大数据技术与应用

北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

第二章 大数据存储
—HBase数据库（下）

鄂海红 计算机学院（国家示范性软件学院）教授
ehaihong@bupt.edu.cn 微信: 87837001 QQ: 3027581960

大数据存储—HBase数据库

北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

大数据存储—HBase数据库

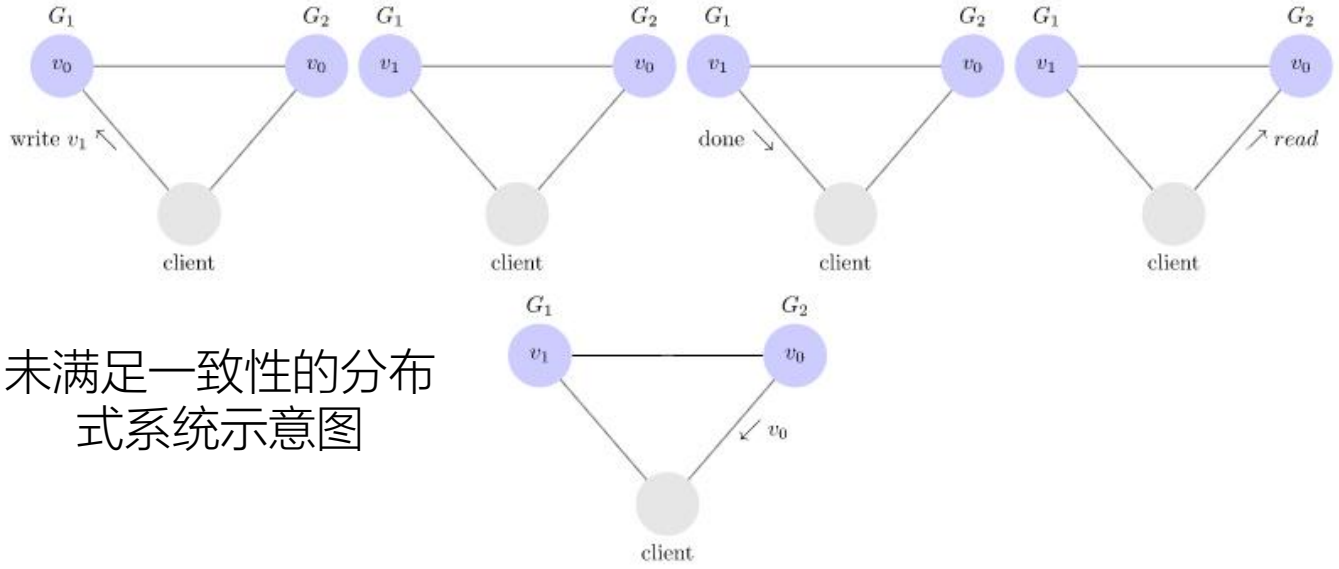
北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

“理解HBase的分布式系统本质”

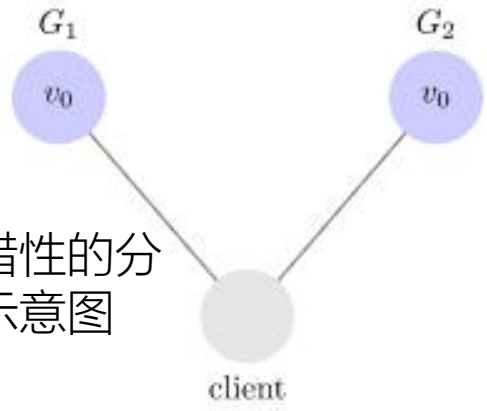
分布式CAP理论及HBase架构

分布式CAP理论

- CAP 定理是分布式系统中的基本定理，这个理论表明任何分布式系统最多可以满足以下三个属性中的两个：
- 一致性（Consistency）：在一致的系统中，一旦客户端将值写入任何服务器并获得响应，那么后续的读客户端将从分布式系统中任何的服务器中读取到这个值
- 可用性（Availability）：在可用系统中，客户端向服务器发送请求并且该服务器未崩溃，则该服务器必须最终响应客户端
- 分区容错性（Partition tolerance）：系统中任意信息的丢失或失败不会影响系统的继续运作，系统如果不能在某一个时限内达成数据一致性，就必须在上面两个操作之间做出选择



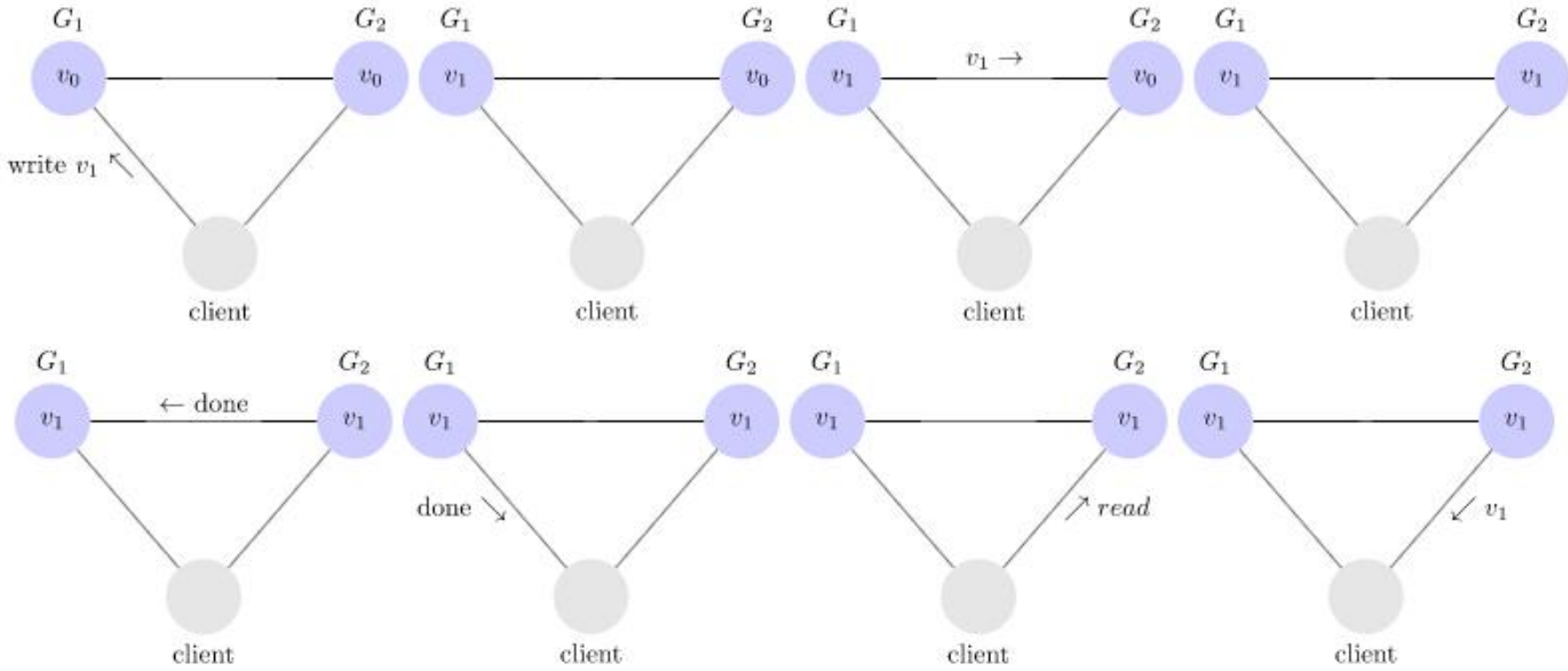
未满足一致性的分布式系统示意图



满足分区容错性的分布式系统示意图

满足一致性的分布式系统

- 在这个系统中，G1 服务器在响应客户端之前将 v 的值复制到 G2 服务器上，这时候客户端从 G2 获取 v 的值得到的结果是 v1



CAP理论：当发生网络分区的时候，如果我们要继续服务，那么强一致性和可用性只能 2 选 1

北京郵電大學 BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

- 反证法：假设存在满足这三个条件的分布式系统
- 图1-该分布式系统网络发生分区
- 图2-客户端 C_1 更新 G_1 服务器上的 v_0 为 v_1，因为系统是可用的所以 G_1 服务器会做出响应，但因为网络发生了分区，G_1无法将数据复制到G_2
- 图3-写完数据之后，另外一个客户端 C_2 向 G_2 服务器发出读取 v 的请求，但是因为网络分区的存在，G_2 服务器上 v 还是更新之前的值，所以客户端 C_2 得到的结果为 v_0
- 这种情况下C_2并没有获取到 C_1 写入的值，也就不满足数据一致性
- 由此可以得出：若系统出现“分区”，系统中的某个节点在进行写操作；为了保证 C，必须要禁止其他节点的读写操作，这就和 A 发生冲突了；如果为了保证 A，其他节点的读写操作正常的话，那就和 C 发生冲突了。

图1-某时刻网络发生分区

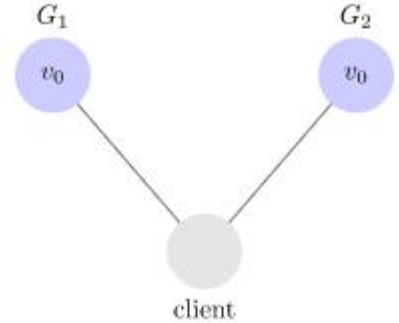


图2-客户端 C_1 更新 G_1 服务器上的 v_0 为 v_1

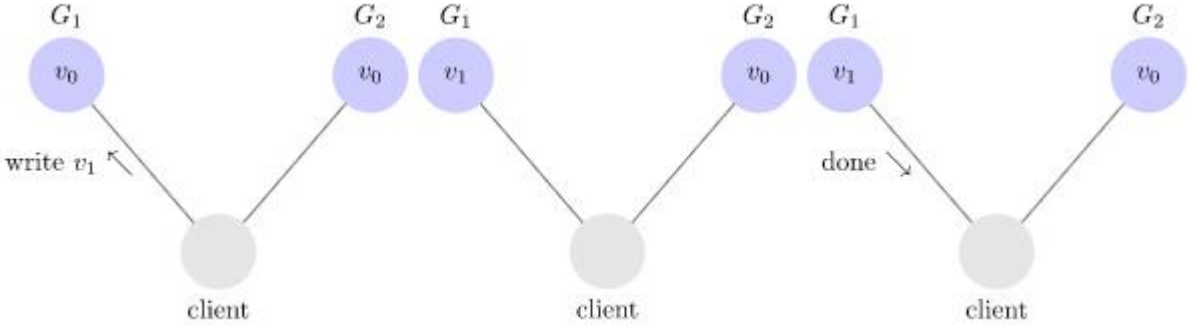
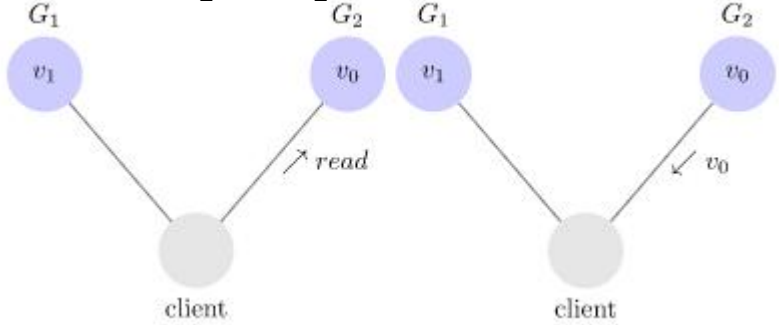


图2- C_2 向 G_2 服务器发出读取 v 的请求



CA CP AP

北京郵電大學 BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

- 一个数据存储系统不可能同时满足CAP，只能同时满足其两个特性CA/CP/AP
- 当前所有的数据存储解决方案，都可以归类的上述三种类型：
- CA：**满足一致性和高可用性**
  - 满足数据的一致性和高可用性，但没有可扩展性，如传统的关系型数据，基本上满足是这个解决方案，如ORACLE，MYSQL 的单节点，满足数据的一致性和高可用性。
- CP：**满足一致性和分区性**
  - 满足数据的一致性和分区性，如Oracle RAC ，Sybase 集群。虽然Oracle RAC具备一点的扩展性，但当节点达到一定数目时，性能（也即可用性）就会下降很快，并且节点之间的网络开销还在，需要实时同步各节点之间的数据。
- AP：**满足高可用性和分区性**
  - 在性能和可扩展性方面表现不错，但在数据一致性方面会用牺牲，各节点的之间数据同步没有那么快，但能保存数据的最终一致性。NoSQL大多类是典型的AP类型数据库。

✓ 分布式系统的网络分区是一定会存在的，所以分布式系统必须满足 P，否则就不是一个正真的分布式系统；所以各类分布式系统需要在网络分区发生时，在 A 和 C 之间做出选择
✓ 如果分布式系统不要求强的可用性，也就是容忍系统停机或者长时间无响应的话，可以考虑舍弃 A
✓ 如果系统可用性要求非常高，那么考虑牺牲一致性来满足。牺牲一致性一般都是说牺牲强一致性，而保证最终一致性。也就是说系统短暂是不一致性的，过段时间能保证一致，也就是最终一致性。

HBase VS Cassandra

北京郵電大學 BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

HBase

主要特点：

- 分布式和可扩展的庞大数据存储系统
- 强一致性
- 建立在Hadoop HDFS的基础上
- CAP中的CP

适合于：

- 针对读取进行了优化
- 很适合基于范围的扫描
- 严格一致性
- 快速读取和写入，具有可扩展性

- ✓ 因为对每一个region同时只有一台RegionServer为它服务，对一个region所有的操作请求，都由这一台region server来响应，自然是强一致性的。
- ✓ 在这台RegionServer fail的时候，它管理的region会failover故障迁移到其他Region Server时，需要根据WAL log来redo，这时候进行redo的region应该是unavailable（不可用）的，所以HBase降低了可用性，提高了一致性。

不适合于：

典型的事务型应用程序或甚至关系分析

应用程序需要全表扫描

数据需要跨聚合、累积以及跨行分析

HBase VS Cassandra

北京郵電大學 BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

Cassandra

主要特点：

- 高可用性
- 逐步可扩展性
- 最终一致性
- 兼顾一致性和延迟
- 没有单一故障点——Cassandra中所有节点都一样
- CAP中的AP

适合于：

- 简单的安装，维护节点
- 快速随机性读取/写入
- 灵活的解析/宽列需求
- 不需要多个辅助索引

不适合于：

- 辅助索引
- 关系数据
- 事务型操作（回滚和提交）
- 对数据需要严格的授权
- 针对列数据的动态查询/搜索
- 低延迟

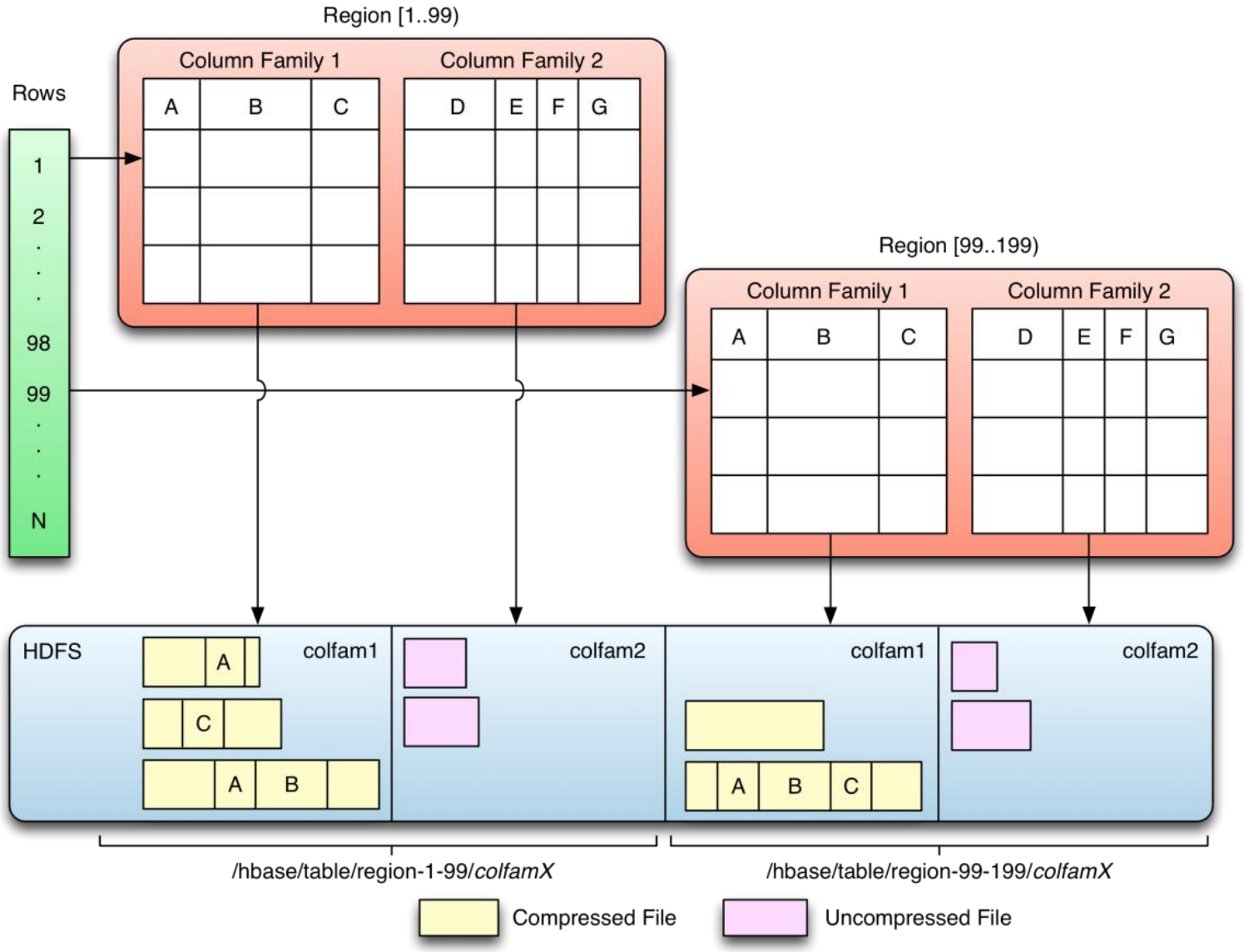
大数据存储—HBase数据库

北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

HBase数据写入、读取机制

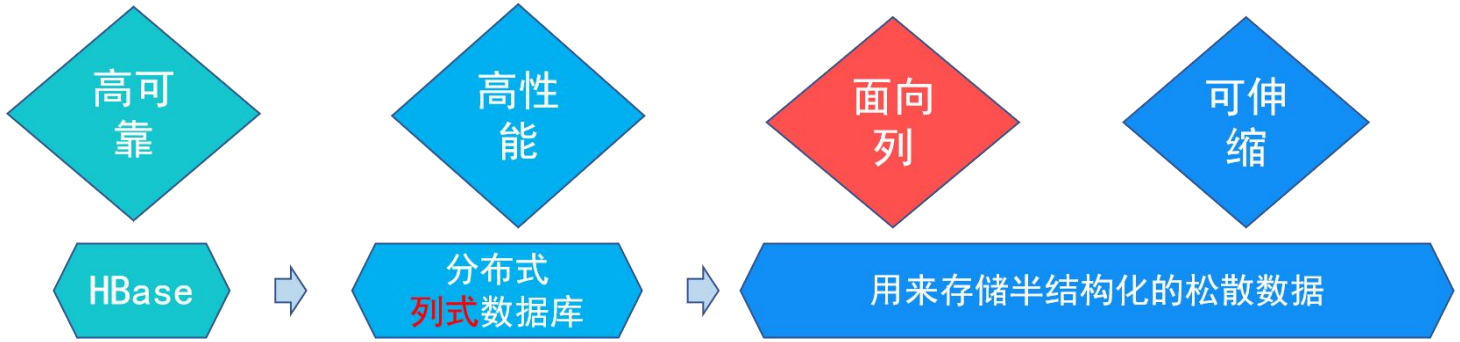
读写的关键是定位RegionServer

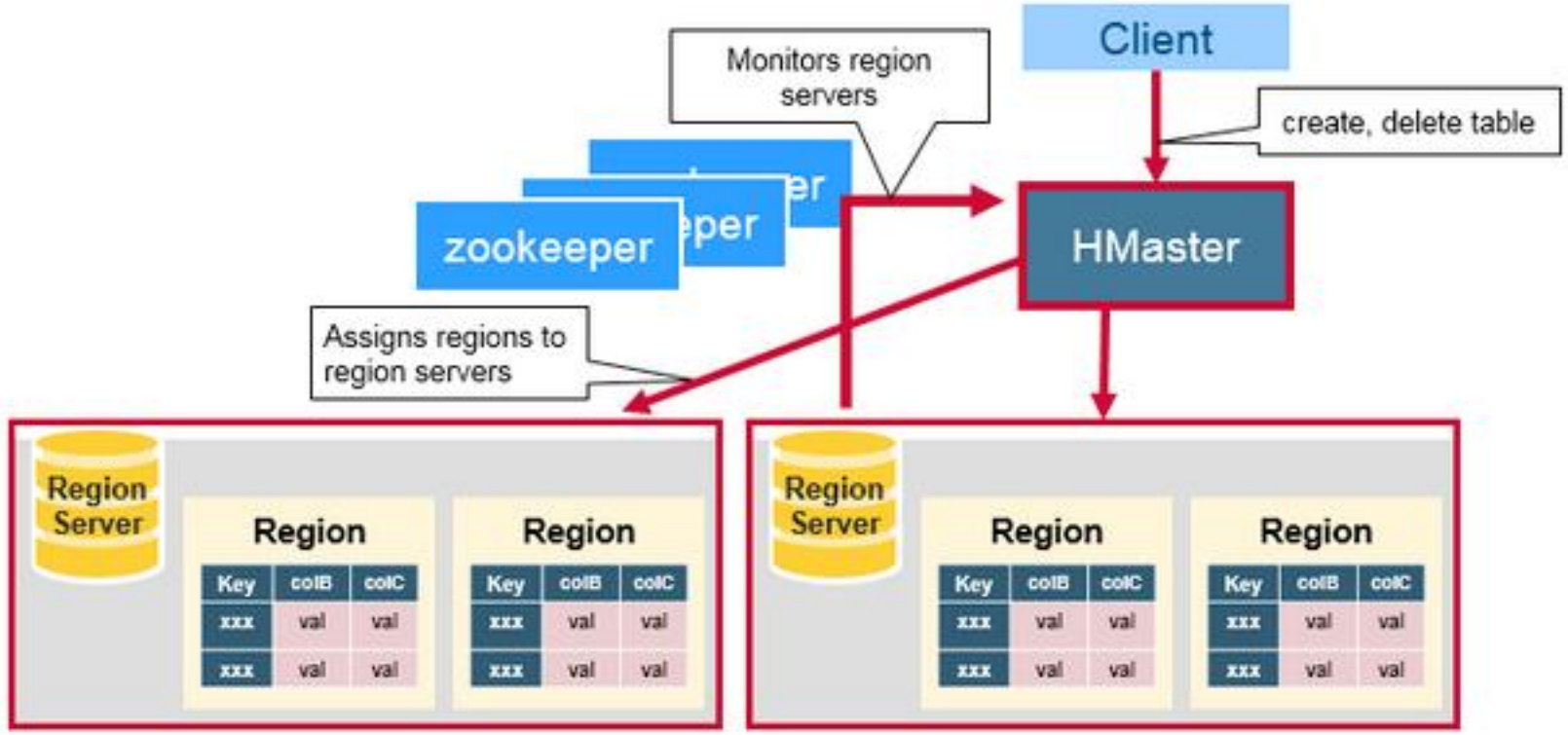
- HBase的数据以HFile的形式存储在RegionServer中
- 对HBase的数据表进行读写的关键：如何通过表名和行关键字RowKey找到所在的RegionServer



高性能随机读/写

- 随机读
  - K/V存储
  - Cache
  - Split
  - Balance
- 随机写（相对而言）
  - Cache+WAL
  - Compact
  - Split
  - Balance

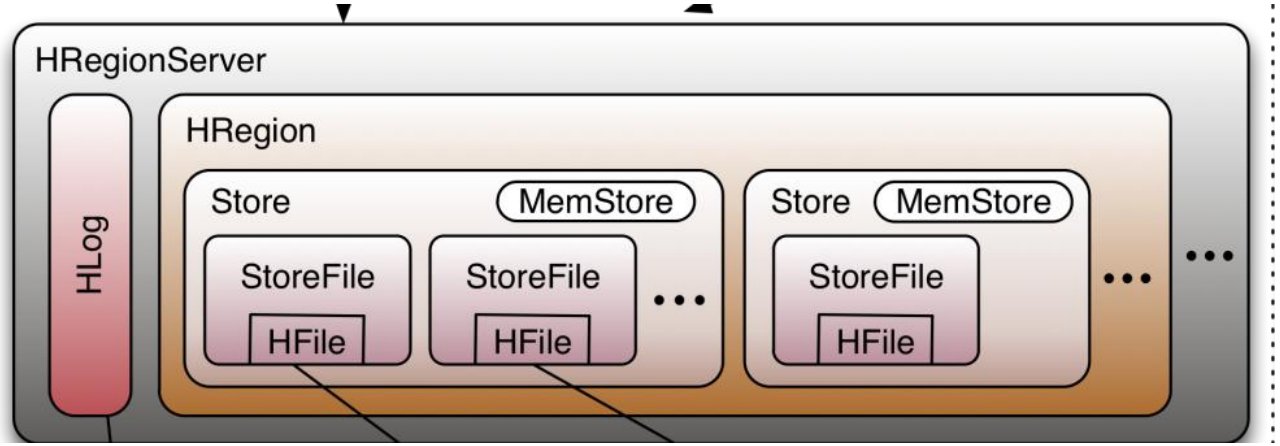




HRegionServer中的Hlog

- HMaster是整个框架中的控制节点，负责：
  - 管理用户对table的增、删、改、查操作
  - 管理HRegionServer的负载均衡，调整region分布
  - 在region split后，负责新region的分配
  - 在HRegionServer停机后，负责失效HRegionServer上的region迁移
  - HDFS的垃圾文件回收
  - 处理schema更新请求
- HBase中可以启动多个HMaster，通过Zookeeper的Master Election机制保证总有一个HMaster运行

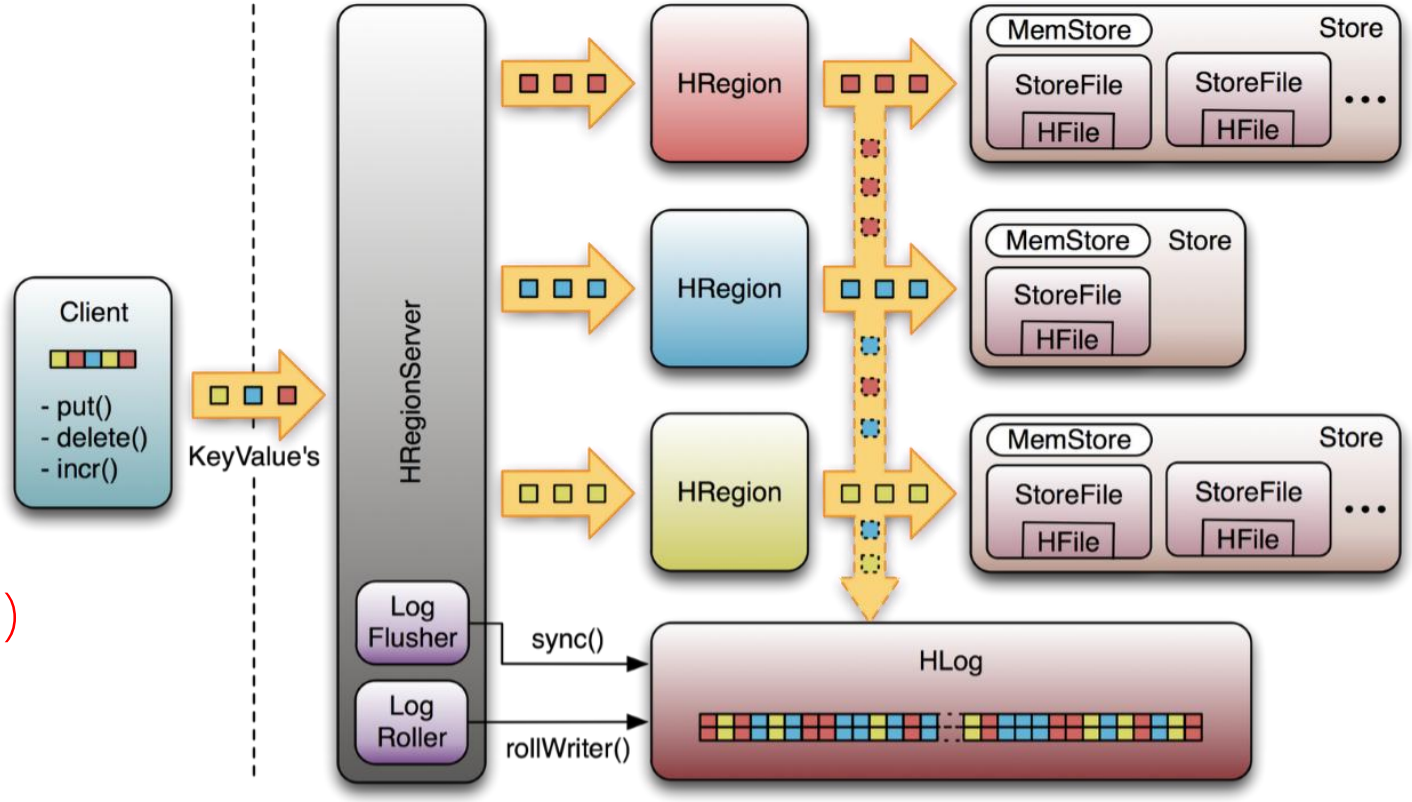
- HRegionServer负责维护HMaster分配给它的region，响应用户I/O请求，向HDFS文件系统中读写数据
- HRegionServer内部管理了一系列HRegion对象
- **HRegion**：一个HRegion可以包含多个Store
- **Store**：每个Store包含一个Memstore和若干个StoreFile
- **StoreFile**：**表数据真实存储的地方**，HFile是表数据在HDFS上的文件格式
- Hlog：多个Store1个HLog；Hlog保障可靠性；MemStore数据镜像，持久化到文件



Write-Ahead Logging 预写日志机制

借助HDFS+WAL机制保证数据可靠性

- 为了防止Region服务器发生故障时，MemStore 缓存中还没有被写入文件的数据会全部丢失，HBase 采用 HLog 来保证系统发生故障时能够恢复到正常的状态
- 每个 Region 服务器都有一个 HLog 文件，同一个 Region 服务器的 Region 对象共用一个 HLog，HLog 是一种预写日志（Write Ahead Log）文件
- 用户更新数据必须先被记入日志后才能写入 MemStore 缓存，当缓存内容对应的日志已经被写入磁盘后，即日志写成功后，缓存的内容才会被写入磁盘

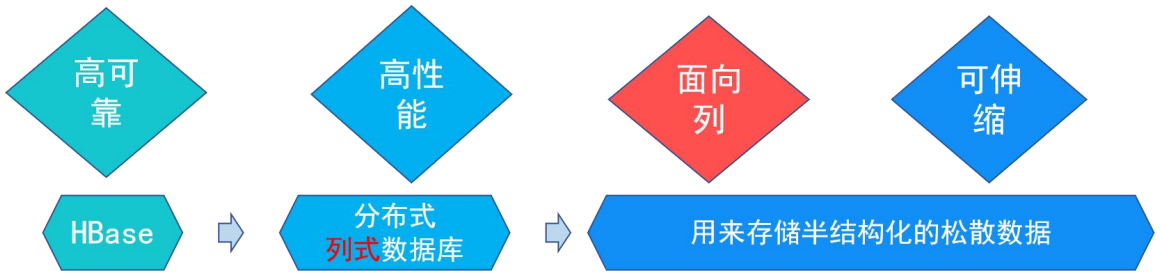


多个Region 对象共用一个 HLog 的方式中

- ✓ 多个 Region 对象在进行更新操作需要修改日志时，只需要不断地把日志记录追加到单个日志文件中
- ✓ 而不需要同时打开、写入多个日志文件中，因此可以减少磁盘寻址次数，提高对表的写操作性能

Write-Ahead Logging 预写日志机制

- ZooKeeper 会实时监测每个 Region 服务器的状态，当某个 Region 服务器发生故障时，ZooKeeper 会通知 Master，Master 首先会处理该故障 Region 服务器上遗留的 HLog 文件
- 由于一个 Region 服务器上可能会维护着多个 Region 对象，这些 Region 对象共用一个 HLog 文件，因此这个遗留的 HLog 文件中包含了来自多个 Region 对象的日志记录
- 系统会根据每条日志记录所属的 Region 对象对 HLog 数据进行拆分，并分别存放到相应 Region 对象的目录下
- 再将失效的 Region 重新分配到可用的 Region 服务器中，并在可用的 Region 服务器中重新进行日志记录中的各种操作， 把日志记录中的数据写入 MemStore 然后刷新到磁盘的 StoreFile 文件中，完成数据恢复



大数据存储—HBase数据库

北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

HBase数据库使用

HBase数据操作方式

北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

- Shell
- JavaAPI
- 非JavaAPI（**REST**、**Thrift**、**Avro**）

Shell工具方式

HBase的Shell工具

- 通过Shell可以连接到本地或远程的HBase服务器上操纵数据
- 命令行启动：> $HBASE_HOME/bin/hbase shell

五类命令：

- 表管理：创建、删除和修改表的相关操作指令
  - create 'table', 'col_f1', 'col_f2'
- 数据管理：对表中的数据进行操作
  - put 'table', 'row1', 'col_f1', 'value'
  - get 'table', 'row1', {COLUMN => 'col_f1'}
- 工具：管理和优化数据存储方式的功能
- 复制：将数据备份到多个节点的相关操作指令
- 其他：查看HBase集群状态和版本

数据表管理命令

北京郵電大學 BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

HBase Shell 数据表命令	
命令	描述
create	创建指定模式的新表
alter	修改表的结构，如添加新的列族
describe	展示表结构的信息，包括列族的数量与属性
list	列出 HBase 中已有的表
disable/enable	为了删除或更改表而禁用一个表，更改完后需要解禁表
disable_all	禁用所有的表，可以用正则表达式匹配表
is_disable	判断一个表是否被禁用
drop	删除表
truncate	如果只是想删除数据而不是表结构，则可用 truncate 来禁用表、删除表并自动重建表结构

```
hbase(main):001:0> create 'test', 'cf'
0 row(s) in 0.4170 seconds

=> Hbase::Table - test
```

```
hbase(main):002:0> list 'test'
TABLE
test
1 row(s) in 0.0180 seconds

=> ["test"]
```

数据增删改查操作命令

	cf					
	cf：a	cf：b	c			
row1	value1					
row2		value2				

命令	描述
put	添加一个值到指定单元格中；put ‘表名’, ‘行键’, ‘列族名’, ‘列值’
get	通过表名、行键等参数获取行或单元格数据； • get ‘表名’, ‘行键’ • get ‘表名’, ‘行键’, ‘列族名’
scan	遍历表并输出满足指定条件的行记录 • 扫描某个列族： scan ‘表名’, {COLUMN=>‘列族名’} • 扫描某个列族的某个列： scan ‘表名’, {COLUMN=>‘列族名:列名’} • 查询同一个列族的多个列： scan ‘表名’, {COLUMNS => [‘列族名1:列名1’, ‘列族名1:列名2’, ...]}
count	计算表中的逻辑行数
delete	删除表中列族或列的数据； • 删除列族的某个列： delete ‘表名’, ‘行键’, ‘列族名:列名’
incr	增加指定表行或列的值；incr ‘表名’, ‘行键’, ‘列族:列名’, 步长值
get_counter	获取计数器；get_counter ‘表名’, ‘行键’, ‘列族:列名’
deleteall	删除指定行的所有元素值，deleteall ‘表名’, ‘行键’

```
0 row(s) in 0.0850 seconds

hbase(main):004:0> put 'test', 'row2', 'cf:b',
'value2'
0 row(s) in 0.0110 seconds

hbase(main):005:0> put 'test', 'row3', 'cf:c',
'value3'
0 row(s) in 0.0100 seconds
```

```
hbase(main):006:0> scan 'test'
ROW COLUMN+CELL
row1 column=cf:a,
timestamp=1421762485768, value=value1
row2 column=cf:b,
timestamp=1421762491785, value=value2
row3 column=cf:c,
timestamp=1421762496210, value=value3
3 row(s) in 0.0230 seconds
```

```
hbase(main):007:0> get 'test', 'row1'
COLUMN CELL
cf:a timestamp=1421762485768,
value=value1
1 row(s) in 0.0350 seconds
```

Java API

北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

- HBaseConfiguration 配置类
- HBaseAdmin 对数据表进行操作的接口
- HTableDescriptor 数据表列族操作及表属性的操作接口
- HColumnDescriptor 数据列相关的数据和操作接口
- HTable，Put/Get/Delete 数据的插入、检索和删除操作

配置Java连接HBase

- Java连接HBase需要两个类：
  - HBaseConfiguration
  - ConnectionFactory
- 首先，配置一个HBase连接：

比如zookeeper的地址端口
hbase.zookeeper.quorum
hbase.zookeeper.property.clientPort

```
<configuration>

<property>
<name>hbase.master</name>
<value>hdfs://host1:60000</value>
</property>

<property>
<name>hbase.zookeeper.quorum</name>
<value>host1,host2,host3</value>
</property>

<property>
<name>hbase.zookeeper.property.clientPort</name>
<value>2181</value>
</property>
</configuration>
```

编写hbase-site.xml文件，实现配置文件的加载：

```
<property>
  <name>hbase.zookeeper.quorum</name>
  <value>name-number-0002:2181,name-number-0003:2181,name-number-0004:2181</value>
</property>
```

加载配置文件，创建连接：

```
config.addResource(new Path(System.getenv("HBASE_CONF_DIR"), "hbase-site.xml"));
Connection connection = ConnectionFactory.createConnection(config);
```

创建表

北京郵電大學 BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

- 要创建表我们需要首先创建一个Admin对象

```
Admin admin = connection.getAdmin(); //使用连接对象获取Admin对象
TableName tableName = TableName.valueOf("test");//定义表名
HTableDescriptor htd = new HTableDescriptor(tableName);//定义表对象
HColumnDescriptor hcd = new HColumnDescriptor("data");//定义列族对象
htd.addFamily(hcd); //添加
admin.createTable(htd);//创建表
```

- HBase2.X 的版本中创建表使用了新的 API

```
TableName tableName = TableName.valueOf("test");//定义表名
//TableDescriptor对象通过TableDescriptorBuilder构建；
TableDescriptorBuilder tableDescriptor = TableDescriptorBuilder.newBuilder(tableName);
ColumnFamilyDescriptor family =
ColumnFamilyDescriptorBuilder.newBuilder(Bytes.toBytes("data")).build();//构建列族对象
tableDescriptor.setColumnFamily(family);//设置列族
admin.createTable(tableDescriptor.build());//创建表
```

添加数据　获取指定行数据


```
Table table = connection.getTable(tableName);//获取Table对象
try {
        byte[] row = Bytes.toBytes("row1"); //定义行
        Put put = new Put(row); //创建Put对象
        byte[] columnFamily = Bytes.toBytes("data"); //列
        byte[] qualifier = Bytes.toBytes(String.valueOf(1)); //列族修饰词
        byte[] value = Bytes.toBytes("张三丰"); //值
        put.addColumn(columnFamily, qualifier, value);
        table.put(put); //向表中添加数据
} finally {//使用完了要释放资源
        table.close();
}
```

```
//获取数据
        Get get = new Get(Bytes.toBytes("row1")); //定义get对象
        Result result = table.get(get); //通过table对象获取数据
        System.out.println("Result: " + result);
//很多时候我们只需要获取“值”这里表示获取 data:1 列族的值
        byte[] valueBytes = result.getValue(Bytes.toBytes("data"), Bytes.toBytes("1")); //获取到的是字节数组
//将字节转成字符串
        String valueStr = new String(valueBytes,"utf-8");
        System.out.println("value:" + valueStr);
```

扫描表中的数据　删除表

扫描表中的数据

```
Scan scan = new Scan();
ResultScanner scanner = table.getScanner(scan);
try {
        for (Result scannerResult: scanner) {
                System.out.println("Scan: " + scannerResult);
                byte[] row = scannerResult.getRow();
                System.out.println("rowName:" + new String(row,"utf-8"));
                }
    } finally {
        scanner.close();
}
```

删除表

```
TableName tableName =
TableName.valueOf("test");
admin.disableTable(tableName); //禁用表
admin.deleteTable(tableName); //删除表
```

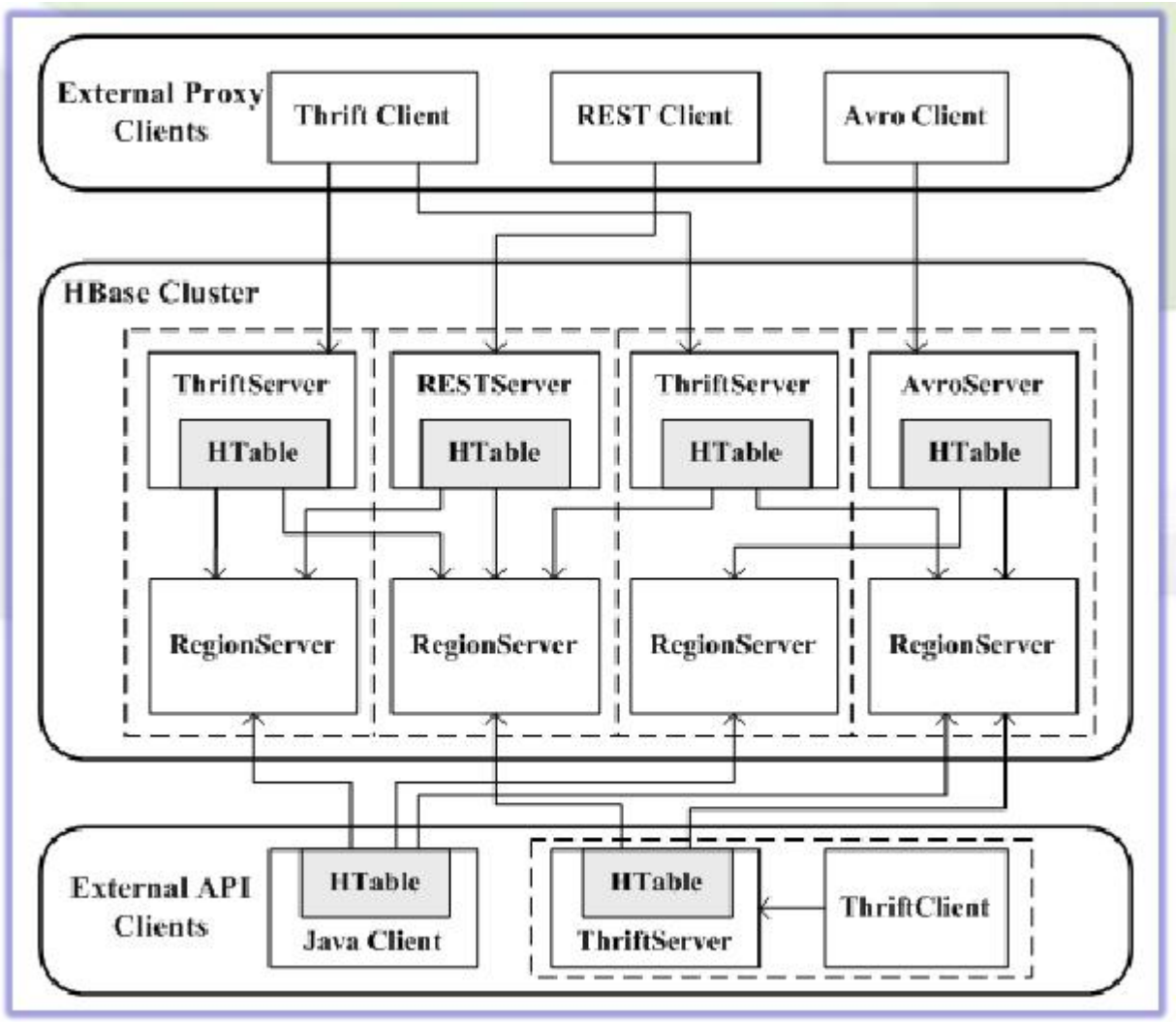
完整代码

```
package com.example.hbase.admin;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.HColumnDescriptor;
import org.apache.hadoop.hbase.HConstants;
import org.apache.hadoop.hbase.HTableDescriptor;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.Admin;
import org.apache.hadoop.hbase.client.Connection;
import org.apache.hadoop.hbase.client.ConnectionFactory;
import org.apache.hadoop.hbase.io.compress.Compression.Algorithm;

public class Example {
private static final String TABLE_NAME = "MY_TABLE_NAME_TOO";
private static final String CF_DEFAULT = "DEFAULT_COLUMN_FAMILY";
public static void createOrOverwrite(Admin admin, HTableDescriptor table) throws
IOException {
if (admin.tableExists(table.getTableName())) {
admin.disableTable(table.getTableName());
admin.deleteTable(table.getTableName());
}
admin.createTable(table);
}

public static void createSchemaTables(Configuration config) throws IOException {
try (Connection connection = ConnectionFactory.createConnection(config);
Admin admin = connection.getAdmin()) {

HTableDescriptor table = new HTableDescriptor(TableName.valueOf(TABLE_NAME));
table.addFamily(new
HColumnDescriptor(CF_DEFAULT).setCompressionType(Algorithm.NONE));

System.out.print("Creating table. ");
createOrOverwrite(admin, table);
System.out.println(" Done.");
}
}
```

```
public static void modifySchema (Configuration config) throws IOException {
try (Connection connection = ConnectionFactory.createConnection(config);
Admin admin = connection.getAdmin()) {

TableName tableName = TableName.valueOf(TABLE_NAME);
if (!admin.tableExists(tableName)) {
System.out.println("Table does not exist.");
System.exit(-1);
}

HTableDescriptor table = admin.getTableDescriptor(tableName);
// 更新表格
HColumnDescriptor newColumn = new HColumnDescriptor("NEWCF");
newColumn.setCompactionCompressionType(Algorithm.GZ);
newColumn.setMaxVersions(HConstants.ALL_VERSIONS);
admin.addColumn(tableName, newColumn);
// 更新列族
HColumnDescriptor existingColumn = new HColumnDescriptor(CF_DEFAULT);
existingColumn.setCompactionCompressionType(Algorithm.GZ);
existingColumn.setMaxVersions(HConstants.ALL_VERSIONS);
table.modifyFamily(existingColumn);
admin.modifyTable(tableName, table);
// 禁用表格
admin.disableTable(tableName);
// 删除列族
admin.deleteColumn(tableName, CF_DEFAULT.getBytes("UTF-8"));
// 删除表格(需提前禁用)
admin.deleteTable(tableName);
}
}
public static void main(String... args) throws IOException {
Configuration config = HBaseConfiguration.create();

//添加必要配置文件(hbase-site.xml, core-site.xml)
config.addResource(new Path(System.getenv("HBASE_CONF_DIR"), "hbase-site.xml"));
config.addResource(new Path(System.getenv("HADOOP_CONF_DIR"), "core-
site.xml"));
createSchemaTables(config);
modifySchema(config);
}
}
```

非Java API

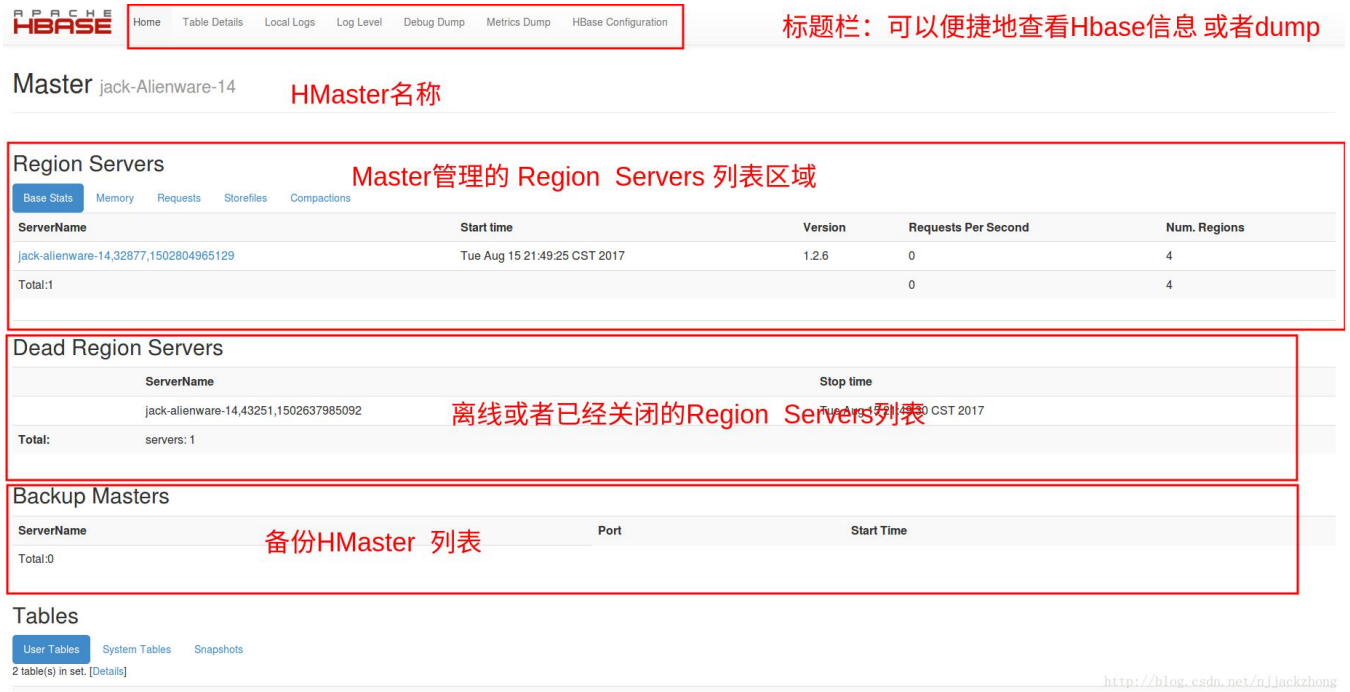
- 由于HBase是用Java写的，因此原生地提供了Java接口
- 同时为采用其他语言来编写HBase的客户端,通过Thrift接口服务器，提供接口（比如HBase python接口）
  - REST代理服务器：支持HTTP接口
  - Thrift代理服务器：支持RPC接口
  - Avro代理服务器：支持RPC接口



HBase Web UI

北京郵電大學 BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

- HBase提供了一种Web方式的用户接口，用户可以通过Web界面查看HBase集群的属性等状态信息，web页面分为：Master状态界面，和Zookeeper统计信息页面
- Master状态界面会看到Master状态的详情：HBase集群信息，任务信息，表信息，RegionServer信息
- RegionServer状态界面会看到RegionServer状态的详情
- RegionServer的节点属性信息，任务信息和Region信息
- Zookeeper统计信息页面是非常简单的半结构化文本打印信息



HBase设计经验—考虑Region规模

- 一般Regions的大小定在10~50GB；每个表的Regions数量控制在50~100个（Region是一段连续的列族）
- 通常情况下，1个表的列族控制在1~3个
  - 因为某个column family在flush的时候，它邻近的column family也会因关联效应被触发flush，最终导致系统产生更多的I/O
- 列族作为若干列的集合，其所有成员都有同样的前缀
  - 比如courses：history 和 courses：math都是courses列族的成员，冒号是分隔符
  - 因此列族前缀必须是可输出字符，列可由任意字节数组组成
  - 尽量使列族的名称简短，因为每个值都会存储列族名
- 一个Cells的大小不要超过10MB，如果要存储中型数据（超过50MB），可以选择将数据存储在HDFS上，然后在HBase中存储引用指针

HBase设计经验- Rowkey设计

- 在表中是通过Row Key的字典序来对一行行的数据来进行排序的，表中每一块区域的划分都是通过开始Row Key和结束Row Key来决定的
- 唯一性： RowKey必须能够唯一的识别一行数据
- 考虑最高频的查询场景：目标是如何高效的读取和消费数据，而不仅是为数据存储本身
- 充分考虑HBase是按照RowKey从小到大排序的特性：结合具体的负载特点，对选取的RowKey字段值进行改造，组合字段场景下需要重点考虑字段的顺序
- 1、Rowkey是以字典序排序
  - 原生HBase只支持从小到大排序，要想实现从大到小，可以采用 Rowkey=Integer.MAX_VALUE-Rowkey的方式，在应用层再转回来完成需求
- 2、Rowkey尽量散列
  - Rowkey要尽量散列，这样可以保证数据不在一个Region上，从而避免了读写的集中（例如：20%的用户产生了80%的数据；或者如果存储基于时间的数据或日志数据，row key是基于设备ID或服务ID加时间，这样的模式可能会导致新加的数据都写到新的Region，而旧的Region则不会被写，形成写热点）
  - 比如可以设计 userid_videoid 拼接字符串 这样的话user就会不均匀；或者反转userid、散列userid、将userid取模后进行MD5加密 取前6位加入userid中
- 3、Rowkey长度要尽量短
  - Rowkey过长，存储开销会大；Rowkey过长，会导致内存的利用率降低，进而降低索引命中率

总结-1

- HBase是一种面向列的（稀疏），基于HDFS的（海量），高性能（快速）分布式数据库系统
  - 利用Hadoop HDFS作为其文件存储系统，提供高可靠性、高性能、列存储、可伸缩、实时读写的数据库系统
  - 利用Hadoop MapReduce来处理HBase中的海量数据
  - 利用Zookeeper作为协同服务

总结-2 HBase Vs RDBMS

	HBase	RDBMS
数据类型	只有字符串	丰富的数据类型
数据操作	简单的增删改查	各种各样的函数，表连接
存储模式	基于列存储	基于表格结构和行存储
数据保护	更新后旧版本仍然会保留	替换
可伸缩性	轻易的进行增加节点，兼容性高	需要中间层，牺牲功能

北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

谢谢聆听
批评指正

ehaihong@bupt.edu.cn