

Hbase 原理及基本架构

主讲人：倪忠文

HBase概述

HBase应用场景

HBase数据模型

HBase物理模型

HBase基础架构

HBase读写流程

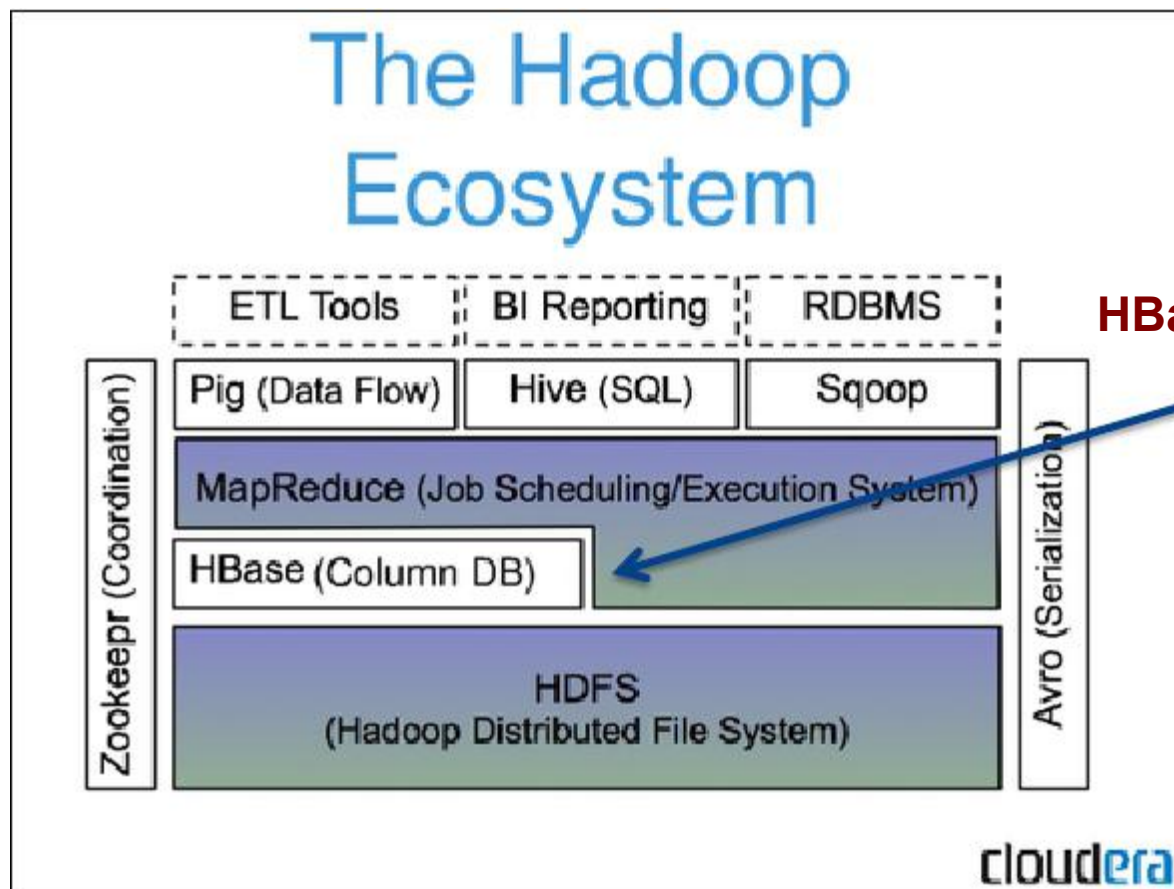
总结

HBase概述

HBase是一个构建在HDFS之上的、分布式的、面向列的开源数据库，由Google BigTable的开源实现，它主要用于存储海量数据，是Hadoop生态系统中的重要一员。

- Hadoop Database，是一个高可靠性、高性能、面向列、可伸缩、实时读写的分布式数据库
- 利用Hadoop HDFS作为其文件存储系统,利用Hadoop MapReduce来处理HBase中的海量数据,利用Zookeeper作为其分布式协同服务
- 主要用来存储非结构化和半结构化的松散数据（列存 NoSQL 数据库）

HBase概述



HBase 构建在HDFS之上

Hbase内部管理的文件
全部存储在**HDFS**中

Hbase是Hadoop生态系统的一个组成部分

Hbase表的特点

大

大：一个表可以有数十亿行，上百万列；

无模式

无模式：每行都有一个可排序的主键和任意多的列，列可以根据需要动态的增加，同一张表中不同的行可以有截然不同的列；

面向列

面向列：面向列（族）的存储和权限控制，列（族）独立检索；

稀疏

稀疏：对于空（null）的列，并不占用存储空间，表可以设计的非常稀疏

数据多版本

数据多版本：每个单元中的数据可以有多个版本，默认情况下版本号自动分配，是单元格插入时的时间戳；

数据类型单一

数据类型单一：Hbase中的数据都是字符串，没有类型。

目录



HBase概述

HBase应用场景

HBase数据模型

HBase物理模型

HBase基础架构

HBase读写流程

总结

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。

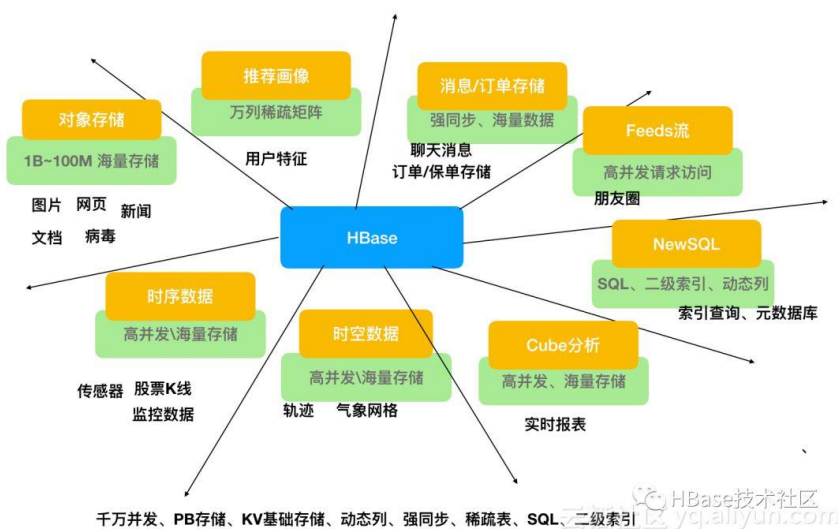
- storing large amounts of data (100s of TBs)
- need high write throughput
- need efficient random access (key lookups) within large data sets
- need to scale gracefully with data
- for structured and semi-structured data
- don't need full RDMS capabilities (cross row/cross table transactions, joins,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

什么公司在使用HBase



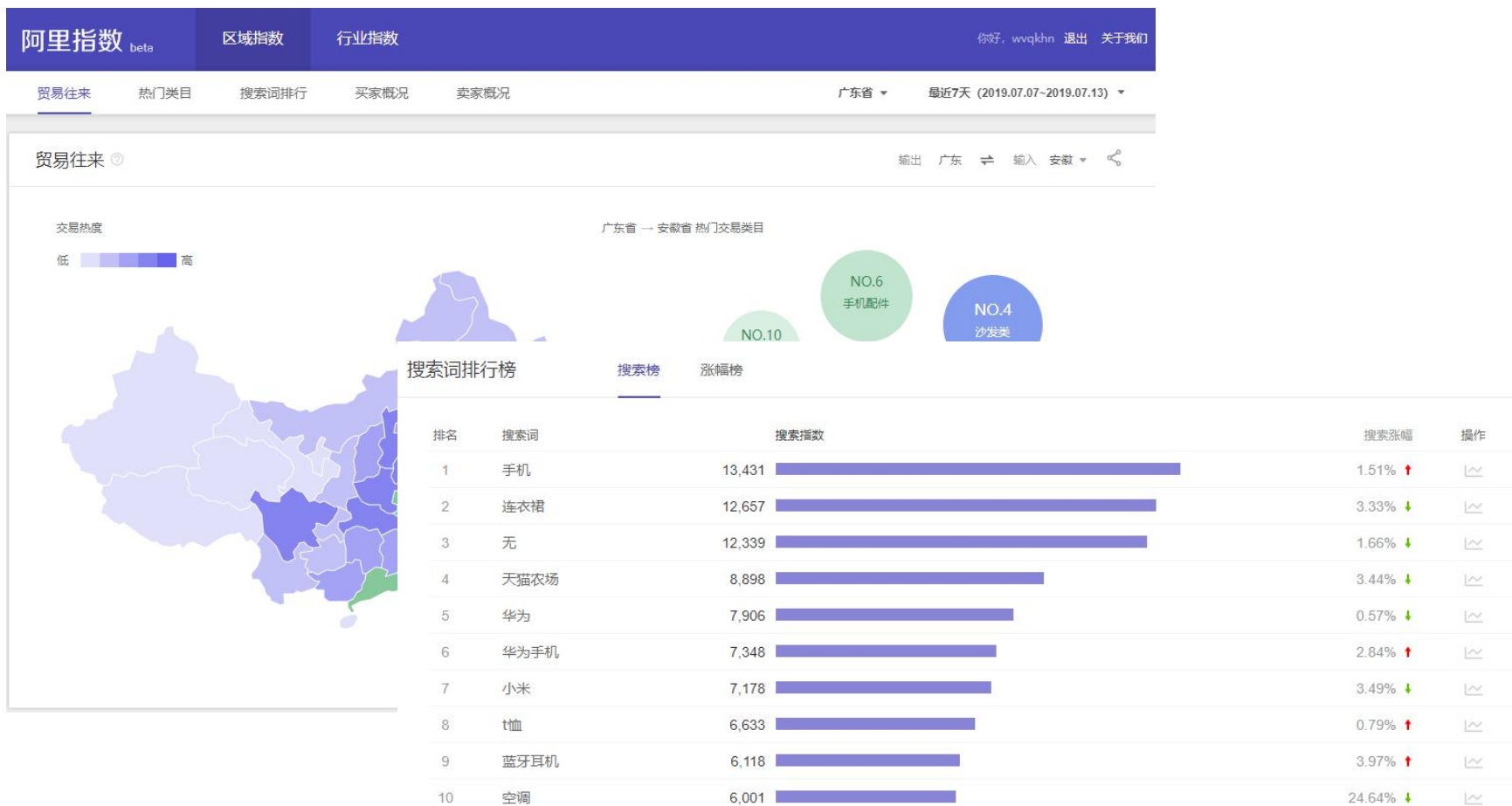
什么公司在使用HBase





- 一. 对象存储：我们知道不少的头条类、新闻类的新闻、网页、图片存储在HBase之中，一些病毒公司的病毒库也是存储在HBase之中
- 二. 时序数据：HBase之上有OpenTSDB模块，可以满足时序类场景的需求
- 三. 推荐画像：特别是用户的画像，是一个比较大的稀疏矩阵，蚂蚁的风控就是构建在HBase之上
- 四. 时空数据：主要是轨迹、气象网格之类，滴滴打车的轨迹数据主要存在HBase之中，另外在技术所有大一点的数据量的车联网企业，数据都是存在HBase之中
- 五. CubeDB OLAP：Kylin一个cube分析工具，底层的数据就是存储在HBase之中，不少客户自己基于离线计算构建cube存储在hbase之中，满足在线报表查询的需求
- 六. 消息/订单：在电信领域、银行领域，不少的订单查询底层的存储，另外不少通信、消息同步的应用构建在HBase之上
- 七. Feeds流：典型的应用就是xx朋友圈类似的应用
- 八. NewSQL：之上有Phoenix的插件，可以满足二级索引、SQL的需求，对接传统数据需要SQL非事务的需求

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

- Facebook创建了Cassandra，最后却弃用Cassandra，使用了HBase；
- 消息系统（聊天系统、邮件系统等）需求：
 - ✓ 一个较小的临时数据集，是经常变化的。
 - ✓ 一个不断增加的数据集，是很少被访问的。
- Hbase同时解决了以上两种需求
- 参照：

http://www.facebook.com/note.php?note_id=454991608919#

目录



HBase概述

HBase应用场景

HBase数据模型

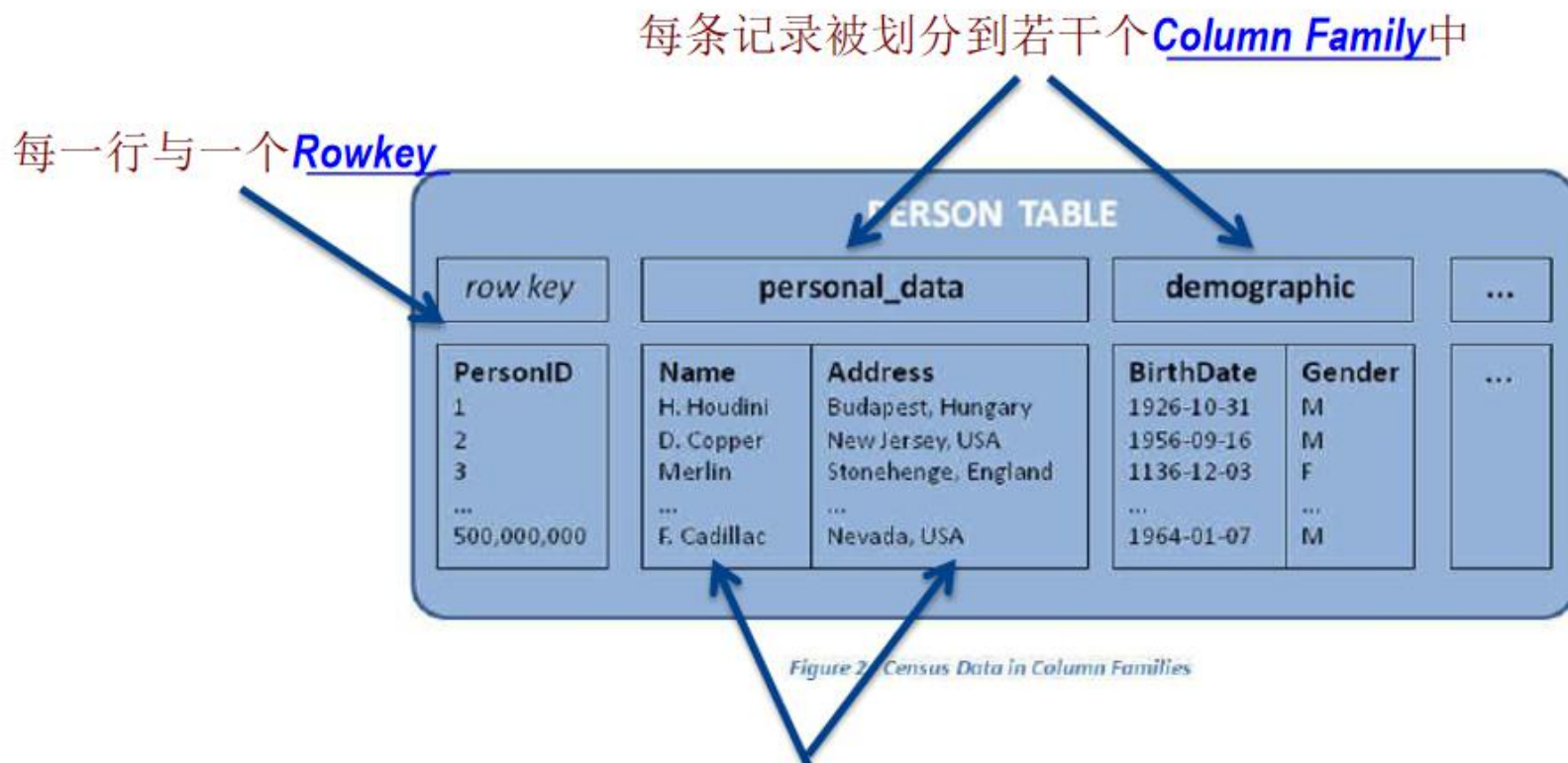
HBase物理模型

HBase基础架构

HBase读写流程

总结

Rowkey与ColumnFamily



每个 **column family** 由一个或者多个 **Column** 组成

➤ Row Key

- ✓ Byte array
- ✓ 表中每条记录的“主键”
- ✓ 方便快速查找

➤ Column Family

- ✓ 拥有一个名称(string)
- ✓ 包含一个或者多个相关列

➤ Column

属于某一个column family
包含在某一行中

- *familyName:columnName*

名称为“Contents”的column family

名称为“anchor”的column family

Row key	Time Stamp	Column “content s:”	Column “anchor:”	
“com.apac he.ww w”	t12	“<html> ...”		
	t11	“<html> ...”		
	t10		“anchor:apache .com”	“APACH E”
“com.cnn.w ww”	t15		“anchor:cnnsi.co m”	“CNN”
	t13		“anchor:my.look ca”	“CNN.co m”
	t6	“<html> ...”		
	t5	“<html> ...”		
	t3	“<html> ...”		

名称为“apache.com”的列

Hbase基本概念

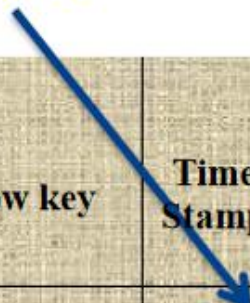
每一行有一个版本号

➤ Version Number

- ✓ 每个rowkey唯一
- ✓ 默认值 → 系统时间戳
- ✓ 类型为Long

➤ Value (Cell)

- ✓ Byte array



Row key	Time Stamp	Column "content s:"	Column "anchor:"	
"com.apache.ww"	t12	"<html>..."		value
	t11	"<html>..."		
	t10		"anchor:apache.com"	"APACHE"
"com.cnn.ww"	t15		"anchor:cnn.com"	"CNN"
	t13		"anchor:my.look.ca"	"CNN.com"
	t6	"<html>..."		
	t5	"<html>..."		
	t3	"<html>..."		

Hbase数据模型

HBase schema可以有多个 *Table*
 每个表可由多个 *Column Family* 组成

HBase 可以有 *Dynamic Column*

- 列名称是编码在cell中的
- 不同的cell可以拥有不同的列


“Roles” column family has
 different columns in
 different cells



Row key	Data
cutting	info: { 'height': '9ft', 'state': 'CA' } roles: { 'ASF': 'Director', 'Hadoop': 'Founder' }
tlipcon	info: { 'height': '5ft7', 'state': 'CA' } roles: { 'Hadoop': 'Committer'@ts=2010, 'Hadoop': 'PMC'@ts=2011, 'Hive': 'Contributor' }

Hbase数据模型

- *version number* 可由用户提供
 - ✓ 无需以递增的顺序插入
 - ✓ 每一行的rowkey必须是唯一的
- **Table** 可能非常稀疏
 - ✓ 很多 cell 可以是空的
- *Row Key* 是主键



Row Key	Time Stamp	ColumnFamily contents	ColumnFamily anchor
"com.cnn.www"	t9		anchor:cnnsi.com = "CNN"
"com.cnn.www"	t8		anchor:my.look.ca = "CNN.com"
"com.cnn.www"	t6	contents:html = "<html>..."	
"com.cnn.www"	t5	contents:html = "<html>..."	
"com.cnn.www"	t3	contents:html = "<html>..."	

- 所有操作均是基于rowkey的；
- 支持CRUD（Create、Read、Update和Delete）和 Scan；
- 单行操作
 - ✓ Put
 - ✓ Get
 - Scan
- 多行操作
 - ✓ Scan
 - ✓ MultiPut
- 没有内置join操作，可使用MapReduce解决。

目录



HBase概述

HBase应用场景

HBase数据模型

HBase物理模型

HBase基础架构

HBase读写流程

总结

- 每个column family存储在HDFS上的一个单独文件中；
- Key 和 Version number在每个 column family中均由一份；
- 空值不会被保存。

Table 5.3. ColumnFamily contents

Row Key	Time Stamp	ColumnFamily "contents:"
"com.cnn.www"	t6	contents:html = "<html>..."
"com.cnn.www"	t5	contents:html = "<html>..."
"com.cnn.www"	t3	contents:html = "<html>..."

Table 5.2. ColumnFamily anchor

Row Key	Time Stamp	Column Family anchor
"com.cnn.www"	t9	anchor:cnnsci.com = "CNN"
"com.cnn.www"	t8	anchor:my.look.ca = "CNN.com"

HBase 为每个值维护了多级索引，即：
<key, column family, column name, timestamp>

Hbase物理模型-一个实例

Row key	Data
cutting	info: { 'height': '9ft', 'state': 'CA' } roles: { 'ASF': 'Director', 'Hadoop': 'Founder' }
tlipcon	info: { 'height': '5ft7', 'state': 'CA' } roles: { 'Hadoop': 'Committer'@ts=2010, 'Hadoop': 'PMC'@ts=2011, 'Hive': 'Contributor' }

info Column Family

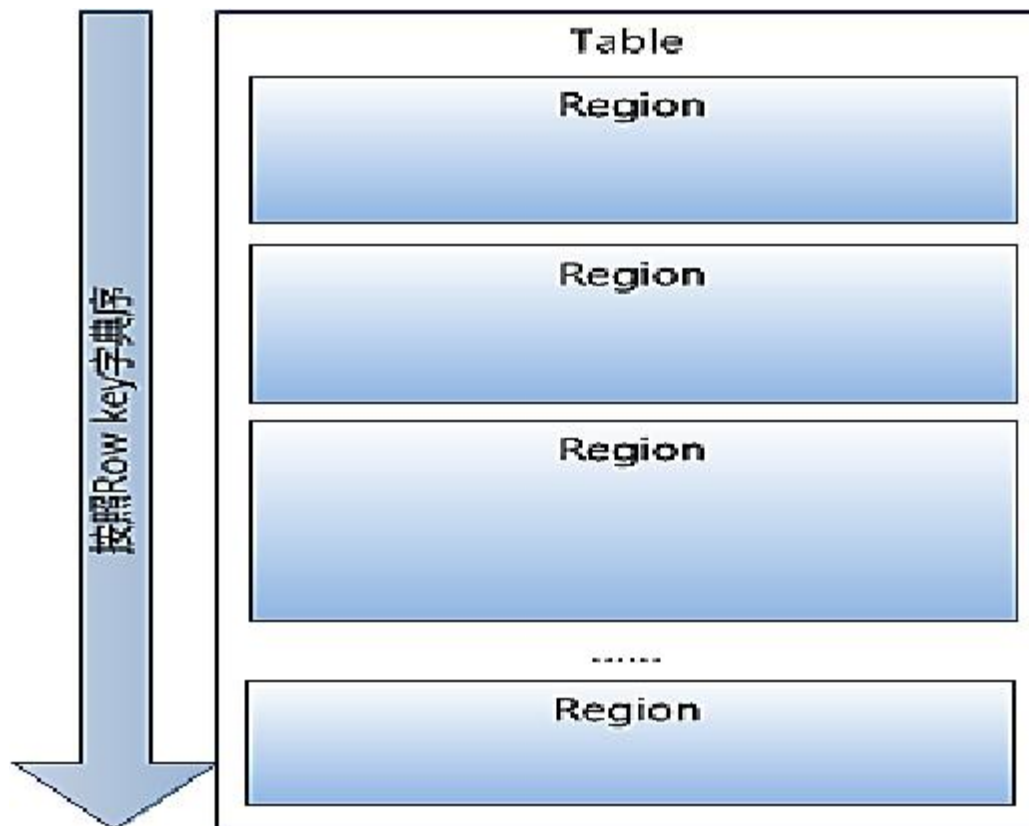
Row key	Column key	Timestamp	Cell value
cutting	info:height	1273516197868	9ft
cutting	info:state	1043871824184	CA
tlipcon	info:height	1273878447049	5ft7
tlipcon	info:state	1273616297446	CA

roles Column Family

Row key	Column key	Timestamp	Cell value
cutting	roles:ASF	1273871823022	Director
cutting	roles:Hadoop	1183746289103	Founder
tlipcon	roles:Hadoop	1300062064923	PMC
tlipcon	roles:Hadoop	1293388212294	Committer
tlipcon	roles:Hive	1273616297446	Contributor

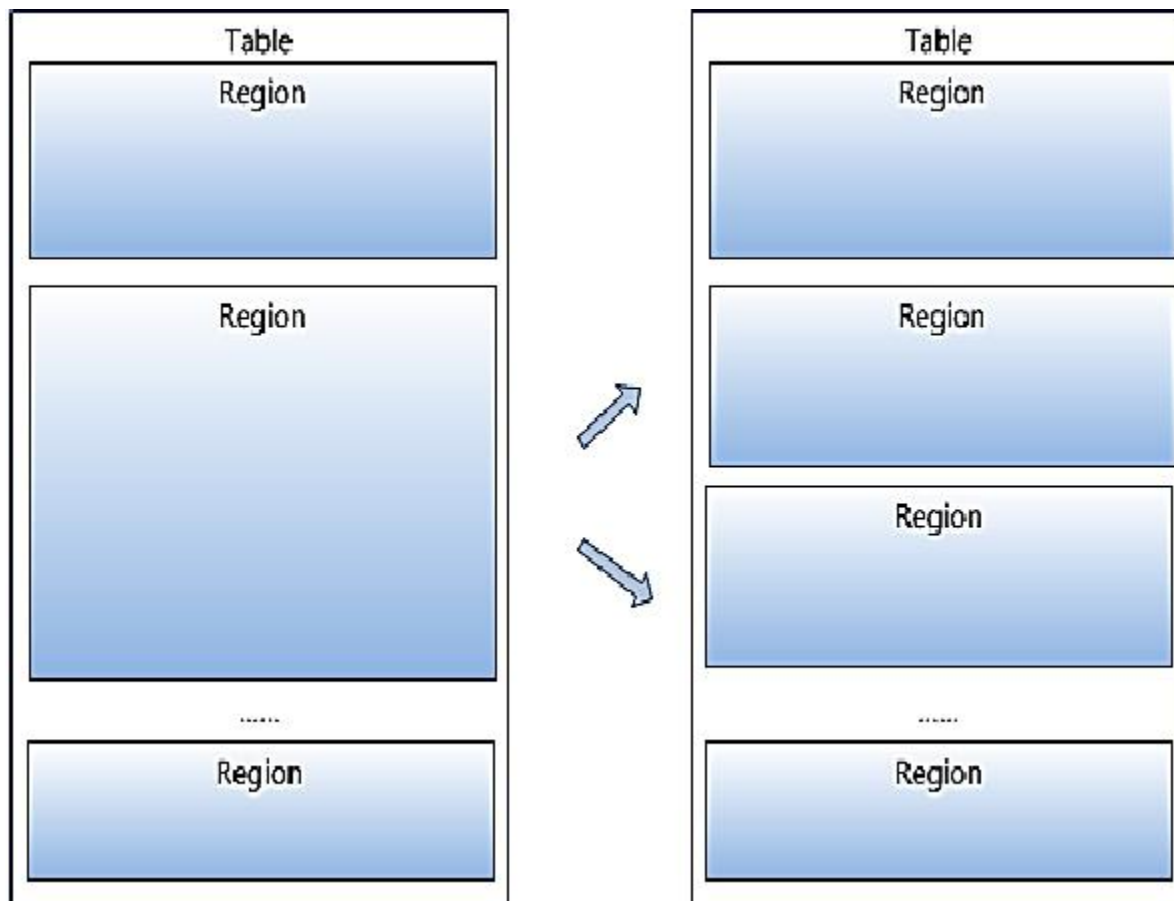
Sorted on disk by Row key, Col key, descending timestamp

- 1、**Table**中的所有行都按照**row key**的字典序排列；
- 2、**Table** 在行的方向上分割为多个**Region**；

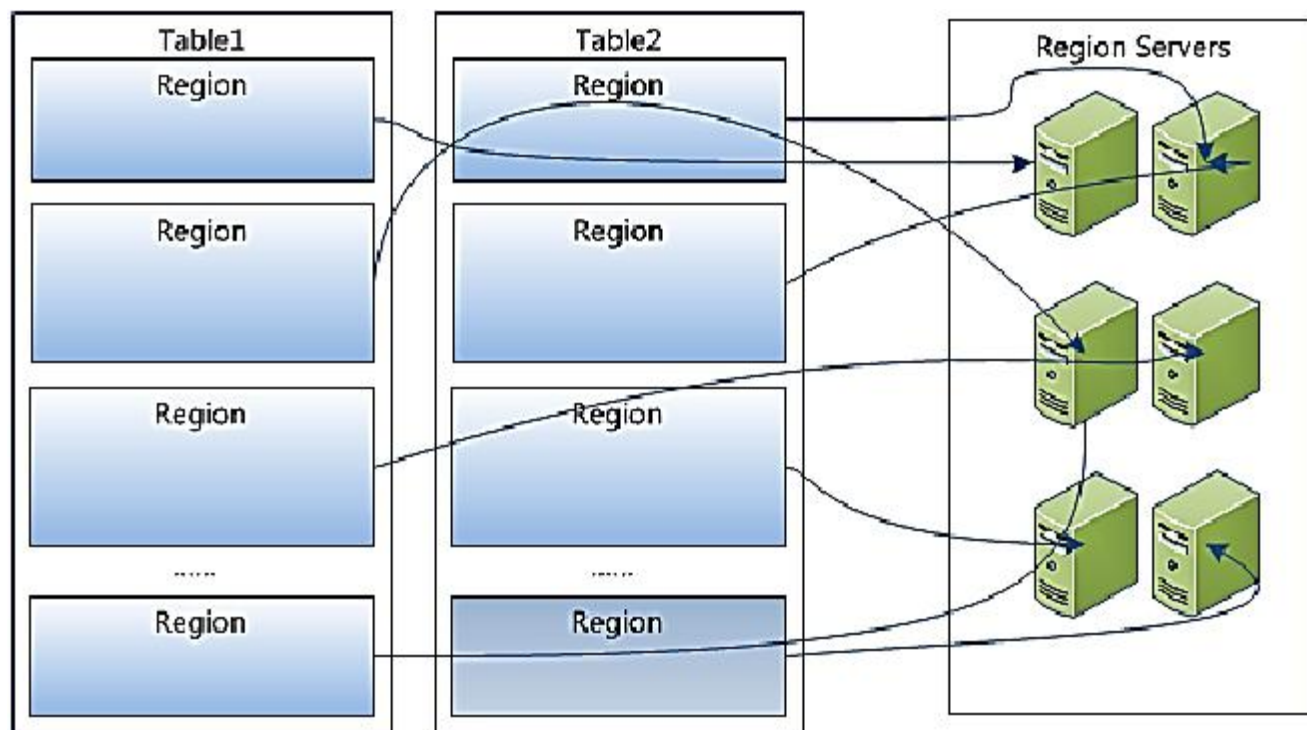


物理存储

3、**Region**按大小分割的，每个表开始只有一个**region**，随着数据增多，**region**不断增大，当增大到一个阈值的时候，**region**就会等分会两个新的**region**，之后会有越来越多的**region**；



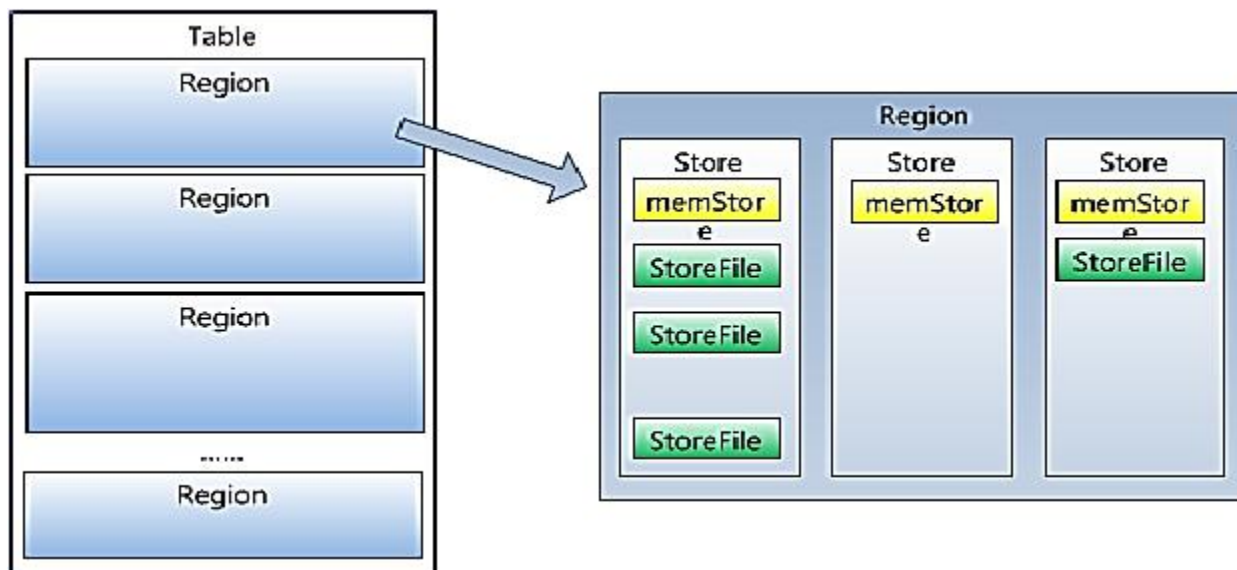
4、Region是HBase中分布式存储和负载均衡的最小单元。
不同Region分布到不同RegionServer上；



物理存储

5、Region虽然是分布式存储的最小单元，但并不是存储的最小单元。

- Region由一个或者多个Store组成，每个store保存一个columns family;
- 每个Store又由一个memStore和0至多个StoreFile组成;
- memStore存储在内存中，StoreFile存储在HDFS上。



The diagram illustrates the internal structure of an HBase StoreFile. It is composed of several key components:

- StoreFile** (Root):
 - DataBlocks**: A collection of data blocks. Each DataBlock points to a specific **DataBlock MAGIC (88)**, which contains a sequence of **Key-value** pairs (Key-value(first) to Key-value(Last)).
 - MetaBlocks (optional)**: Point to a **MetaBlock MAGIC (88)**, which contains metadata such as **Key-Len(int)**, **Val-Len(int)**, **Key(byte[])**, and **Value(byte[])**.
 - FileInfo**: Points to a **FileInfo MAGIC (88)**, which contains file-level information like **Size or Item Num**, **Last key(byte[])**, **Avg key len(int)**, **Avg val len(int)**, **Comparator(classname)**, and **User define**.
 - DataIndex**: Points to an **Index Block Magic**, which contains **Index of Data block 0** through **Index of Data block n**.
 - MetaBlock**: Points to a **MetaBlock MAGIC (88)**, which contains **Index of Meta block 0** through **Index of Meta block n**.
 - Trailer**: Points to a **Trailer MAGIC (88)**, which contains **Offset(long)**, **Data size(int)**, **Keylen(int)**, and **Key(byte[])**.

The diagram also shows the layout of the StoreFile on disk, with fields like **Total Uncompressed Data Bytes (long)**, **Entry Count or Data K-V Count (int)**, **Compression Codec (int)**, **Version (int)**, **TRAILER BLOCK MAGIC (88)**, **File Info Offset (long)**, **Data Index Offset (long)**, **Data Index Count (int)**, **Meta Index Offset (long)**, and **Meta Index Count (int)**.

- ◆ **Data Block段**：保存表中的数据，这部分可以被压缩。
- ◆ **Meta Block段（可选的）**：保存用户自定义的键值对，可以被压缩。
- ◆ **File Info段**：**Hfile**的元信息，不被压缩，用户也可以在这一部分添加自己的元信息。
- ◆ **Data Block Index段**：**Data Block**的索引，每条索引的**key**是被索引的**Block**的第一条记录的**key**。采用**LRU**机制淘汰。
- ◆ **Meta Block Index段（可选的）**：**Meta Block**的索引。
- ◆ **Trailer段**：这一段是定长的，保存了每一段的偏移量。

目录

HBaSe概述

HBaSe应用场景

HBaSe数据模型

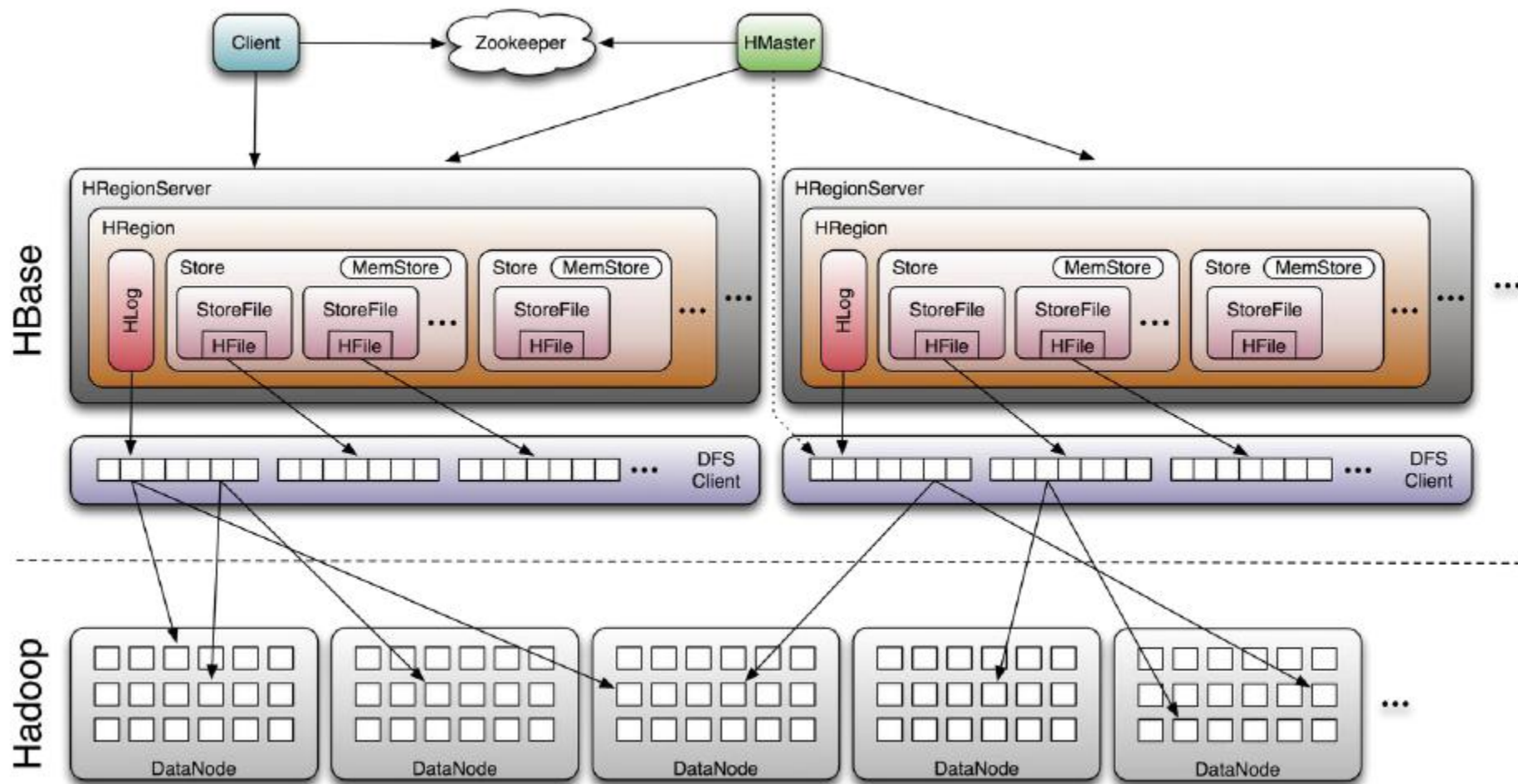
HBaSe物理模型

HBaSe基础架构

HBaSe读写流程

总结

HBase架构



➤ Client

- ✓ 包含访问HBase的接口，并维护cache来加快对HBase的访问

➤ Zookeeper

- ✓ 保证任何时候，集群中只有一个master
- ✓ 存贮所有Region的寻址入口
- ✓ 实时监控Region server的上线和下线信息。并实时通知给Master
- ✓ 存储HBase的schema和table元数据

➤ Master

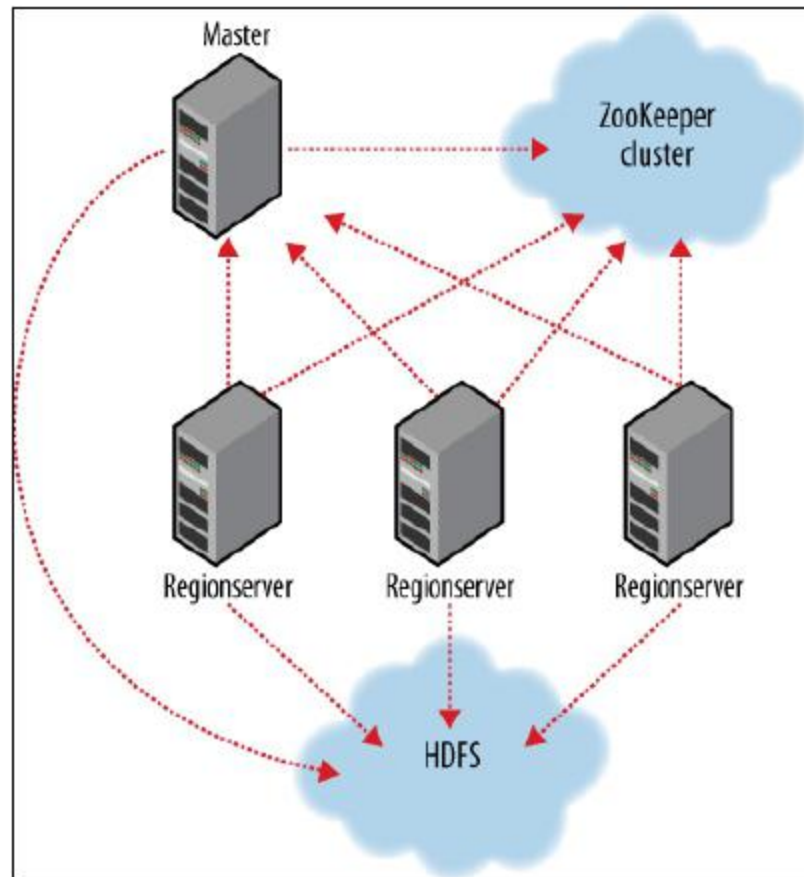
- ✓ 为Region server分配region
- ✓ 负责Region server的负载均衡
- ✓ 发现失效的Region server并重新分配其上的region
- ✓ 管理用户对table的增删改查操作

➤ Region Server

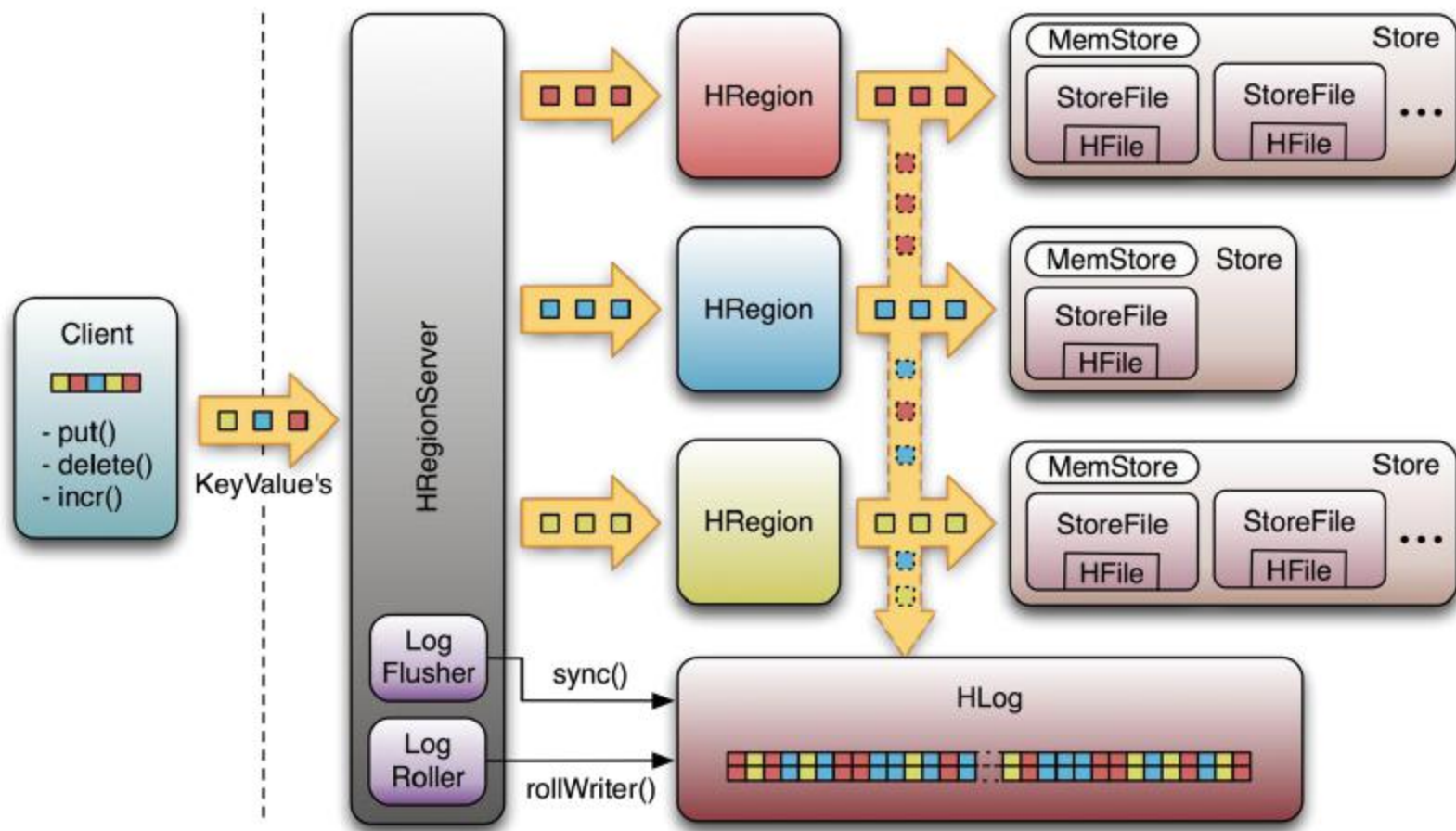
- ✓ Region server维护region，处理对这些region的IO请求
- ✓ Region server负责切分在运行过程中变得过大的region

ZooKeeper作用

- **HBase 依赖ZooKeeper**
- 默认情况下，**HBase** 管理**ZooKeeper** 实例
 - ✓ 比如，启动或者停止**ZooKeeper**
- **Master**与**RegionServers**启动时会向**ZooKeeper**注册
- **Zookeeper**的引入使得**Master**不再是单点故障



Write-Ahead-Log (WAL)



➤ **Master容错**: **Zookeeper**重新选择一个新的**Master**

✓无Master过程中，数据读取仍照常进行；

✓无master过程中，region切分、负载均衡等无法进行；

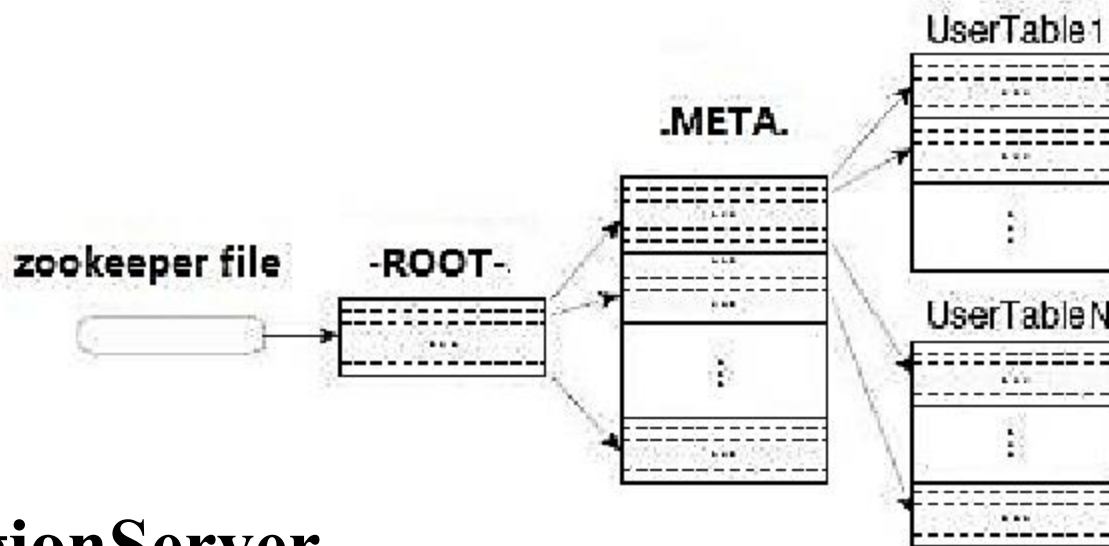
➤ **RegionServer容错**: 定时向**Zookeeper**汇报心跳，如果一旦时间内未出现心跳

✓Master将该RegionServer上的Region重新分配到其他RegionServer上；

✓失效服务器上“预写”日志由主服务器进行分割并派送给新的RegionServer

➤ **Zookeeper容错**: **Zookeeper**是一个可靠地服务

✓一般配置3或5个Zookeeper实例。



➤ 寻找RegionServer

- ✓ ZooKeeper
- ✓ -ROOT-(单Region)
- ✓ .META.
- ✓ 用户表

➤ **-ROOT-**

- ✓ 表包含.META.表所在的region列表，该表只会有一个Region；
- ✓ Zookeeper中记录了-ROOT-表的location。

➤ **.META.**

- ✓ 表包含所有的用户空间region列表，以及RegionServer的服务器地址。

关系数据库与Hbase比较

	RDBMS	HBase
Data layout	Row-oriented	Column-family-oriented
Transactions	Multi-row ACID	Single row only
Query language	SQL	get/put/scan/etc *
Security	Authentication/Authorization	Work in progress
Indexes	On arbitrary columns	Row-key only
Max data size	TBs	~1PB
Read/write throughput limits	1000s queries/second	Millions of queries/second

目录



HBase概述

HBase应用场景

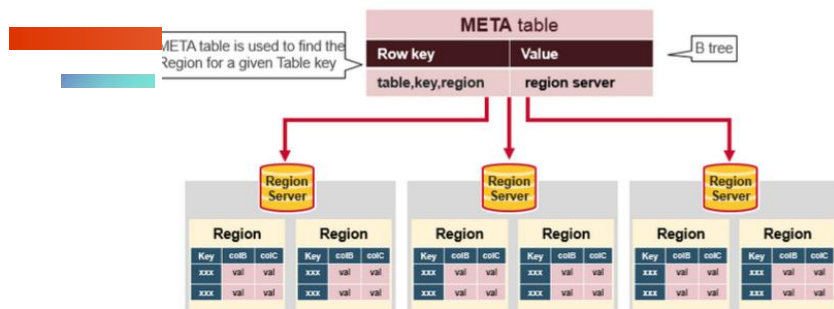
HBase数据模型

HBase物理模型

HBase基础架构

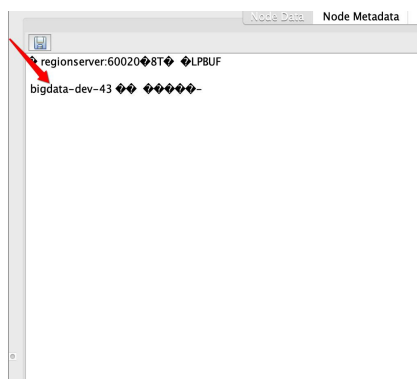
HBase读写流程

总结



HBase读写流程- meta表

- 有一个特殊的Hbase 目录表叫做Meta表，它拥有Region 在集群中的位置信息，ZooKeeper存储着Meta表的位置
- meta表location info以非临时znode的方式注册到zk上，如上图，可知meta region位置为bigdata-dev-43机器。



hive

0,1562590412400
0,1562590410719

```
hbase(main):002> describe 'hbasemeta'
Table hbasemeta is ENABLED
hbasemeta {TABLE_ATTRIBUTES => {IS_META => 'true', REGION_REPLICATION => '1', coprocessor$1 => 'org.apache.hadoop.hbase.coprocessor.MultiRowMutationEndpoint[566f7011]'}
COLUMN FAMILIES DESCRIPTION
{NAME => 'info', BLOOMFILTER => 'NONE', VERSIONS => '10', IN_MEMORY => 'true', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', COMPRESSION => 'NONE', CACHE_DATA_IN_L1 => 'true', ...
IN_VERSIONS => '0', BLOOMCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
1 row(s) in 0.0230 seconds
```

meta表只有一个列簇info，并且包含四列：

1、regioninfo：当前region的startKey与endKey，name等消息

2、seqnumDuringOpen:

3、server: region所在服务器及端口

4、serverstartcode: 服务开始的时候的timestamp

[illegible]

meta表结构

- Rowkey

hash值=MD5Hash.getMD5AsHex(byte(表名,region startKey,创建时间)), meta表的rowkey组成为: 表名,region startKey,创建时间.hash值。如果当前region为table的第一个region时 (第一个region无start key) 时, region startKey=null。

- mete info:

meta表只有一个列簇info, 并且包含四列:

- 1、regioninfo : 当前region的startKey与endKey, name等消息
- 2、seqnumDuringOpen:
- 3、server: region所在服务器及端口
- 4、serverstartcode: 服务开始的时候的timestamp

读数据

流程总览

1. 从zookeeper中获取meta信息，并通过meta信息找到需要查找的table的startkey所在的region信息
2. 和该region所在的regionserver进行rpc交互获取result
3. region server查询memstore（memstore是一个按key排序的树形结构的缓冲区），如果有该rowkey，则直接返回，若没有进入步骤4
4. 查询blockcache，如果有该rowkey，则直接返回，若没有进入步骤5
5. 查询storefile，不管有没有都直接返回

写数据

流程总览

1. zookeeper中获取meta信息，并通过meta信息找到需要查找的table的startkey所在的region信息（和读数据相似）
2. 将put缓存在内存中，参考`BufferedMutatorImpl`，并计算其内存大小（计算方式会考虑java对象占用的大小，参考《java对象在内存的大小》），当超过`hbase.client.write.buffer`默认2097152字节也就是2MB，client会将put 通过rpc交给region server
3. region server接收数据后分别写到HLog和MemStore上一份
4. MemStore达到一个阈值后则把数据刷成一个StoreFile文件。若MemStore中的数据有丢失，则可以从HLog上恢复
5. 当多个StoreFile文件达到一定的数量，会触发Compact和Major Compaction操作，这里不对compaction的细节做展开。
6. 当Compact后，逐步形成越来越大的StoreFile后，会触发Split操作，把当前的StoreFile分成两个，这里相当于把一个大的region分割成两个region，细节也不展开了。



HBase概述

HBase应用场景

HBase数据模型

HBase物理模型

HBase基础架构

HBase读写流程

总结

谢谢

