `select * from student where cfcq = 'A004'`
- 全局索引（二级索引）
 - 通过在某个或某些列上建立二级索引，实现基于列的快速条件查询
 - 一张表可以建多个全局索引

原表（一级索引）

RowKey	cf:cq	value
A	A002	52
B	A001	23
C	A004	78
D	A003	13



全局索引（二级索引）

cf:cq	RowKey
A001	B
A002	A
A003	D
A004	C

➤ 创建全局索引

- 新建的索引中没有数据，数据在以下两种情况下才会生成
 - 原表中有新的数据插入时，系统会为新插入的数据自动生成索引
 - 对索引执行重建（Rebuild）操作，系统会为表中所有数据生成索引

```
CREATE GLOBAL INDEX <index_name> ON <tableName> (  
  <column1> <SEGMENT LENGTH length1> | <(length1)> ①  
  [, <column2> <SEGMENT LENGTH length2> | <(length2)>, ...] ②  
);
```

① <column1>: 在某个列上建全局索引，但不可以是首列，因首列映射为RowKey

② <SEGMENT LENGTH length2>: 只有索引列为String类型时，才需要指定长度，简写为(length2)

eg: CREATE GLOBAL INDEX index_name ON hyper_student(name(12));

➤ 查看索引信息

- 索引创建后，通过查看表的索引信息，确认是否创建成功

```
DESCRIBE FORMATTED <tableName>;
```


➤ 生成全局索引

- 在HBase Shell中执行Rebuild命令来生成索引
- 不支持SQL方式生成索引

```
/* 为全表生成索引 */  
# rebuild_global_index 'table_name', 'index_name'  
  
/* 在指定的RowKey范围内生成索引，生产环境中的大表应采用此方法分批建立索引 */  
# rebuild_global_index_with_range 'table_name', 'index_name', 'start_key', 'end_key'
```

➤ 删除全局索引

```
DROP INDEX [IF EXISTS] <index_name> ON <tableName>;
```

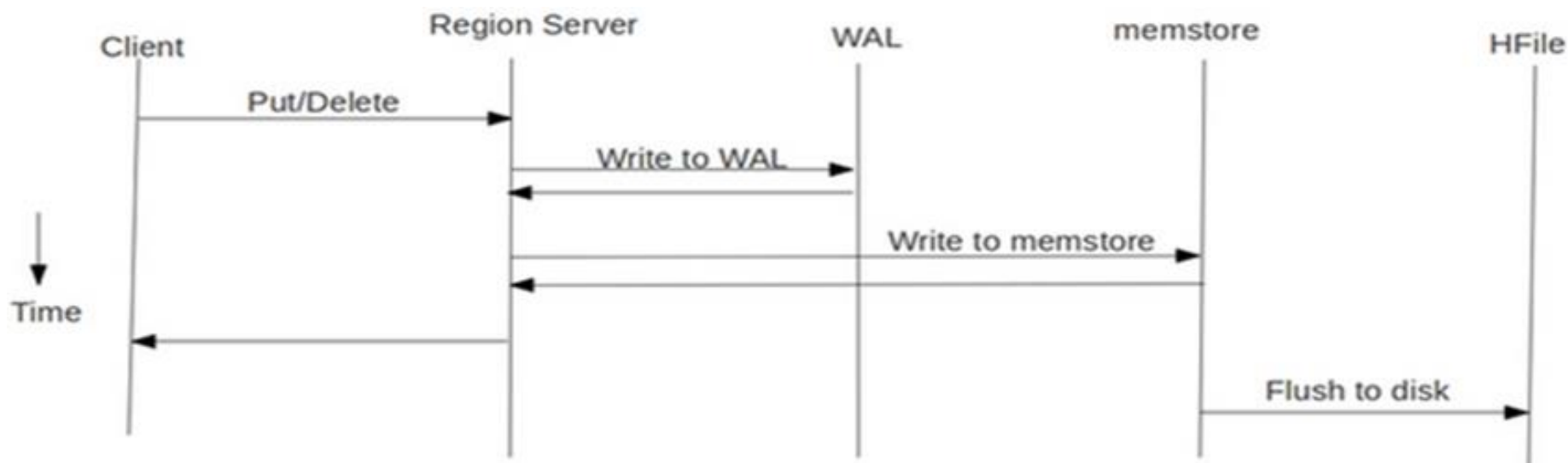


6 chapter

Hyperbase数据快速入库

- ✓ 传统数据入库
- ✓ BulkLoad数据入库

- TableOutputFormat方式：利用MapReduce将数据批量写入（put）到Hyperbase中
- 在大规模数据入库时，TableOutputFormat方式效率低下
 - 导致频繁的Flush、Split、Compact等IO操作
 - 在持续写数据的过程中，Memstore会占用大量内存
 - 被迫使用更大的HLog，或完全停用HLog
 - 对节点的稳定性造成一定影响，如：GC时间过长，系统响应过慢，导致节点超时退出



➤ 概念

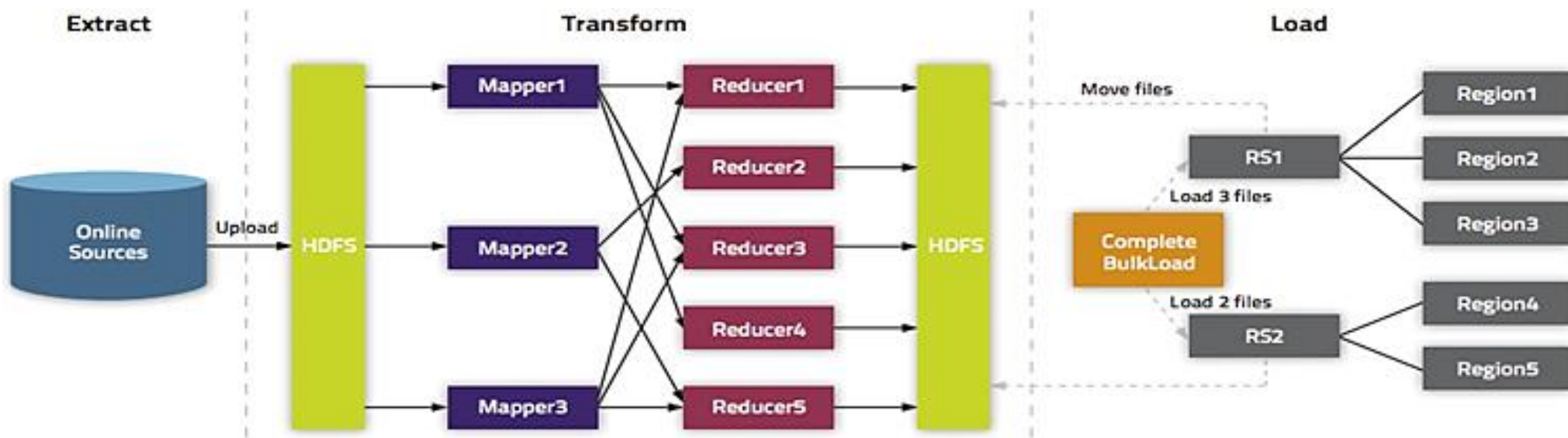
- 含义：由于HBase中数据以HFile文件的形式存储于HDFS，所以我们绕过HBase API，直接将数据加工成HFile文件，再将其加载到HBase中，从而完成大规模数据的快速入库
- 优点
 - 极大地提高了大规模数据的入库效率
 - 不占用HRegionServer资源，显著减轻了Hyperbase集群的写入压力
 - 利用MapReduce或Spark加工HFile文件，非常高效

➤ HBase BulkLoad的基本流程

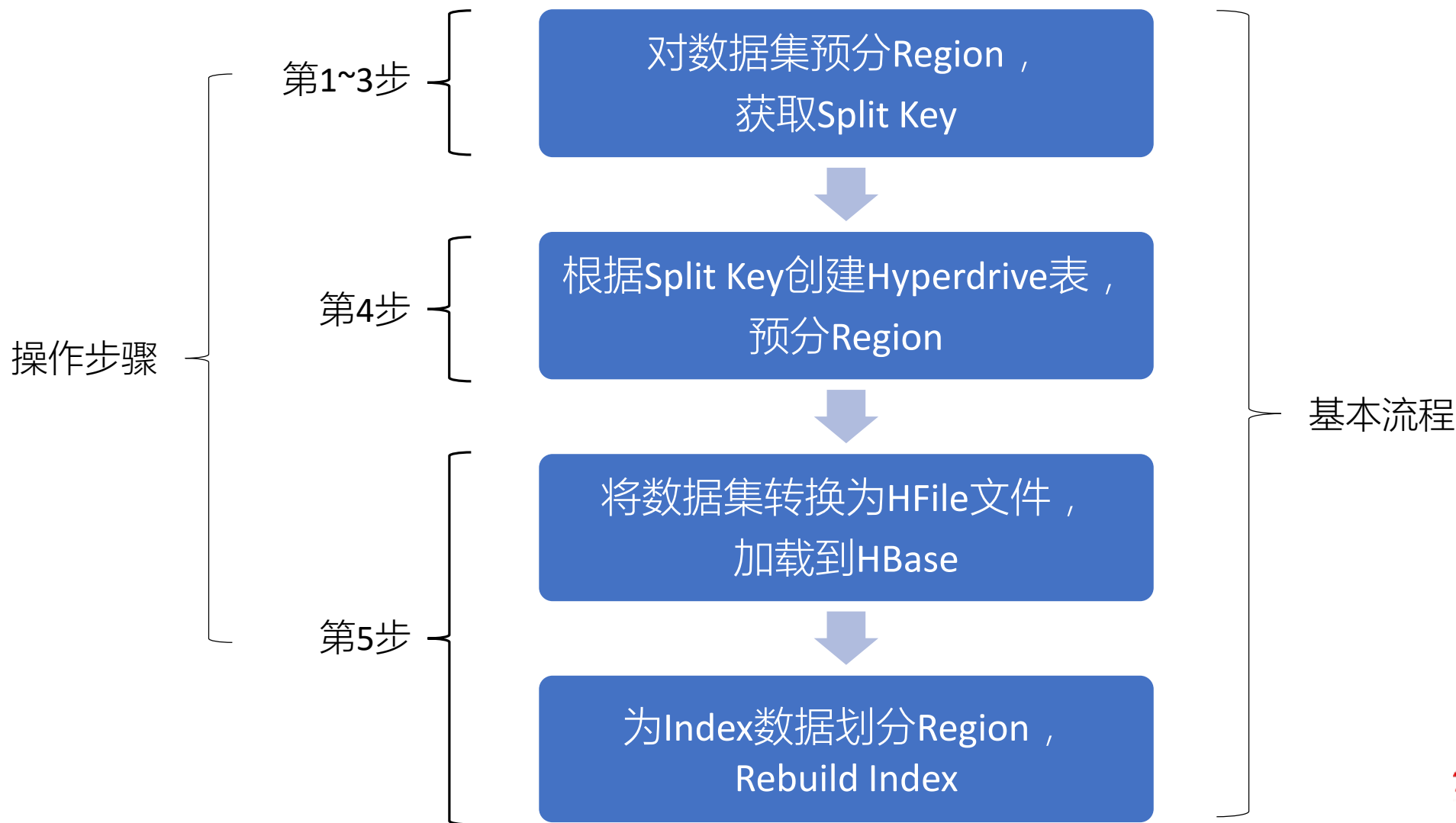
- 提取：从数据源中提取数据
 - 对于MySQL，运行mysqldump命令导出数据
- 转换：利用MapReduce，将数据转换为HFile文件
 - 对于TSV或CSV文件，使用[HBase ImportTsv](#)工具将其转换成HFile文件
 - 每个输出文件夹中的每个区域都会创建一个HFile文件

➤ HBase BulkLoad的基本流程

- 转换：利用MapReduce，将数据转换为HFile文件
 - HDFS中的可用磁盘空间至少为原始输入文件的两倍。例如，对于100GB的mysqldump导出文件，HDFS中至少预留不少于200GB的磁盘空间，可以在任务结束后删除原始输入文件
- 加载：将HFile文件加载到HBase
 - 利用[HBase CompleteBulkLoad](#)工具，将HFile文件移动到HBase表的相应目录中，完成加载



➤ SQL BulkLoad的基本流程



➤ SQL BulkLoad的操作步骤

- 第1步：将数据集上传至HDFS
- 第2步：为HDFS中的数据集创建Inceptor外表
- 第3步：对外表（第2步创建）预分Region，获取Split Key
 - 人工选取Split Key
 - 通过SQL自动生成Split Key
 - ① 对外表进行采样
 - ② 利用采样结果生成一张Split Key表
- 第4步：在Inceptor中创建Hyperdrive表（HBase表的二维映射表），利用Split Key（第3步获取）对HBase表预分Region
- 第5步：在Inceptor中使用SQL BulkLoad语句，将外表中的数据导入Hyperdrive表
 - 实际过程是通过Hyperbase表，将数据转换为HFile文件后，加载到HBase表的相应目录中，并生成索引

➤ SQL BulkLoad的操作步骤

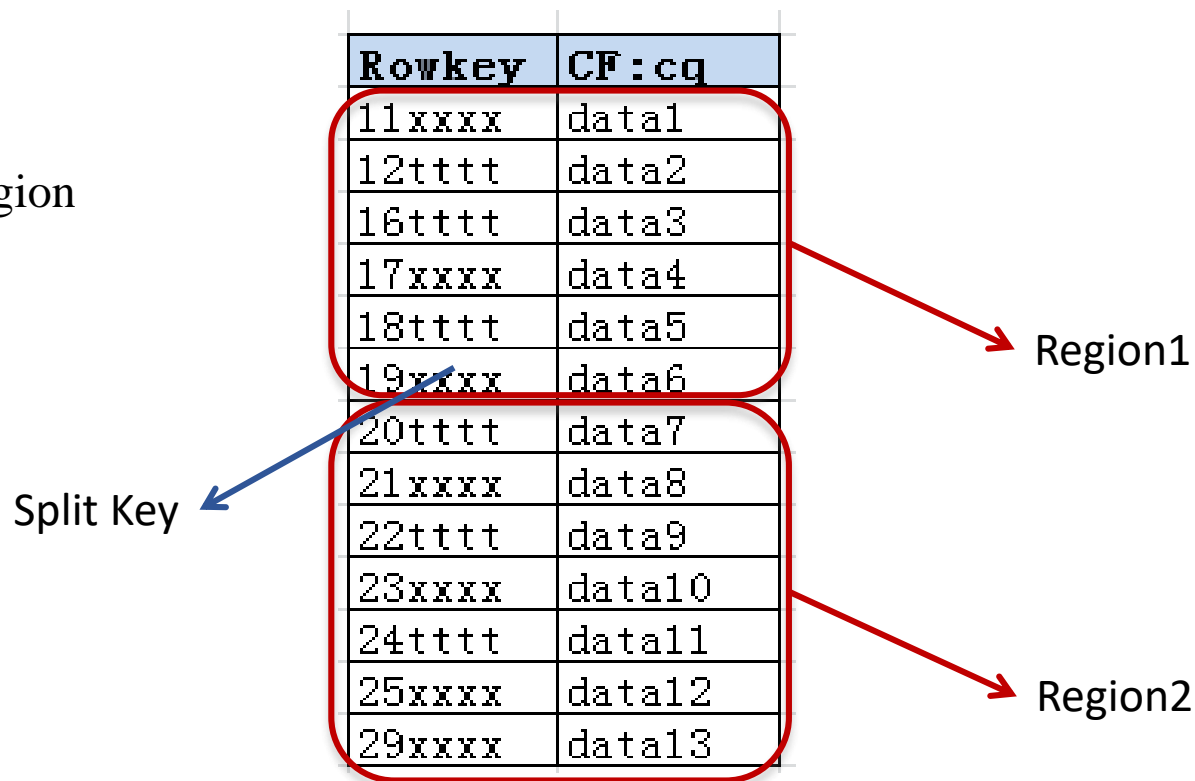
- 第3步：对Inceptor外表（HDFS数据集）预分Region

- 根据RowKey对数据集预先划分Region，使各个Region的大小均匀、行数接近

- Region分配原则

- ① 每个Region的大小为1~3GB

- ② 每个HResionServer管理10~1000个Region



➤ SQL BulkLoad的操作步骤

- 第3步：通过SQL自动生成Split Key

```
/* 计算外表（数据集）的记录数 */
select count(*) from text_table;
/* 创建采样表 */
create table sample_table(id int, name string);
/* 在外表中进行采样，并将采样结果插入sample_table
   sample函数：x是采样率（=count(*) / 97 / region数），第2~N个参数指定需要选出的列名（至少需要指定一列），实际运用中需要选出所有用来拼接 RowKey的字段*/
insert into table sample_table select sample(x, key1, value1) from text_table;
/* 自动生成Split Key
   RowKey由id和name组成，ntile()的参数为预分Region数，limit值为(Region数-1) */
select mid from (
  select max(c) as mid from (
    select concat(id,name) c, ntile(10) over (order by concat(id,name)) nt from sample_table
  ) group by nt
) order by mid limit 9;
```

➤ SQL BulkLoad的操作步骤

- 第4步：根据Split Key创建Hyperdrive表，并预分Region

```
create table holo_hyperdrive (  
  rowkey string,  
  num int,  
  country string,  
  rd string)  
stored as hyperdrive  
tblproperties ('hyperdrive.table.splitkey' = '"1100000000", "1200000000", "1300000000", "1400000000",  
  "1500000000", "1600000000", "1700000000", "1800000000", "1900000000"');
```

- 第5步：将外表中的数据导入Hyperdrive表

```
insert overwrite table holo_hyperdrive  
select /*+USE_BULKLOAD*/ rowkey, num, country, rd  
from text_table order by rowkey;
```



Q&A

TRANSWARP
星环科技