

# 深入理解大数据-大数据处理与编程实践

## Ch. 6. HBase与Hive程序设计

南京大学计算机科学与技术系

主讲人：黄宜华，顾荣

鸣谢：本课程得到Google（北京）与Intel公司  
中国大学合作部精品课程计划资助

# Ch.6. HBase与Hive程序设计

1. HBase基本工作原理
2. HBase基本操作与编程方法示例
3. Hive基本工作原理
4. Hive基本操作示例
5. HBase与Hive编程实验

# 1. HBase基本工作原理

## 关系数据库的理论局限性

- RDBMS选择ACID (CAP定理中的C), 然后是A
  - 网络分片(Network Partitions)在分布式系统中不可避免
  - 系统扩展时性能和可靠性下降
- Scale up, not out
  - 并行数据库 的扩展性
    - 经验定律: 当集群节点数每增加4~16台, 每个节点的效率下降一半
  - 无法扩展超过~40节点
    - 最新的TPC-C世界纪录: 27 servers
    - 最新的TPC-H世界纪录: 32 servers
- “One size does not fit all”
  - 在所有数据库的主要应用领域, 新的架构轻易地有10x倍性能提升 (数据仓库, 流处理, 科学计算, 非结构化数据处理, OLTP在线事务处理)\*

\* ACM SIGACT News, Volume 33 Issue 2 (2002): “Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services”

\* ICDE 2005: “One Size Fits All”: An Idea Whose Time Has Come and Gone

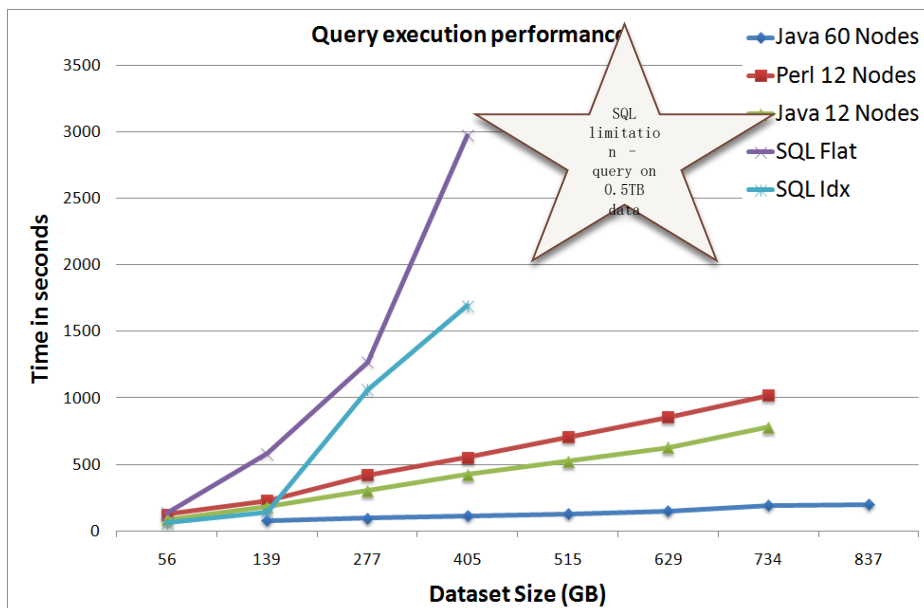
\* VLDB 2007: The End of an Architectural Era (It's Time for a Complete Rewrite)

# 1. HBase基本工作原理

## RDBMS的实现局限性

- RDBMS实现和操作上的局限性 – 不适合新的应用
  - 大表 - 在一张表中存储500GB的数据?
  - 灵活动态可变的表结构 - 为大表修改表结构(Alter Table )?
  - 无停机时间的在线大表分区和动态扩容 - ...

关系数据库 vs. Hadoop/Hive



Intel silicon design environment  
usage analysis, 40M records/day

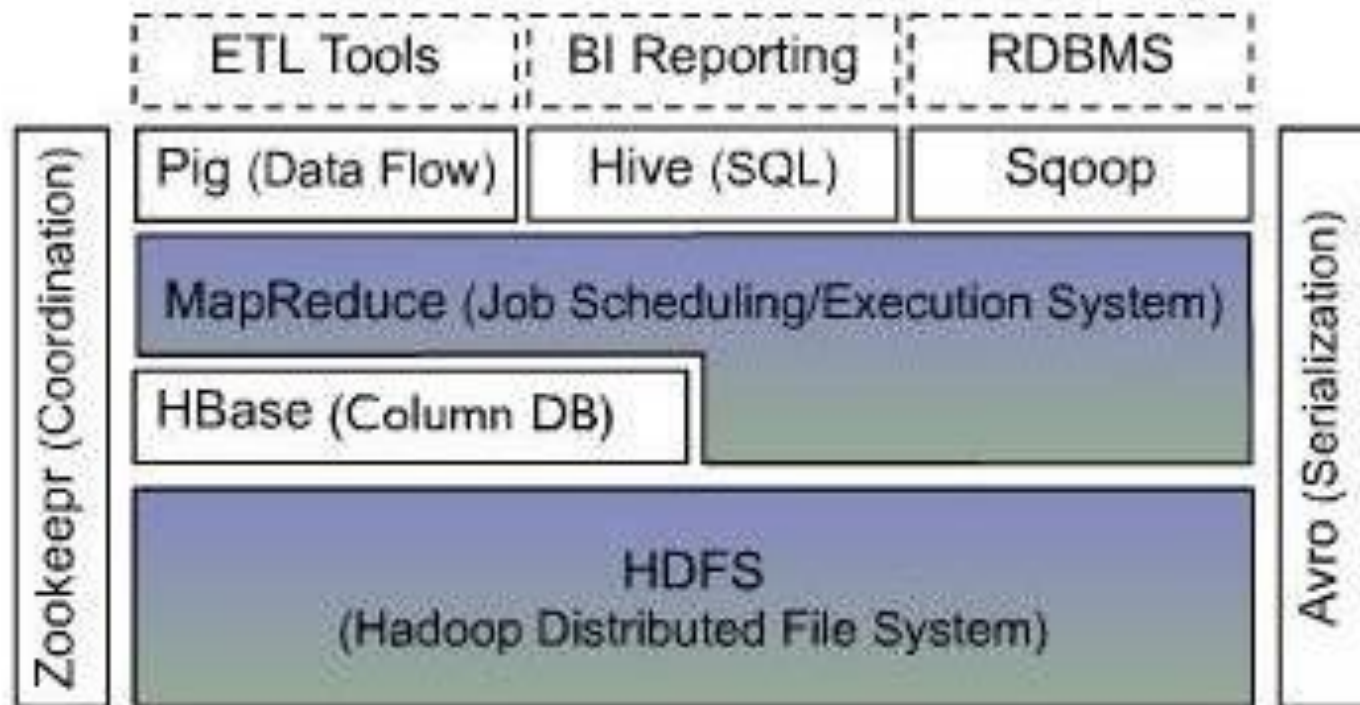
# 1. HBase基本原理

## HBase的设计目标和功能特点

- 针对HDFS缺少结构化半结构化数据存储访问能力的缺陷，提供一个分布式数据管理系统，解决大规模的结构化和半结构化数据存储访问问题
- Google BigTable的一个开源实现
- 提供基于列存储模式的大数据表管理能力
- 可存储管理数十亿以上的数据记录，每个记录可包含百万以上的数据列
- HBase试图提供随机和实时的数据读写访问能力
- 具有高可扩展性、高可用性、容错处理能力、负载平衡能力、以及实时数据查询能力

### HBase在Hadoop中的生态环境

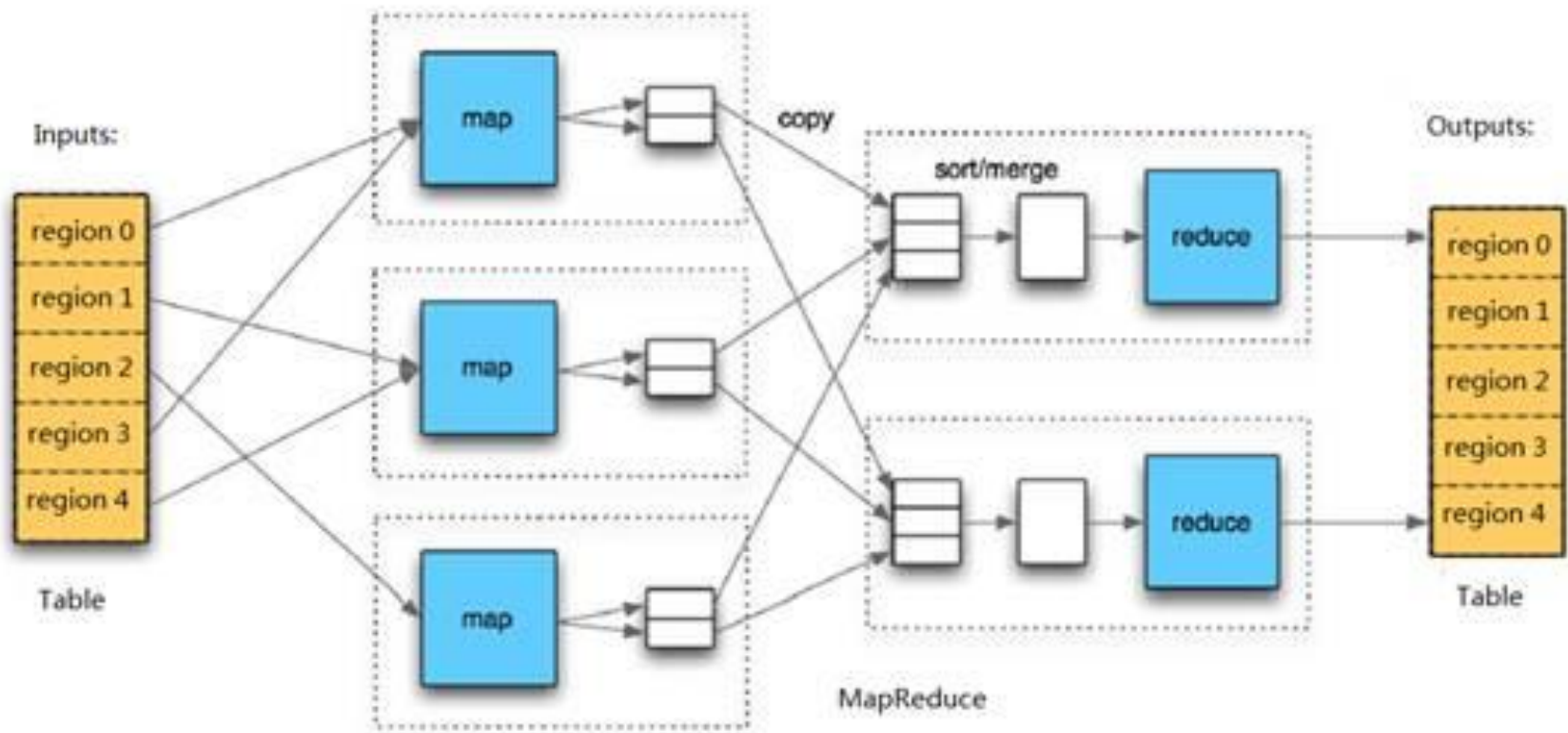
- 构建于分布式文件系统HDFS之上
- 为上层应用提供结构化半结构化海量数据存储访问能力



# HBase基本工作原理

## HBase在Hadoop中的生态环境

- 可与MapReduce协同工作，为MapReduce提供数据输入输出，以完成数据的并行化处理





## HBase数据模型

### 逻辑数据模型

- 数据存储逻辑模型与BigTable类似,但实现上有一些不同之处。
- 是一个分布式多维表, 表中的数据通过:
  - 一个行关键字(row key)
  - 一个列关键字(column key)
  - 一个时间戳(time stamp)进行索引和查询定位的。

行 关 键 字	时 间 戳	列"contents:"	列"anchor:"		列"mime:"
"com.cnn.www"	t9		"anchor:cnnsi.com"	"CNN"	
	t8		"anchor:my.look.ca"	"CNN.com"	
	t6	"<html>..."			"text/html"
	t5	"<html>..."			
	t3	"<html>..."			



## HBase数据模型

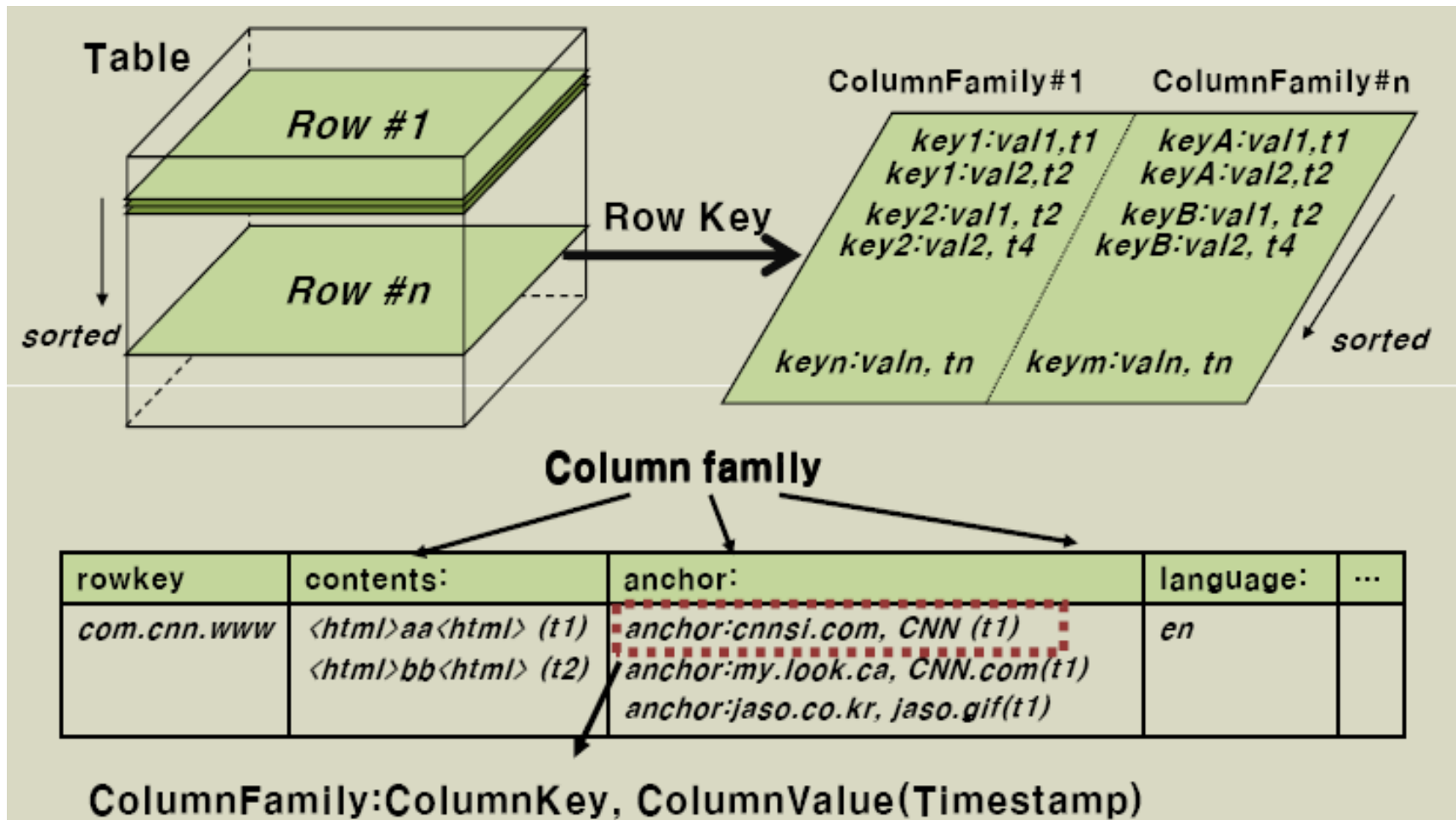
## HBase物理存储格式

行 关 键 字	时 间 戳	列 "contents:"	
"com.cnn.www"	t6	"<html>..."	
	t5	"<html>..."	
	t3	"<html>..."	
行关键字	时 间 戳	列 "anchor:"	
"com.cnn.www"	t9	"anchor:cnnsi.com"	"CNN"
	t8	"anchor:my.look.ca"	"CNN.com"
行关键字	时 间 戳	列 "mime:"	
"com.cnn.www"	t6	"text/html"	

- 按照列存储的稀疏行/列矩阵。物理存储格式上按逻辑模型中的行进行分割，并按照列族存储。
- 值为空的列不予存储，节省存储空间。

## HBase数据模型

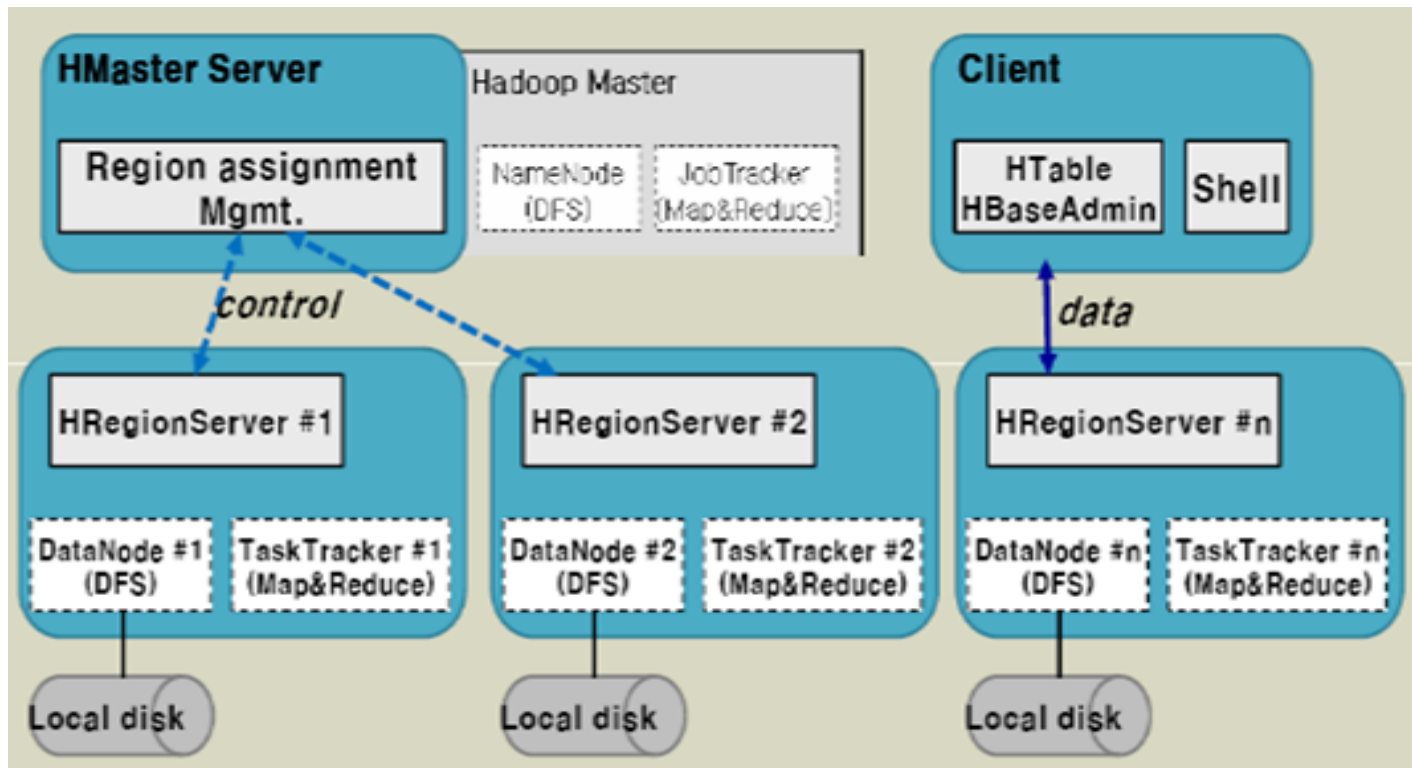
## HBase物理存储格式



# HBase基本工作原理

## HBase的基本构架

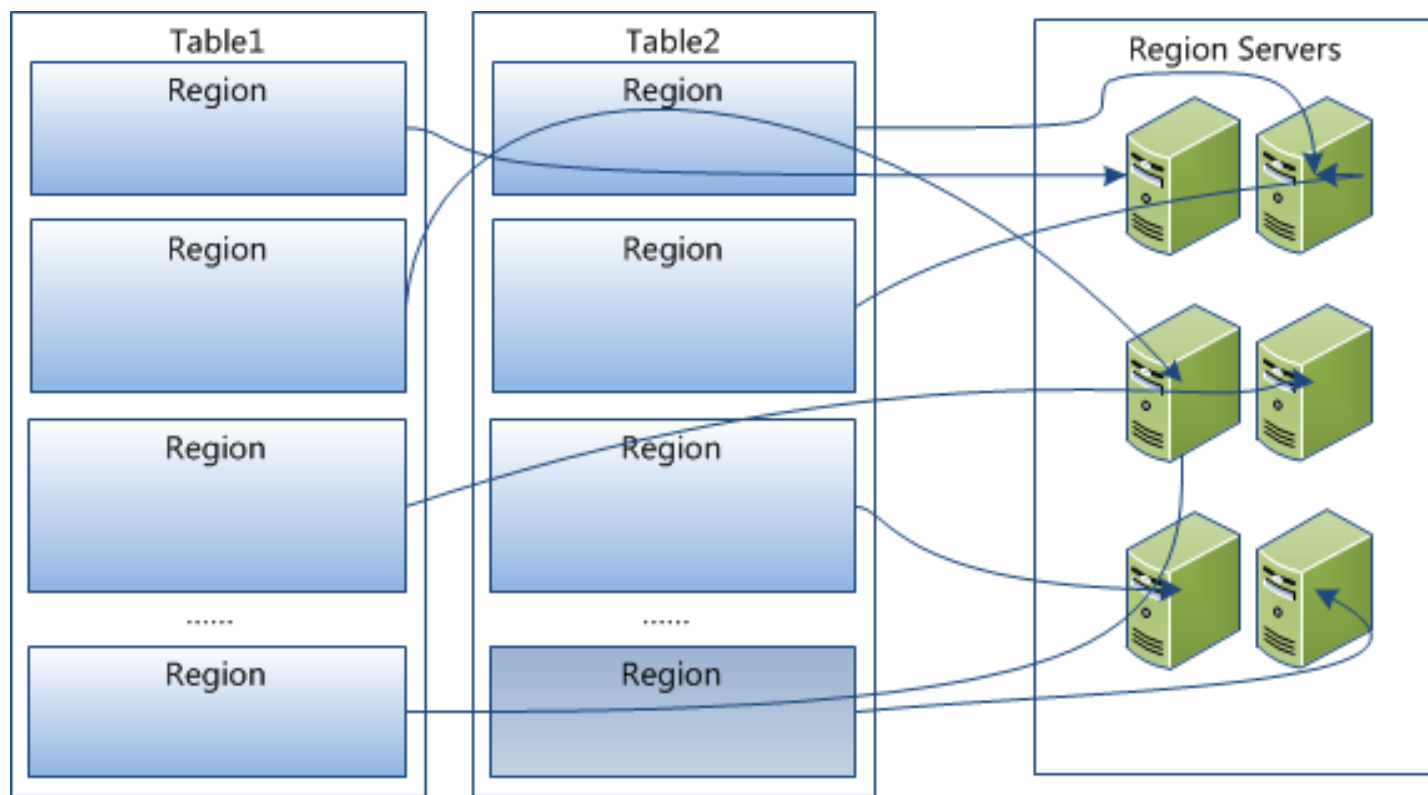
- 由一个MasterServer和由一组子表数据区服务器RegionServer构成，分别存储逻辑大表中的部分数据
- 大表中的底层数据存于HDFS中



## HBase数据存储管理方法

### HBase子表数据存储与子表服务器

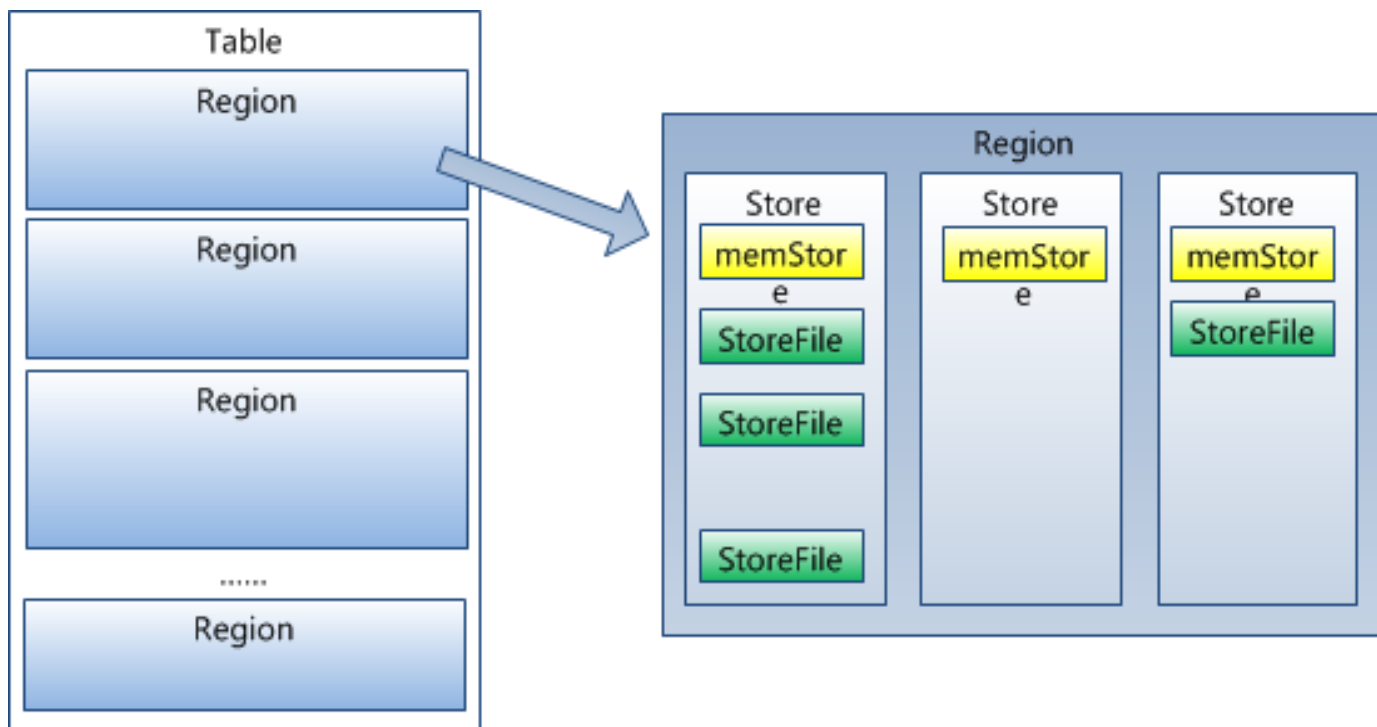
- 与BigTable类似，大表被分为很多个子表（Region），每个子表存储在一个子表服务器RegionServer上



## HBase数据存储管理方法

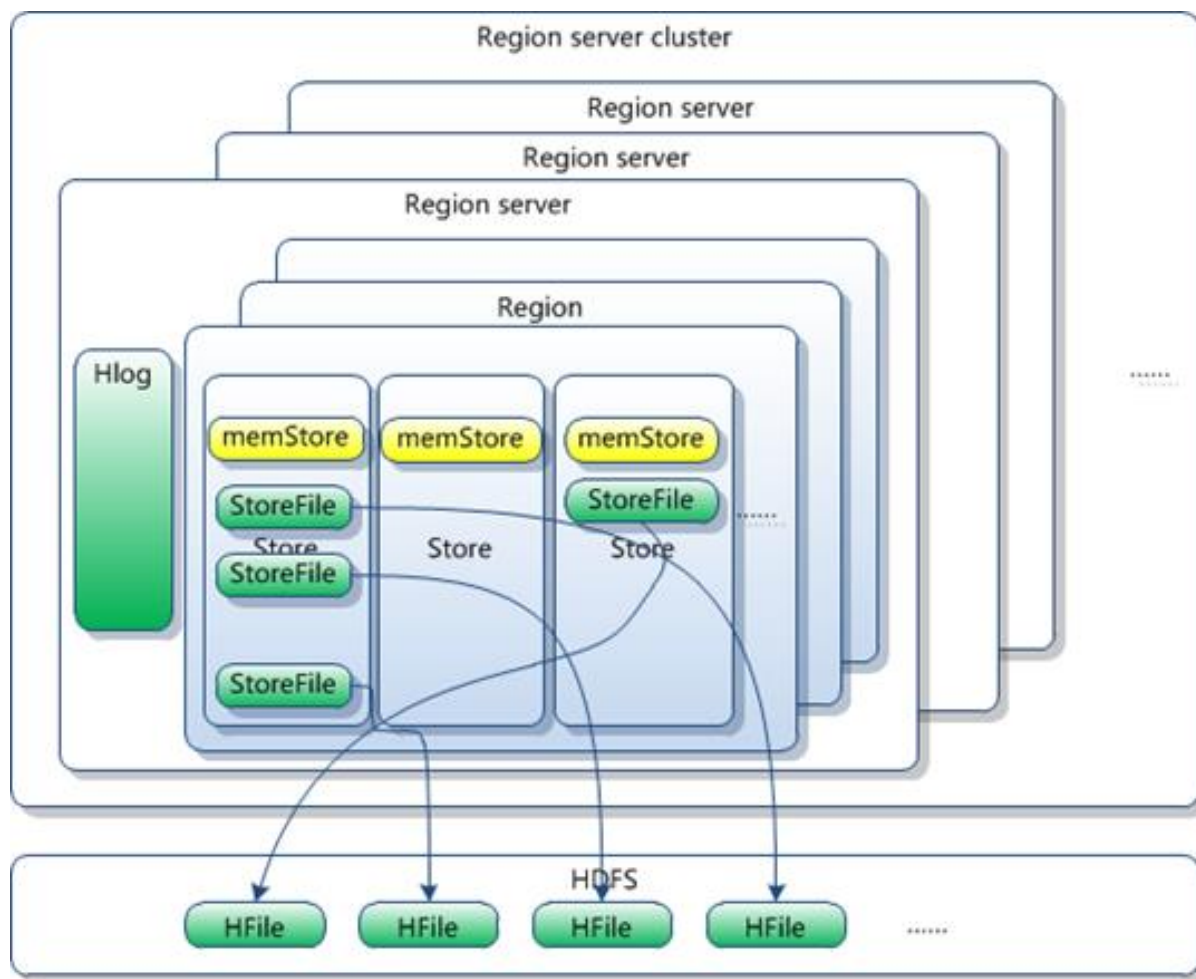
### HBase子表数据存储与子表服务器

- 每个子表中的数据区Region由很多个数据存储块Store构成
- 而每个Store数据块又由存放在内存中的memStore和存放在文件中的StoreFile构成



## HBase数据存储管理方法

## HBase子表数据存储与子表服务器



### HBase数据存储管理方法

#### HBase数据的访问

- 当客户端需要进行数据更新时，先查到子表服务器,然后向子表提交数据更新请求。提交的数据并不直接存储到磁盘上的数据文件中，而是添加到一个基于内存的子表数据对象memStore中，当memStore中的数据达到一定大小时，系统将自动将数据写入到文件数据块StoreFile中。
- 每个文件数据块StoreFile最后都写入到底层基于HDFS的文件中



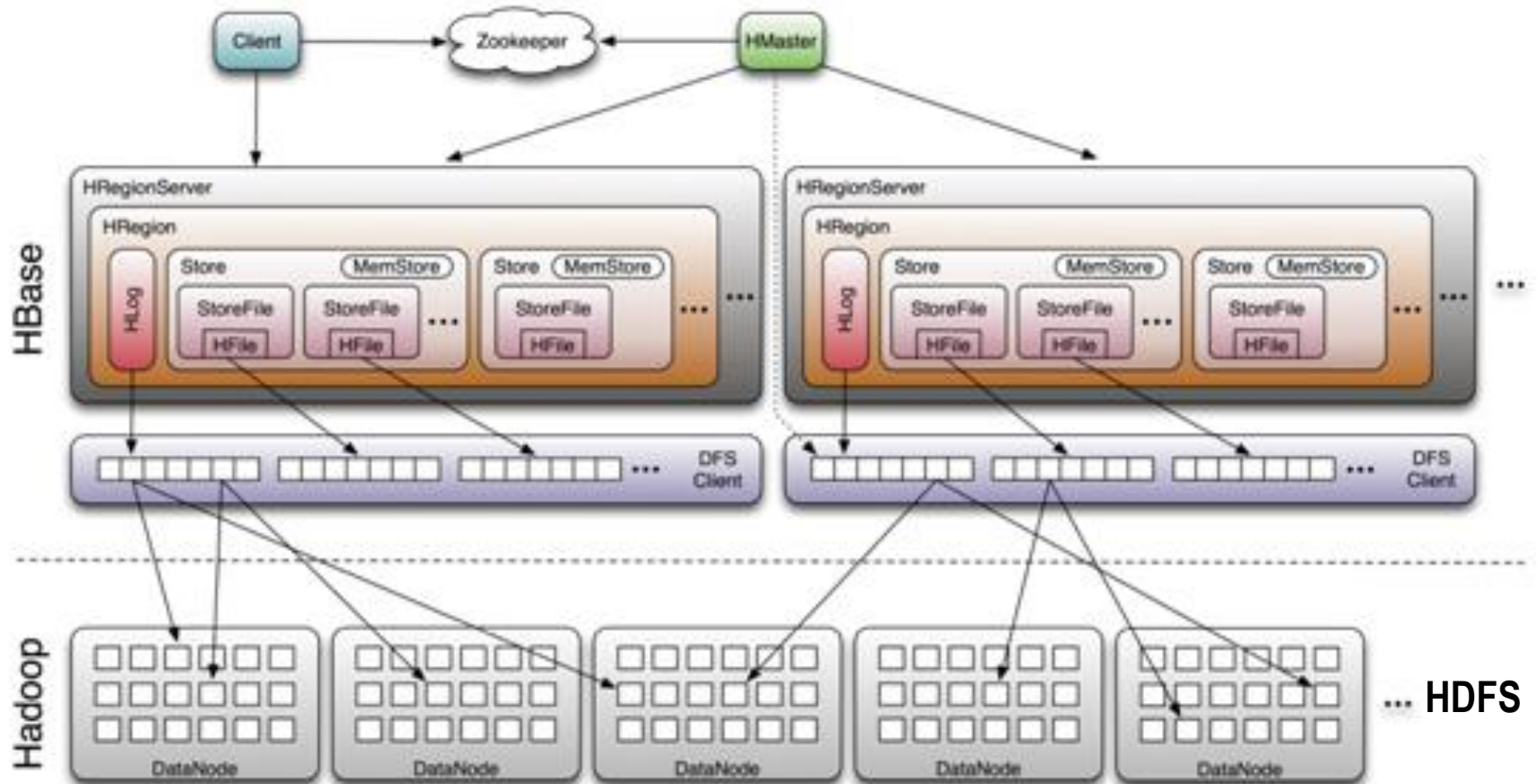
### HBase数据存储管理方法

#### HBase数据的访问

- 需要查询数据时，子表先查memStore。如果没有，则再查磁盘上的StoreFile。每个StoreFile都有类似B树的结构，允许进行快速的数据查询。StoreFile将定时压缩，多个压缩为一个
- 两个小的子表可以进行合并
- 子表大到超过某个指定值时，子表服务器就需要调用HRegion.closeAndSplit(),把它分割为两个新的子表。

## HBase数据存储管理方法

## HBase子表服务器与主服务器



### HBase数据存储管理方法

#### HBase主服务器HServer

- 与BigTable类似，HBase使用主服务器HServer来管理所有子表服务器。主服务器维护所有子表服务器在任何时刻的状态
- 当一个新的子表服务器注册时，主服务器让新的子表服务器装载子表
- 若主服务器与子表服务器连接超时，那么子表服务器将自动停止，并重新启动；而主服务器则假定该子表服务器已死机，将其上的数据转移至其它子表服务器，将其上的子表标注为空闲，并在重新启动后另行分配使用。

### HBase数据存储管理方法

#### HBase数据记录的查询定位

- 描述所有子表和子表中数据块的元数据都存放在专门的元数据表中,并存储在特殊的子表中。子表元数据会不断增长,因此会使用多个子表来保存
- 所有元数据子表的元数据都保存在根子表中。主服务器会扫描根子表,从而得到所有的元数据子表位置,再进一步扫描这些元数据子表即可获得所寻找子表的位置。

## HBase数据记录的查询定位

- HBase使用三层类似B+树的结构来保存region位置

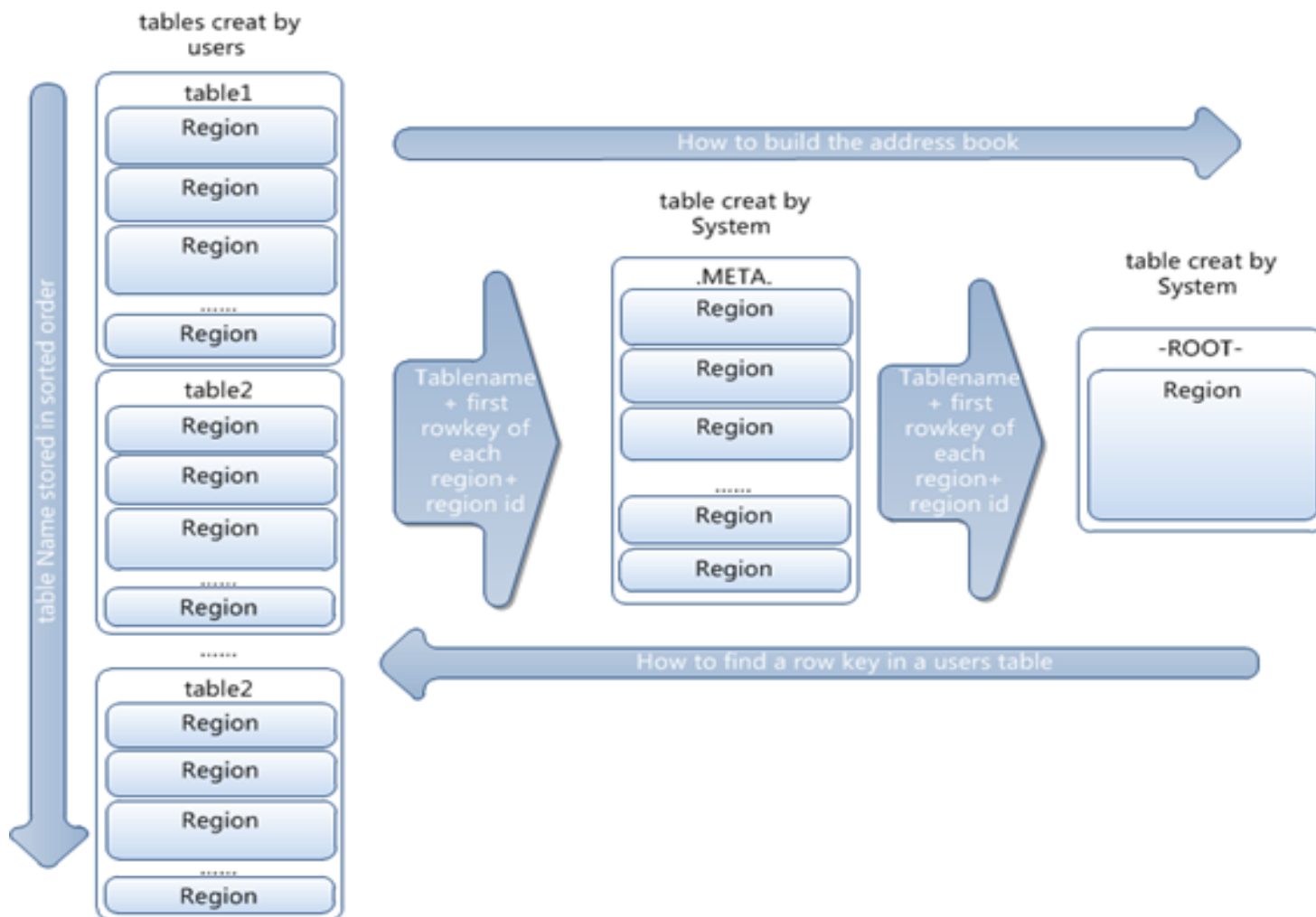
第一层是保存zookeeper里面的文件，它持有root region的位置。

第二层root region是.META.表的第一个region，其中保存了.META.表其它region的位置。通过root region，我们就可以访问.META.表的数据。

.META.是第三层，它是一个特殊的表，保存了HBase中所有数据表的region位置信息。

## HBase数据存储管理方法

## HBase数据记录的查询定位

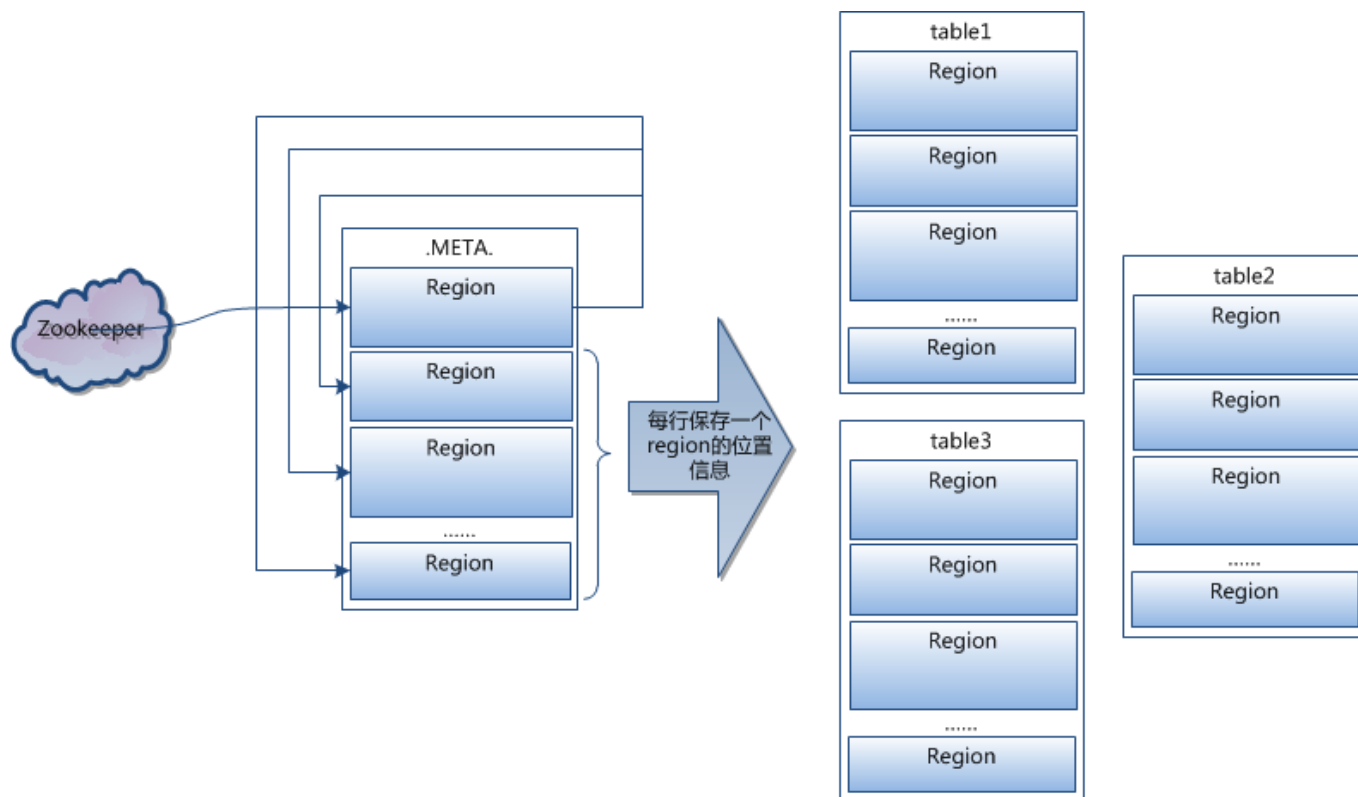


## HBase数据存储管理方法

## HBase数据记录的查询定位

- 元数据子表采用三级索引结构：

根子表→用户表的元数据表→用户表





## 2. HBase基本操作与编程方法

### Hbase shell 操作

- HBase shell常用的操作命令有
  - Create
  - Describe
  - Disable
  - enable,drop,list,scan,put,get,delete,deleteall,count,status等
  - 通过help可以看到详细的用法。

# HBase Shell操作

- 我们建立如下的表以及数据 table:students

ID	Description		Courses			Home
	Name	Height	Chinese	Math	Physics	Province
001	Li Lei	176	80	90	95	Zhe Jiang
002	Han Meimei	183	88	77	66	Bei Jing
003	Xiao Ming	162	90	90	90	Shang Hai

# 创建表格与列举表格

```
hadoop@master: ~  
File Edit View Terminal Help  
hadoop@master:~$ ls  
cleanhadooplogs  hadoop-1.0.3  runhive  TempStatsStore  
cleanhbaselogs   hadoopcode    starthadoop  tutorial  
csapp            hbase-0.94.1  starthbase   updatehadoopconfig  
data            hive-0.9.0    startzookeeper  updatehbaseconfig  
derby.log        metastore_db  stophadoop   Videos  
Desktop          Music         stophbase    zkdata  
Documents        Pictures      stopzookeeper zookeeper-3.4.3  
Downloads        Public        temp         zookeeper.out  
examples.desktop ratings.dat   Templates  
hadoop@master:~$ hbase shell  
HBase Shell; enter 'help<RETURN>' for list of supported commands.  
Type "exit<RETURN>" to leave the HBase Shell  
Version 0.94.1, r1365210, Tue Jul 24 18:40:10 UTC 2012  
  
hbase(main):001:0> list  
TABLE  
scores  
1 row(s) in 3.8490 seconds  
  
hbase(main):002:0> create 'students', 'ID', 'Description', 'Courses', 'Home'  
0 row(s) in 7.8970 seconds  
  
hbase(main):003:0>
```

# 插入数据

```
hadoop@master: ~  
File Edit View Terminal Help  
Documents Pictures stopzookeeper zookeeper-3.4.3  
Downloads Public temp zookeeper.out  
examples.desktop ratings.dat Templates  
hadoop@master:~$ hbase shell  
HBase Shell; enter 'help<RETURN>' for list of supported commands.  
Type "exit<RETURN>" to leave the HBase Shell  
Version 0.94.1, r1365210, Tue Jul 24 18:40:10 UTC 2012  
  
hbase(main):001:0> list  
TABLE  
scores  
1 row(s) in 3.8490 seconds  
  
hbase(main):002:0> create 'students','ID','Description','Courses','Home'  
0 row(s) in 7.8970 seconds  
  
hbase(main):003:0> put 'students','001','Description:Name','Li Lei'  
0 row(s) in 0.4350 seconds  
  
hbase(main):004:0> put 'students','001','Description:Height','176'  
0 row(s) in 0.0280 seconds  
  
hbase(main):005:0> put 'students','001','Course:Chinese','80'
```

# 显示描述表信息

```
hadoop@master: ~  
File Edit View Terminal Help  
SSION => 'NONE', MIN_VERSIONS => '0', TTL => '21474  
83647', KEEP_DELETED_CELLS => 'false', BLOCKSIZE =>  
'65536', IN_MEMORY => 'false', ENCODE_ON_DISK => '  
true', BLOCKCACHE => 'true'}}}  
1 row(s) in 0.1260 seconds  
  
hbase(main):042:0> list  
TABLE  
scores  
students  
2 row(s) in 0.0810 seconds  
  
hbase(main):043:0> describe 'students'  
DESCRIPTION  
ENABLED  
{NAME => 'students', FAMILIES => [{NAME => 'Courses', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'NONE', REPLICATION_SCOPE => '0', VERSIONS => '3', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => '2147483647', KEEP_DELETED_CELLS => 'false', BLOCKSIZE => '65536', IN_MEMORY => 'false', ENCODE_ON_DISK => 'true', BLOCKCACHE => 'true'}, {NAME => 'Description', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'NONE', REPLICATION_SCOPE => '0', VERSIONS => '3', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => '2147483647', KEEP_DELETED_CELLS => 'false', BLOCKSIZE => '65536', IN_MEMORY => 'false', ENCODE_ON_DISK => 'true', BLOCKCACHE => 'true'}, {NAME => 'Home', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'NONE', REPLICATION_SCOPE => '0', VERSIONS => '3', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => '2147483647', KEEP_DELETED_CELLS => 'false', BLOCKSIZE => '65536', IN_MEMORY => 'false', ENCODE_ON_DISK => 'true', BLOCKCACHE => 'true'}, {NAME => 'ID', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'NONE', REPLICATION_SCOPE => '0', VERSIONS => '3', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => '2147483647', KEEP_DELETED_CELLS => 'false', BLOCKSIZE => '65536', IN_MEMORY => 'false', ENCODE_ON_DISK => 'true', BLOCKCACHE => 'true'}}}  
1 row(s) in 0.0780 seconds  
  
hbase(main):044:0> █
```

# 输入数据与扫描数据

```
hadoop@master: ~  
File Edit View Terminal Help  
hbase(main):051:0> put 'students','002','Home:Province','Bei Jing'  
0 row(s) in 0.0420 seconds  
  
hbase(main):052:0> put 'students','003','Home:Province','Shang Hai'  
0 row(s) in 0.0270 seconds  
  
hbase(main):053:0> put 'students','002','Description:Name','Han Meimei'  
0 row(s) in 0.0470 seconds  
  
hbase(main):054:0> put 'students','002','Description:Height','183'  
0 row(s) in 0.0360 seconds  
  
hbase(main):055:0> put 'students','003','Description:Height','162'  
0 row(s) in 0.0240 seconds  
  
hbase(main):056:0> put 'students','003','Description:Name','Xiao Ming'  
0 row(s) in 0.0190 seconds  
  
hbase(main):057:0> scan 'students'  
ROW COLUMN+CELL  
001 column=Courses:Chinese, timestamp=1351776664719, value=80  
001 column=Courses:Math, timestamp=1351776678749, value=90  
001 column=Courses:Physics, timestamp=1351776693609, value=95  
001 column=Description:Height, timestamp=1351776295307, value=176  
001 column=Description:Name, timestamp=1351776277573, value=Li Lei  
001 column=Home:Province, timestamp=1351776717090, value=Zhe Jiang  
002 column=Description:Height, timestamp=1351776794960, value=183  
002 column=Description:Name, timestamp=1351776780963, value=Han Meimei  
002 column=Home:Province, timestamp=1351776742893, value=Bei Jing  
003 column=Description:Height, timestamp=1351776806923, value=162  
003 column=Description:Name, timestamp=1351776822022, value=Xiao Ming  
003 column=Home:Province, timestamp=1351776757867, value=Shang Hai  
3 row(s) in 0.2140 seconds  
  
hbase(main):058:0> █
```

## 限制列进行扫描

```
hbase(main):098:0* scan 'students',{COLUMNS=>'Courses:'}
```

ROW	COLUMN+CELL
001	column=Courses:Chinese, timestamp=1351776664719, value=80
001	column=Courses:Math, timestamp=1351776678749, value=90
001	column=Courses:Physics, timestamp=1351776693609, value=95
002	column=Courses:Chinese, timestamp=1351776979636, value=88
002	column=Courses:Math, timestamp=1351776989455, value=77
002	column=Courses:Physics, timestamp=1351776999603, value=66
003	column=Courses:Chinese, timestamp=1351776946110, value=90
003	column=Courses:Math, timestamp=1351776952592, value=90
003	column=Courses:Physics, timestamp=1351776963651, value=90

3 row(s) in 0.1560 seconds

```
hbase(main):099:0>
```



## HBase中的disable和enable

- disable和enable都是HBase中比较常见的操作，很多对table的修改都需要表在disable的状态下才能进行
- disable 'students'将表students的状态更改为disable的时候，HBase会在zookeeper中的table结点下做记录
- 在zookeeper记录下修改该表的同时，还会将表的region全部下线，region为offline状态
- enable的过程和disable相反，会把表的所有region上线，并删除zookeeper下的标志。如果在enable前，META中有region的server信息，那么此时会在该server上将该region 上线；如果没有server的信息，那么此时还要随机选择一台机器作为该region的server

# HBase的Java编程

- HBase Java编程接口概述HBaseConfiguration

HBaseConfiguration是每一个hbase client都会使用到的对象，它代表的是HBase配置信息。

默认的构造方式会尝试从hbase-default.xml和hbase-site.xml中读取配置。如果classpath没有这两个文件，就需要你自己设置配置。

配置Java代码示例：

```
Configuration HBASE_CONFIG = new Configuration();
HBASE_CONFIG.set( "hbase.zookeeper.quorum", "zkServer" );
HBASE_CONFIG.set( "hbase.zookeeper.property.clientPort",
    "2181" );
HBaseConfiguration cfg = new BaseConfiguration(HBASE_CONFIG);
```

## HBase的Java编程:创建表

- 创建表是通过HBaseAdmin对象来操作的。HBaseAdmin负责表的META信息处理。HBaseAdmin提供了createTable这个方法：  
`public void createTable(HTableDescriptor desc)`

HTableDescriptor 代表的是表的schema

HColumnDescriptor 代表的是column的schema

### Java代码示例：

```
HBaseAdmin hAdmin = new HBaseAdmin(hbaseConfig);
HTableDescriptor t = new HTableDescriptor(tableName);
t.addFamily(new HColumnDescriptor("f1"));
t.addFamily(new HColumnDescriptor("f2"));
t.addFamily(new HColumnDescriptor("f3"));
t.addFamily(new HColumnDescriptor("f4"));
hAdmin.createTable(t);
```

## HBase的Java编程:插入数据

- HTable通过put方法来插入数据，可以传递单个批Put对象或者List put对象来分别实现单条插入和批量插入
- Put对象的常用的方法：

```
public static void addData (String tableName, String rowKey,  
                           String family, String qualifier, String value)  
    throws Exception  
{ try {  
    HTable table = new HTable(conf, tableName);  
    Put put = new Put(Bytes.toBytes(rowKey));  
    put.add(Bytes.toBytes(family), Bytes.toBytes(qualifier),  
            Bytes.toBytes(value));  
    table.put(put);  
    System.out.println("insert recored success!");  
} catch (IOException e) { e.printStackTrace(); }  
}
```

## HBase的Java编程:插入数据

- HTable通过put方法来插入数据，可以传递单个批Put对象或者List put对象来分别实现单条插入和批量插入
  - **setTimeStamp**:指定所有cell默认的timestamp,如果一个Cell没有指定timestamp,就会用到这个值。如果没有调用，HBase会将当前时间作为未指定timestamp的cell的timestamp.
  - **setWriteToWAL**: WAL是Write Ahead Log的缩写，指的是HBase在插入操作前是否写Log。默认是打开，关掉会提高性能，但是如果系统出现故障(负责插入的Region Server挂掉)，数据可能会丢失。
  - 另外HTable也有两个方法也会影响插入的性能
  - **setAutoFlush**: AutoFlush指的是在每次调用HBase的Put操作时，是否提交到HBase Server。默认是true,每次会提交。如果此时是单条插入，就会有更多的I/O,从而降低性能。
  - **setWriteBufferSize**: Write Buffer Size在AutoFlush为false的时候起作用，默认是2MB,也就是当插入数据超过2MB,就会自动提交到Server

## HBase的Java编程:删除表

- 删除表也通过HBaseAdmin来操作，删除表之前首先要disable表。这是一个非常耗时的操作，所以不建议频繁删除表。  
disableTable和deleteTable分别用来disable和delete表

### Java代码示例:

```
HBaseAdmin hAdmin = new HBaseAdmin(hbaseConfig);  
if (hAdmin.tableExists(tableName)) {  
    hAdmin.disableTable(tableName);  
    hAdmin.deleteTable(tableName);  
}
```

## HBase的Java编程:查询数据

- 查询分为单条随机查询和批量查询
  - 单条查询是通过rowkey在table中查询某一行的数据。HTable提供了get方法来完成单条查询。
  - 批量查询是通过制定一段rowkey的范围来查询。HTable提供了个getScanner方法来完成批量查询。

### Java代码示例:

```
Scan s = new Scan();
s.setMaxVersions();
ResultScanner ss = table.getScanner(s);
for(Result r:ss){
    System.out.println(new String(r.getRow()));
    for(KeyValue kv:r.raw()){
        System.out.println(new String(kv.getColumn()));
    }
}
```



## HBase的Java编程:删除数据

- HTable 通过delete方法来删除数据: delete(final Delete delete)

Delete常用方法:

deleteFamily或删除Columns:

指定要删除的family或者column的数据。如果不调用任何这样的方法, 将会删除整行

```
HTable table = new HTable(hbaseConfig, "mytest");
```

```
Delete d = new Delete("row1".getBytes());
```

```
table.delete(d)
```

## HBase的Java编程:切分表

- 参数hbase.hregion.max.filesize指示在当前ReigonServer上单个Reigon的最大存储空间，单个Region超过该值时，这个Region会被自动split成更小的region。

HBaseAdmin提供split方法来将table 进行手工split.

```
public void split(final String tableNameOrRegionName)
```

如果提供tableName，那么会将table所有region进行split ;如果提供region Name，那么只会split这个region.

由于split是一个异步操作，并不能确切地控制region的个数。

## HBase的Java编程:程序的编写与运行

- 与Hadoop编程类似，在进行HBase Java编程的时候也需要配置相关的类包文件jar文件，在程序执行的时候，还需要给出类库的目录

在开发环境中准备导入jar包：

将HBase目录中的jar包以及lib中的所有jar包都加入

# HBase的Java编程:程序的编写与运行

与编写其它Java代码类似，可以通过代码编写参考例子代码进行代码分析，例子代码演示了数据库的各项操作，包括创建表，插入记录，删除记录，查询等

```
import java.io.IOException;

public class HBaseSample {

    private static Configuration conf = null;

    static {
        conf = HBaseConfiguration.create();
    }

    public static void creatTable(String tableName, String[] familys) throws Exception {
        HBaseAdmin admin = new HBaseAdmin(conf);
        if (admin.tableExists(tableName)) {
            System.out.println("table already exists!");
        } else {
            HTableDescriptor tableDesc = new HTableDescriptor(tableName);
            for(int i=0; i<familys.length; i++){
                tableDesc.addFamily(new HColumnDescriptor(familys[i]));
            }
            admin.createTable(tableDesc);
            System.out.println("create table " + tableName + " ok.");
        }
    }
}
```

### 3. Hive基本工作原理

#### 使用Hadoop进行数据分析

- 通过前面的课程知道，很多分析任务可以通过Hadoop集群进行，即可以通过Hadoop集群将任务分布到数百甚至上千个节点中进行分析，通过并行执行降低分析的时间
- 如果原始数据使用数据库形式的话，则需要数据进行转换，将数据存储到分布式文件系统HDFS中，然后通过MapReduce程序进行分析
- Hadoop通过MapReduce的并行化方式进行并行处理，能够充分利用数目庞大的服务器节点
- 但是，MapReduce是一个底层的编程接口，对于数据分析人员来说，这个编程接口并不是十分友好，还需要进行大量的编程以及调试工作



## Hive简介

- Hive可以被认为是一种数据仓库，包括数据的存储以及查询
- Hive包括一个高层语言的执行引擎，类似于SQL的执行引擎
- Hive建立在Hadoop的其它组成部分之上，包括Hive依赖于HDFS进行数据保存，依赖于MapReduce完成查询操作
- Hive最初的开发由Facebook推动，在Facebook内部每天会搜集大量的数据，并需要在这些数据上进行大量分析
- 最初的分析是通过手工的python脚本形式进行
- 数据分析的数量十分巨大，在2006年每天需要分析数十个10 GB左右的数据，在2007年增长到大约TB的量级，现在数据分析的数量可能是这个数量的10倍

## Hive的应用范围举例

- 日志分析：在互联网公司中，每天会产生大量的日志数据，分析这些日志数据是每天都需要进行的重要工作；日志分析可以优化系统，可以获知用户行为，也可以获知数据的统计信息
- 数据挖掘：通过对结构化数据的挖掘，能够获得原先使用者没有意识到的信息
- 文档索引：可以对一系列文档进行分析，并形成文档的索引结构，不一定是完整的倒排表，可能是关联信息的索引
- 商业智能信息处理：可以对商业信息进行查询分析，从中可以获得一些智能决策的信息
- 即时查询以及数据验证：数据分析人员可能临时需要验证数据的特性，需要查询引擎迅速进行数据统计分析

## Hive的组成模块（1）

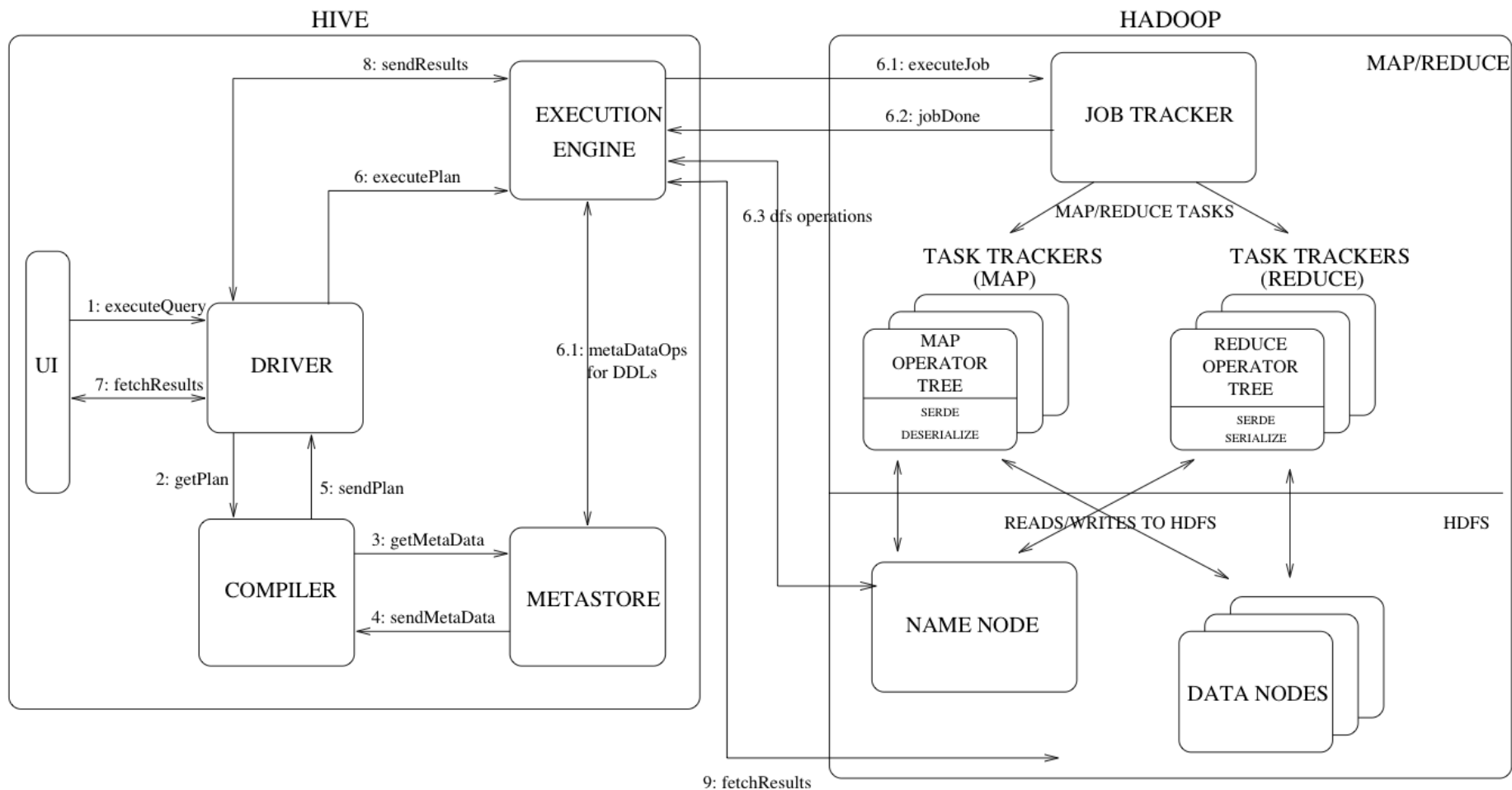
- Hive的模块非常类似于传统的数据库的模块，下面是Hive的必要组成模块以及对应的功能介绍
- HiveQL：这是Hive的数据查询语言，与SQL非常类似。Hive提供了这个数据查询语言与用户的接口，包括一个shell的接口，可以进行用户的交互以及网络接口与JDBC接口。JDBC接口可以用于编程，与传统的数据库编程类似，使得程序可以直接使用Hive功能而无需更改
- Driver: 执行驱动程序，用以将各个组成部分形成一个有机的执行系统，包括会话的处理，查询获取以及执行驱动



## Hive的组成模块（2）

- Compiler: Hive需要一个编译器，将HiveQL语言编译成中间表示，包括对于HiveQL语言的分析，执行计划的生成以及优化等工作
- Execution Engine: 执行引擎，在Driver的驱动下，具体完成执行操作，包括MapReduce执行，或者HDFS操作，或者元数据操作
- Metastore: 用以存储元数据：存储操作的数据对象的格式信息，在HDFS中的存储位置的信息以及其他的用于数据转换的信息SerDe等

# Hive的系统结构



## Hive的数据模型

- 每一个类似于数据库的系统都首先需要定义一个数据模型，然后才是在这个数据模型之上的各种操作
- Tables: Hive的数据模型由数据表组成  
数据表中的列是有类型的 (int, float, string, data, boolean)  
也可以是复合的类型, 如list: map (类似于JSON形式的数据)
- Partitions: 数据表可以按照一定的规则进行划分Partition  
例如, 通过日期的方式将数据表进行划分
- Buckets: 数据存储的桶  
在一定范围内的数据按照Hash的方式进行划分 (这对于数据的抽样以及对于join的优化很有意义)

## 元数据存储：Metastore

- 在Hive中由一系列的数据表格组成一个命名空间，关于这个命名空间的描述信息会保存在Metastore的空间中
- 元数据使用SQL的形式存储在传统的关系数据库中，因此可以使用任意一种关系数据库，例如Derby(apache的关系数据库实现)，MySQL以及其他的多种关系数据库存储方法
- 在数据库中，保存最重要的信息是有关数据库中的数据表格描述，包括每一个表的格式定义，列的类型，物理的分布情况，数据划分情况等

## 数据的物理分布情况

- Hive在HDFS中有固定的位置，通常被放置在HDFS的如下目录中：  
[/home/hive/warehouse](#)
- 每个数据表被存放在warehouse的子目录中  
进一步来说，数据划分Partition、数据桶Buckets形成了数据表的子目录
- 数据可能以任意一种形式存储，例如：
  - 使用分隔符的文本文件，或者是SequenceFile
  - 使用用户自定义的SerDe，则可以定义任意格式的文件

## Hive系统的配置

- 要想Hive使用HDFS进行数据仓库的存储，使用MapReduce进行HQL语言的执行，需要进行相应的配置。首先，需要创建Hive的配置文件hive-site.xml(注意，Hive安装包中包含一个配置文件模板—hive-default.xml.template)
- fs.default.name用以告知Hive对应的HDFS文件系统的名字节点在什么位置
- mapred.job.tracker用以告知Hive对应的MapReduce处理系统的任务管理器在什么位置
- 通过上面两点的配置，就可以让Hive系统与对应的HDFS以及MapReduce进行交互，完成数据的存取以及查询的操作

## 4. Hive基本操作示例

### 启动Hive的命令行界面shell

- 完成Hive系统的合适配置之后，打开任意一个命令行界面，执行下面的命令就可以启动hive的界面
- `$cd /hive/install/directory` 进入hive的安装目录
- `$bin/hive` 如果已经将hive加入到执行路径，则可以直接执行hive即可
- 执行成功的话，我们就可以看到hive的主界面  
hive>
- 等待用户输入查询命令

## 创建数据表的命令

- 显示所有的数据表

```
hive> show tables;
```

- 创建一个表，这个表包括两列，分别是整数类型以及字符串类型，使用文本文件表达，数据域之间的分隔符为\t

```
hive> create table shakespeare (freq int, word string) row format  
delimited fields terminated by '\t' stored as textfile;
```

- 显示所创建的数据表的描述，即创建时候对于数据表的定义

```
hive> describe shakespeare;
```



## 装入数据

- 数据装入到Hive中

```
hive> load data inpath "shakespeare_freq" into table shakespeare;
```

## SELECTS and FILTERS

- `hive> select * from shakespeare limit 10;`
- `hive> select * from shakespeare  
          where freq > 100 sort by freq asc limit 10;`
- `hive> SELECT a.foo FROM invites a WHERE a.ds='2008-08-15';`
- `hive> INSERT OVERWRITE DIRECTORY '/tmp/hdfs_out'  
          SELECT a.* FROM invites a WHERE a.ds='2008-08-15';`
- `hive> INSERT OVERWRITE LOCAL DIRECTORY '/tmp/local_out'  
          SELECT a.* FROM pokes a;`

## Group By

- hive> INSERT OVERWRITE TABLE events  
SELECT a.bar, count(\*)  
FROM invites a  
WHERE a.foo > 0 GROUP BY a.bar;

## Join

- hive> SELECT t1.bar, t1.foo, t2.foo  
FROM pokes t1 JOIN invites t2 ON t1.bar = t2.bar;

## Hive总结

- Hive提供了一种类似于SQL的查询语言，使得其能够用于用户的交互查询
- 与传统的数据库类似，Hive提供了多个数据表之间的联合查询，能够完成高效的多个数据表之间的查询
- 通过底层执行引擎的工作，Hive将SQL语言扩展到很大的查询规模

# 实验4：HBase和Hive的安装与使用

## 实验内容与要求

1. 在自己本地电脑上正确安装和运行HBase和Hive。
2. 在HBase中创建一张表Wuxia，用于保存下一步的输出结果。
3. 修改第3次实验的MapReduce程序，在Reduce阶段将倒排索引的信息通过文件输出，而每个词语及其对应的“平均出现次数”信息写入到HBase的表“Wuxia”中。
4. 编写Java程序，遍历上一步中保存在HBase中的表，并把表格的内容（词语以及平均出现次数）保存到本地文件中。
5. Hive安装完成后，在Hive Shell命令行操作创建表（表名：Wuxia(word STRING, count DOUBLE))、导入平均出现次数的数据、查询（出现次数大于300的词语）和前100个出现次数最多的词。
6. 实验结果提交：要求书写一个实验报告，其中包括：
  1. 实验输出结果的屏幕拷贝、相关操作步骤的屏幕拷贝
  2. 实验体会
  3. 实验报告文件命名规则：MPLab4-组号-组长姓名.doc
  4. 实验完成时间：14天, 11月10日前完成并提交报告

**Thanks !**

# Backups

# 行主键

- 行主键row key是用来检索记录的主键。这样的话，访问HBase table中的行，有三种方式：1 通过单个row key访问；2 通过row key的范围range来访问；3 全表扫描
- 行键 (Row key)可以是任意字符串(最大长度是 64KB，实际应用中长度一般为 10-100bytes)，在HBase内部，row key保存为字节数组。
- 存储时，数据按照Row key的字典序(byte order)排序存储。这样的话，设计key时，要充分排序存储这个特性，将经常一起读取的行存储放到一起。(空间局部性)
- 字典序对int排序的结果是  
1, 10, 100, 11, 12, 13, 14, 15, 16, 17, 18, 19, 2, 20, 21, ..., 9, 91, 92, 93, 94, 95, 96, 97, 98, 99。要保持整形的自然序，行键必须用0作左填充。
- 行的一次读写是原子操作（不论一次读写多少列）。这样的设计兼顾了用户可以理解在一行中的读写行为以及设计上的可扩展性



# 列族

- 列族的设计与传统数据库中的列不一致
- 与BigTable中的模式一样，hbase表中的每个列，都归属与某个列族。列族是表的schema的一部分，必须在使用表之前定义。列名都以列族作为前缀。例如 *courses:history*, *courses:math* 都属于 *courses* 这个列族。
- 访问控制、磁盘和内存的使用统计都是在列族层面进行的。实际应用中，列族上的控制权限能帮助管理不同类型的应用：允许一些应用可以添加新的基本数据、一些应用可以读取基本数据并创建继承的列族、一些应用则只允许浏览数据（甚至可能因为隐私的原因不能浏览所有数据）

# 时间戳timestamp

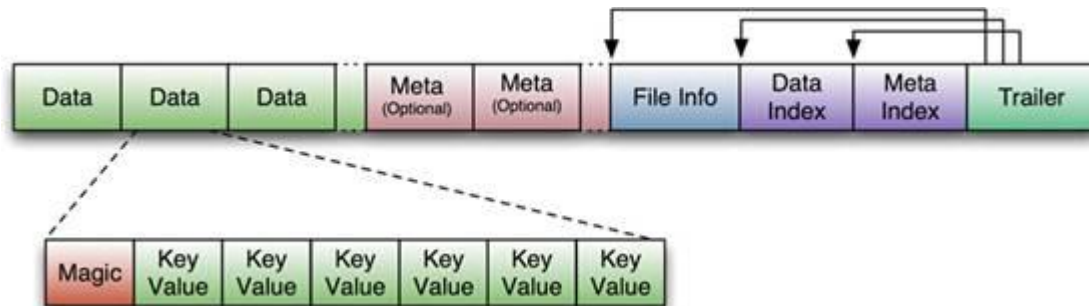
- HBase中通过row和columns确定的为一个存储单元称为单元格cell。每个 cell都保存着同一份数据的多个版本。版本通过时间戳来索引。时间戳的类型是 64位整型。时间戳可以由HBase(在数据写入时自动 )赋值，此时时间戳是精确到毫秒的当前系统时间。时间戳也可以由客户显式赋值。如果应用程序要避免数据版本冲突，就必须自己生成具有唯一性的时间戳。每个cell中，不同版本的数据按照时间倒序排序，即最新的数据排在最前面。
- 为了避免数据存在过多版本造成的管理（包括存储和索引）负担，HBase提供了两种数据版本回收方式。一是保存数据的最后n个版本，二是保存最近一段时间内的版本（比如最近七天）。用户可以针对每个列族进行设置。

# HBase存储格式

- HBase中的所有数据文件都存储在Hadoop HDFS文件系统上，主要包括上述提出的两种文件类型：
- 1. HFile， HBase中KeyValue数据的存储格式，HFile是Hadoop的二进制格式文件，实际上StoreFile就是对HFile做了轻量级包装，即StoreFile底层就是HFile
- 2. HLogFile， HBase中WAL（Write Ahead Log） 的存储格式，物理上是Hadoop的Sequence File

# HFile

- 首先HFile文件是不定长的，长度固定的只有其中的两块：Trailer和FileInfo。Trailer中有指针指向其他数据块的起始点。File Info中记录了文件的一些Meta信息，例如：AVG\_KEY\_LEN, AVG\_VALUE\_LEN, LAST\_KEY, COMPARATOR, MAX\_SEQ\_ID\_KEY等。Data Index和Meta Index块记录了每个Data块和Meta块的起始点。
- Data Block是HBase I/O的基本单元，为了提高效率，HRegionServer中有基于LRU的Block Cache机制。每个Data块的大小可以在创建一个Table的时候通过参数指定，大号的Block有利于顺序Scan，小号Block利于随机查询。每个Data块除了开头的Magic以外就是一个个KeyValue对拼接而成，Magic内容就是一些随机数字，目的是防止数据损坏。后面会详细介绍每个KeyValue对的内部构造。

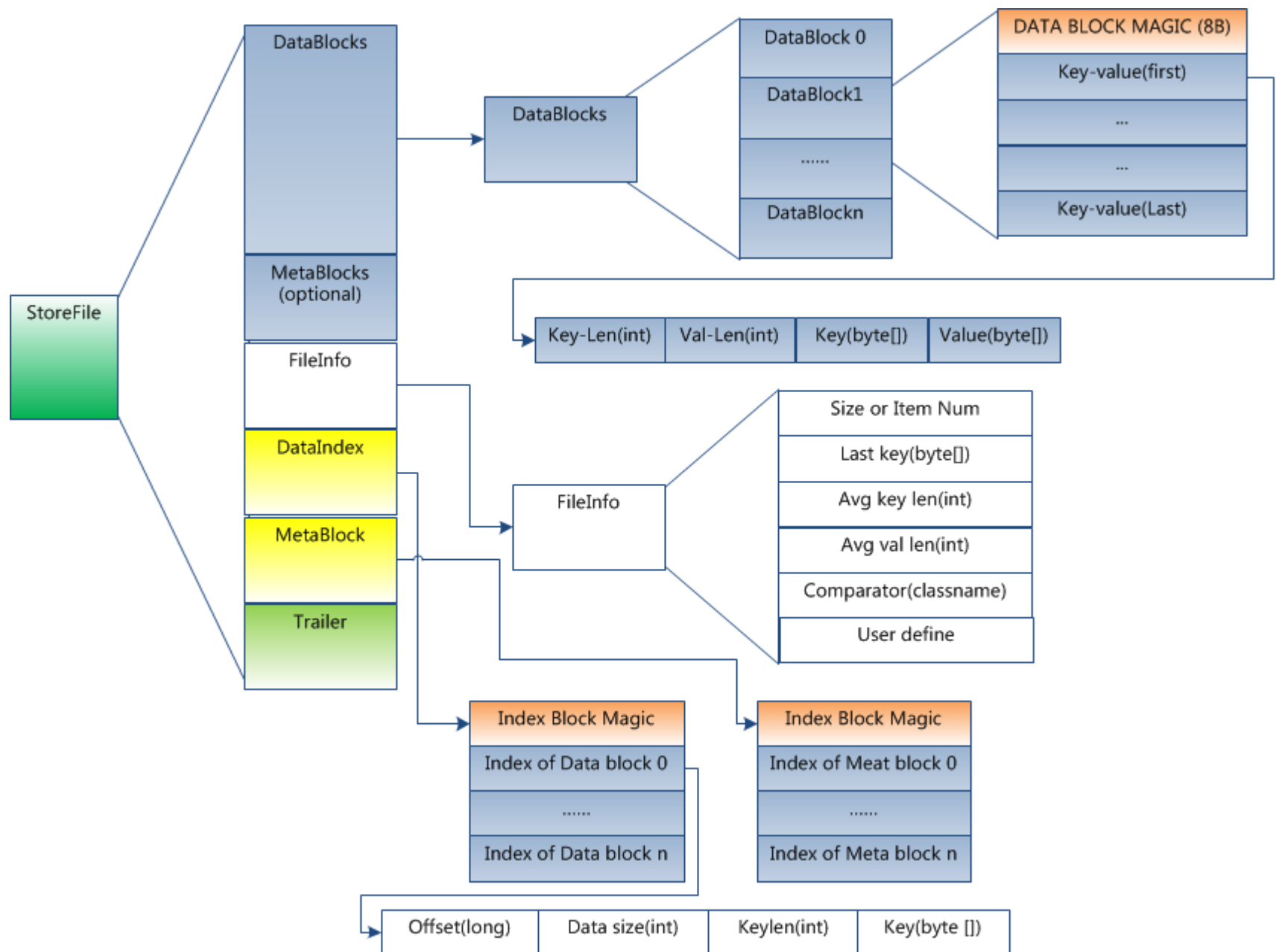


# HFile文件的格式描述 (1)

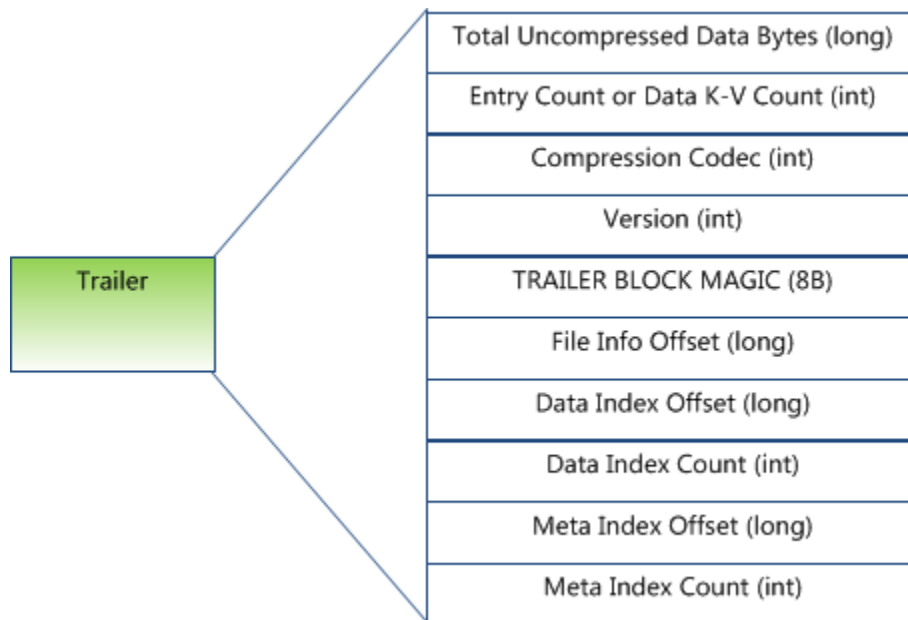
- Data Block 段 - 保存表中的数据，这部分可以被压缩
- Meta Block 段 (可选的) - 保存用户自定义的kv对，可以被压缩。
- File Info 段 - HFile的元信息，不被压缩，用户也可以在这一部分添加自己的元信息。
- Data Block Index 段 - Data Block的索引。每条索引的key是被索引的block的第一条记录的key。
- Meta Block Index段 (可选的) - Meta Block的索引。
- Trailer - 这一段是定长的。保存了每一段的偏移量，读取一个HFile时，会首先读取Trailer，Trailer保存了每个段的起始位置(段的Magic Number用来做安全check)，然后，DataBlock Index会被读取到内存中，这样，当检索某个key时，不需要扫描整个HFile，而只需从内存中找到key所在的block，通过一次磁盘io将整个block读取到内存中，再找到需要的key。DataBlock Index采用LRU机制淘汰。

## HFile文件的格式 (2)

- HFile的Data Block, Meta Block通常采用压缩方式存储, 压缩之后可以大大减少网络IO和磁盘IO, 随之而来的开销当然是需要花费cpu进行压缩和解压缩。目标Hfile的压缩支持两种方式: Gzip, Lzo。
- HLog(WAL log): WAL 意为Write ahead log, 用来做灾难恢复只用, Hlog记录数据的所有变更, 一旦数据修改, 就可以从log中进行恢复。
- 每个Region Server维护一个Hlog, 而不是每个Region一个。这样不同region(来自不同table)的日志会混在一起, 这样做的目的是不断追加单个文件相对于同时写多个文件而言, 可以减少磁盘寻址次数, 因此可以提高对table的写性能。带来的麻烦是, 如果一台region server下线, 为了恢复其上的region, 需要将region server上的log进行拆分, 然后分发到其它region server上进行恢复。
- HLog文件就是一个普通的Hadoop Sequence File, Sequence File 的Key是HLogKey对象, HLogKey中记录了写入数据的归属信息, 除了table和region名字外, 同时还包括 sequence number和timestamp, timestamp是“写入时间”, sequence number的起始值为0, 或者是最近一次存入文件系统中sequence number。HLog Sequence File的Value是HBase的KeyValue对象, 即对应HFile中的KeyValue,



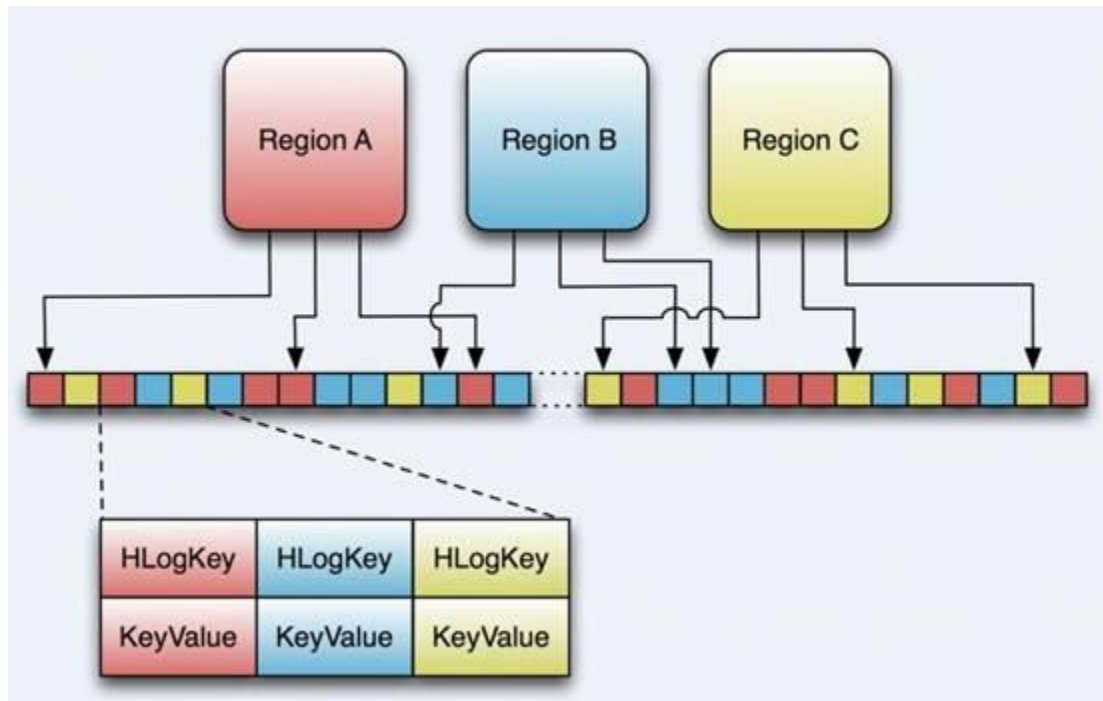
# Trailer部分的格式



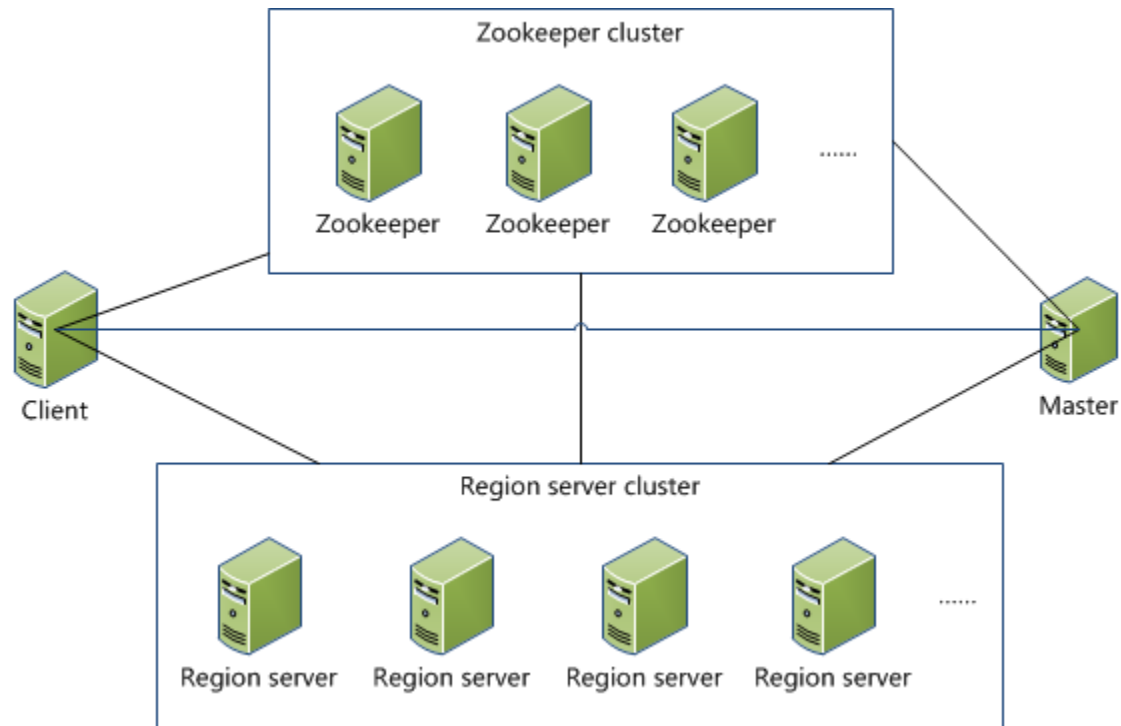


# HLogFile

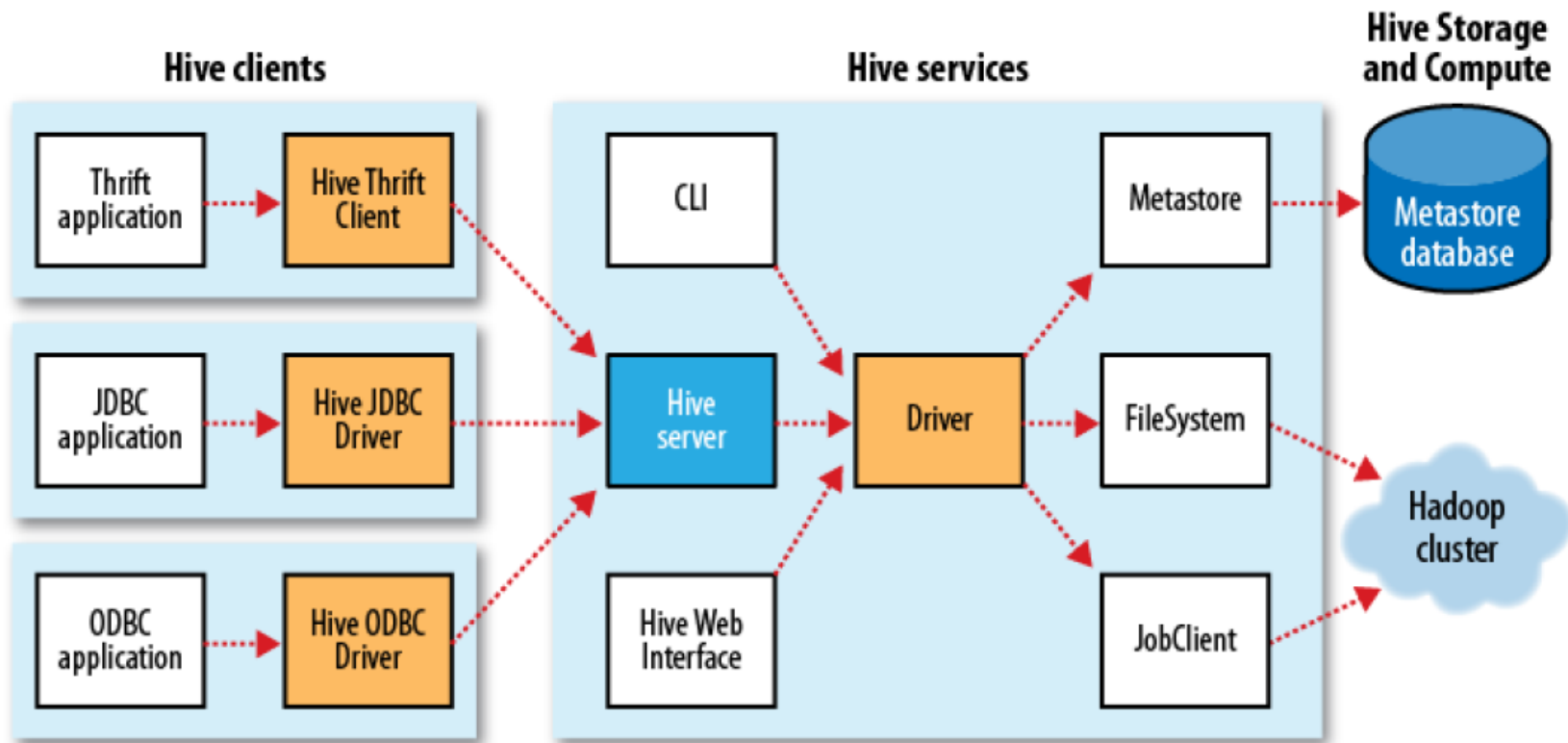
- 其实HLog文件就是一个普通的Hadoop Sequence File, Sequence File 的Key是HLogKey对象, HLogKey中记录了写入数据的归属信息, 除了table和region名字外, 同时还包括 sequence number和timestamp, timestamp是“写入时间”, sequence number的起始值为0, 或者是最近一次存入文件系统中sequence number。
- HLog Sequece File的Value是HBase的KeyValue对象, 即对应HFile中的KeyValue



# HBase的运行组成



# Hive客户端



通过将Hive运行为服务器hive - service hiveserver可以启动一个服务，而后可通过不同的客户端连接到这个服务去访问Hive提供的功能

# Hive客户端

Thrift客户端：被绑定在多种语言中，包括C++，Java，PHP，Python以及Ruby等

JDBC驱动：提供纯Java的JDBC驱动，连接字符串类似于 `jdbc:hive://host:port/dbname`，除非使用了嵌入的模式，否则JDBC模式的驱动访问了Thrift服务器

ODBC驱动：与JDBC类似，但尚未成熟

# Hive中的元数据存储metastore

metastore包括两个部分：服务和后台的数据存储

**Embedded metastore:** 默认使用内嵌的Derby数据库实例，每次只能打开一个Hive会话

**Local metastore:** 可以使用运行在一个进程中的metastore进程来访问独立的数据库，可通过JDBC进行设置具体的数据库访问

**Remote metastore:** 一个或者多个metastore服务器和Hive服务运行在不同的进程内

# metastore的配置情况

