复 习

HDFS，GFS，Blob FS

- 业务需求和设计目的
- 体系架构及其特点
- 具体市场应用场景和适用性
- 大数据分布式系统的未来发展趋势

大数据技术及应用

第四章 分布式数据库HBase

提纲

4.1 概述

4.2 HBase访问接口

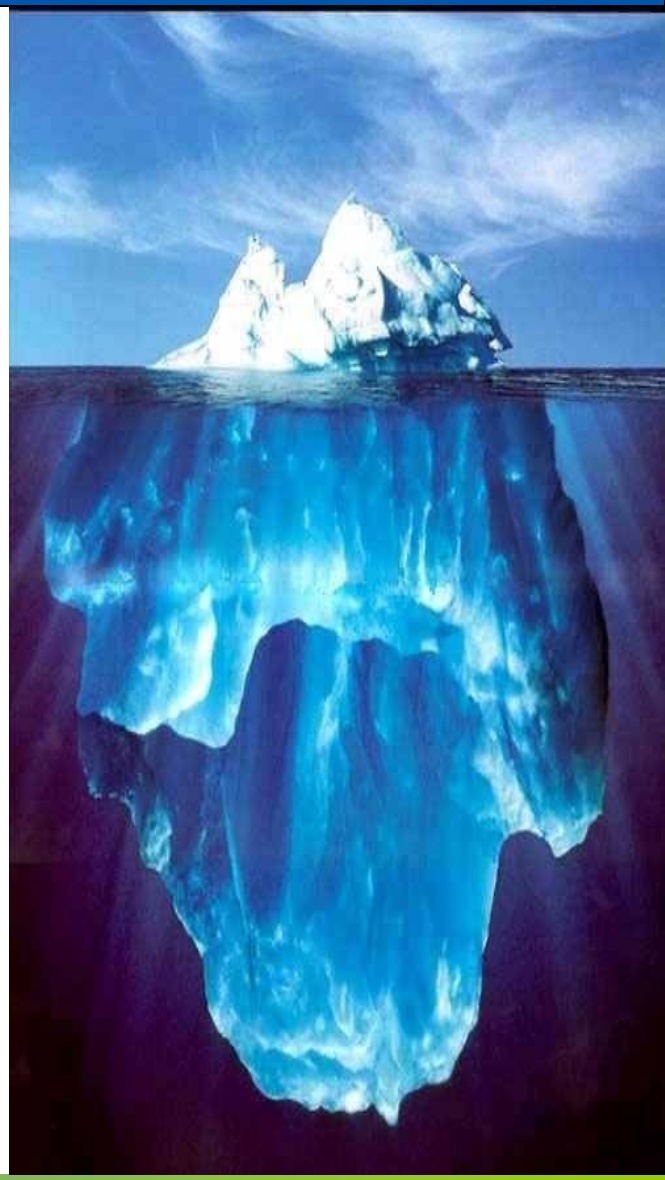
4.3 HBase数据模型

4.4 HBase的实现原理

4.5 HBase运行机制

4.6 HBase应用方案

4.7 HBase编程实践



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

主流解决方案——Google云计算

Google云计算：以用户数据为中心

- 数据存储在“云”中
- 数据访问不受地理位置限制
- 数据能够很方便的共享

主流解决方案——Google云计算

Google云计算技术具体包括：

- **Google文件系统海量数据分布存储技术(GFS)、**
- **分布式计算编程模型MapReduce、**
- **分布式锁服务Chubby**
- **分布式结构化数据存储系统Bigtable等。**

分布式结构化数据表Bigtable

设计动机

需要存储的数据种类繁多：Google目前向公众开放的服务很多，需要处理的数据类型也非常多。包括URL、网页内容、用户的个性化设置在内的数据都是Google需要经常处理的

海量的服务请求：Google运行着目前世界上最繁忙的系统，它每时每刻处理的客户服务请求数量是普通的系统根本无法承受的

商用数据库无法满足Google的需求：一方面现有商用数据库设计着眼点在于通用性，根本无法满足Google的苛刻服务要求；另一方面对于底层系统的完全掌控会给后期的系统维护、升级带来极大的便利

4.1.1从BigTable说起

- BigTable是一个分布式存储系统，起初用于解决典型的互联网搜索问题，
- 利用谷歌提出的MapReduce分布式并行计算模型来处理海量数据，
- 使用GFS作为底层数据存储，采用Chubby提供协同服务管理，
- 可以扩展到PB级别的数据和上千台机器，具备广泛应用性、可扩展性、高性能和高可用性等特点。

建立互联网索引

1 爬虫持续不断地抓取新页面，这些页面每页一行地存储到BigTable里

2 MapReduce计算作业运行在整张表上，生成索引，为网络搜索应用做准备

搜索互联网

3 用户发起网络搜索请求

4 网络搜索应用查询建立好的索引，从BigTable得到网页

5 搜索结果提交给用户

数据模型

- Bigtable是一个分布式多维映射表，表中的数据通过一个行关键字（Row Key）、一个列关键字（Column Key）以及一个时间戳（Time Stamp）进行索引
- Bigtable对存储在其中的数据不做任何解析，一律看做字符串
- Bigtable的存储逻辑可以表示为：

(row:string, column:string, time:int64)→string

数据模型

➢**行:**行是表的第一级索引，可以把该行的列、时间和值看成一个整体，简化为一维键值映射。

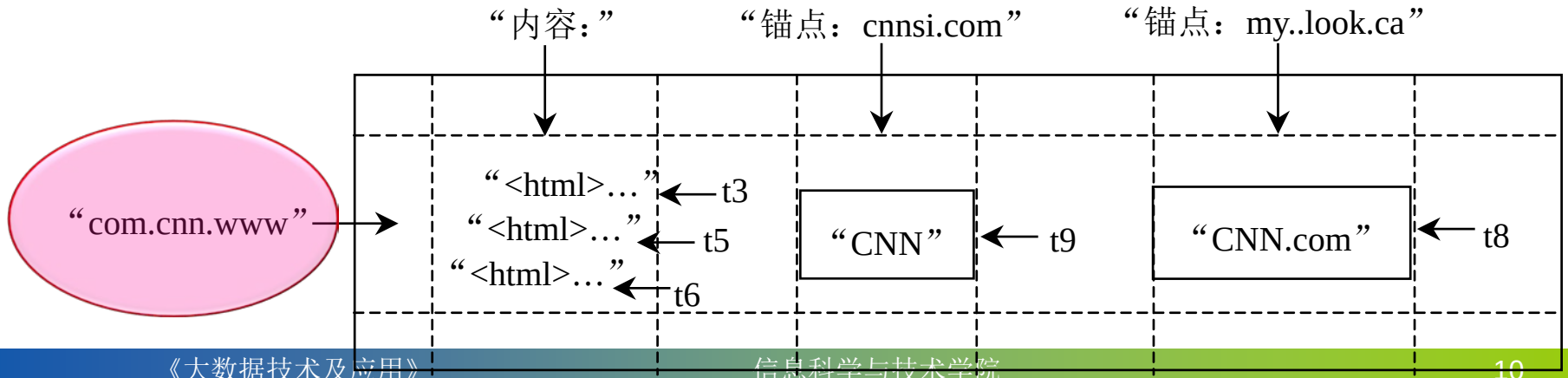
●Bigtable的行关键字可以是任意的字符串，但是大小不能超过64KB。Bigtable和传统的关系型数据库有很大不同，它不支持一般意义上的事务，但能保证对于行的读写操作具有原子性（Atomic）

●表中数据都是根据行关键字进行排序的，排序使用的是词典序。

●一个典型实例，其中com.cnn.www就是一个行关键字。不直接存储网页地址而将其倒排是Bigtable的一个巧妙设计。带来两个好处：

☞同一地址域的网页会被存储在表中的连续位置，有利于用户查找和分析

☞倒排便于数据压缩，可以大幅提高压缩率



由于规模的问题，单个的大表不利于数据处理，因此Bigtable将一个表分成了多个子表，每个子表包含多个行。

子表是Bigtable中**数据划分和负载均衡的基本单位**。

➢**列:**列是第二级索引，每行拥有的列是不受限制的，可以随时增加减少。

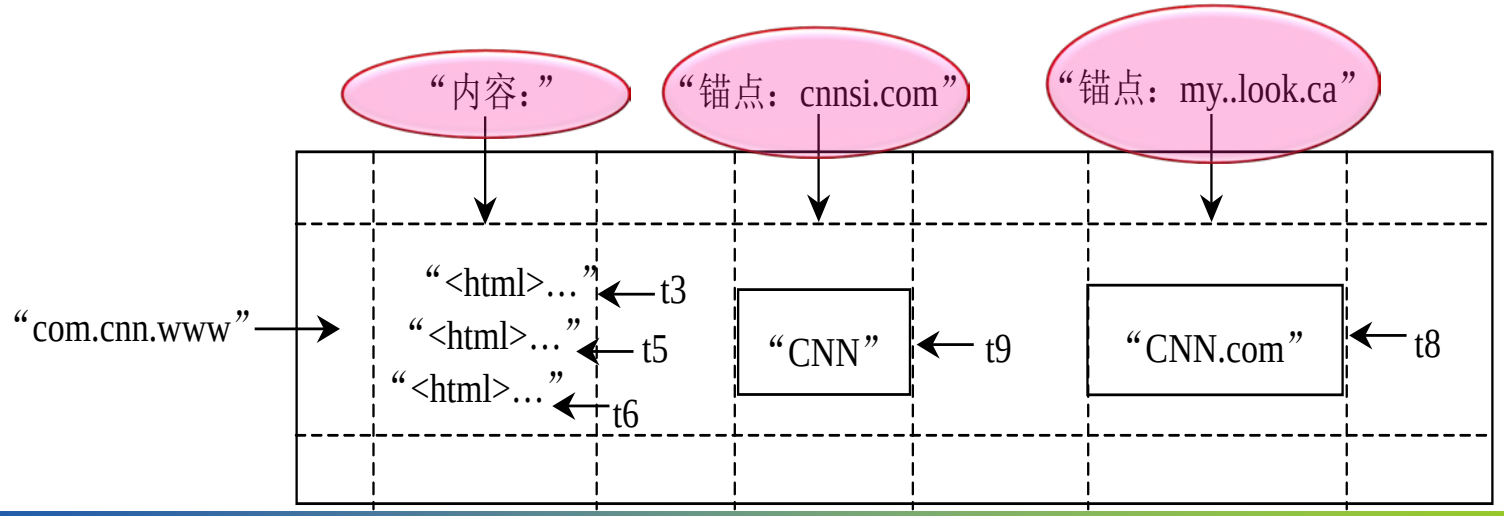
●Bigtable并不是简单地存储所有的列关键字，为了方便管理，列被分为多个列族，一个列族里的列一般存储相同类型的数据。一行的列族很少变化，但是列族里的列可以随意添加删除。同族的数据会被**压缩在一起保存**。引入了列族的概念之后，列关键字就采用下述的语法规则来定义：

●族名：限定词（family：qualifier)

☞族名必须有意义，限定词则可以任意选定

☞图中，内容、锚点都是不同的族。而cnnsi.com和my.look.ca则是锚点族中不同的限定词

☞**族同时也是Bigtable中访问控制（Access Control）基本单元，也就是说访问权限的设置是在族这一级别上进行的**



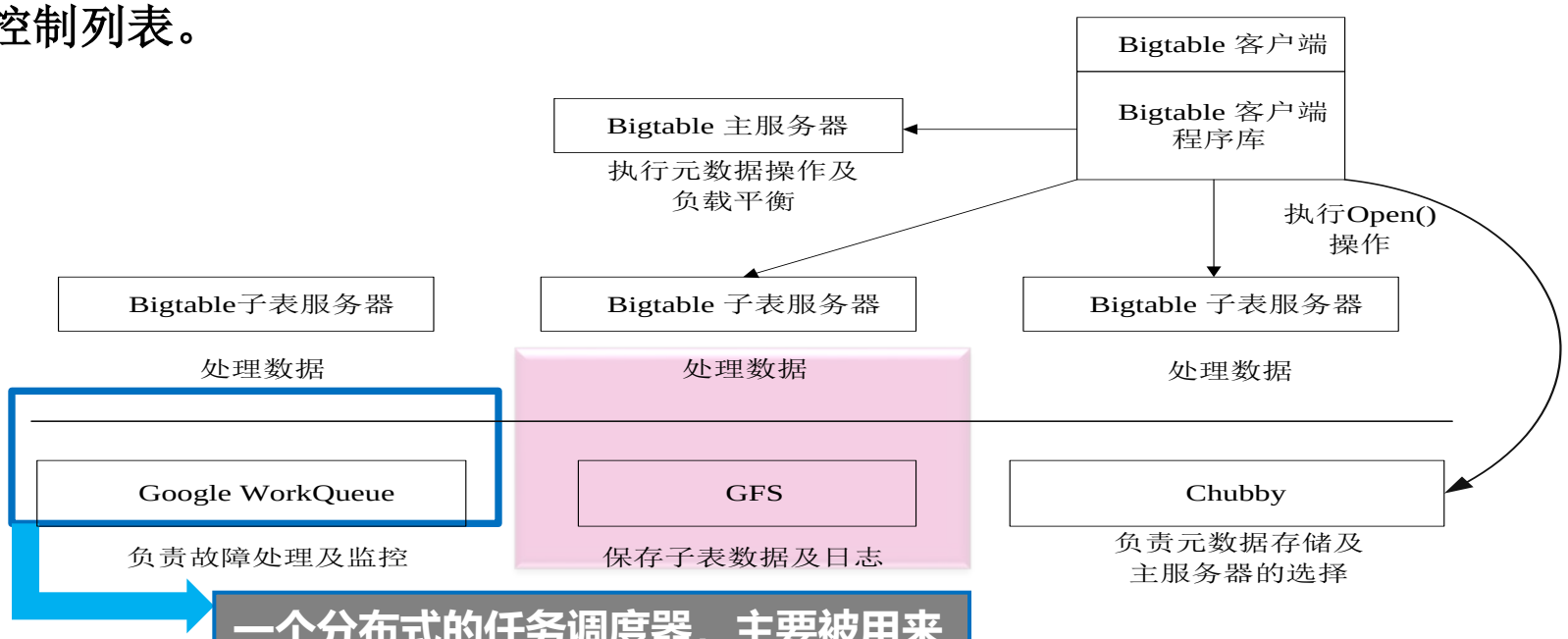
➢**时间戳**:时间戳是第三级索引。

为了简化不同版本的数据管理，Bigtable目前提供了两种设置：

- 一种是保留最近的*N*个不同版本，图中数据模型采取的就是这种方法，它保存最新的三个版本数据。
- 另一种就是保留限定时间内的所有不同版本，比如可以保存最近10天的所有不同版本数据。失效的版本将会由Bigtable的垃圾回收机制自动处理.

Google的很多服务比如网页检索和用户的个性化设置等都需要保存不同时间的数据，这些不同的数据版本必须通过时间戳来区分。

Bigtable数据库的架构，由主服务器和分服务器构成，把数据库看成是一张大表，那么可将其划分为许多基本的小表，这些小表就称为tablet，是bigtable中最小的处理单位了。
主服务器负责将Tablet分配到Tablet服务器、检测新增和过期的Tablet服务器、平衡Tablet服务器之间的负载、GFS垃圾文件的回收、数据模式的改变（例如创建表）等。
Tablet服务器负责处理数据的读写，并在Tablet规模过大时进行拆分。
Bigtable使用集群管理系统来调度任务、管理资源、监测服务器状态并处理服务器故障。
BigTable将数据存储分为两部分：最近的更新存储在内存memtable中，较老的更新则以SSTable的格式存储在GFS，后者是主体部分，不可变的数据结构。写操作的内容插入到memtable中，当memtable的大小达到一个阈值时就冻结，然后创建一个新的memtable，旧的就转换成一个SSTable写入GFS。
使用分布式的锁服务Chubby来保证集群中主服务器的唯一性、保存Bigtable数据的引导区位置、发现Tablet服务器并处理Tablet服务器的失效、保存Bigtable的数据模式信息、保存存取控制列表。

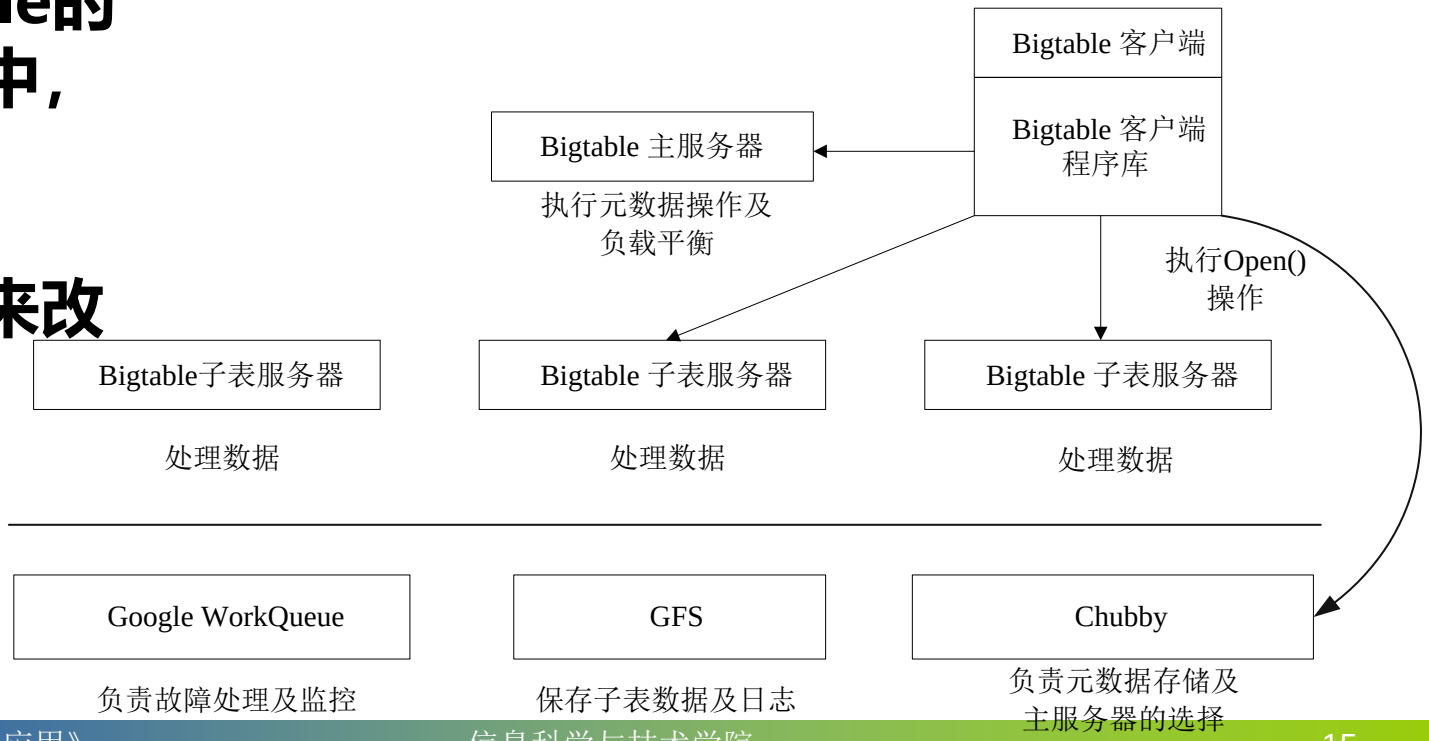


系统架构

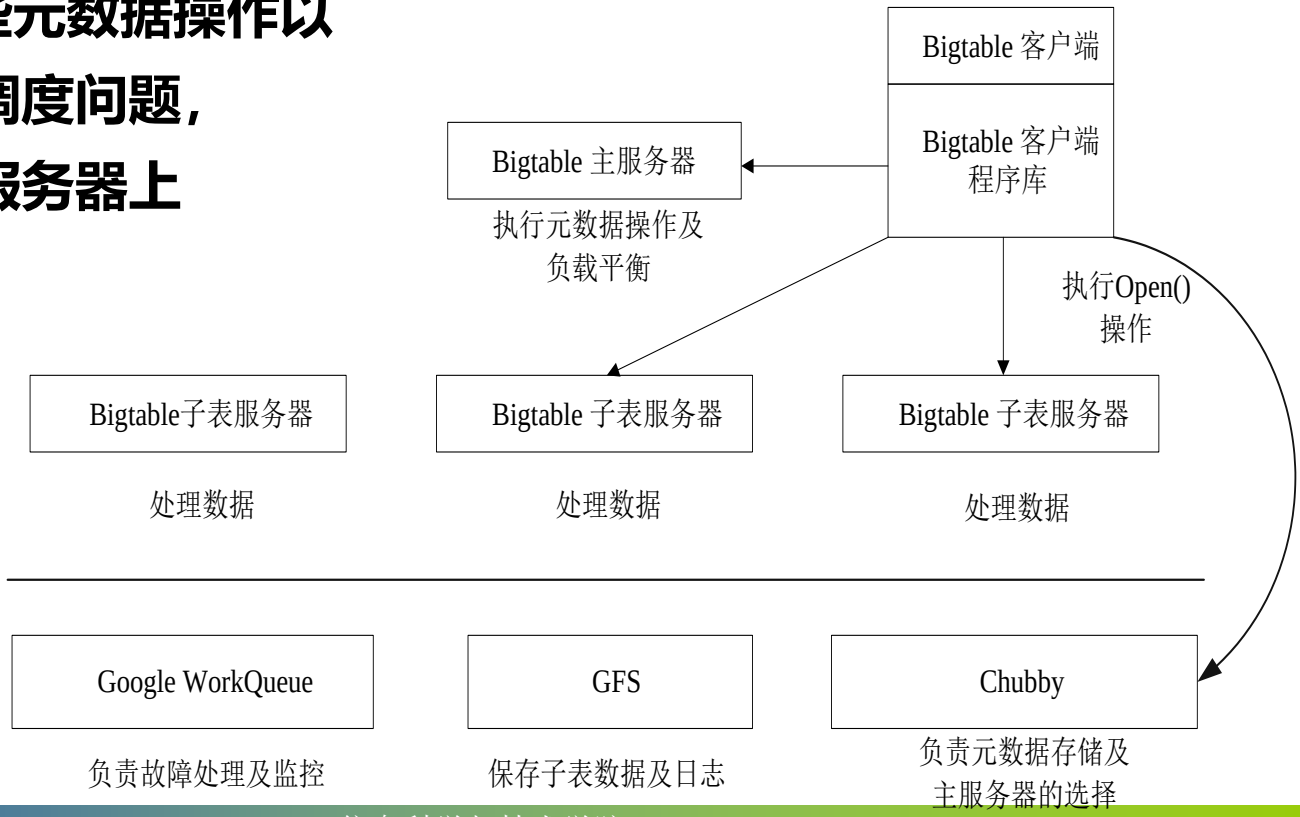
- 在Bigtable中Chubby主要有以下几个作用：

1. 选取并保证同一时间内只有一个主服务器（Master Server）
2. 获取子表的位置信息
3. 保存Bigtable的模式信息及访问控制列表

另外在Bigtable的实际执行过程中，Google的MapReduce等技术也被用来改善其性能

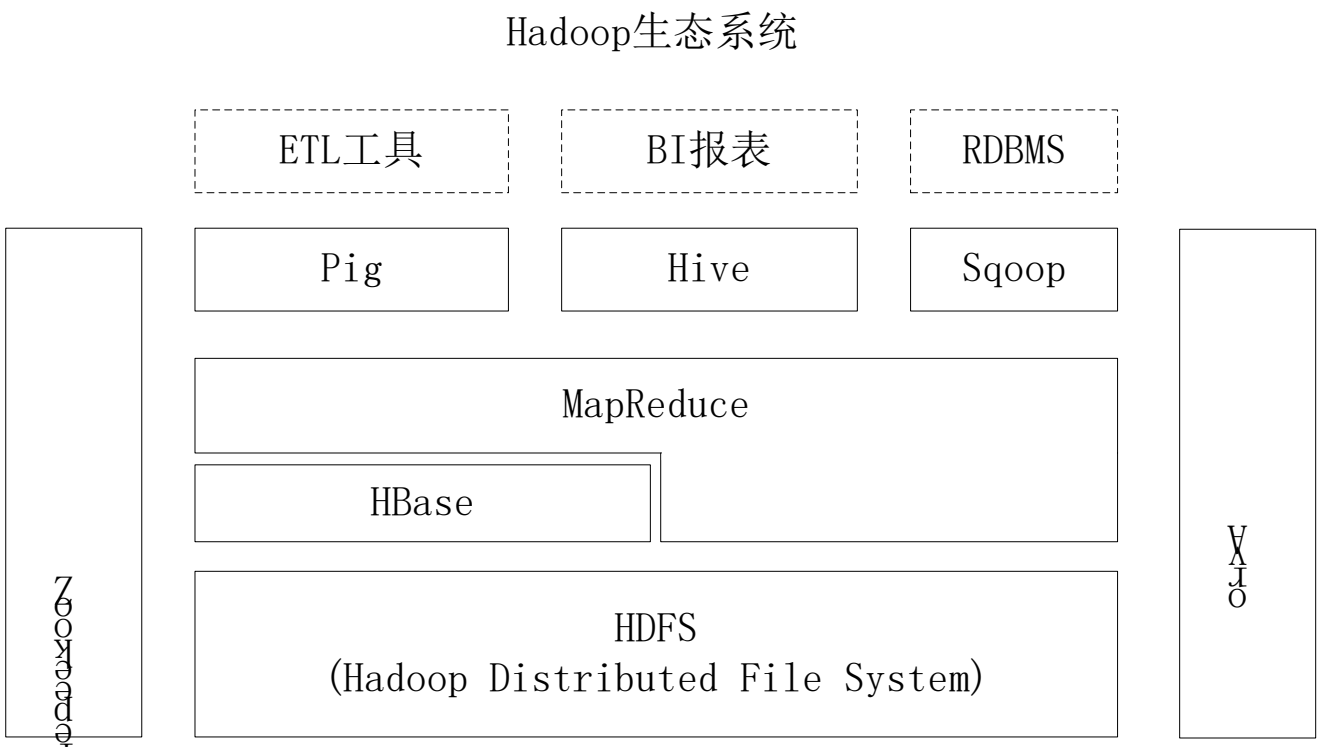


- Bigtable主要由三个部分组成：客户端程序库（Client Library），一个主服务器（Master Server）和多个子表服务器（Tablet Server）
  - 客户访问Bigtable服务时，首先要利用其库函数执行Open()操作来打开一个锁（实际上就是**获取了文件目录**），锁打开以后客户端就可以和子表服务器进行通信
  - 和许多具有单个主节点分布式系统一样，客户端主要与子表服务器通信，几乎不和主服务器进行通信，这使得主服务器的**负载大大降低**
  - 主服务主要进行一些元数据操作以及子表服务器之间负载调度问题，实际数据是存储在子表服务器上



4.1.2HBase简介

HBase是一个高可靠、高性能、面向列、可伸缩的分布式数据库，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



Hadoop生态系统中HBase与其他部分的关系

4.1.2HBase简介

	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

HBase和BigTable的底层技术对应关系

4.1.2HBase简介

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，但受限于**Hadoop MapReduce**编程框架的高延迟数据处理机制，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（分库分表也不能很好解决）
- 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，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**已经成功应用于互联网服务领域和传统行业的众多在线数据分析处理系统中。

4.1.3HBase与传统关系数据库的对比分析

HBase与传统的关系数据库的区别主要体现在以下几个方面：

（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。

（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。

（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。

4.1.3HBase与传统关系数据库的对比分析

（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，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

（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。而在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。相反，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

4.2 HBase访问接口

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

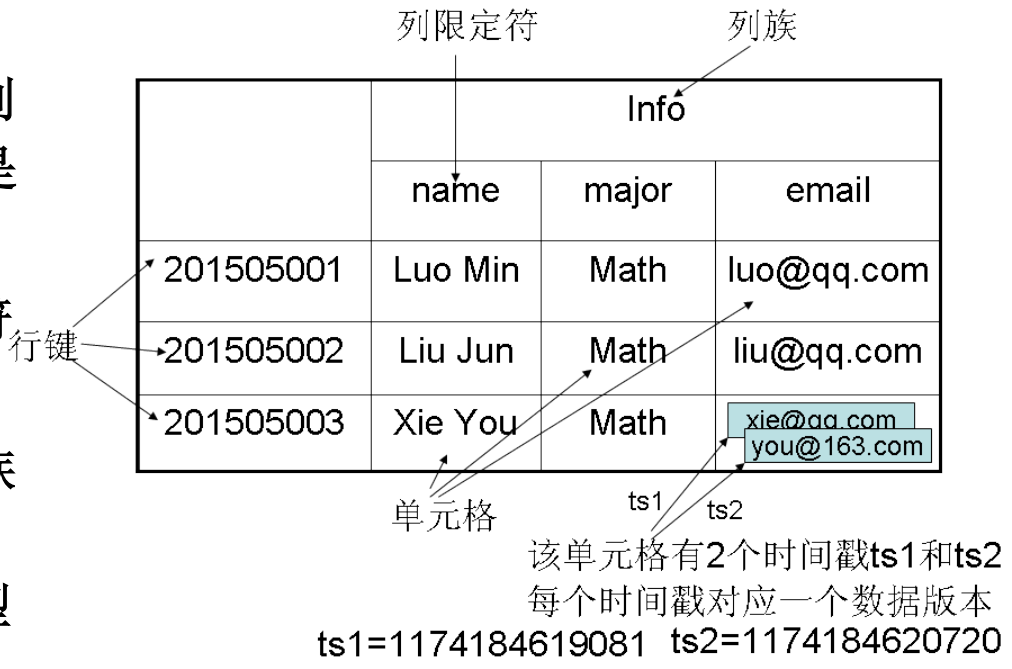
- 4.3.1 数据模型概述
- 4.3.2 数据模型相关概念
- 4.3.3 数据坐标
- 4.3.4 概念视图
- 4.3.5 物理视图
- 4.3.6 面向列的存储

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，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
- 列族支持动态扩展，可以很轻松地添加一个列族或列，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的）

4.3.2数据模型相关概念

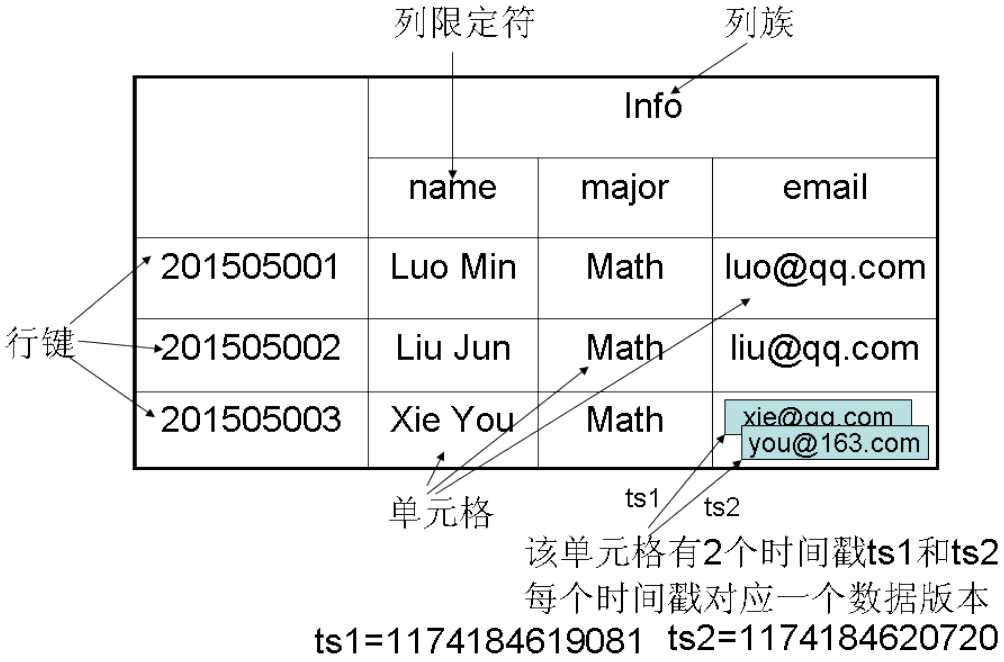
- 表：HBase采用表来组织数据，表由行和列组成，列划分为若干个列族
- 行：每个HBase表都由若干行组成，每个行由行键（row key）来标识。
- 列族：一个HBase表被分组成许多“列族”（Column Family）的集合，它是基本的访问控制单元
- 列限定符：列族里的数据通过列限定符（或列）来定位
- 单元格：在HBase表中，通过行、列族和列限定符确定一个“单元格”（cell），单元格中存储的数据没有数据类型，总被视为字节数组byte[]
- 时间戳：每个单元格都保存着同一份数据的多个版本，这些版本采用时间戳进行索引



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”



4.3.4概念视图

HBase数据的概念视图

行键	时间戳	列族contents	列族anchor
"com.cnn.www"	t5		anchor:cnnsi.com=”CNN”
	t4		anchor:my.look.ca="CNN.com"
	t3	contents:html="<html>..."	
	t2	contents:html="<html>..."	
	t1	contents:html="<html>..."	

4.3.5物理视图

HBase数据的物理视图:**基于列的存储**

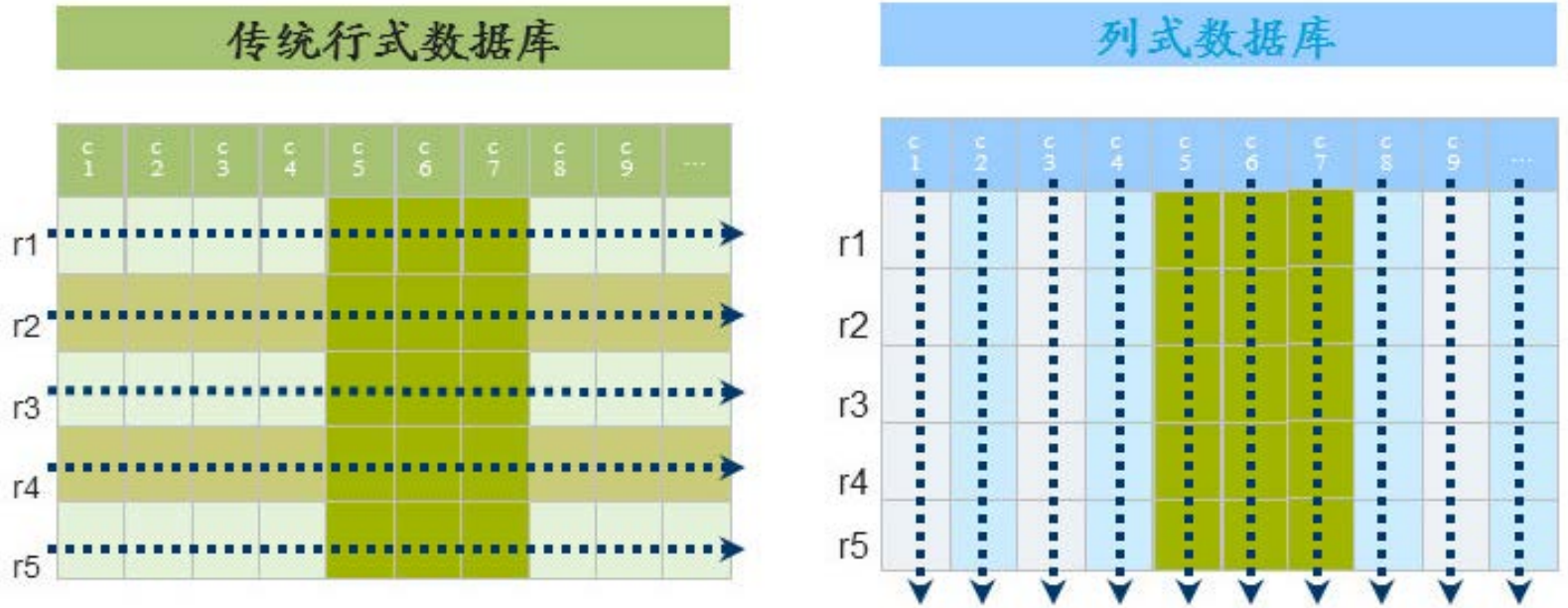
列族contents

行键	时间戳	列族contents
"com.cnn.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.cnn.www"	t5	anchor:cnnsi.com=”CNN”
	t4	anchor:my.look.ca="CNN.com"

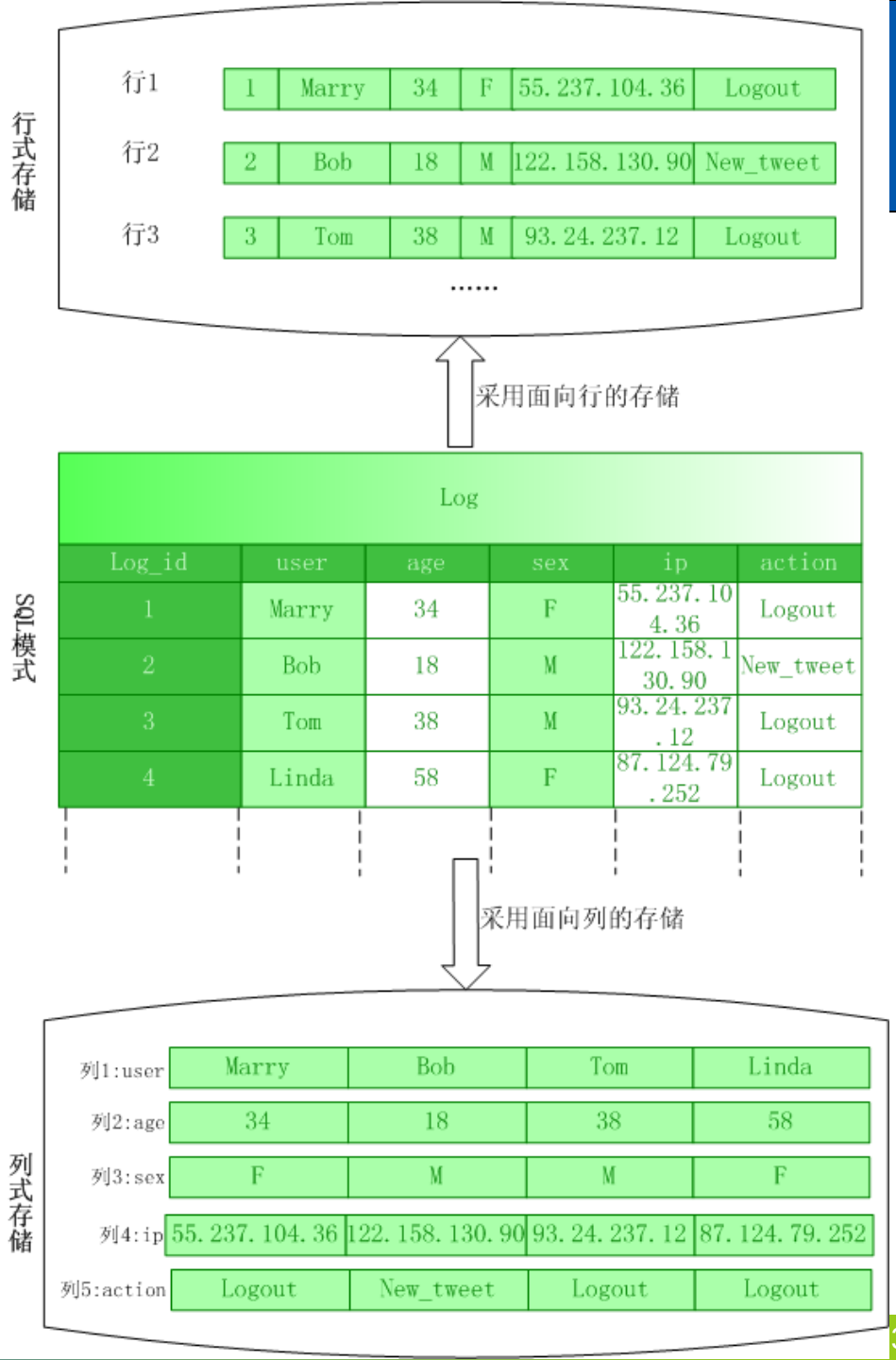
4.3.6面向列的存储



行式数据库和列式数据库示意图

4.3.6面向列的存储

行式存储结构和列式存储结构



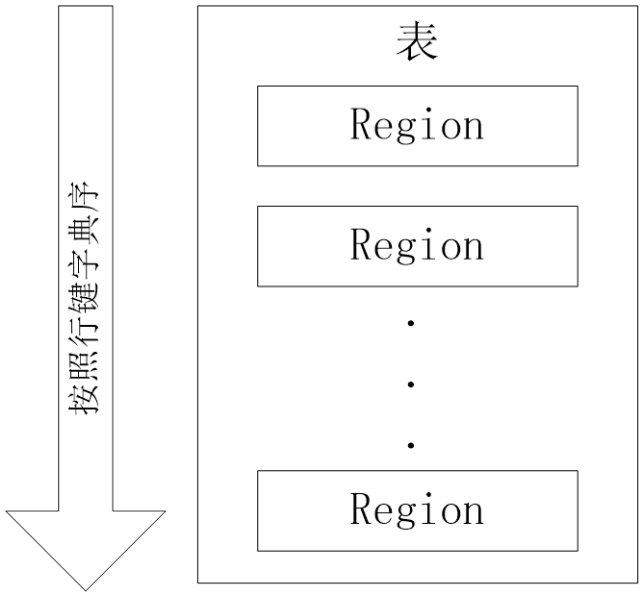
4.4 HBase的实现原理

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

4.4.1HBase功能组件

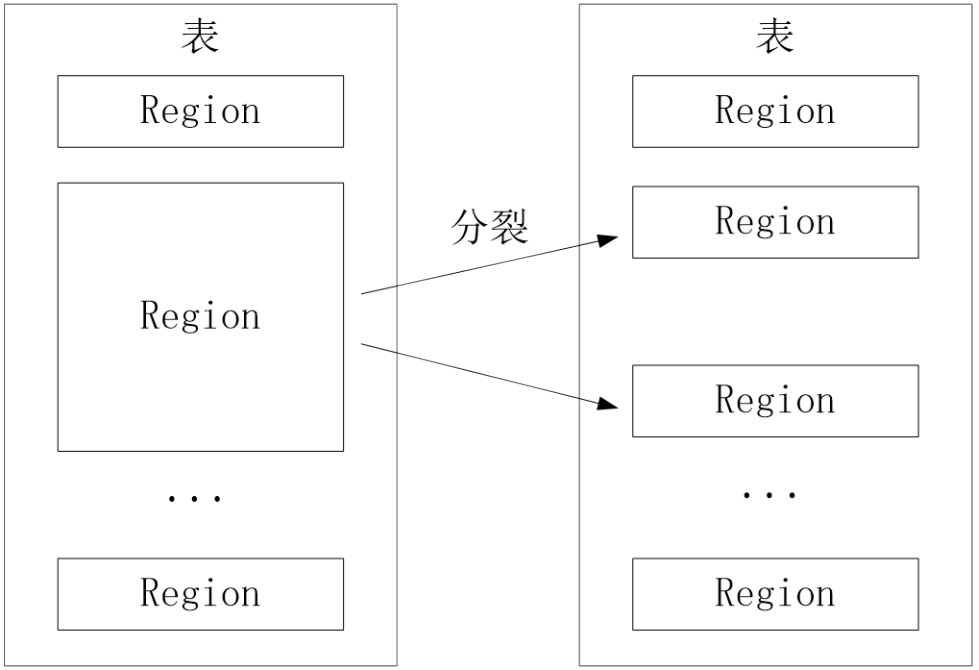
- HBase的实现包括三个主要的功能组件：
  - （1）库函数：链接到每个客户端
  - （2）一个Master主服务器
  - （3）许多个Region服务器
- 主服务器Master负责管理和维护HBase表的分区信息，维护Region服务器列表，分配Region，负载均衡
- Region服务器负责存储和维护分配给自己的Region，处理来自客户端的读写请求
- 客户端并不是直接从Master主服务器上读取数据，而是在获得Region的存储位置信息后，直接从Region服务器上读取数据
- 客户端并不依赖Master，而是通过Zookeeper来获得Region位置信息，大多数客户端甚至从来不和Master通信，这种设计方式使得Master负载很小

4.4.2表和Region



一个HBase表被划分成多个Region

- 开始只有一个**Region**，后来不断分裂
- **Region**拆分操作非常快，接近瞬间，因为拆分之后的**Region**读取的仍然是原存储文件，直到“合并”过程把存储文件异步地写到独立的文件之后，才会读取新文件



一个Region会分裂成多个新的Region

4.4.2表和Region

- 每个Region默认大小是100MB到200MB（2006年以前的硬件配置）
  - 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（2013年以后的硬件配置）
- 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

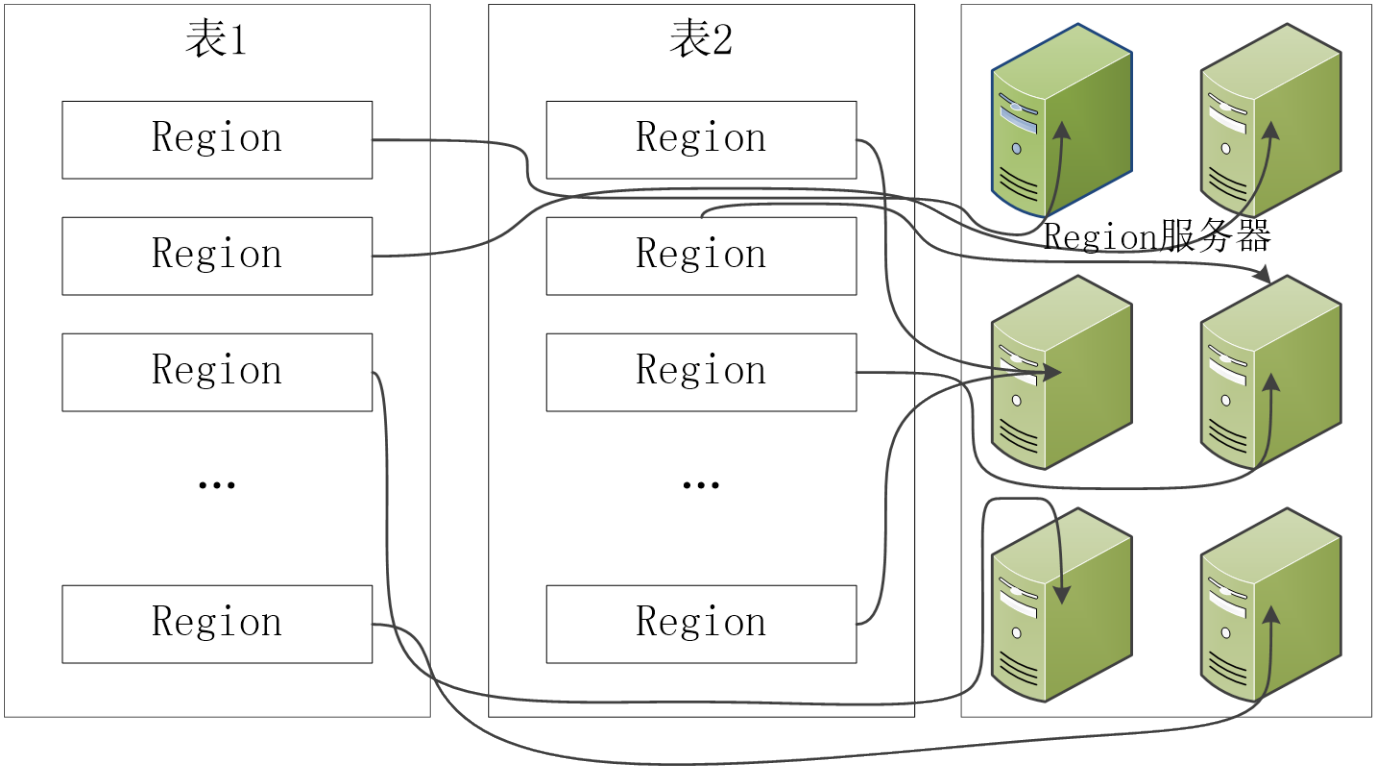
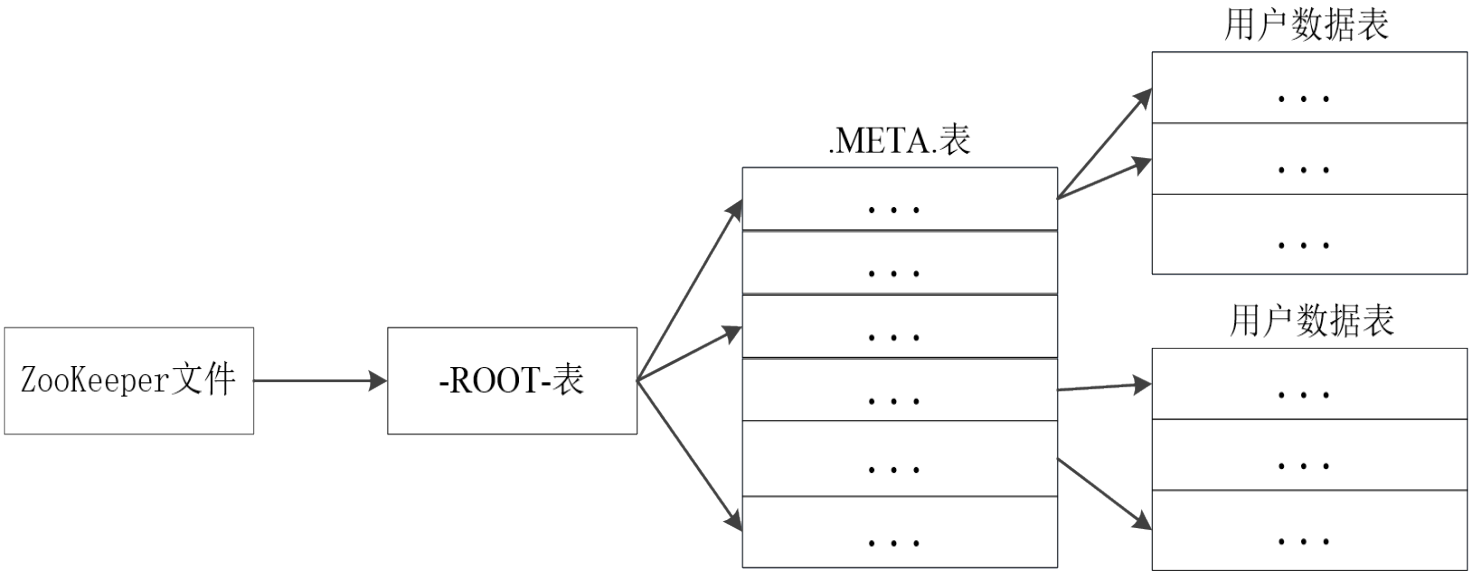


图4-7 不同的Region可以分布在不同的Region服务器上

4.4.3Region的定位

- 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



HBase的三层结构

4.4.3Region的定位

HBase的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

4.4.3Region的定位

为了加快访问速度，.META.表的全部Region都会被保存在内存中。

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，上面的三层结构可以保存的用户数据表的Region数目的计算方法是：

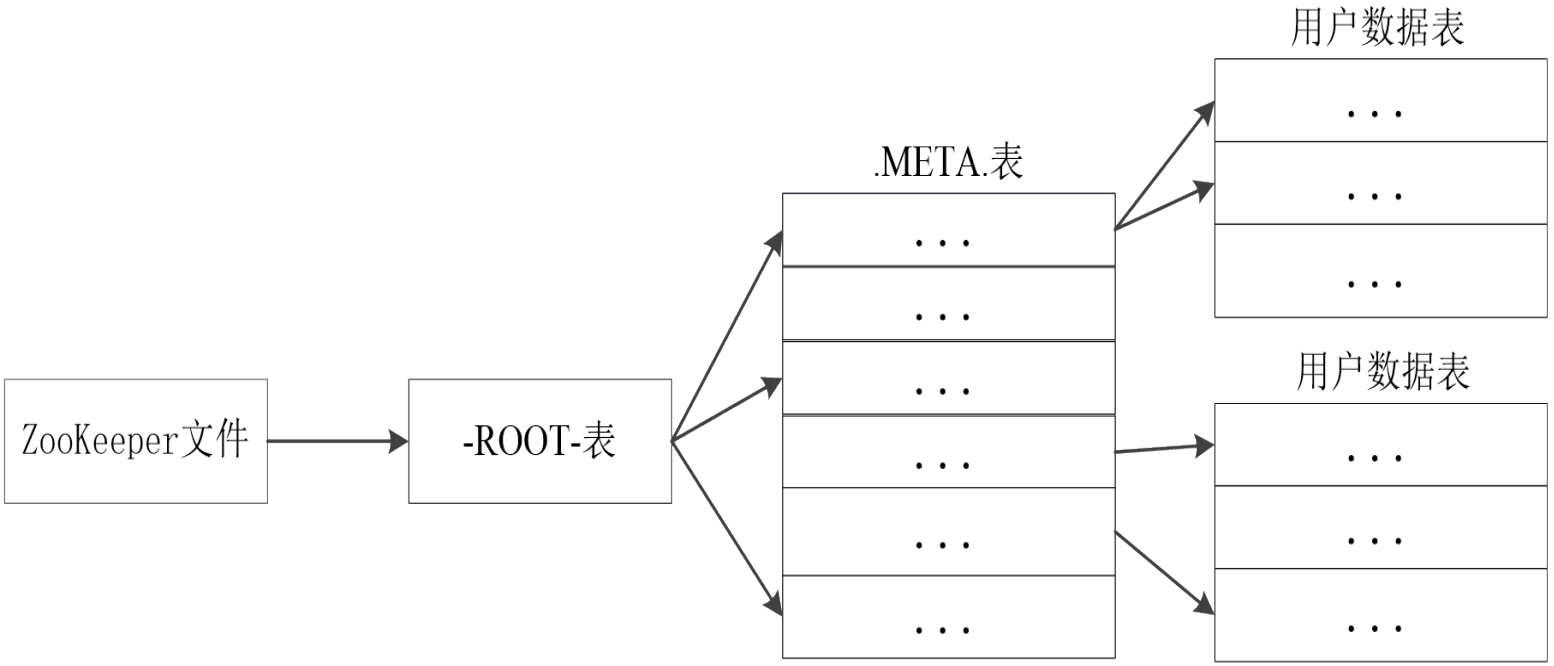
（-ROOT-表能够寻址的.META.表的Region个数）×（每个.META.表的Region可以寻址的用户数据表的Region个数）

- 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

4.4.3Region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



Bigtable数据模型

➢**行:**行是表的第一级索引，可以把该行的列、时间和值看成一个整体，简化为一维键值映射。

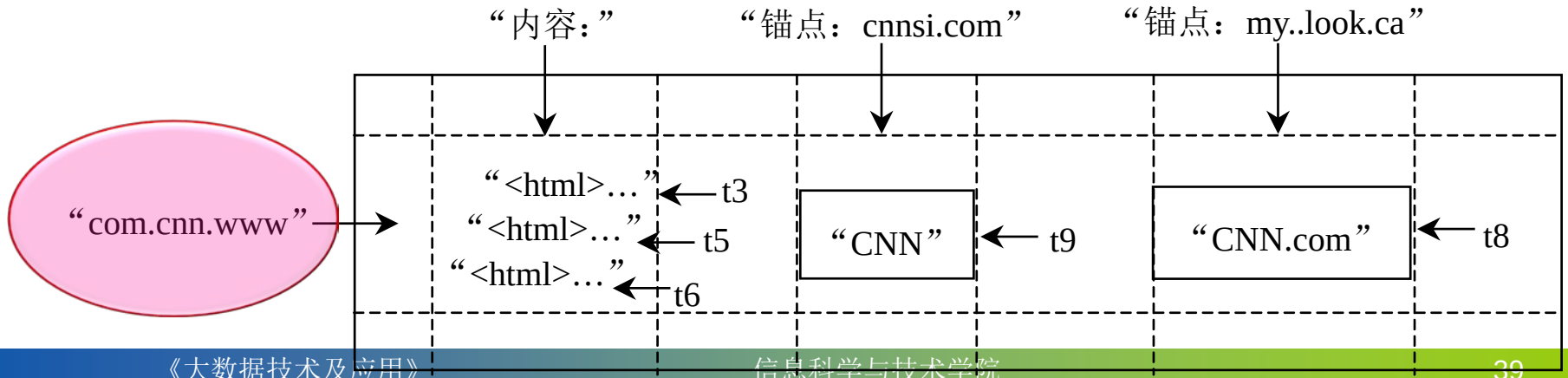
●Bigtable的行关键字可以是任意的字符串，但是大小不能超过64KB。Bigtable和传统的关系型数据库有很大不同，它不支持一般意义上的事务，但能保证对于行的读写操作具有原子性（Atomic）

●表中数据都是根据行关键字进行排序的，排序使用的是词典序。

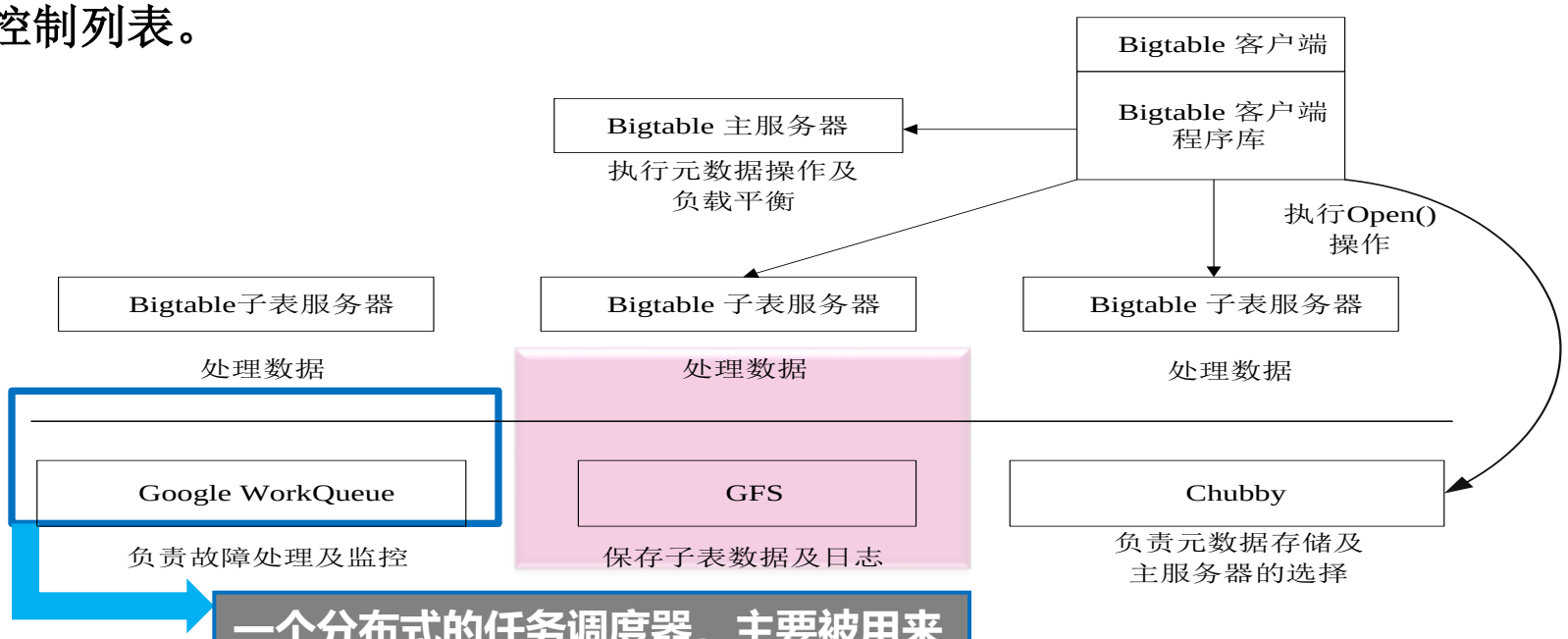
●一个典型实例，其中com.cnn.www就是一个行关键字。不直接存储网页地址而将其倒排是Bigtable的一个巧妙设计。带来两个好处：

☞同一地址域的网页会被存储在表中的连续位置，有利于用户查找和分析

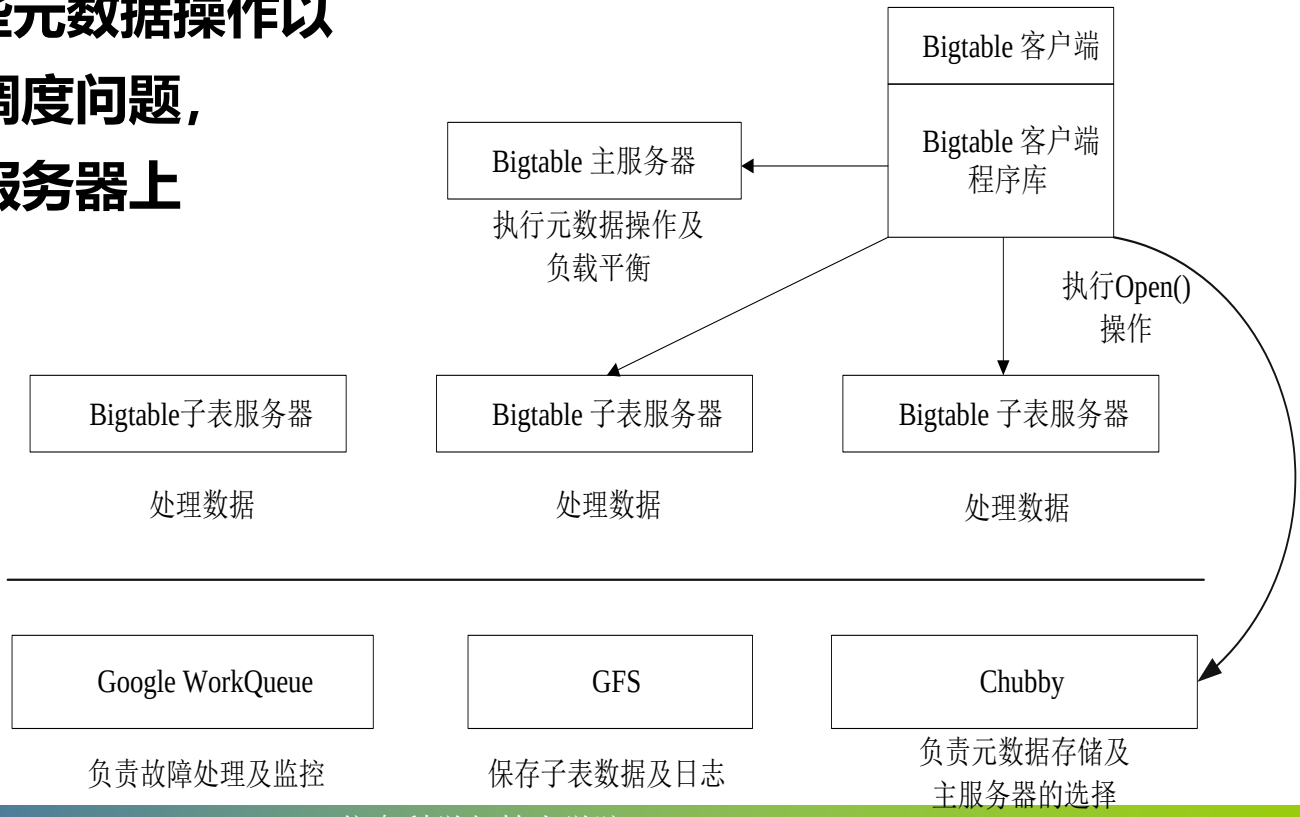
☞倒排便于数据压缩，可以大幅提高压缩率



Bigtable数据库的架构，由主服务器和分服务器构成，把数据库看成是一张大表，那么可将其划分为许多基本的小表，这些小表就称为tablet，是bigtable中最小的处理单位了。
主服务器负责将Tablet分配到Tablet服务器、检测新增和过期的Tablet服务器、平衡Tablet服务器之间的负载、GFS垃圾文件的回收、数据模式的改变（例如创建表）等。
Tablet服务器负责处理数据的读写，并在Tablet规模过大时进行拆分。
Bigtable使用集群管理系统来调度任务、管理资源、监测服务器状态并处理服务器故障。
BigTable将数据存储分为两部分：最近的更新存储在内存memtable中，较老的更新则以SSTable的格式存储在GFS，后者是主体部分，不可变的数据结构。写操作的内容插入到memtable中，当memtable的大小达到一个阈值时就冻结，然后创建一个新的memtable，旧的就转换成一个SSTable写入GFS。
使用分布式的锁服务Chubby来保证集群中主服务器的唯一性、保存Bigtable数据的引导区位置、发现Tablet服务器并处理Tablet服务器的失效、保存Bigtable的数据模式信息、保存存取控制列表。

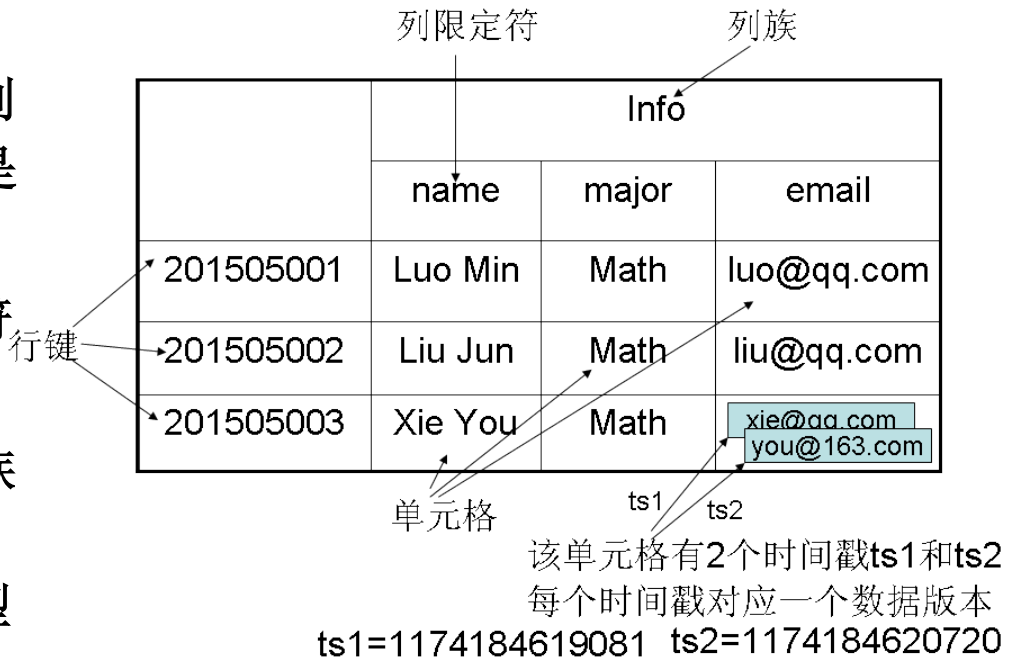


- Bigtable主要由三个部分组成：客户端程序库（Client Library），一个主服务器（Master Server）和多个子表服务器（Tablet Server）
  - 客户访问Bigtable服务时，首先要利用其库函数执行Open()操作来打开一个锁（实际上就是**获取了文件目录**），锁打开以后客户端就可以和子表服务器进行通信
  - 和许多具有单个主节点分布式系统一样，客户端主要与子表服务器通信，几乎不和主服务器进行通信，这使得主服务器的**负载大大降低**
  - 主服务主要进行一些元数据操作以及子表服务器之间负载调度问题，实际数据是存储在子表服务器上



HBASE数据模型

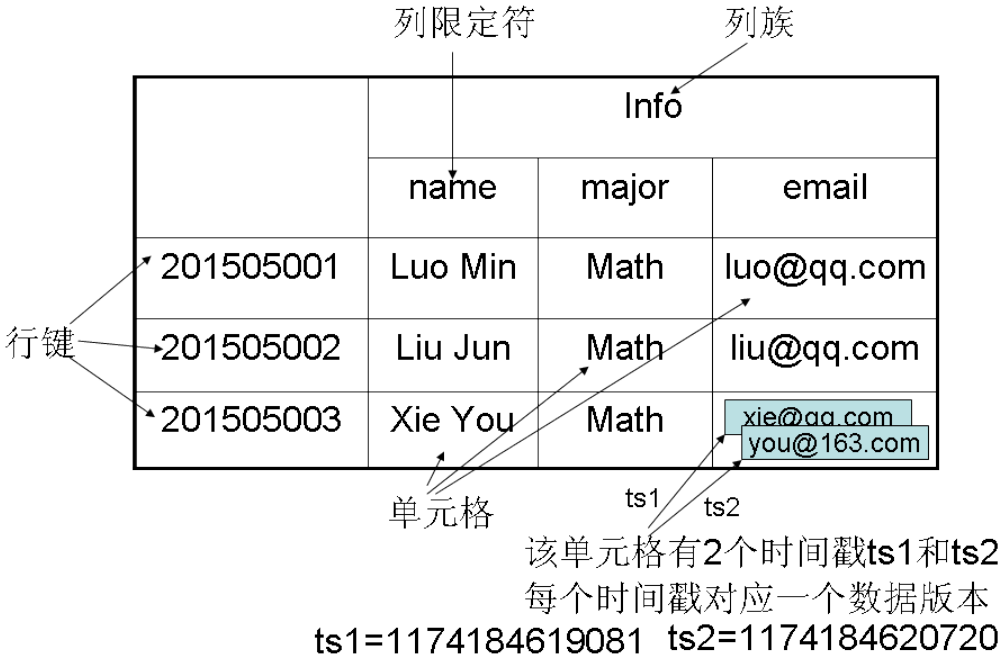
- 表：HBase采用表来组织数据，表由行和列组成，列划分为若干个列族
- 行：每个HBase表都由若干行组成，每个行由行键（row key）来标识。
- 列族：一个HBase表被分组成许多“列族”（Column Family）的集合，它是基本的访问控制单元
- 列限定符：列族里的数据通过列限定符（或列）来定位
- 单元格：在HBase表中，通过行、列族和列限定符确定一个“单元格”（cell），单元格中存储的数据没有数据类型，总被视为字节数组byte[]
- 时间戳：每个单元格都保存着同一份数据的多个版本，这些版本采用时间戳进行索引



HBASE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]**

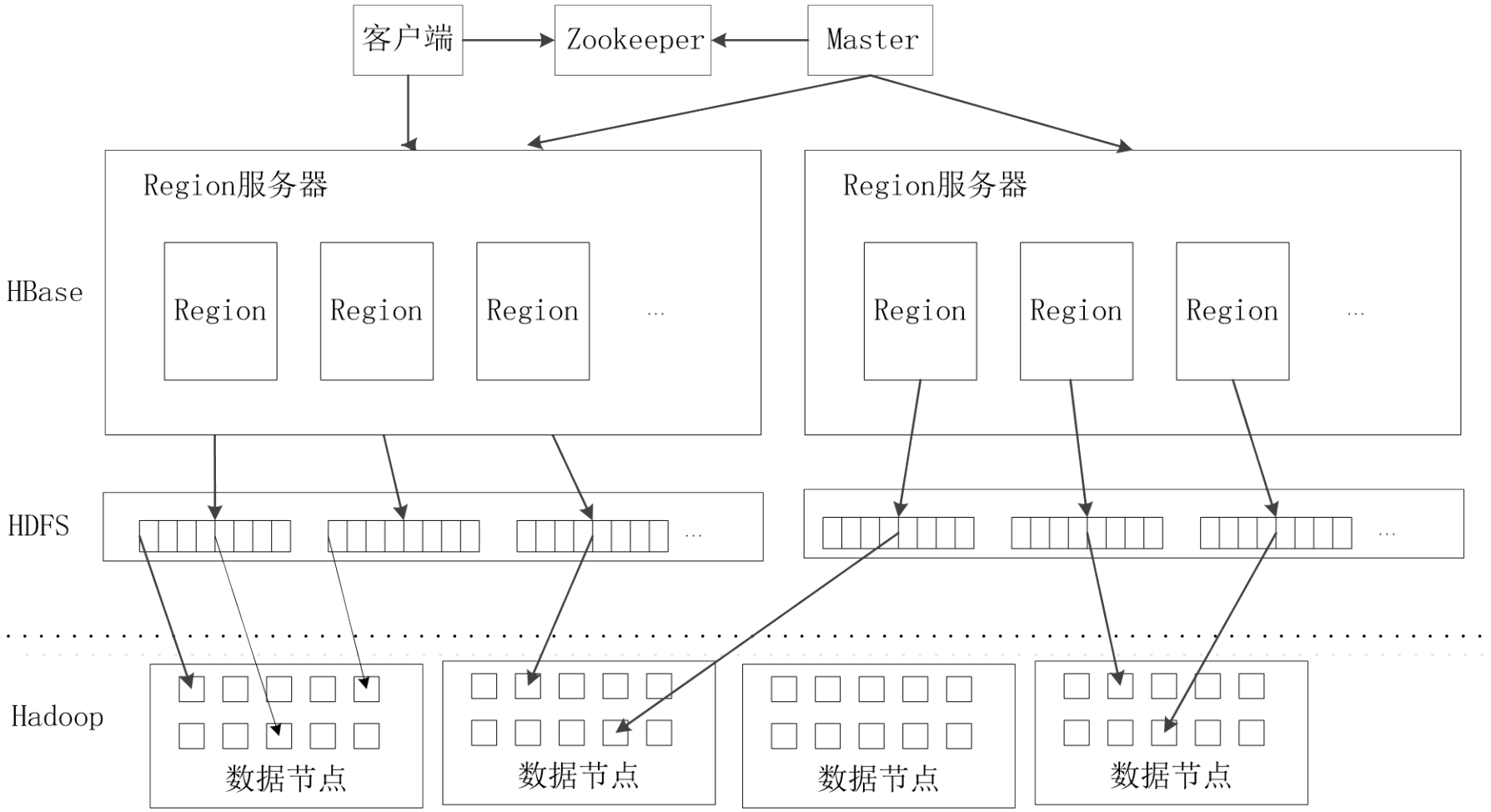
键	值
[“201505003”, “Info”, “email”, 1174184619081]	“xie@qq.com”
[“201505003”, “Info”, “email”, 1174184620720]	“you@163.com”



4.5 HBase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

4.5.1HBase系统架构

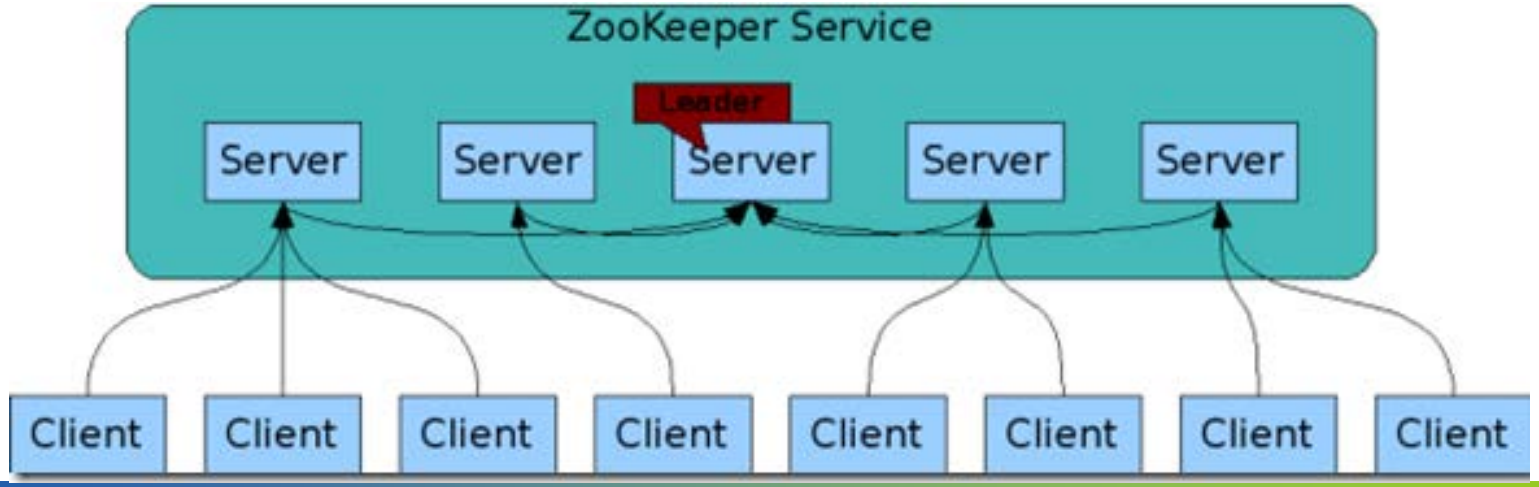


HBase的系统架构

4.5.1HBase系统架构

- **1. 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
- **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，这就避免了**Master**的“单点失效”问题

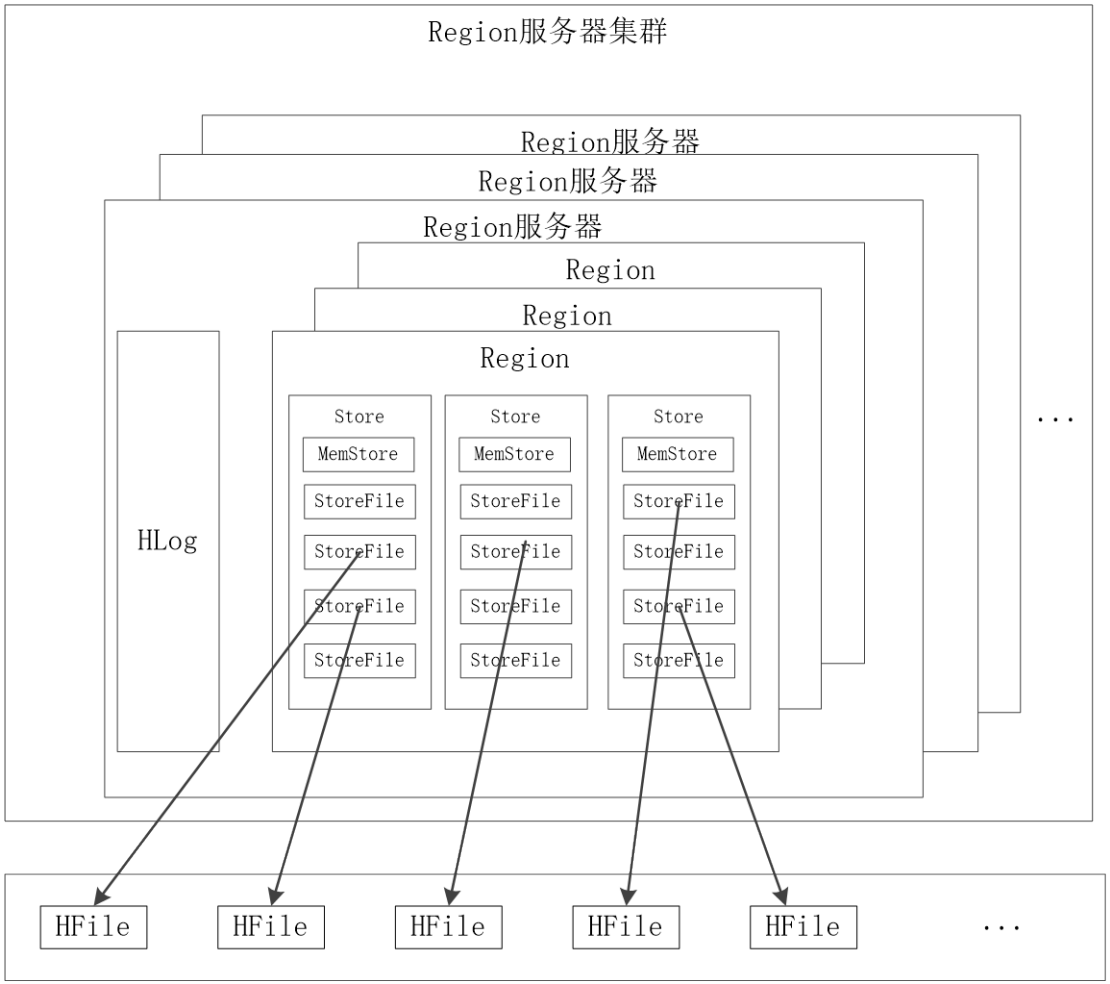
Zookeeper是一个很好的集群管理工具，被大量用于分布式计算，提供配置维护、域名服务、分布式同步、组服务等。



4.5.1HBase系统架构

- **3. Master**
- 主服务器Master主要负责表和Region的管理工作：
  - 管理用户对表的增加、删除、修改、查询等操作
  - 实现不同Region服务器之间的负载均衡
  - 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Region服务器上的Region进行迁移
- **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，并响应用户的读写请求

4.5.2Region服务器工作原理



Region服务器向HDFS文件系统中读写数据

1. 用户读写数据过程
2. 缓存的刷新
3. StoreFile的合并

4.5.2Region服务器工作原理

1. 用户读写数据过程

- 用户写入数据时，被分配到相应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
- 当用户读取数据时，Region服务器会首先访问MemStore缓存，如果找不到，再去磁盘上面的StoreFile中寻找

4.5.2Region服务器工作原理

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

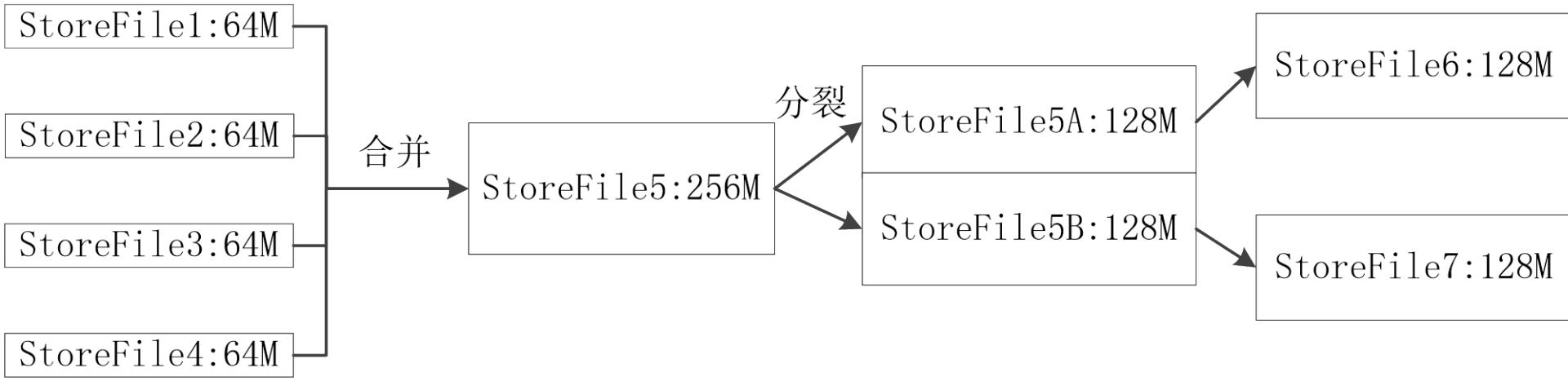
4.5.2Region服务器工作原理

3. StoreFile的合并

- 每次刷写都生成一个新的StoreFile，数量太多，影响查找速度
- 调用Store.compact()把多个合并成一个
- 合并操作比较耗费资源，只有数量达到一个阈值才启动合并

4.5.3Store工作原理

- **Store是Region服务器的核心**
- **多个StoreFile合并成一个**
- **单个StoreFile过大时，又触发分裂操作，1个父Region被分裂成两个子Region**



StoreFile的合并和分裂过程

4.5.4HLog工作原理

- 分布式环境必须要考虑系统出错。**HBase采用HLog保证系统恢复**
- **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（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

4.5.4HLog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
- 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：恢复时需要分拆日志

4.6 HBase访问方式

- Native Java API
  - 最常规和高效的访问方式
- HBase Shell
  - 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
  - 支持REST 风格的Http API访问HBase，解除了语言限制
- MapReduce
  - 直接使用MapReduce作业处理Hbase数据
  - 使用Pig/hive处理Hbase数据

HBase Java API概述

- Hbase是用Java语言编写的，支持Java编程是自然而然的事情
- 支持CRUD操作：
  - Create, Read, Update, Delete
- Java API包含Hbase shell支持的所有功能，甚至更多
- Java API是访问Hbase最快的方式

Java API程序设计步骤

- 步骤1：创建一个Configuration对象
  - 包含各种配置信息
  - 例如：Configuration conf = HbaseConfiguration.create();
- 步骤2：构建一个HTable句柄
  - 提供Configuration对象
  - 提供待访问Table的名称
  - 例如：HTable table = new HTable(conf, tableName);
- 步骤3：执行相应的操作
  - 执行put、get、delete、scan等操作
  - 例如：table.getTableName();
- 步骤4：关闭HTable句柄
  - 将内存数据刷新到磁盘上
  - 释放各种资源
  - 例如：table.close();

程序示例

- 框架程序

```
public class ConstructHTable {
    public static void main(String[] args) throws IOException {
        Configuration conf = HBaseConfiguration.create();
        HTable hTable = new HTable(conf, "-ROOT-");
        System.out.println("Table is: " + Bytes.toString(hTable.getTableName()));
        hTable.close();
    }
}
```

包含建立连接所需的全部信息

表名称

释放资源

4.7 HBase应用方案

4.7.1 HBase实际应用中的性能优化方法

4.7.2 HBase性能监视

4.7.3 在HBase之上构建SQL引擎

4.7.4 构建HBase二级索引

4.7.1 HBase实际应用中的性能优化方法

行键（Row Key）

- 行键是按照字典序存储，因此，设计行键时，要充分利用这个排序特点，将经常一起读取的数据存储到一块，将最近可能会被访问的数据放在一块。

- 例子：如果最近写入HBase表中的数据是最可能被访问的，可以考虑将时间戳作为行键的一部分，由于是字典序排序，所以可以使用Long.MAX_VALUE - timestamp作为行键，这样能保证新写入的数据在读取时可以被快速命中。

4.7.1 HBase实际应用中的性能优化方法

- **InMemory**

—创建表的时候，可以通过**HColumnDescriptor.setInMemory(true)**将表放到**Region**服务器的缓存中，保证在读取的时候被**cache**命中。

- **Max Version**

—创建表的时候，可通过**HColumnDescriptor.setMaxVersions(int maxVersions)**设置表中数据的最大版本，如果只需要保存最新版本的数据，那么可以设置**setMaxVersions(1)**。

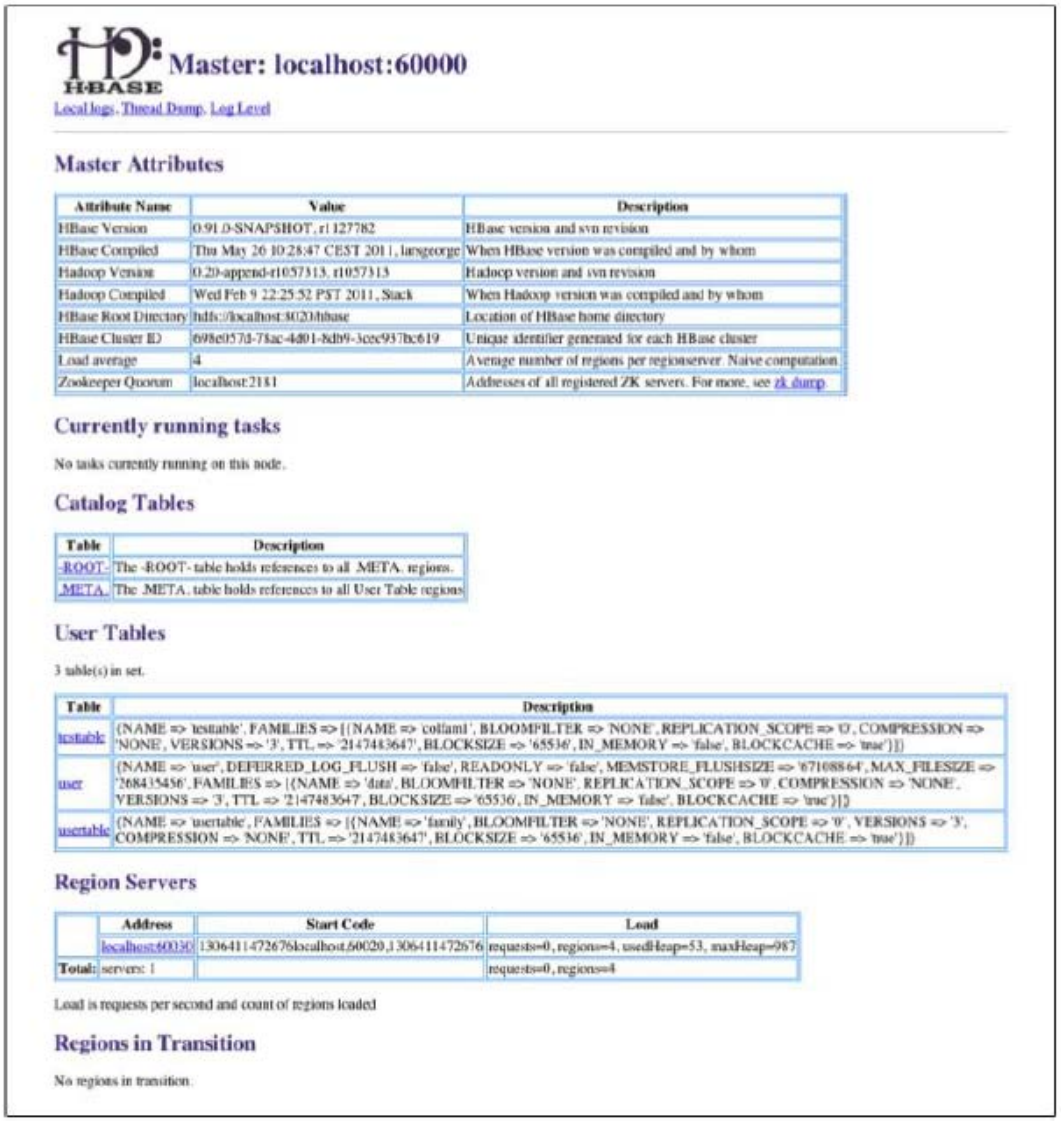
- **Time To Live**

—创建表的时候，可以通过**HColumnDescriptor.setTimeToLive(int timeToLive)**设置表中数据的存储生命期，过期数据将自动被删除，例如如果只需要存储最近两天的数据，那么可以设置**setTimeToLive(2 * 24 * 60 * 60)**。

4.7.2 HBase性能监视

- Master-status(自带)
- Ganglia
- OpenTSDB
- Ambari

Master-status



Master: localhost:60000

Local logs, Thread Dump, Log Level

Master Attributes

Attribute Name	Value	Description
HBase Version	0.91.0-SNAPSHOT, r1127782	HBase version and svn revision
HBase Compiled	Thu May 26 10:28:47 CEST 2011, lars george	When HBase version was compiled and by whom
Hadoop Version	0.20-append-r1057313, r1057313	Hadoop version and svn revision
Hadoop Compiled	Wed Feb 9 22:25:52 PST 2011, Stack	When Hadoop version was compiled and by whom
HBase Root Directory	hdfs://localhost:8020/hbase	Location of HBase home directory
HBase Cluster ID	698e057d-78ac-4d01-8db9-3cec937bc619	Unique identifier generated for each HBase cluster
Load average	4	Average number of regions per regionserver. Naive computation.
Zookeeper Quorum	localhost:2181	Addresses of all registered ZK servers. For more, see zk dump.

Currently running tasks

No tasks currently running on this node.

Catalog Tables

Table	Description
-ROOT-	The -ROOT- table holds references to all .META. regions.
.META.	The .META. table holds references to all User Table regions

User Tables

3 table(s) in set.

Table	Description
testable	{NAME => 'testable', FAMILIES => [{NAME => 'colfam1', BLOOMFILTER => 'NONE', REPLICATION_SCOPE => '0', COMPRESSION => 'NONE', VERSIONS => '3', TTL => '2147483647', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}]}
user	{NAME => 'user', DEFERRED_LOG_FLUSH => 'false', READONLY => 'false', MEMSTORE_FLUSHSIZE => '67108864', MAX_FILESIZE => '268435456', FAMILIES => [{NAME => 'data', BLOOMFILTER => 'NONE', REPLICATION_SCOPE => '0', COMPRESSION => 'NONE', VERSIONS => '3', TTL => '2147483647', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}]}
usertable	{NAME => 'usertable', FAMILIES => [{NAME => 'family', BLOOMFILTER => 'NONE', REPLICATION_SCOPE => '0', VERSIONS => '3', COMPRESSION => 'NONE', TTL => '2147483647', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}]}

Region Servers

	Address	Start Code	Load
	localhost:60030	1306411472676localhost,60020,1306411472676	requests=0, regions=4, usedHeap=53, maxHeap=987
Total:	servers: 1		requests=0, regions=4

Load is requests per second and count of regions loaded

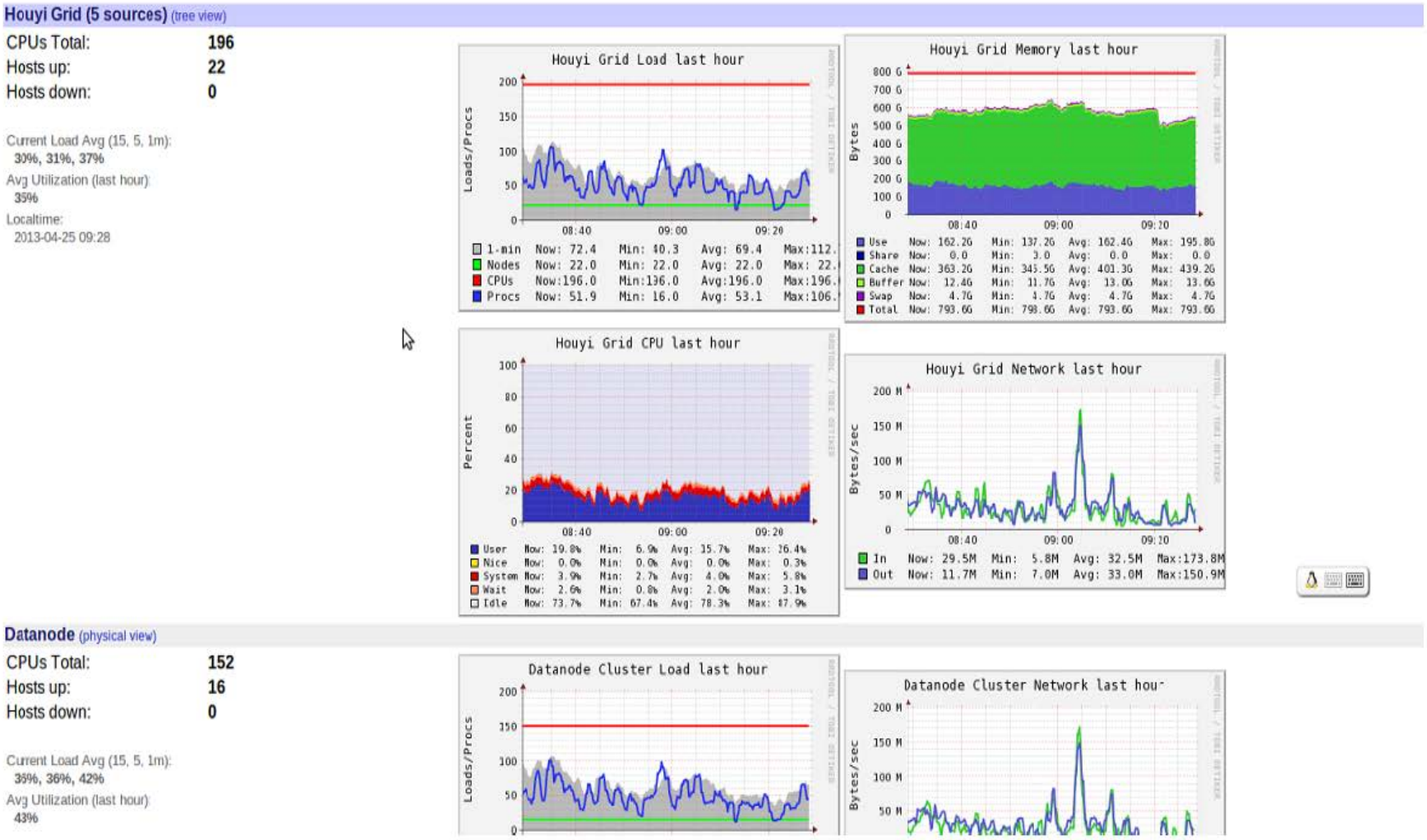
Regions in Transition

No regions in transition.

- **HBase Master默认基于Web的UI服务端口为60010，HBase region服务器默认基于Web的UI服务端口为60030.** 如果**master运行在名为master.foo.com的主机中，mater的主页地址就是** **http://master.foo.com:60010**，用户可以通过Web浏览器输入这个地址查看该页面.
- 可以查看HBase集群的当前状态.

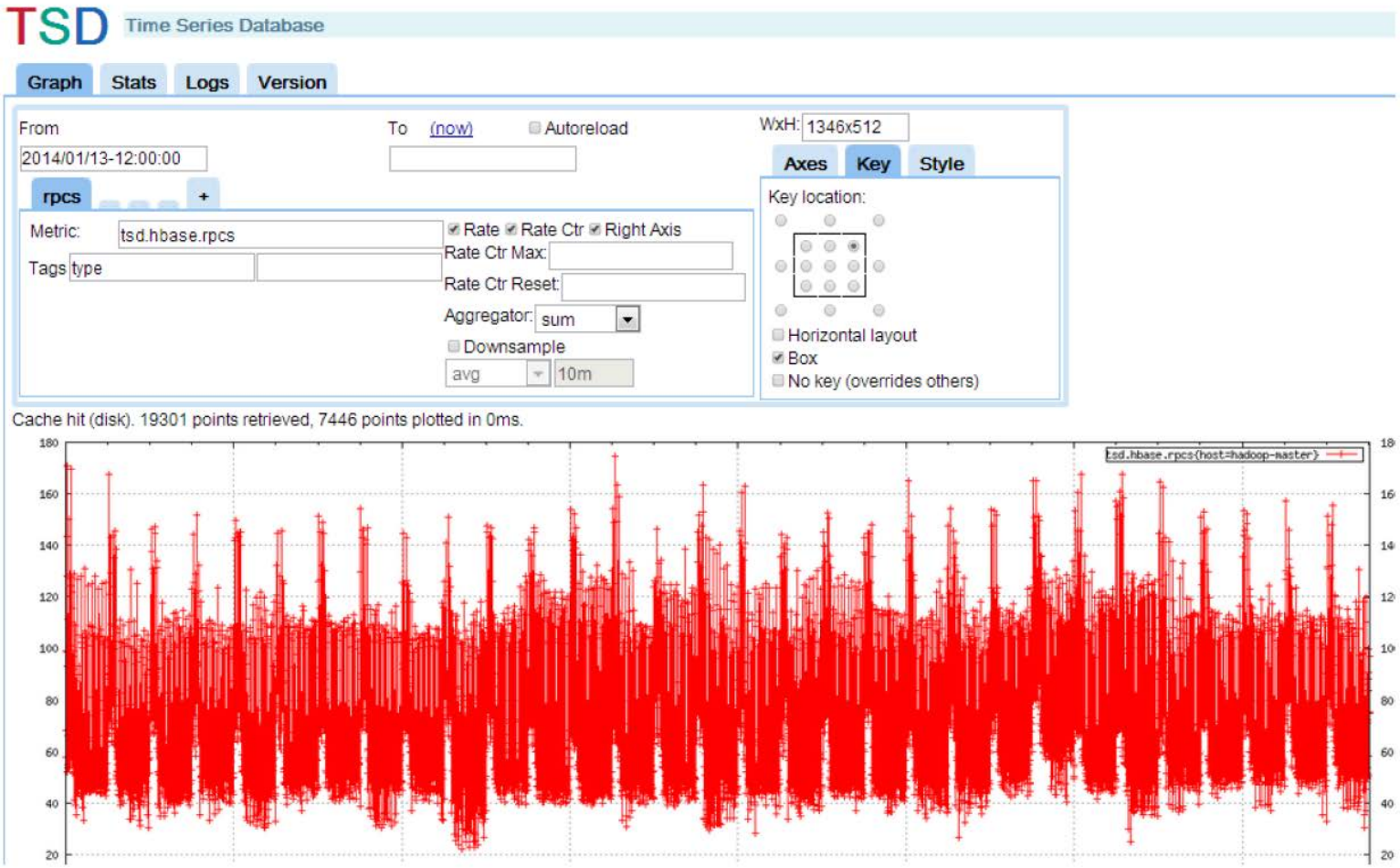
Ganglia

Ganglia是UC Berkeley发起的一个开源集群监视项目，用于监控系统性能



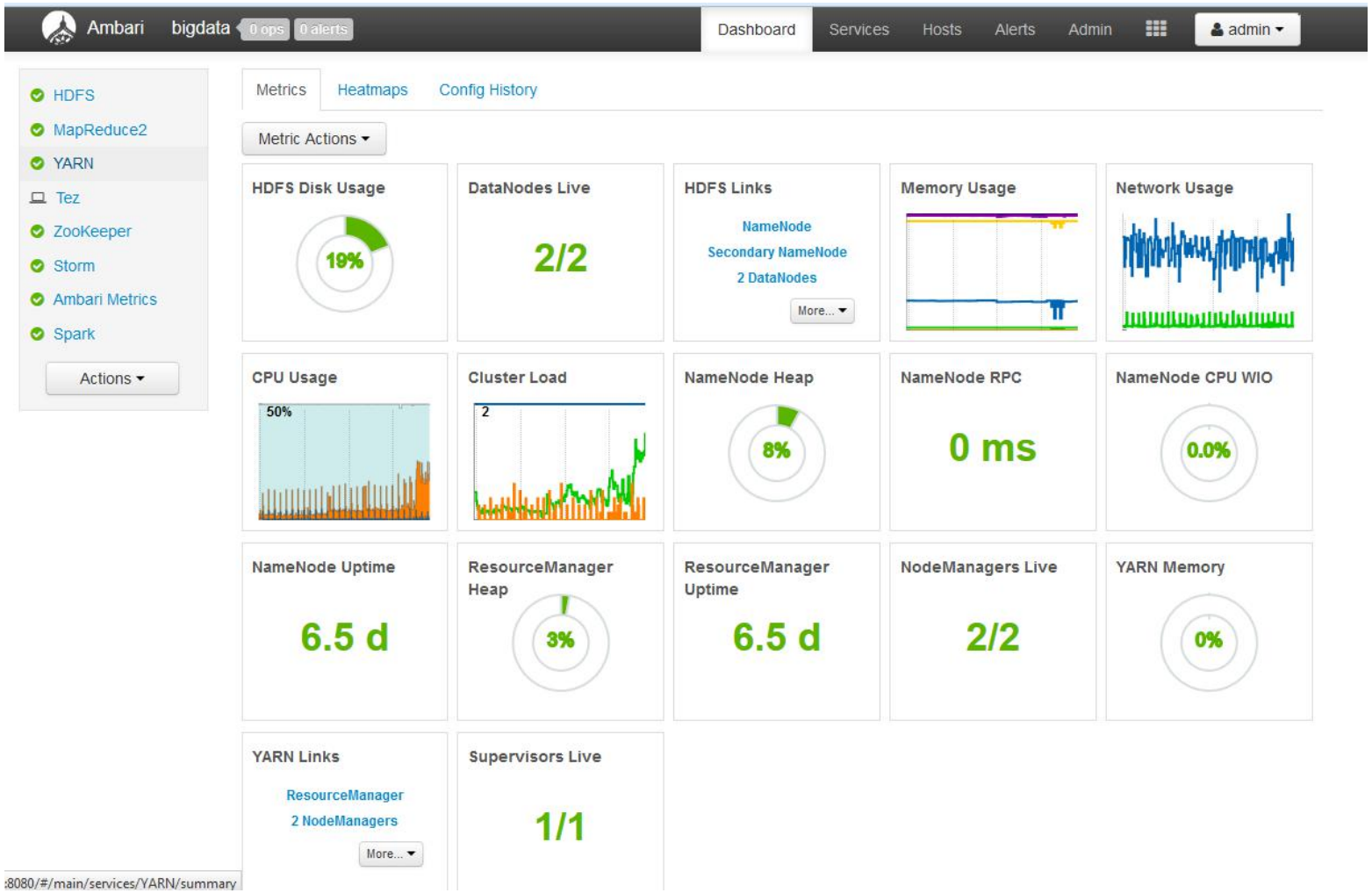
OpenTSDB

OpenTSDB可以从大规模的集群（包括集群中的网络设备、操作系统、应用程序）中获取相应的metrics并进行存储、索引以及服务，从而使得这些数据更容易让人理解，如web化，图形化等



Ambari

Ambari 的作用就是创建、管理、监视 Hadoop 的集群



4.7.3 在HBase之上构建SQL引擎

➢NoSQL区别于关系型数据库的一点就是NoSQL不使用SQL作为查询语言，至于为何在NoSQL数据存储HBase上提供SQL接口，有如下原因：

1.易使用。使用诸如SQL这样易于理解的语言，使人们能够更加轻松地使用HBase。

2.减少编码。使用诸如SQL这样更高层次的语言来编写，减少了编写的代码量。

方案：

1.Hive整合HBase

2.Phoenix

4.7.3 在HBase之上构建SQL引擎

1.Hive整合HBase

Hive与HBase的整合功能从Hive0.6.0版本已经开始出现，利用两者对外的API接口互相通信，通信主要依靠hive_hbase-handler.jar工具包(Hive Storage Handlers)。由于HBase有一次比较大的版本变动，所以并不是每个版本的Hive都能和现有的HBase版本进行整合，所以在使用过程中特别注意的就是两者版本的一致性。

2.Phoenix

Phoenix由Salesforce.com开源，是构建在Apache HBase之上的一个SQL中间层，可以让开发者在HBase上执行SQL查询。

4.7.4 构建HBase二级索引

HBase只有一个针对行健的索引

访问HBase表中的行，只有三种方式：

- 通过单个行健访问
- 通过一个行健的区间来访问
- 全表扫描

使用其他产品为HBase行健提供索引功能：

- **Hindex二级索引**
- **HBase+Redis**
- **HBase+solr**

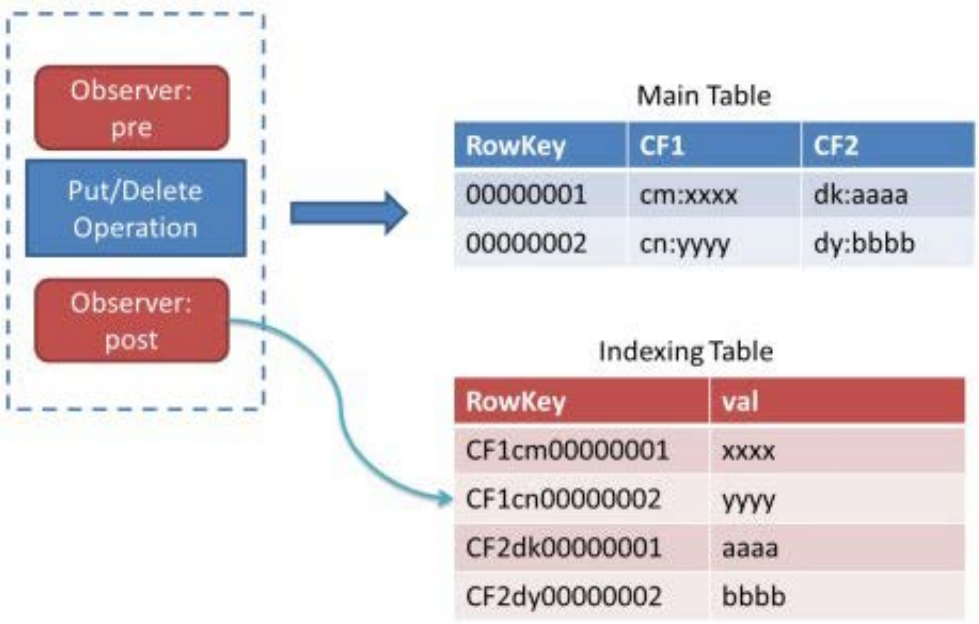
4.7.4 构建HBase二级索引

- **Coprocessor构建二级索引**

- **Coprocessor提供了两个实现：endpoint和observer，endpoint相当于关系型数据库的存储过程，而observer则相当于触发器.observer允许我们在记录put前后做一些处理，因此，而我们可以在插入数据时同步写入索引表.**

优点：
非侵入性：引擎构建在HBase之上，既没有对HBase进行任何改动，也不需要上层应用做任何妥协

- 缺点：每插入一条数据需要向索引表插入数据，即耗时是双倍的，对HBase的集群的压力也是双倍的



4.8 HBase编程实践

4.8.1 HBase的安装与配置

4.8.2 HBase常用Shell命令

4.8.3 HBase常用Java API及应用实例

4.8.1 HBase的安装与配置

1. HBase安装

- 下载安装包hbase-1.1.2-bin.tar.gz
- 解压安装包hbase-1.1.2-bin.tar.gz至路径 /usr/local
- 配置系统环境,将hbase下的bin目录添加到系统的path中

备注：安装完Hadoop时，只包含HDFS和MapReduce等核心组件，并不包含HBase，因此，HBase需要单独安装

2. HBase配置

HBase有三种运行模式，单机模式、伪分布式模式、分布式模式。
以下先决条件很重要，比如没有配置JAVA_HOME环境变量，就会报错。

– JDK
– Hadoop(单机模式不需要，伪分布式模式和分布式模式需要)
– SSH

启动关闭Hadoop和HBase的顺序一定是：

启动Hadoop—>启动HBase—>关闭HBase—>关闭Hadoop

HBASE_MANAGES_ZK=true，由HBase管理Zookeeper,否则启动独立的Zookeeper.
建议：单机版HBase，使用自带Zookeeper；集群安装HBase则采用单独Zookeeper集群

4.8.2 HBase常用Shell命令

Shell命令帮助

```
hbase(main):001:0> help
HBase Shell, version 0.91.0-SNAPSHOT, r1130916, Sat Jul 23 12:44:34 CEST 2011
Type 'help "COMMAND"', (e.g. 'help "get"' -- the quotes are necessary) for
help on a specific command. Commands are grouped. Type 'help "COMMAND_GROUP"',
(e.g. 'help "general"') for help on a command group.

COMMAND GROUPS:
  Group name: general
  Commands: status, version

  Group name: ddl
  Commands: alter, create, describe, disable, drop, enable, exists,
            is_disabled, is_enabled, list
```

4.8.2 HBase常用Shell命令

查询数据库状态

- hbase(main):024:0>status
- 3 servers, 0 dead,1.0000 average load

查询数据库版本

- hbase(main):025:0>version
- 0.90.4, r1150278,Sun Jul 24 15:53:29 PDT 2011

4.8.2 HBase常用Shell命令

- **create：创建表**
- **list：列出HBase中所有的表信息**

例子1：创建一个表，该表名称为tempTable，包含3个列族f1，f2和f3

（1）利用命令create创建表tempTable，表中有f1，f2，f3三个列族

（2）利用list列出Hbase中所有的表信息

```
hbase(main):002:0> create 'tempTable', 'f1', 'f2', 'f3'
0 row(s) in 1.3560 seconds

hbase(main):003:0> list
TABLE
tempTable
testTable
wordcount
3 row(s) in 0.0350 seconds
```

4.8.2HBase常用Shell命令

put： 向表、行、列指定的单元格添加数据

一次只能为一个表的一行数据的一个列添加一个数据

scan： 浏览表的相关信息

例子2：继续向表tempTable中的第r1行、第“f1:c1”列，添加数据值为“hello,dblab”

（1）利用put命令向表tempTalble，行r1，列f1:c1中插入数据

（2）利用scan命令浏览表tempTable的相关信息

```
hbase(main):005:0> put 'tempTable', 'r1', 'f1:c1', 'hello, dblab'
0 row(s) in 0.0240 seconds

hbase(main):006:0> scan 'tempTable'
ROW                           COLUMN+CELL
 r1                           column=f1:c1, timestamp=1430036599391, value=hello, dblab
1 row(s) in 0.0160 seconds
```

在添加数据时，HBase会自动为添加的数据添加一个时间戳，当然，也可以在添加数据时人工指定时间戳的值.

4.8.2HBase常用Shell命令

get：通过表名、行、列、时间戳、时间范围和版本号来获得相应单元格的值

例子3：

（1）从tempTable中，获取第r1行、第“f1:c1”列的值

（2）从tempTable中，获取第r1行、第“f1:c3”列的值

备注：f1是列族，c1和c3都是列

```
hbase(main):012:0> get 'tempTable', 'r1', {COLUMN=>'f1:c1'}
COLUMN                          CELL
 f1:c1                          timestamp=1430036599391, value=hello, dblab
1 row(s) in 0.0090 seconds

hbase(main):013:0> get 'tempTable', 'r1', {COLUMN=>'f1:c3'}
COLUMN                          CELL
0 row(s) in 0.0030 seconds
```

从运行结果可以看出： tempTable中第r1行、第“f1:c3”列的值当前不存在

4.8.2HBase常用Shell命令

- **enable/disable：使表有效或无效**
- **drop：删除表**

例子4：使表tempTable无效、删除该表

（1）利用disable命令使表tempTable无效

（2）利用drop命令删除表tempTable

（3）利用list命令展示删除表tempTable后的效果

```
hbase(main):016:0> disable 'tempTable'
0 row(s) in 1.3720 seconds

hbase(main):017:0> drop 'tempTable'
0 row(s) in 1.1350 seconds

hbase(main):018:0> list
TABLE
testTable
wordcount
2 row(s) in 0.0370 seconds
```

4.8.3HBase常用Java API及应用实例

HBase是Java编写的，它的原生的API也是Java开发的，不过，可以使用Java或其他语言调用API来访问HBase:

首先要在工程中导入一下jar包：

这里只需要导入hbase安装目录中的lib文件中的所有jar包，此处不用再导入Hadoop中的jar包，避免由于Hadoop和HBase的版本冲突引起错误。

4.8.3HBase常用Java API及应用实例

任务要求：创建表、插入数据、浏览数据

创建一个学生信息表，用来存储学生姓名（姓名作为行键，并且假设姓名不会重复）以及考试成绩，其中，考试成绩是一个列族，分别存储了各个科目的考试成绩。逻辑视图如表所示。

学生信息表的表结构

name	score		
	English	Math	Computer

需要添加的数据

name	score		
	English	Math	Computer
zhangsan	69	86	77
lisi	55	100	88

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.Bytes;
import java.io.IOException;
public class Chapter4{
    public static Configuration configuration;
    public static Connection connection;
    public static Admin admin;
    public static void main(String[] args)throws IOException{
        createTable("student",new String[]{"score"});
        insertData("student","zhangsan","score","English","69");
        insertData("student","zhangsan","score","Math","86");
        insertData("student","zhangsan","score","Computer","77");
        getData("student", "zhangsan", "score", "English");
    }
……
public static void init(){……}//建立连接
public static void close(){……}//关闭连接
public static void createTable(){……}//创建表
public static void insertData() {……}//插入数据
public static void getData{……}//浏览数据
}
```

4.8.3HBase常用Java API及应用实例

- 建立连接，关闭连接

```
//建立连接
    public static void init(){
        configuration  = HBaseConfiguration.create();
        configuration.set("hbase.rootdir","hdfs://localhost:9000/hbase");
        try{
            connection = ConnectionFactory.createConnection(configuration);
            admin = connection.getAdmin();
        }catch (IOException e){
            e.printStackTrace();
        }
    }
```

备注：**hbase-site.xml**

```
<configuration>
<property>
<name>hbase.rootdir</name>
<value>hdfs://localhost:9000/hbase</value>
</property>
</configuration>
```

（单机版）file:///DIRECTORY/hbase

4.8.3HBase常用Java API及应用实例

- 建立连接，关闭连接

```
//关闭连接
    public static void close(){
        try{
            if(admin != null){
                admin.close();
            }
            if(null != connection){
                connection.close();
            }
        }catch (IOException e){
            e.printStackTrace();
        }
    }
```

①建表

name	score		
	English	Math	Computer

在运行程序时，需要指定参数myTableName为“student”，colFamily为“{“score”}”
程序的运行效果与如下HBase Shell命令等效：
create ‘student’, ‘score’

```
/*创建表*/
  /**
   * @param myTableName 表名
   * @param colFamily列族数组
   * @throws Exception
   */
      public static void createTable(String myTableName,String[] colFamily) throws IOException {

      TableName tableName = TableName.valueOf(myTableName);
       if(admin.tableExists(tableName)){
          System.out.println("table exists!");
       }else {
          HTableDescriptor hTableDescriptor = new HTableDescriptor(tableName);
          for(String str: colFamily){
             HColumnDescriptor hColumnDescriptor = new HColumnDescriptor(str);
             hTableDescriptor.addFamily(hColumnDescriptor);
          }
          admin.createTable(hTableDescriptor);
       }
}
```

②添加数据

name	score		
	English	Math	Computer
zhangsan	69	86	77
lisi	55	100	88

```
/*添加数据*/
/**
        * @param tableName 表名
        * @param rowKey 行键
        * @param colFamily 列族
        * @param col 列限定符
        * @param val 数据
        * @throws Exception
        */
    public static void insertData(String tableName, String rowKey, String colFamily,
String col, String val) throws IOException {
        Table table = connection.getTable(TableName.valueOf(tableName));
        Put put = new Put(Bytes.toBytes(rowkey));
        put.addColumn(Bytes.toBytes(colFamily), Bytes.toBytes(col),
Bytes.toBytes(val));
        table.put(put);
        table.close();
}
```

4.8.3 HBase常用Java API及应用实例

添加数据时，需要分别设置参数myTableName、rowkey、colFamily、col、val的值，然后运行上述代码
例如添加第一行数据时，为insertData()方法指定相应参数，并运行如下3行代码：

```
insertData("student","zhangsan","score","English","69");
insertData("student","zhangsan","score","Math","86");
insertData("student","zhangsan","score","Computer","77");
```

上述代码与如下HBase Shell命令等效：

```
put 'student',’zhangsan’,’score:English’,’69’;
put 'student',’zhangsan’,’score:Math’,’86’;
put 'student',’zhangsan’,’score:Computer’,’77’;
```

4.8.3HBase常用Java API及应用实例

③浏览数据

```
    /*获取某单元格数据*/
    /**      * @param tableName 表名
             * @param rowKey 行键
             * @param colFamily 列族
             * @param col 列限定符
           * @throws IOException     */
      public static void getData(String tableName,String rowKey,String
colFamily,String col)throws  IOException{
           Table table = connection.getTable(TableName.valueOf(tableName));
           Get get = new Get(Bytes.toBytes(rowkey));
           get.addColumn(Bytes.toBytes(colFamily),Bytes.toBytes(col));
           //获取的result数据是结果集，还需要格式化输出想要的数据才行
           Result result = table.get(get);
           System.out.println(new
String(result.getValue(colFamily.getBytes(),col==null?null:col.getBytes())));
           table.close();
    }
```

4.8.3HBase常用Java API及应用实例

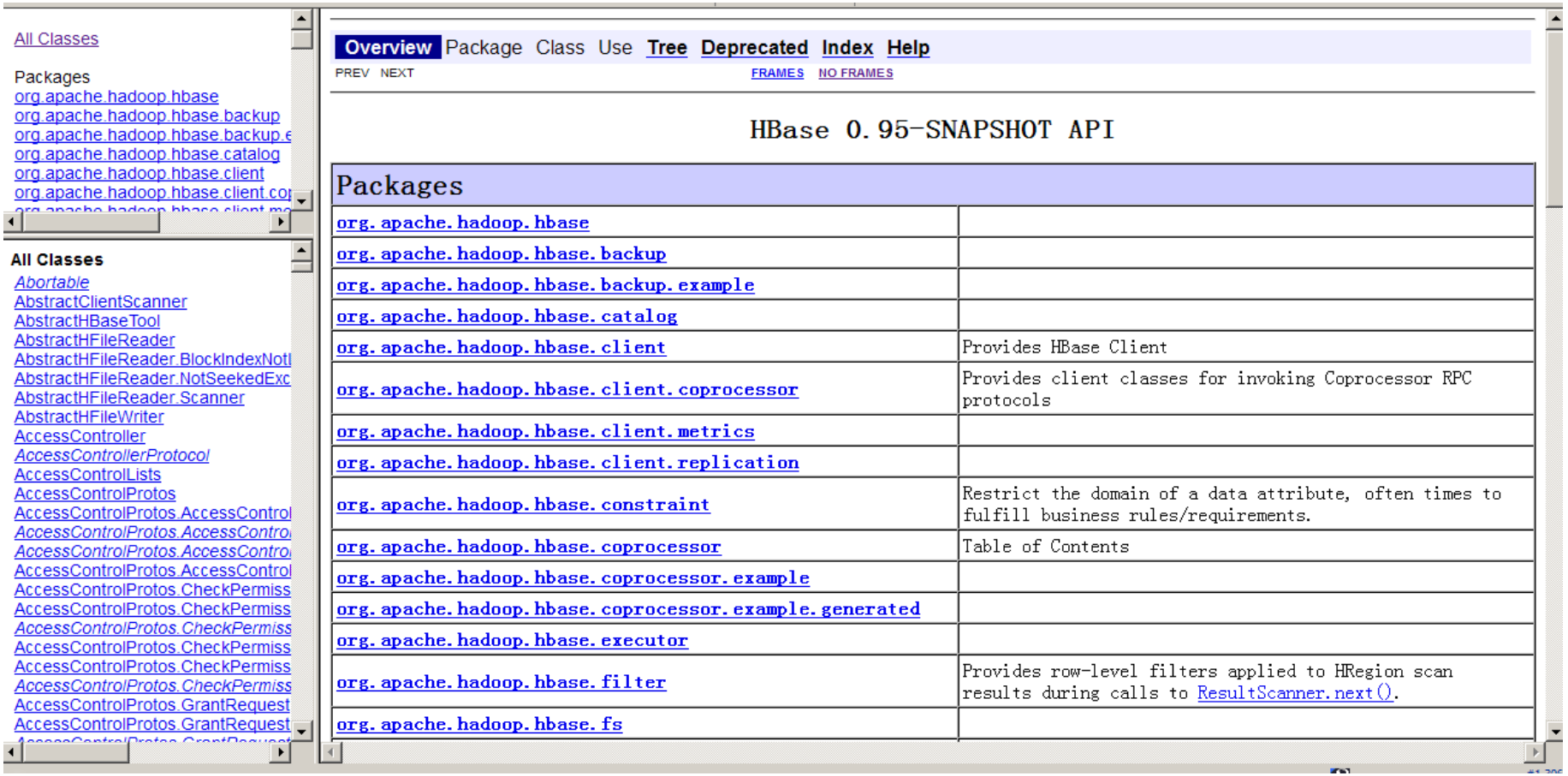
比如，现在要获取姓名为“zhangsan”在“English”上的数据，就可以在运行上述代码时，指定参数tableName为“student”、rowKey为“zhangsan”、colFamily为“score”、col为“English”。

```
getData("student", "zhangsan", "score", "English");
```

上述代码与如下HBase Shell命令等效：

```
get ‘student','zhangsan',{COLUMN=>'score:English'}”
```

Hbase API


All Classes
Packages
org.apache.hadoop.hbase
org.apache.hadoop.hbase.backup
org.apache.hadoop.hbase.catalog
org.apache.hadoop.hbase.client
All Classes
Abortable
AbstractClientScanner
AbstractHBaseTool
AbstractHFileReader
AccessController
AccessControlLists
AccessControlProtos
Overview Package Class Use Tree Deprecated Index Help
PREV NEXT
FRAMES NO FRAMES
HBase 0.95-SNAPSHOT API
Packages
org.apache.hadoop.hbase
org.apache.hadoop.hbase.backup
org.apache.hadoop.hbase.backup.example
org.apache.hadoop.hbase.catalog
org.apache.hadoop.hbase.client
Provides HBase Client
org.apache.hadoop.hbase.client.coprocessor
Provides client classes for invoking Coprocessor RPC protocols
org.apache.hadoop.hbase.client.metrics
org.apache.hadoop.hbase.client.replication
org.apache.hadoop.hbase.constraint
Restrict the domain of a data attribute, often times to fulfill business rules/requirements.
org.apache.hadoop.hbase.coprocessor
Table of Contents
org.apache.hadoop.hbase.coprocessor.example
org.apache.hadoop.hbase.coprocessor.example.generated
org.apache.hadoop.hbase.executor
org.apache.hadoop.hbase.filter
Provides row-level filters applied to HRegion scan results during calls to ResultScanner.next().
org.apache.hadoop.hbase.fs

本章小结

- 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，可以根据具体应用选择访问方式。
- HBase实际上就是一个稀疏、多维、持久化存储的映射表，它采用行键、列键和时间戳进行索引，每个值都是未经解释的字符串。
- HBase采用分区存储，一个大的表会被分拆许多个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。
- HBase运行机制和编程实践的知识。