大数据管理
Big Data Management

张海腾

htzhang@ecust.edu.cn

第四章 分布式数据库HBASE

- 4.1 概述
- 4.2 HBASE访问接口
- 4.3 HBASE数据模型
- 4.4 HBASE的实现原理
- 4.5 HBASE的运行机制
- 4.6 HBASE编程实现
- 4.7 本章小节

4.1 概述

4.1.1 从BigTable说起

4.1.2 HBase简介

4.1.3 HBase与传统关系数据库的对比分析

4.1.1 从BigTable说起

- HBase（Hadoop DataBase）是Google BigTable的开源实现
  - BigTable是一个分布式存储系统
  - BigTable起初用于解决典型的互联网搜索问题
    - 建立互联网索引
    - 搜索互联网
  - BigTable利用谷歌提出的MapReduce分布式并行计算模型来处理海量数据
  - BigTable使用谷歌分布式文件系统GFS作为底层数据存储
  - BigTable采用Chubby提供协同服务管理

4.1.2 HBase简介

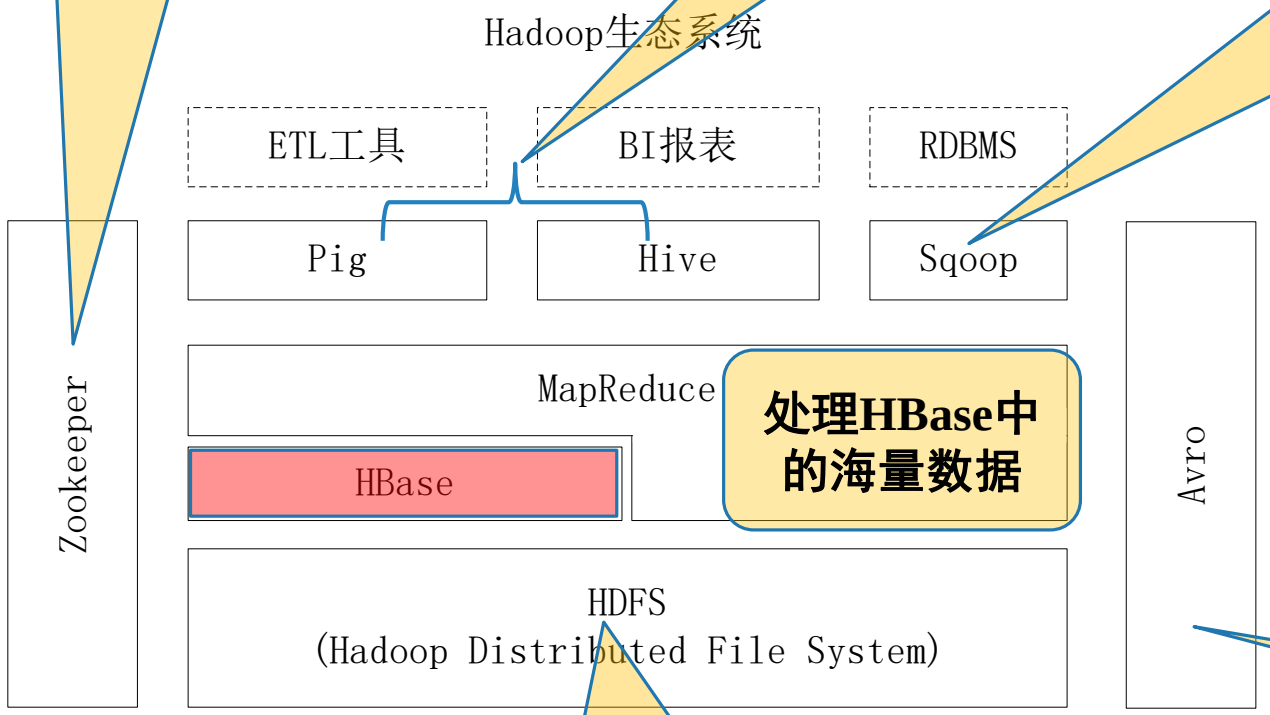
- HBase是一个高可靠、高性能、面向列、可伸缩、实时读写的分布式数据库，是谷歌BigTable的开源实现，主要用来存储非结构化和半结构化的松散数据。
- HBase的目标是处理非常庞大的表，可以通过水平扩展的方式，利用廉价计算机集群处理由超过10亿行数据和数百万列元素组成的数据表。
- HBase常被用来存放一些海量的（通常在TB级别以上）结构比较简单的数据，如历史订单记录，日志数据，监控数据等等，HBase提供了简单的基于Key值的快速查询能力 。

4.1.2 HBase简介

作为协同服务，实现稳定服务和失败恢复

为HBase提供高层语言支持

为HBase提供了高效、便捷的RDBMS数据导入功能



处理HBase中的海量数据

	BigTable	HBase
文件存储系统	GFS	HDFS
海量数据处理	MapReduce	Hadoop MapReduce
协同服务管理	Chubby	Zookeeper

Hadoop生态系统中HBase与其他部分的关系

作为HBase的底层数据存储方式

Avro是Hadoop的一个数据序列化系统，用于支持大批量数据交换的应用。

4.1.3 HBase与传统RDB比较

1. 数据类型
   - 关系数据库采用关系模型，具有丰富的数据类型和存储方式
   - HBase则采用了更加简单的数据模型，它把数据存储为未经解释的字符串byte[]
2. 数据操作
   - 关系数据库中包含了丰富的操作，其中会涉及复杂的多表连接
   - HBase操作则不存在复杂的表与表之间的关系，只有简单的插入、查询、删除、清空等，因为HBase在设计上就避免了复杂的表和表之间的关系

4.1.3 HBase与传统RDB比较

3. 存储模式
   - 关系数据库是基于行模式存储的
   - HBase是基于列存储的，每个列族都由几个文件保存，不同列族的文件是分离的
4. 数据索引
   - 关系数据库通常可以针对不同列构建复杂的多个索引，以提高数据访问性能
   - HBase只有一个索引——行键（Row Key），通过巧妙的设计，HBase中的所有访问方法，或者通过行键访问，或者通过行键扫描，从而使得整个系统不会慢下来

4.1.3 HBase与传统RDB比较

5. 数据维护
   - 在关系数据库中，更新操作会用最新的当前值去替换记录中原来的旧值，旧值被覆盖后就不会存在
   - 在HBase中执行更新操作时，并不会删除数据旧的版本，而是生成一个新的版本，旧有的版本仍然保留
6. 可伸缩性
   - 关系数据库很难实现横向扩展，纵向扩展的空间也比较有限
   - HBase和BigTable这些分布式数据库就是为了实现灵活的水平扩展而开发的，能够轻易地通过在集群中增加或者减少硬件数量来实现性能的伸缩

思考题

- **关系数据库已经流行很多年，并且Hadoop已经有了HDFS和MapReduce，为什么需要HBase?**

- Hadoop可以很好地解决大规模数据的离线批量处理问题，但是，受限于Hadoop MapReduce编程框架的高延迟数据处理机制，使得Hadoop无法满足大规模数据实时处理应用的需求
- HDFS面向批量访问模式，不是随机访问模式
- 传统的通用关系型数据库无法应对在数据规模剧增时导致的系统扩展性和性能问题
  传统关系数据库在数据结构变化时一般需要停机维护；空列浪费存储空间
- 因此，业界出现了一类面向半结构化数据存储和处理的高可扩展、低写入/查询延迟的系统，例如，键值数据库、文档数据库和列族数据库（如BigTable和HBase等）
- HBase已成功应用于互联网服务领域和传统行业的众多在线式数据分析处理系统中

4.2 HBase访问接口

类型	特点	场合
Native Java API	最常规和高效的访问方式	适合Hadoop MapReduce作业并行批处理HBase表数据
HBase Shell	HBase的命令行工具，最简单的接口	适合HBase管理使用
Thrift Gateway	利用Thrift序列化技术，支持C++、PHP、Python等多种语言	适合其他异构系统在线访问HBase表数据
REST Gateway	解除了语言限制	支持REST风格的Http API访问HBase
Pig	使用Pig Latin流式编程语言来处理HBase中的数据	适合做数据统计
Hive	简单	当需要以类似SQL语言方式来访问HBase的时候

4.3 HBase数据模型

- 4.3.1 数据模型概述
- 4.3.2 数据模型相关概念
- 4.3.3 数据坐标
- 4.3.4 HBase的概念视图
- 4.3.5 HBase的物理视图
- 4.3.6 面向列的存储

4.3.1 数据模型概述

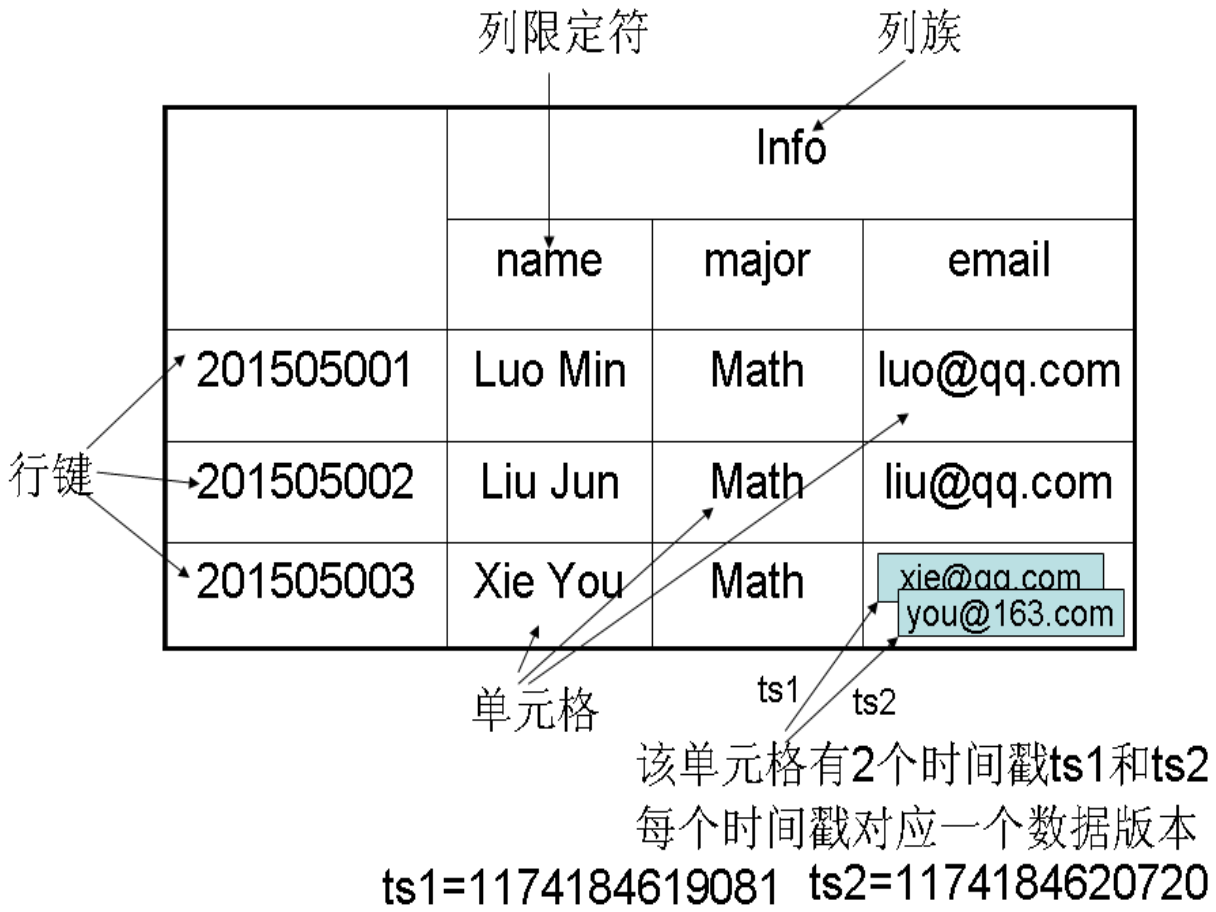
- HBase是一个稀疏、多维度、排序的映射表，这张表的索引是行键、列族、列限定符和时间戳
- 每个值是一个未经解释的字符串byte[]，没有数据类型
- 用户在表中存储数据，每一行都有一个可排序的行键和任意多的列
- 表在水平方向由一个或者多个列族组成，一个列族中可以包含任意多个列，同一个列族里面的数据存储在一起

4.3.1 数据模型概述

- **列族支持动态扩展，可以很轻松地添加一个列限定符或列，无需预先定义列的数量以及类型，所有列均以字符串形式存储，用户需要自行进行数据类型转换**
- **HBase中执行更新操作时，并不会删除数据旧的版本，而是生成一个新的版本，旧有的版本仍然保留（这是和HDFS只允许追加不允许修改的特性相关）**

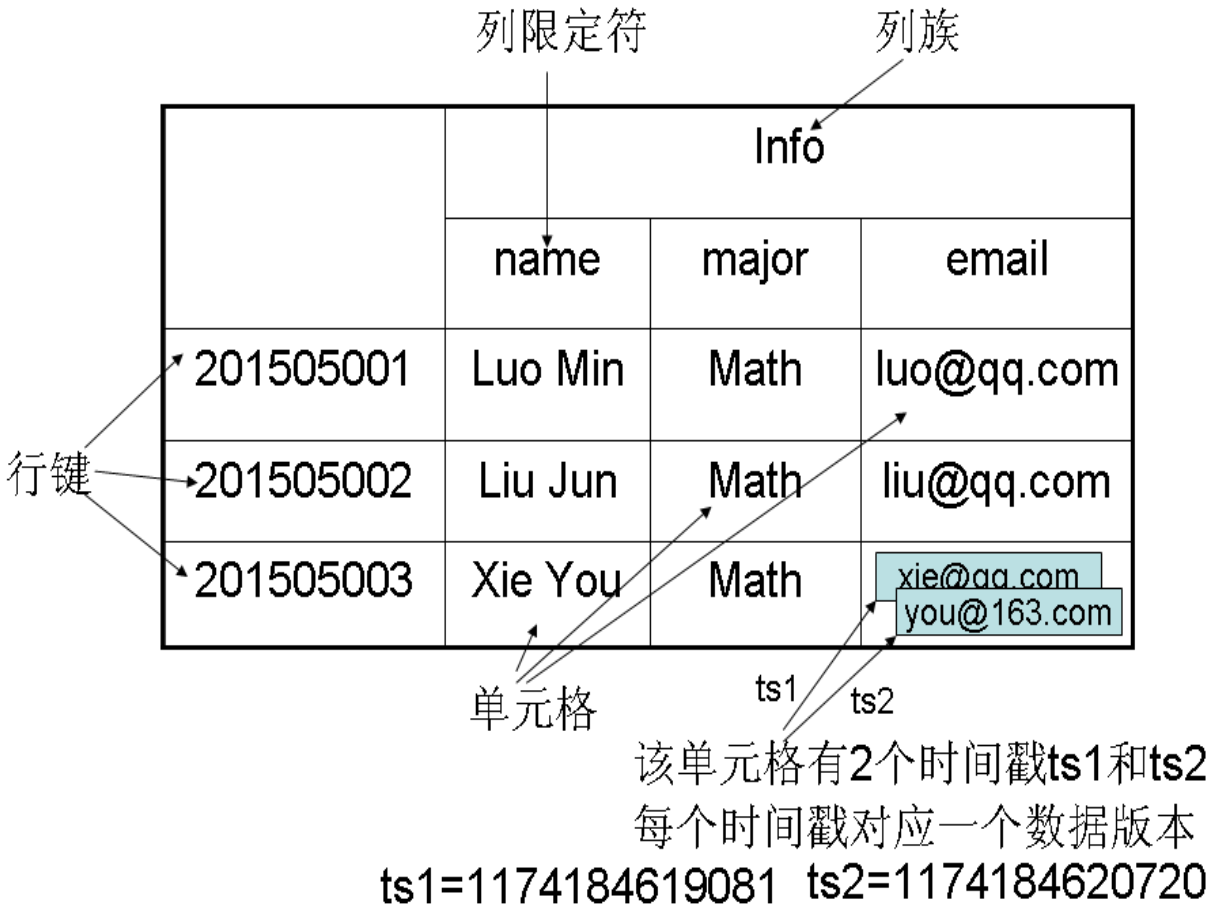
4.3.2 数据模型相关概念

- **表：HBase采用表来组织数据，表由行和列组成，列划分为若干个列族**
- **行：每个HBase表都由若干行组成，每个行由行键（Row Key）来标识。**
- **列族：一个HBase表被分组成许多“列族”（Column Family）的集合，它是基本的访问控制单元，一个没有列族的表是没有意义的**



4.3.2 数据模型相关概念

- **列限定符：**列族里的数据通过列限定符（或列）来定位，类型byte[]
- **单元格：**在HBase表中，通过行、列族和列限定符确定一个“单元格”（cell），单元格中存储的数据没有数据类型，字节数组byte[]
- **时间戳：**每个单元格都保存着同一份数据的多个版本，这些版本采用时间戳进行索引



4.3.3 数据坐标

- HBase中需要根据行键、列族、列限定符和时间戳来确定一个单元格，因此，可以视为一个“四维坐标”，即：

[行键, 列族, 列限定符, 时间戳]

键	值
[“201505003”, “Info”, “email”, 1174184619081]	“xie@qq.com”
[“201505003”, “Info”, “email”, 1174184620720]	“you@163.com”

键值数据库

4.3.4 HBase的概念视图

存储网页内容

包含了任何引用这个页面的锚链接文本

行键	时间戳	列族contents	列族anchor
"com.baidu.www"	t5	是一个稀疏的映射关系	anchor:xmu.edu.cn.com=“BAIDU”
	t4		anchor:pku.edu.cn.com=“BAIDU.com"
	t3	contents:html="<html>..."	
	t2	contents:html="<html>..."	
	t1	contents:html="<html>..."	

反向的URL

4.3.5 HBase的物理视图

列族contents

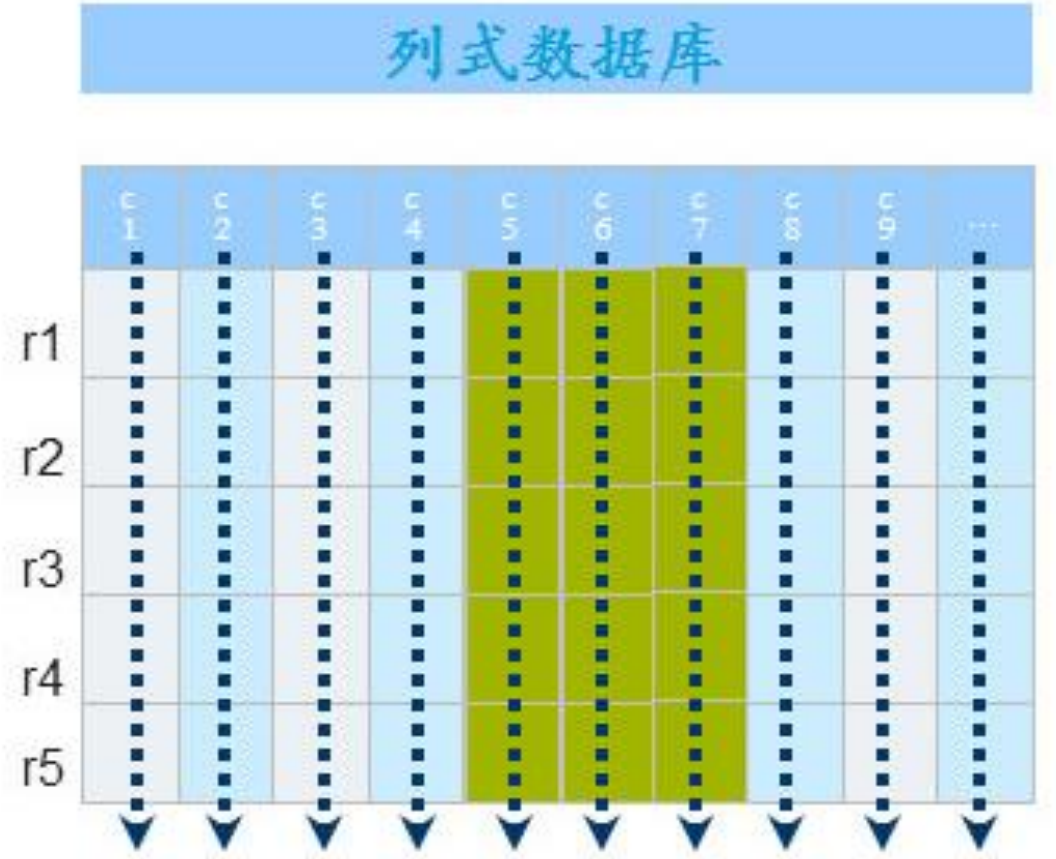
行键	时间戳	列族contents
"com.baidu.www"	t3	contents:html="<html>..."
	t2	contents:html="<html>..."
	t1	contents:html="<html>..."

列族anchor

行键	时间戳	列族anchor
"com.baidu.www"	t5	anchor:xmu.edu.cn.com=“BAIDU”
	t4	anchor:pku.edu.cn.com=“BAIDU.com"

4.3.6 面向列的存储

- **NSM存储模型和DSM存储模型**

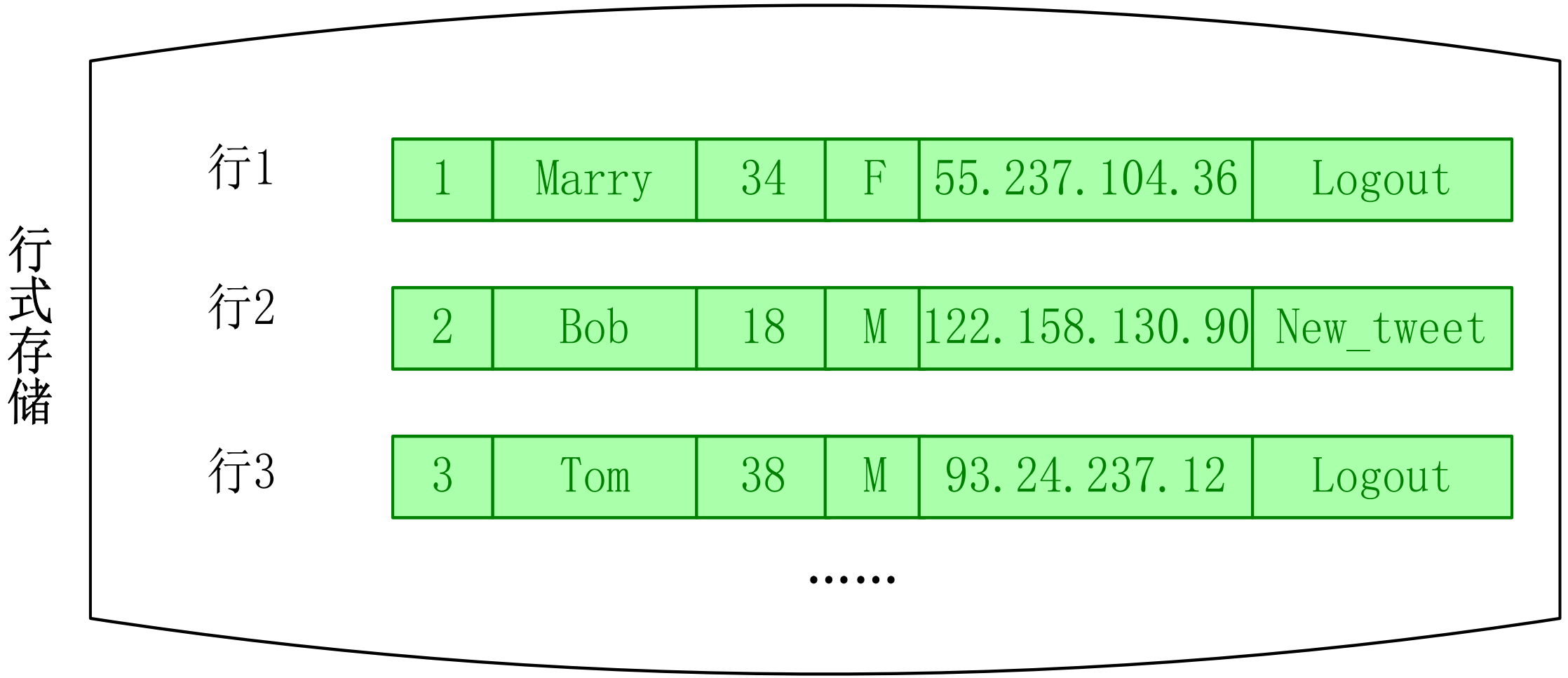


4.3.6 面向列的存储

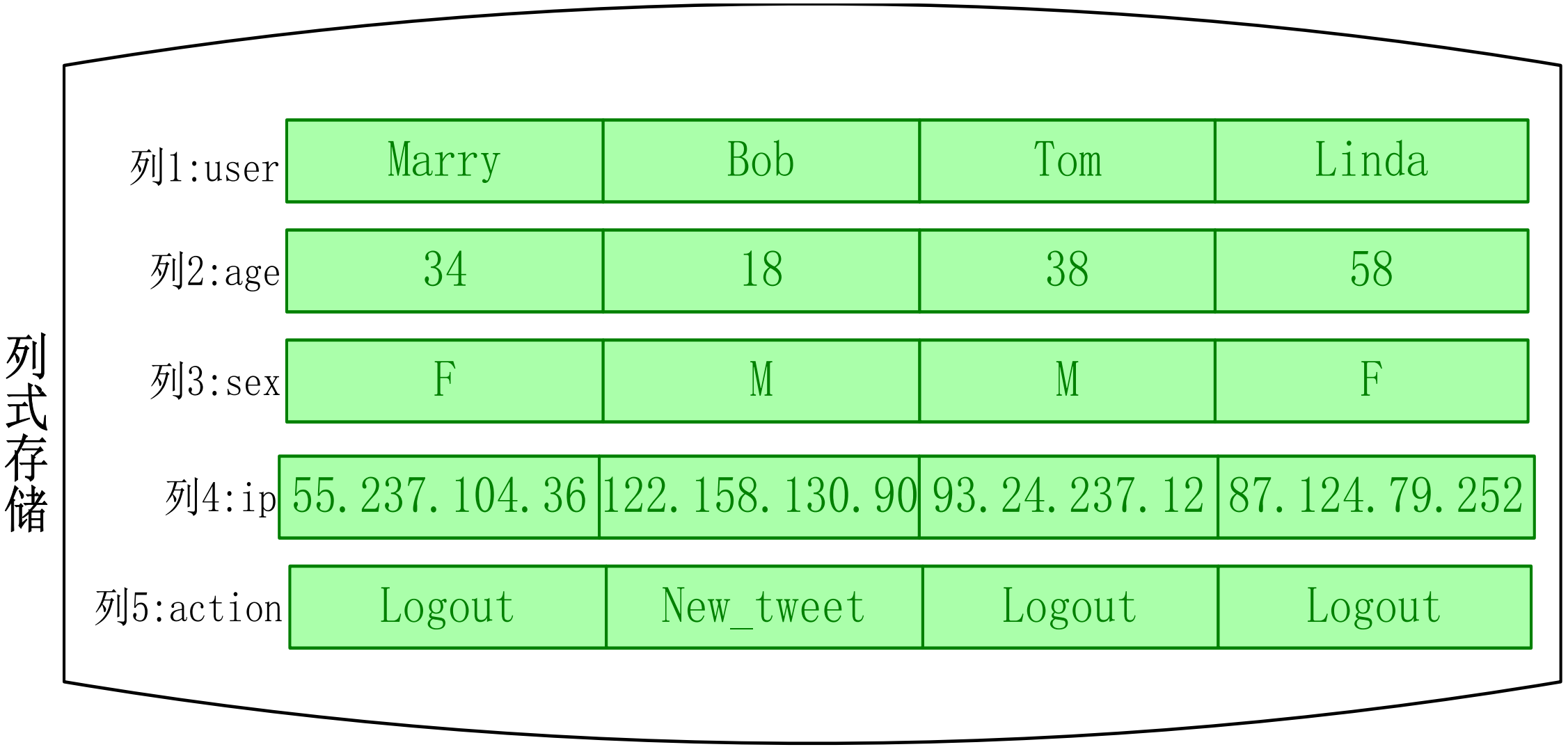
SQL模式

Log					
Log_id	user	age	sex	ip	action
1	Marry	34	F	55.237.104.36	Logout
2	Bob	18	M	122.158.130.90	New_tweet
3	Tom	38	M	93.24.237.12	Logout
4	Linda	58	F	87.124.79.252	Logout

4.3.6 面向列的存储


行式存储
行1
1
Marry
34
F
55.237.104.36
Logout
行2
2
Bob
18
M
122.158.130.90
New_tweet
行3
3
Tom
38
M
93.24.237.12
Logout
……

4.3.6 面向列的存储


列式存储
列1:user
Marry
Bob
Tom
Linda
列2:age
34
18
38
58
列3:sex
F
M
M
F
列4:ip
55.237.104.36
122.158.130.90
93.24.237.12
87.124.79.252
列5:action
Logout
New_tweet
Logout
Logout

4.4 HBase的实现原理

- 4.4.1 HBase功能组件
- 4.4.2 表和Region
- 4.4.3 Region的定位

4.4.1 HBase功能组件

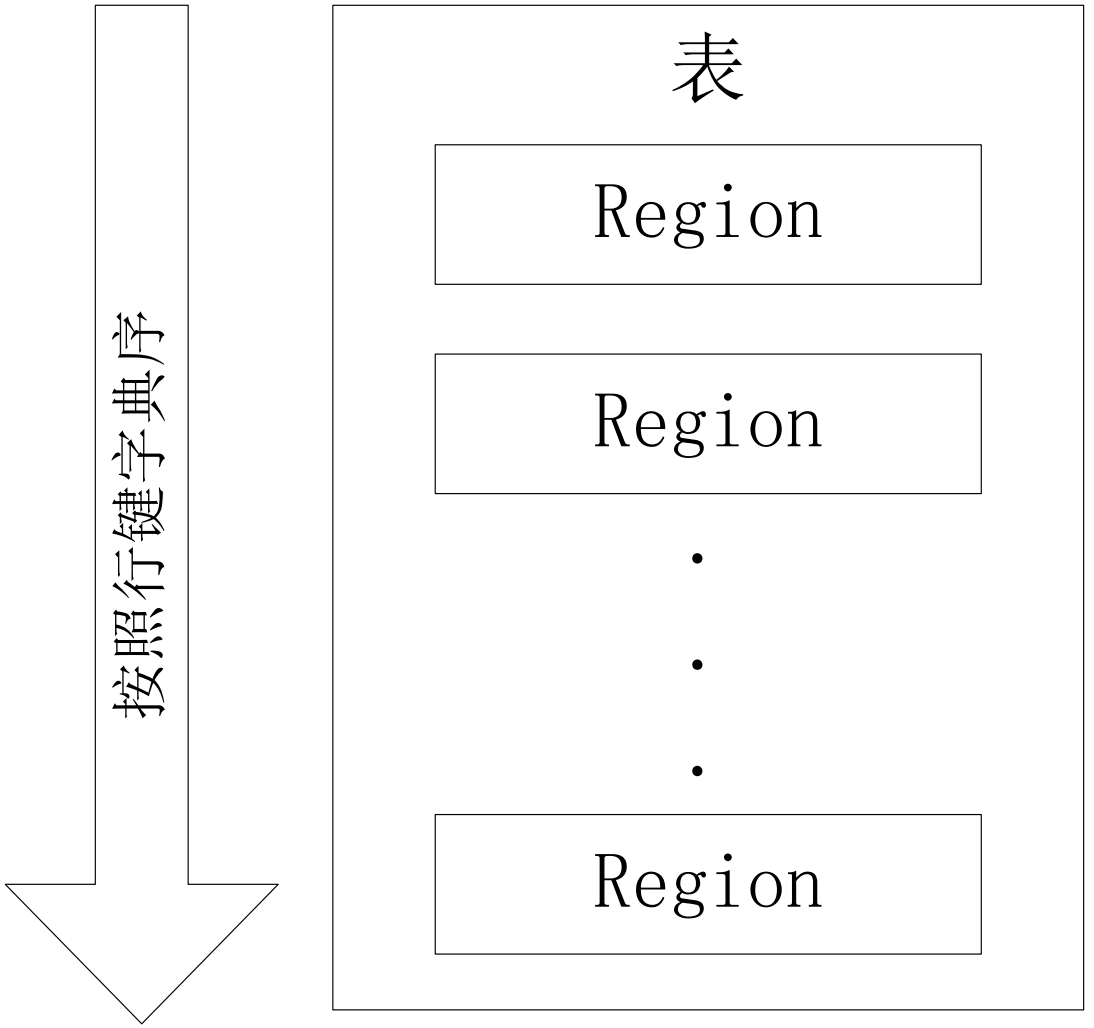
- HBase的实现包括三个主要的功能组件：
  （1）库函数：链接到每个HBase客户端
  （2）一个Master主服务器(Master Server)
  （3）许多个Region服务器(Region Servers)
- Master Server负责管理和维护HBase表的分区信息(create, delete, alter)，维护Region Server列表，分配Region，负载均衡
- Region Server负责存储和维护分配给自己的Region(splitRegion, compactRegion)，处理来自客户端的读写请求(get, put, delete)

4.4.1 HBase功能组件

- HBase客户端并不是直接从Master主服务器上读取数据，而是在获得Region的存储位置信息后，直接从Region服务器上读取数据
- 客户端并不依赖Master，而是通过Zookeeper来获得Region位置信息，大多数客户端甚至从来不和Master通信，这种设计方式使得Master负载很小（即使Master宕机，也不影响客户端对表进行操作）

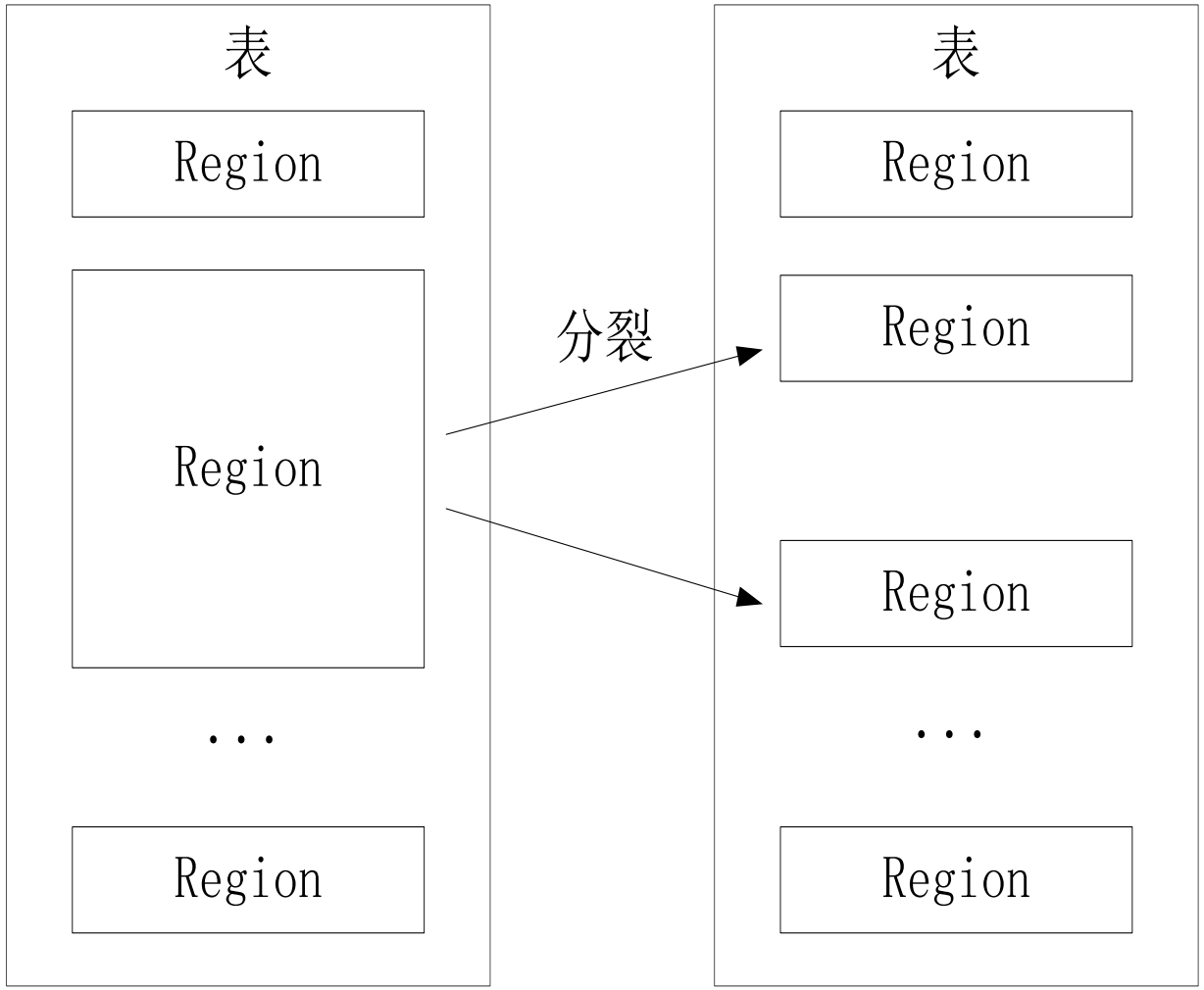
4.4.2 表和Region

- 表中的行根据行键的值的字典顺序进行维护
- 行的数量庞大，需分布存储到多台机器上
- 根据行键的值对表中的行进行分区（Region），包含了位于某个值域区间的所有数据
- Region是负载均衡和数据分发的基本单位，被分发到不同的Region服务器上



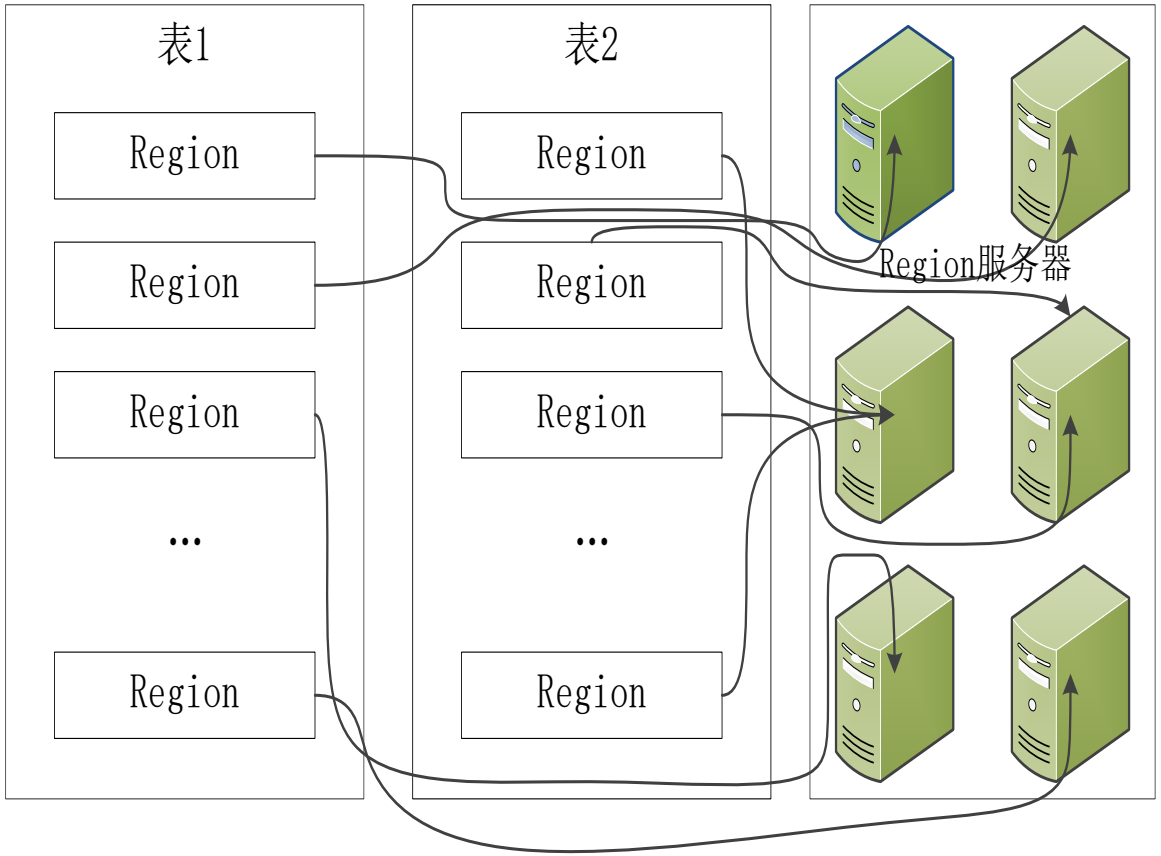
4.4.2 表和Region

- 初始时，每张表只有一个Region
- 随着数据的不断插入，Region会持续增大
- 达到一定阈值时，**会被自动等分成2个新的Region**
- ……



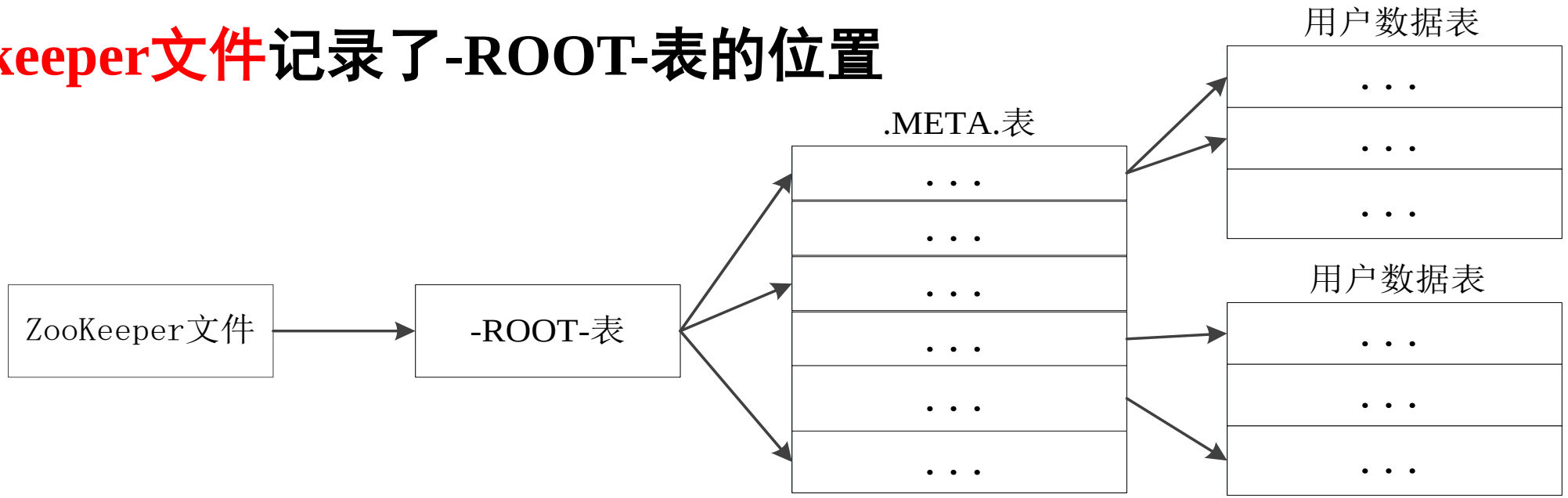
4.4.2 表和Region

- 每个Region的最佳大小取决于单台服务器的有效处理能力，目前每个Region最佳大小建议1GB~2GB
- 同一个Region不会被分拆到多个Region服务器
- 每个Region服务器存储10~1000个Region
- 不同的Region可以分布在不同的Region服务器上



4.4.3 Region的定位

- **定位机制：保证客户端知道到哪里能找到自己需要的数据**
- **元数据表，又名.META.表，存储了Region和Region服务器的映射关系，当HBase表很大时，.META.表也会被分裂成多个Region**
- **根数据表，又名-ROOT-表，记录所有元数据的具体位置,-ROOT-表只有唯一一个Region，名字是在程序中被写死的**
- **Zookeeper文件记录了-ROOT-表的位置**



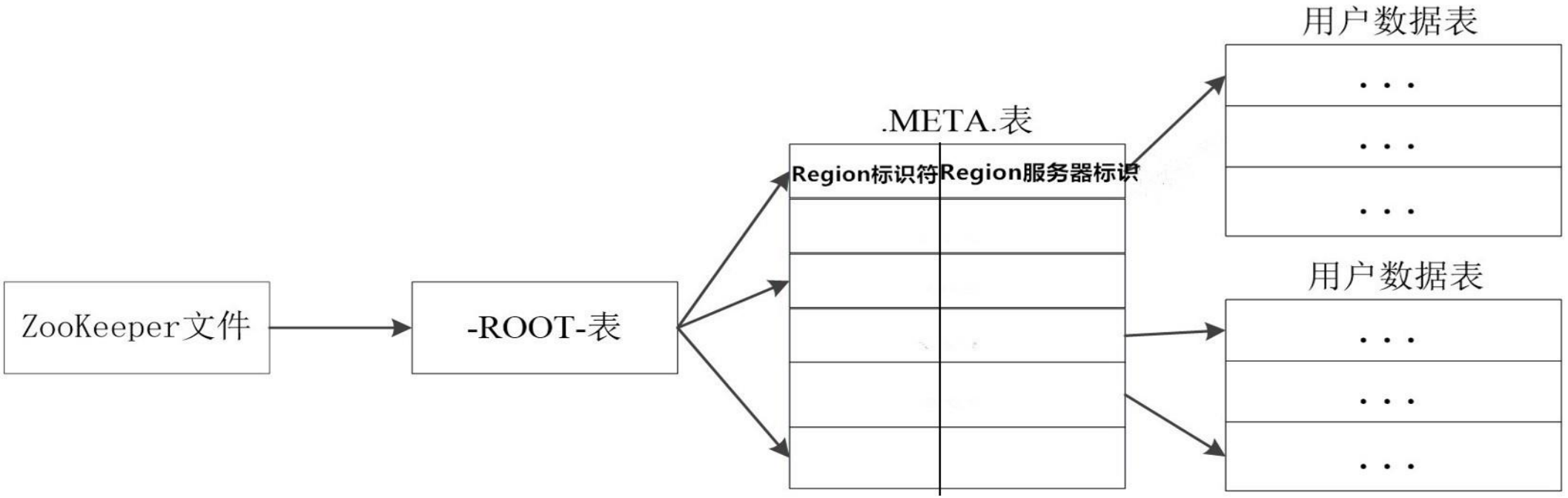
4.4.3 Region的定位

- HBase三层结构中各层次的名称和作用

层次	名称	作用
第一层	Zookeeper文件	记录了-ROOT-表的位置信息
第二层	-ROOT-表	记录了.META.表的Region位置信息 -ROOT-表只能有一个Region。通过-ROOT-表，就可以访问.META.表中的数据
第三层	.META.表	记录了用户数据表的Region位置信息， .META.表可以有多个Region，保存了HBase中所有用户数据表的Region位置信息

4.4.3 Region的定位

- 客户端访问数据时的“三级寻址”
- 为了加速寻址，客户端会缓存位置信息，同时，需要解决缓存失效问题
- 寻址过程客户端只需要询问Zookeeper服务器，不需要连接Master服务器
- 为了加快访问速度，.META.表的全部Region都被保存在内存中



思考题

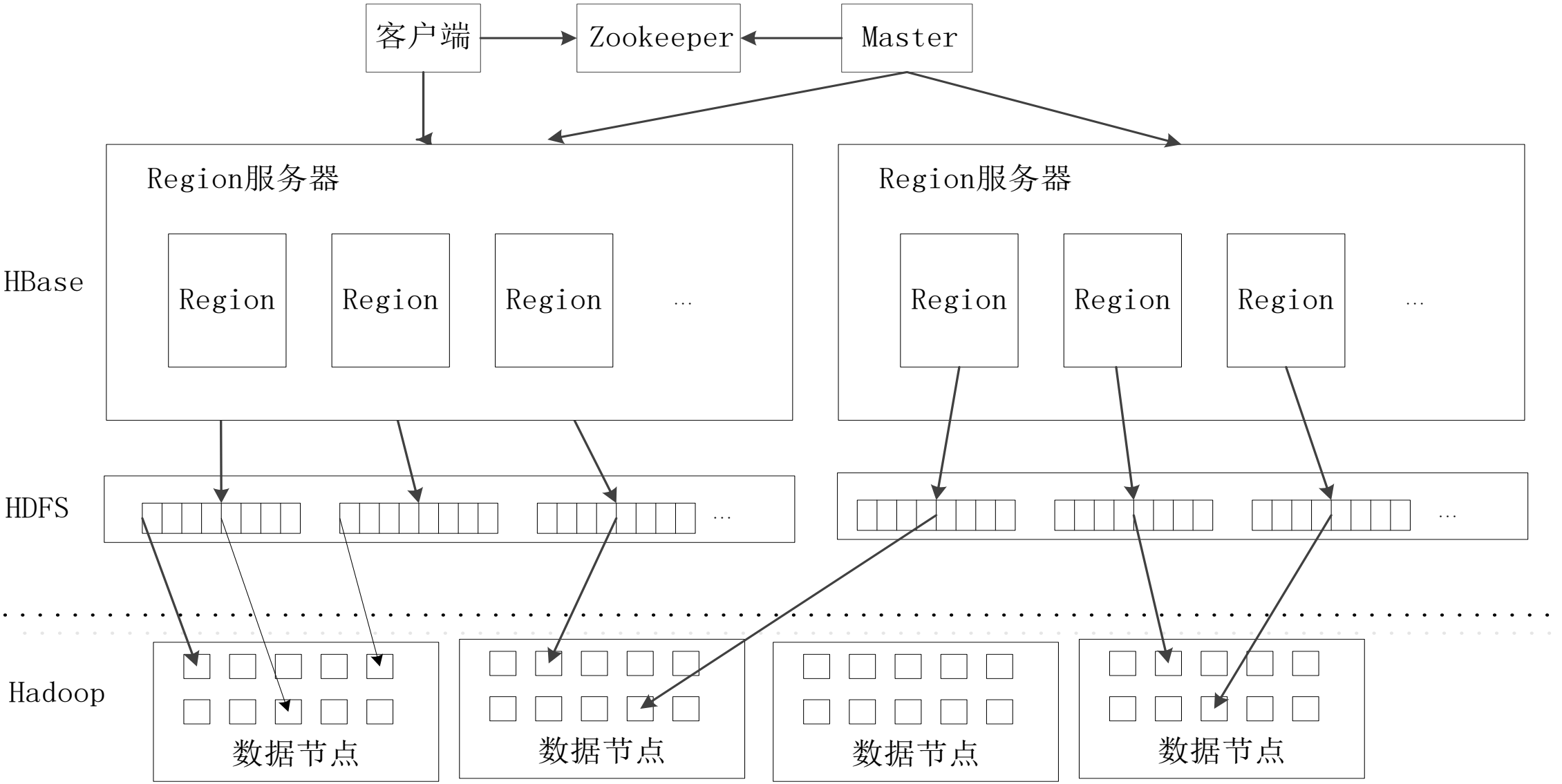
- 假设.META.表的每行（一个映射条目）在内存中大约占用1KB，并且每个Region限制为128MB，那么，这样的三层结构可以保存的用户数据表的Region数目是多少？

- 用户数据表的Region数=（-ROOT-表能够寻址的.META.表的Region个数）×（每个.META.表的 Region可以寻址的用户数据表的Region个数）
- 一个-ROOT-表最多只能有一个Region，也就是最多只能有128MB，按照每行（一个映射条目）占用1KB内存计算，128MB空间可以容纳128MB/1KB=2^{17}行，也就是说，一个-ROOT-表可以寻址2^{17}个.META.表的Region。
- 同理，每个.META.表的 Region可以寻址的用户数据表的Region个数是128MB/1KB=2^{17}。
- 最终，三层结构可以保存的Region数目是(128MB/1KB) × (128MB/1KB) = 2^{34}个Region

4.5 HBase的运行机制

- 4.5.1 HBase系统架构
- 4.5.2 Region服务器工作原理
- 4.5.3 Store工作原理
- 4.5.4 HLog工作原理

4.5.1 HBase系统架构



4.5.1 HBase系统架构

1. Client客户端

- 客户端包含访问HBase的接口，同时在缓存中维护着已经访问过的Region位置信息，用来加快后续数据访问过程
- 客户端首先访问ZooKeeper，找到-ROOT-表，再访问.META.表，最后才能找到用户的数据位置
- 使用HBase RPC机制与Master和Region Server进行通信
  - Client与Master进行管理类操作（DDL）
  - Client与RegionServer进行数据读写类操作（DML）

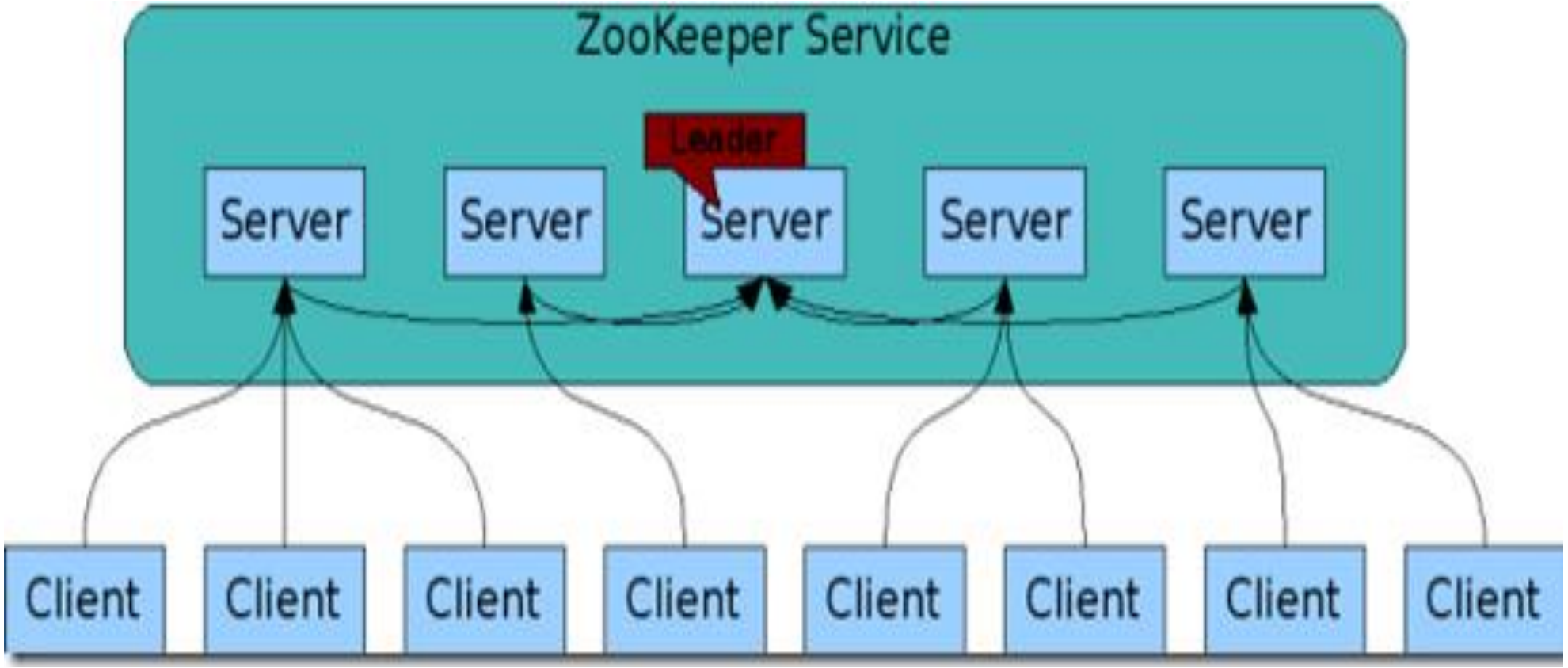
4.5.1 HBase系统架构

2. Zookeeper服务器

- Zookeeper可以帮助选举出一个Master作为集群的总管，并保证在任何时刻总有唯一一个Master在运行，避免了Master的“单点失效”问题
- Zookeeper存储了-ROOT-表地址、Master地址
- Zookeeper存储了HBase的管理模式，包括有哪些表、每个表有哪些列族
- 实时监控RegionServer的状态，将RegionServer的上线和下线信息实时通知给Master

4.5.1 HBase系统架构

- Zookeeper是一个很好的集群管理工具，被大量用于分布式计算，提供配置维护、域名服务、分布式同步、组服务等。



4.5.1 HBase系统架构

3. Master主服务器

- 主服务器Master主要负责表和Region的管理工作：
  - 为RegionServer分配Region
  - 管理用户对表的增加、删除、修改、查询等操作
  - 实现不同RegionServer之间的负载均衡
  - 在Region分裂或合并后，负责重新调整Region的分布
  - 对发生故障失效的RegionServer上的Region进行迁移
  - HDFS上垃圾文件（HBase）的回收

4.5.1 HBase系统架构

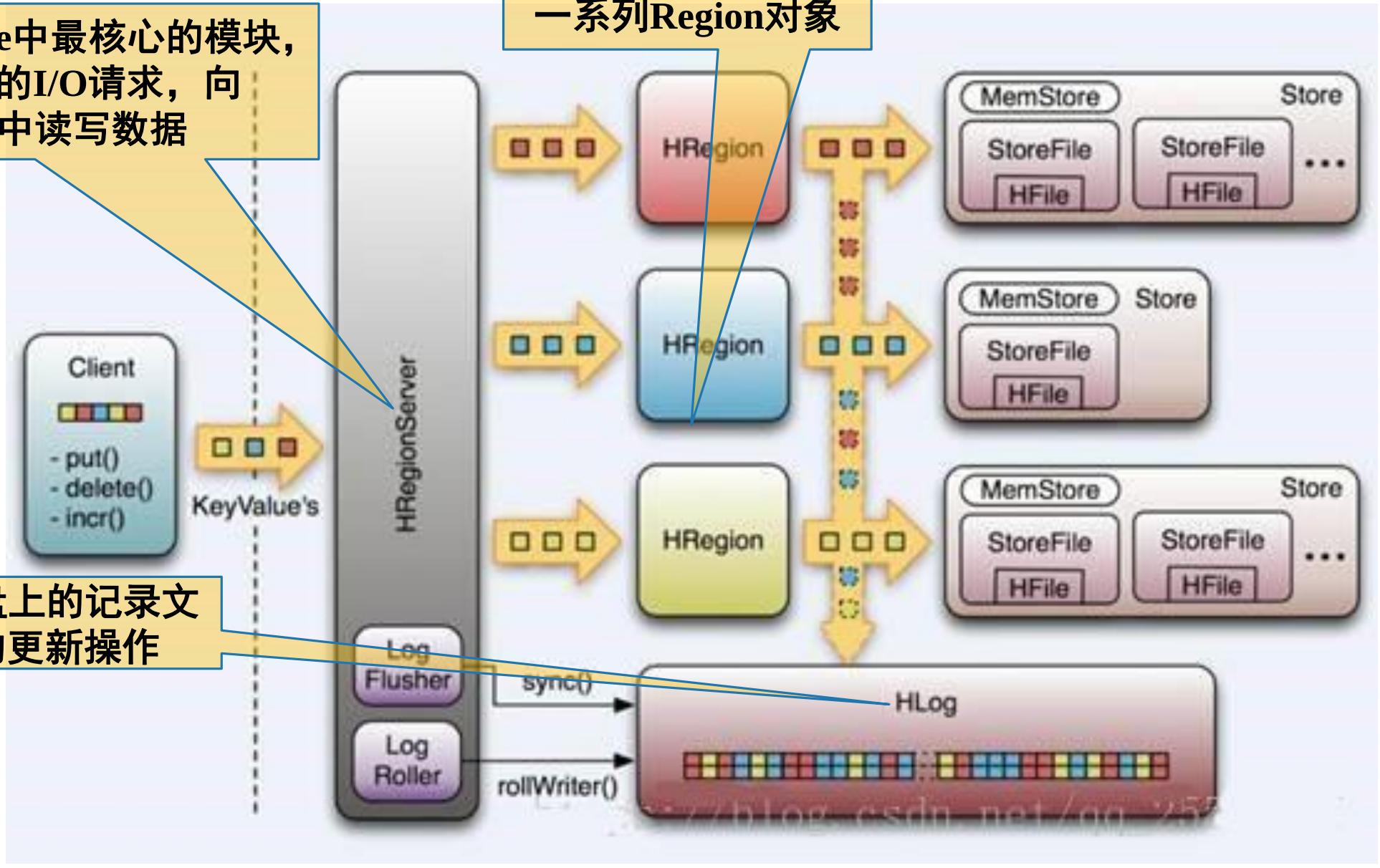
4. Region服务器

- Region服务器是HBase中最核心的模块
- 负责维护分配给自己的Region
- 响应用户的I/O请求，向HDFS文件系统中读写
- 负责Split在运行过程中变得过大的Region，负责Compact操作

4.5.2 Region服务器的工作原理

Region服务器是HBase中最核心的模块，主要负责响应用户的I/O请求，向HDFS文件系统中读写数据

一系列Region对象

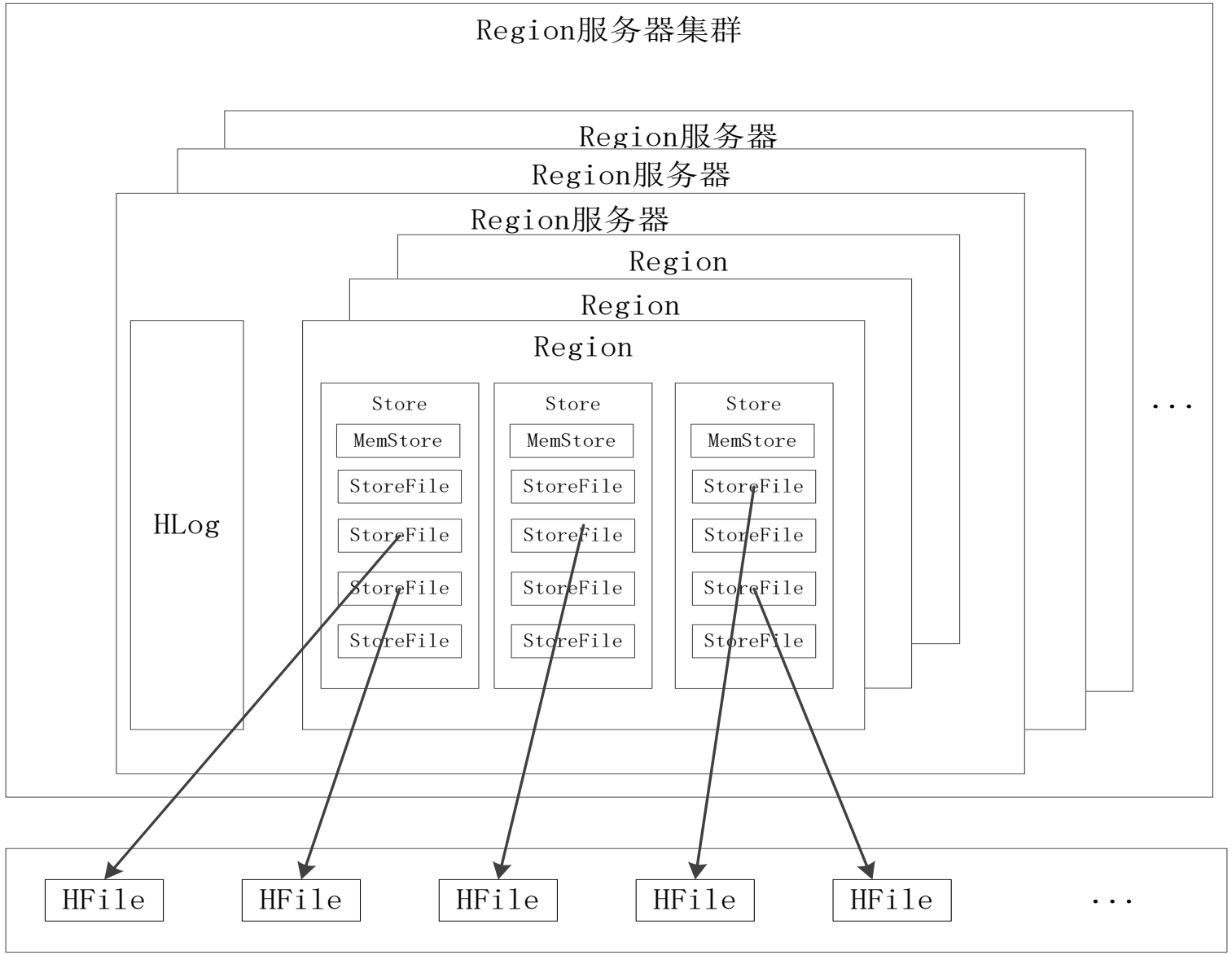


一个Hlog文件，磁盘上的记录文件，记录着所有的更新操作

4.5.2 Region服务器的工作原理

- Region服务器内部管理的对象
  - 一系列Region对象
    - 多个Store——一个Store对应了表中一个列族的存储
      - 一个MemStore（内存中的缓存，保存最新更新的数据）
      - 若干个StoreFile（磁盘中的文件，方便快速读取，在底层的实现方式是HDFS文件系统的HFile）
  - 一个HLog文件（磁盘上的记录文件，记录着所有的更新操作）

4.5.2 Region服务器的工作原理



1. 用户读写数据过程
2. 缓存的刷新
3. StoreFile的合并

4.5.2 Region服务器的工作原理

1、用户读写数据的过程

- 写入数据时：
  - 被分配到相应的Region服务器去执行
  - 用户数据首先被写入到MemStore和HLog中
  - 只有当操作写入HLog之后，commit()调用才会将其返回给客户端
- 读取数据时：
  - Region服务器会首先访问MemStore缓存
  - 如果找不到，再去磁盘上的StoreFile中寻找

4.5.2 Region服务器的工作原理

2、缓存的刷新

- 系统会周期性地把MemStore缓存里的内容刷写到磁盘的StoreFile文件中，清空缓存，并在HLog里面写入一个标记
- 每次刷写都生成一个新的StoreFile文件，因此，每个Store包含多个StoreFile文件
- 每个Region服务器都有一个自己的HLog 文件，每次启动都检查该文件，确认最近一次执行缓存刷新操作之后是否发生新的写入操作；如果发现更新，则先写入MemStore，再刷写到StoreFile，最后删除旧的HLog文件，开始为用户提供服务

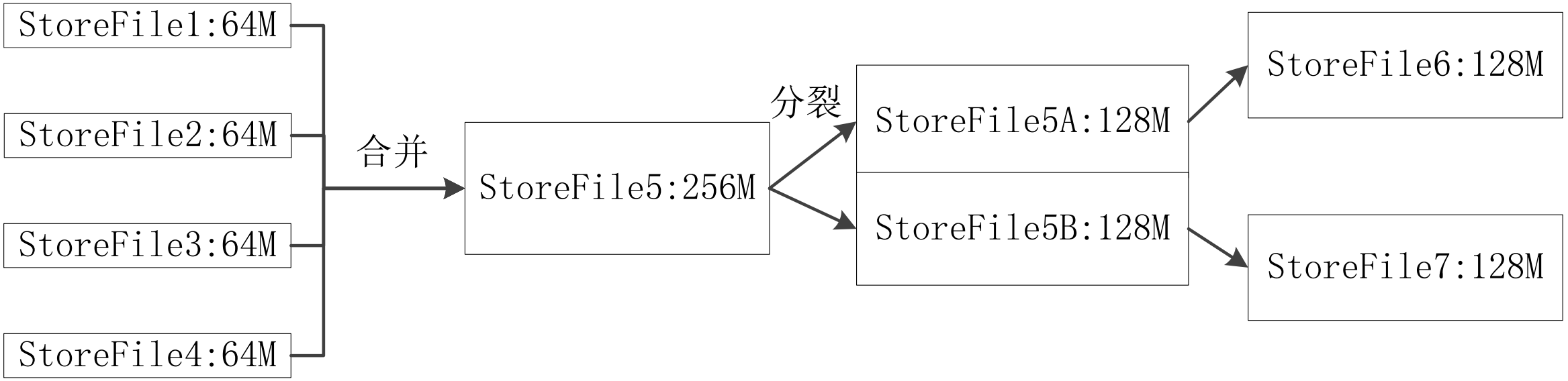
4.5.2 Region服务器的工作原理

3、StoreFile的合并

- 每次MemStore缓存的刷新操作都生成一个新的StoreFile，数量太多，影响查找速度
- 为了减少查找时间，系统会调用Store. compact()把多个StoreFile合并成一个大文件
- 合并操作比较耗费资源，只有在StoreFile文件的数量达到一个阈值才启动合并

4.5.3 Store的工作原理

- Store是Region服务器的核心
- 多个StoreFile合并成一个大的文件
- 单个StoreFile超过一定阈值时，又触发分裂操作，1个父Region被分裂成两个子Region



4.5.4 HLog的工作原理

- 分布式环境必须要考虑系统出错。HBase采用HLog保证系统发生故障时能够恢复到正确的状态
- HBase系统为每个Region服务器配置了一个HLog文件，它是一种预写式日志WAL（Write Ahead Log）
- 用户更新数据必须首先写入日志后，才能写入MemStore缓存，并且，直到MemStore缓存内容对应的日志已经写入磁盘，该缓存内容才能被刷写到磁盘

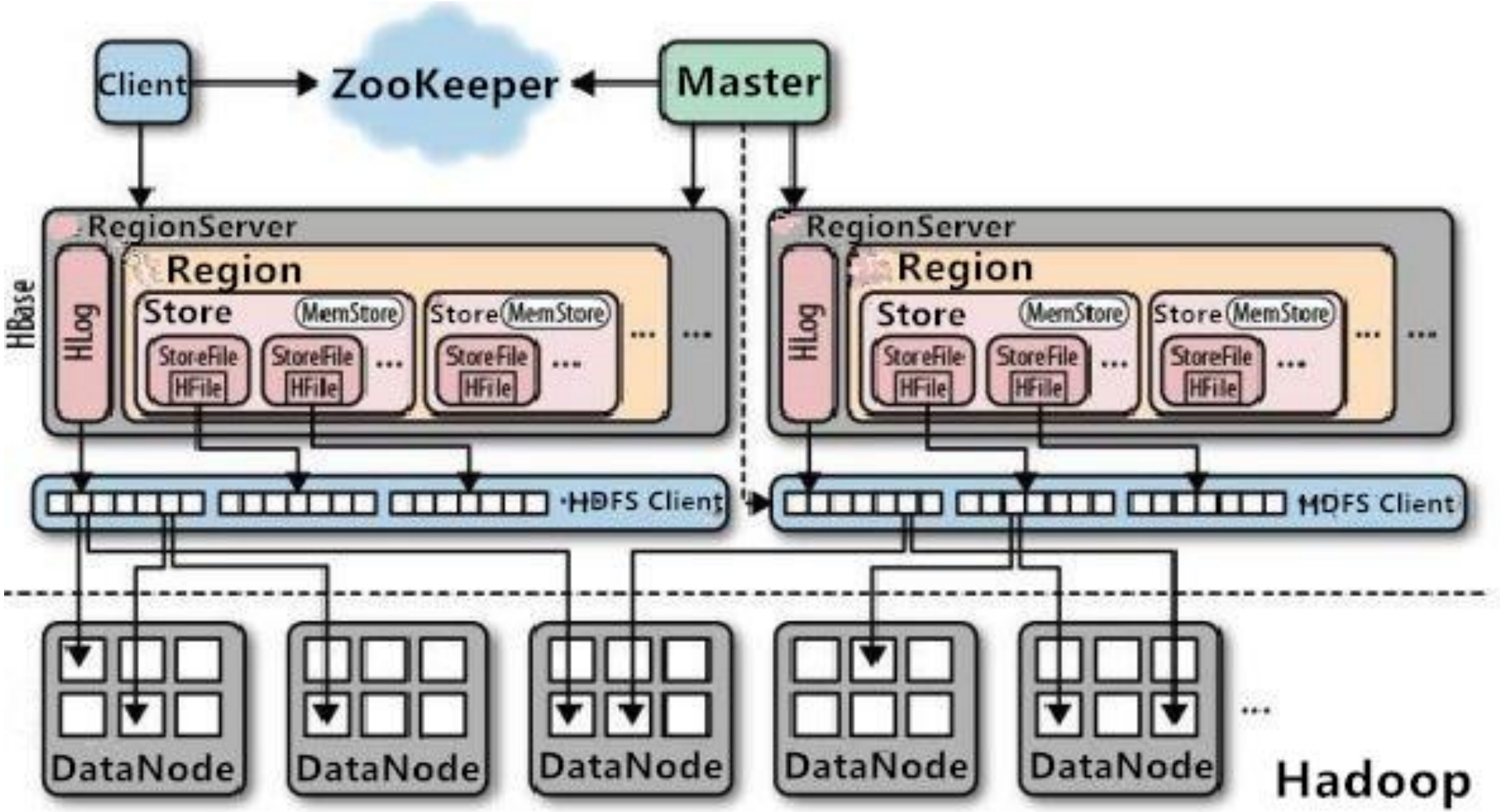
4.5.4 HLog的工作原理

- 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服务器上面遗留的HLog文件，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，分别放到相应Region对象的目录下，然后，再将失效的Region重新分配到可用的Region服务器中，并把与该Region对象相关的HLog日志记录也发送给相应的Region服务器

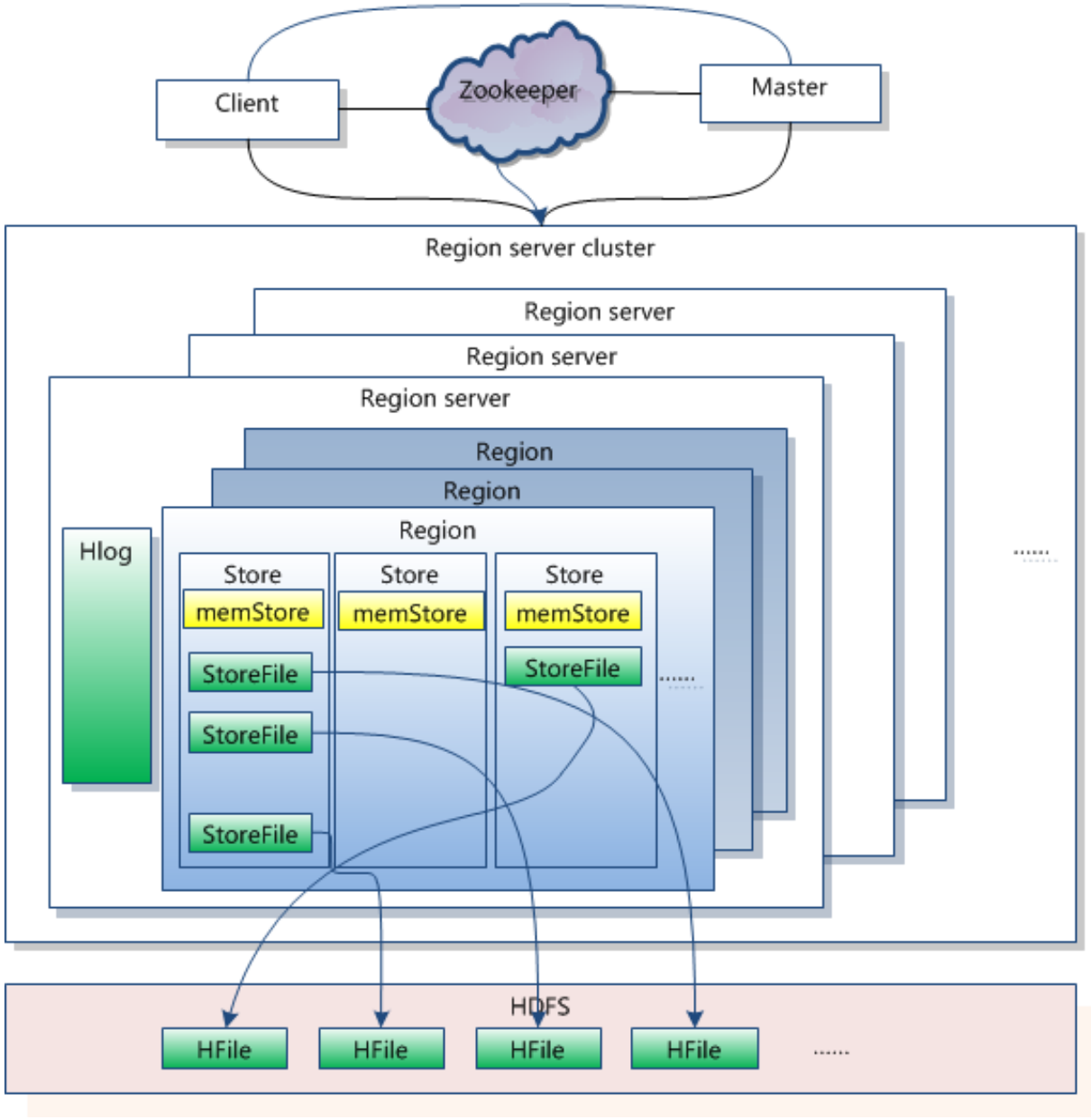
4.5.4 HLog的工作原理

- Region服务器领取到分配给自己的Region对象以及与之相关的HLog日志记录以后，会重新做一遍日志记录中的各种操作，把日志记录中的数据写入到MemStore缓存中，然后，刷新到磁盘的StoreFile文件中，完成数据恢复
- 共用日志优点：提高对表的写操作性能；缺点：恢复时需要拆分日志

HBase的存储结构


Client
ZooKeeper
Master
HBase
RegionServer
Region
HLog
Store
MemStore
StoreFile
HFile
HDFS Client
DataNode
Hadoop

HBase的存储结构



- Client
- Zookeeper
- Master
- 多个Region Server
  - 10~1000个Region
    - 若干个Store
      - 1个memStore
      - 0个或多个StoreFile（是实际数据存储文件HFile的轻量级封装）
  - 1个HLog

4.6 HBase的编程实现

- 4.6.1　HBase的安装与配置
- 4.6.2　HBase常用Shell命令
- 4.6.3　HBase常用Java API及应用实例

https://dblab.xmu.edu.cn/blog/2442/

本章小结

- 本章详细介绍了HBase数据库的知识。HBase数据库是BigTable的开源实现，和BigTable一样，支持大规模海量数据，分布式并发数据处理效率极高，易于扩展且支持动态伸缩，适用于廉价设备
- HBase可以支持Native Java API、HBase Shell、Thrift Gateway、REST Gateway、Pig、Hive等多种访问接口，可以根据具体应用场合选择相应访问方式
- HBase实际上是一个稀疏、多维、持久化存储的映射表，它采用行键、列族和时间戳进行索引，每个值都是未经解释的字符串。介绍了HBase数据在概念视图和物理视图中的差别

本章小结

- HBase采用分区存储，一张大的表会被拆分许多个Region，这些Region会被分发到不同的服务器上实现分布式存储
- HBase的系统架构包括客户端、Zookeeper服务器、Master主服务器、Region服务器。客户端包含访问HBase的接口；Zookeeper服务器负责提供稳定可靠的协同服务；Master主服务器主要负责表和Region的管理工作；Region服务器负责维护分配给自己的Region，并响应用户的读写请求
- 本章最后详细介绍了HBase运行机制和编程实践的知识