获取教材和讲义 PPT 等各种课程资料请访问 http://dblab.xmu.edu.cn/node/422

=课程教材由林子雨老师根据网络资料编著=

厦门大学计算机科学系教师 林子雨 编著

http://www.cs.xmu.edu.cn/linziyu

2013 年 9 月

前言

本教程由厦门大学计算机科学系教师林子雨编著，可以作为计算机专业研究生课程《大数据技术基础》的辅助教材。

本教程的主要内容包括：大数据概述、大数据处理模型、大数据关键技术、大数据时代面临的新挑战、NoSQL 数据库、云数据库、Google Spanner、Hadoop、HDFS、HBase、MapReduce、Zookeeper、流计算、图计算和 Google Dremel 等。

本教程是林子雨通过大量阅读、收集、整理各种资料后精心制作的学习材料，与广大数据库爱好者共享。教程中的内容大部分来自网络资料和书籍，一部分是自己撰写。对于自写内容，林子雨老师拥有著作权。

本教程 PDF 文档及其全套教学 PPT 可以通过网络免费下载和使用（下载地址：http://dblab.xmu.edu.cn/node/422）。教程中可能存在一些问题，欢迎读者提出宝贵意见和建议！

本教程已经应用于厦门大学计算机科学系研究生课程《大数据技术基础》，欢迎访问 2013 班级网站 http://dblab.xmu.edu.cn/node/423。

林子雨的 E-mail 是：ziyulin@xmu.edu.cn。

林子雨的个人主页是：http://www.cs.xmu.edu.cn/linziyu。

林子雨于厦门大学海韵园

2013 年 9 月

第 7 章 HBase

厦门大学计算机科学系教师 林子雨 编著

个人主页：http://www.cs.xmu.edu.cn/linziyu

课程网址：http://dblab.xmu.edu.cn/node/422

2013 年 9 月

第 7 章 HBase

HBase 是一个分布式的、面向列的开源数据库，该技术来源于 Google 公司发表的论文“BigTable：一个结构化数据的分布式存储系统”。HBase 是 Apache 的 Hadoop 项目的一个子项目。就像 BigTable 利用了 Google 文件系统 GFS（Google File System）所提供的分布式数据存储一样，HBase 在 Hadoop 文件系统 HDFS（Hadoop Distributed File System）之上提供了类似于 BigTable 的能力。HBase 不同于一般的关系数据库，它是一个适合于非结构化数据存储的数据库，而且 HBase 采用了基于列而不是基于行的模式。

本章介绍 HBase 相关知识，内容要点如下：

- HBase 简介
- HBase 使用场景和成功案例
- HBase 和传统关系数据库的对比分析
- HBase 访问接口
- HBase 数据模型
- HBase 系统架构
- HBase 存储格式
- 读写数据
- MapReduce on HBase

7.1HBase 简介

HBase—— Hadoop Database，是一个高可靠性、高性能、面向列、可伸缩的分布式存储系统，利用 HBase 技术可在廉价 PC 服务器上搭建起大规模结构化存储集群。

HBase 是 Google BigTable 的开源实现，模仿并提供了基于 Google 文件系统的 BigTable 数据库的所有功能。类似 Google BigTable 利用 GFS 作为其文件存储系统，HBase 利用 Hadoop HDFS 作为其文件存储系统；Google 运行 MapReduce 来处理 BigTable 中的海量数据，HBase 同样利用 Hadoop MapReduce 来处理 HBase 中的海量数据；Google BigTable 利用 Chubby 作

为协同服务，HBase 利用 Zookeeper 作为协同服务。

HBase 仅能通过行键(row key)和行键的值域区间范围（range）来检索数据，并且仅支持单行事务(可通过 Hive 支持来实现多表连接等复杂操作)。HBase 主要用来存储非结构化和半结构化的松散数据。

HBase 可以直接使用本地文件系统或者 Hadoop 作为数据存储方式，不过为了提高数据可靠性和系统的健壮性，发挥 HBase 处理大数据量等功能，需要使用 Hadoop 作为文件系统。与 Hadoop 一样，HBase 目标主要依靠横向扩展，通过不断增加廉价的商用服务器来增加计算和存储能力。

HBase 的目标是处理非常庞大的表，可以用普通的计算机处理超过 10 亿行数据并且由数百万列元素组成的数据表。

HBase 中的表一般有这样的特点：

- **大**：一个表可以有上亿行，上百万列；
- **面向列**：面向列(族)的存储和权限控制，列(族)独立检索；
- **稀疏**：对于为空(null)的列，并不占用存储空间，因此，表可以设计得非常稀疏。

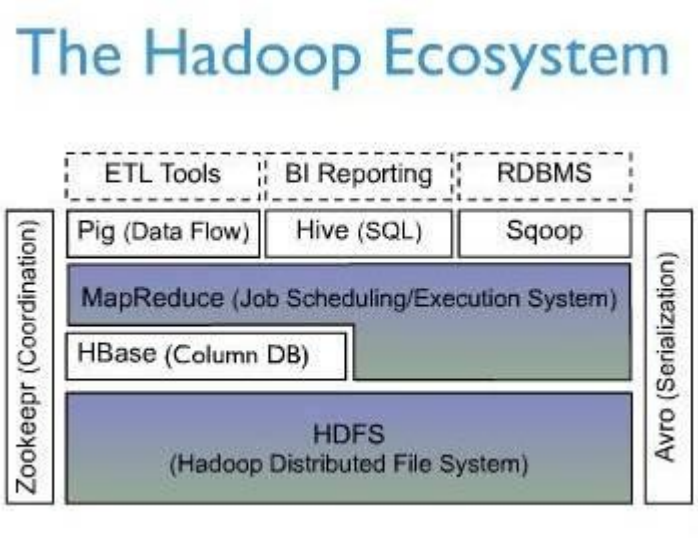

图 7-1 Hadoop 体系架构

图 7-1 描述了 Hadoop 生态系统中的各层系统，其中，HBase 位于结构化存储层，HDFS 为 HBase 提供了高可靠性的底层存储支持，MapReduce 为 HBase 提供了高性能的计算能力，Zookeeper 为 HBase 提供了稳定服务和失败恢复机制。

此外，Pig 和 Hive 还为 HBase 提供了高层语言支持，使得在 HBase 上进行数据统计处理变得非常简单。Sqoop 则为 HBase 提供了方便的 RDBMS 数据导入功能，使得传统数据库

数据向 HBase 中迁移变得非常方便。

7.2HBase 使用场景和成功案例

HBase 被证实是一个强大的工具，尤其是在已经使用 Hadoop 的场合。在其“婴儿期”的时候，它就快速部署到了其他公司的生产环境，并得到开发人员的支持。今天，HBase 已经是 Apache 顶级项目，有着众多的开发人员和兴旺的用户社区。它成为一个核心的基础架构部件，运行在世界上许多公司（如 StumbleUpon、Trend Micro、Facebook、Twitter、Salesforce 和 Adobe）的大规模生产环境中。

HBase 不是数据管理问题的“万能药”，针对不同的使用场景你可能需要考虑其他的技术。让我们看看现在 HBase 是如何使用的，人们用它构建了什么类型的应用系统。通过这个讨论，你会知道哪种数据问题适合使用 HBase。

有时候了解软件产品的最好方法是看看它是怎么用的。它可以解决什么问题和这些解决方案如何适用于大型应用架构，这些能够告诉你很多。因为 HBase 有许多公开的产品部署案例，我们正好可以这么做。下面将详细介绍一些成功使用 HBase 的使用场景。

HBase 模仿了 Google 的 BigTable，让我们先从典型的 BigTable 问题开始：存储互联网。

7.2.1 典型的互联网搜索问题：BigTable 发明的原因

搜索是一种定位你所关心信息的行为。例如，搜索一本书的页码，其中含有你想读的主题，或者搜索网页，其中含有你想找的信息。搜索含有特定词语的文档，需要查找索引，该索引提供了特定词语和包含该词语的所有文档的映射。为了能够搜索，首先必须建立索引。Google 和其他搜索引擎正是这么做的。它们的文档库是整个互联网，搜索的特定词语就是你在搜索框里敲入的任何东西。

BigTable 和模仿出来的 HBase，为这种文档库提供存储，BigTable 提供行级访问，所以，爬虫可以在 BigTable 中插入和更新单个文档，每个文档保存为 BigTable 中的一行。MapReduce 计算作业运行在 BigTable 的整张表上，就可以高效地生成搜索索引，为网络搜索应用做准备。当用户发起网络搜索请求时，网络搜索应用就会查询已经建立好的索引，直接从 BigTable 中得到匹配的文档，然后把搜索结果提交给用户。图 7-2 显示了互联网搜索应用中 BigTable 的关键角色。

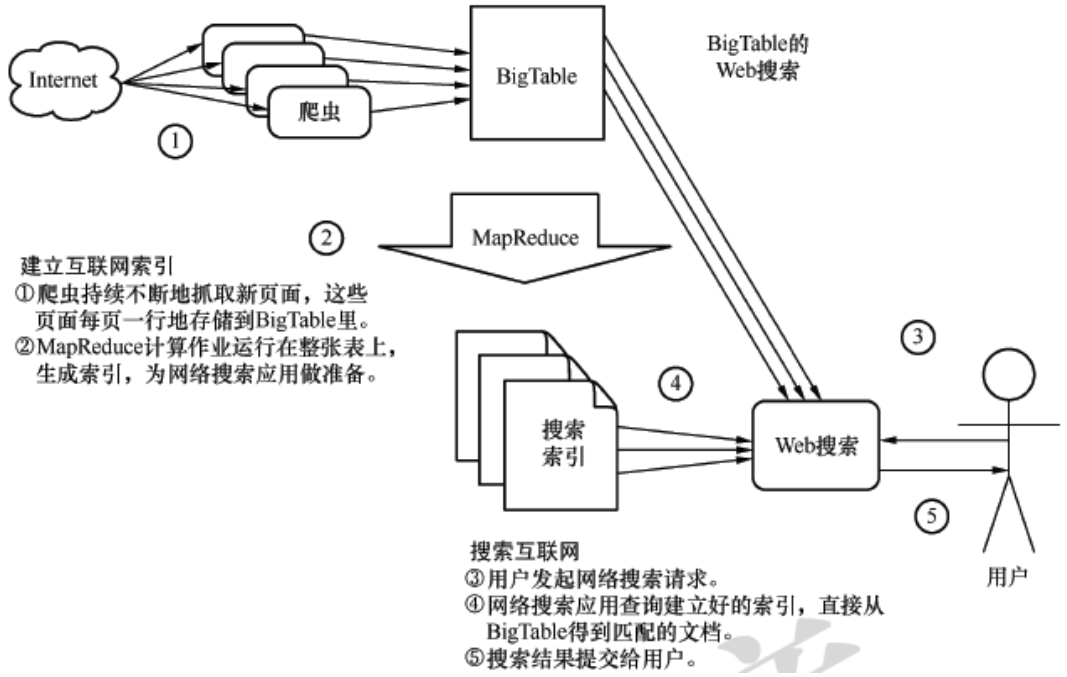

图 7-2 使用 BigTable 提供网络搜索结果

讲完典型 HBase 使用场景以后，我们来看看其他使用 HBase 的地方。愿意使用 HBase 的用户数量在过去几年里迅猛增长，部分原因在于 HBase 产品变得更加可靠且性能变得更好，更多原因在于越来越多的公司开始投入大量资源来支持和使用它。随着越来越多的商业服务供应商提供支持，用户越发自信地把 HBase 应用于关键应用系统中。一个设计初衷是用 HBase 来存储互联网持续更新的网页副本，用在互联网相关的其他方面也是很合适的。例如，HBase 在社交网络公司内部和周围各种各样的需求中，也找到了用武之地。从存储个人之间的通信信息，到通信信息分析，HBase 成为了 Facebook、Twitter 和 StumbleUpon 等公司的关键基础设施。

在这个领域，HBase 有 3 种主要使用场景，但不限于这 3 种。接下来我们将介绍这 3 种主要的使用场景：（1）抓取增量数据；（2）内容服务；（3）信息交换。

7.2.2 抓取增量数据

数据通常是细水长流的，不断累加到已有的数据库中以备将来使用，如分析、处理和服务。许多 HBase 使用场景属于这一类——使用 HBase 作为数据存储，抓取来自各种数据源的增量数据。例如，这种数据源可能是网页爬虫（我们讨论过的 BigTable 典型问题），可能是记录用户看了什么广告和看了多长时间的广告效果数据，也可能是记录各种参数的时间序列数据。我们讨论几个成功的使用场景，以及这些项目涉及的公司。

1．抓取监控指标：OpenTSDB

服务数百万用户的、基于 Web 的产品的后台基础设施，一般都有数百或数千台服务器。这些服务器承担了各种功能——服务流量、抓取日志、存储数据和处理数据等等。为了保证产品正常运行，对服务器及其上面运行的软件的健康状态进行监控，是至关重要的（从 OS 到用户交互应用）。大规模监控整个环境，需要能够采集和存储来自不同数据源各种监控指标的监控系统。每个公司都有自己的办法。一些公司使用商业工具来收集和展示监控指标，而另外一些公司则采用开源框架。

StumbleUpon 创建了一个开源框架，用来收集服务器的各种监控指标。按照时间收集监控指标一般被称为时间序列数据，也就是说，按照时间顺序收集和记录的数据。StumbleUpon 的开源框架叫做 OpenTSDB，它是 Open Time Series Database（开放时间序列数据库）的缩写。这个框架使用 HBase 作为核心平台来存储和检索所收集的监控指标。创建这个框架的目的是为了拥有一个可扩展的监控数据收集系统，一方面，能够存储和检索监控指标数据并保存很长时间，另一方面，如果需要增加功能，也可以添加各种新监控指标。StumbleUpon 使用 OpenTSDB 监控所有基础设施和软件，包括 HBase 集群自身。

2．抓取用户交互数据：Facebook 和 StumbleUpon

抓取监控指标是一种使用方式，还有一种使用方式是抓取用户交互数据。如何跟踪数百万用户在网站上的活动？怎么知道哪一个网站功能最受欢迎？怎样使得这一次网页浏览直接影响到下一次？例如，谁看了什么？某个按钮被点击了多少次？还记得 Facebook 和 Stumble 里的 Like 按钮和 StumbleUpon 里的+1 按钮吗？是不是听起来像是一个计数问题？每次用户喜欢一个特定主题，计数器就增加一次。

StumbleUpon 在开始阶段采用的是 MySQL，但是，随着网站服务越来越流行，这种技术选择遇到了问题。急剧增长的用户在线负载需求，远远超过了 MySQL 集群的能力，最终，StumbleUpon 选择使用 HBase 来替换这些集群。当时，HBase 产品不能直接提供必需的功能。StumbleUpon 在 HBase 上做了一些小的开发改动，后来将这些开发工作贡献回了项目社区。

FaceBook 使用 HBase 的计数器来计量人们喜欢特定网页的次数。内容原创人和网页主人可以得到近乎实时的、多少用户喜欢他们网页的数据信息。他们可以因此更敏捷地判断应该提供什么内容。Facebook 为此创建了一个叫 Facebook Insights 的系统，该系统需要一个可扩展的存储系统。公司考虑了很多种可能的选择，包括关系型数据库管理系统、内存数据库

和 Cassandra 数据库，最后决定使用 HBase。基于 HBase，Facebook 可以很方便地横向扩展服务规模，给数百万用户提供服务，还可以继续沿用他们已有的运行大规模 HBase 集群的经验。该系统每天处理数百亿条事件，记录数百个监控指标。

3．遥测技术：Mozilia 和 Trend Micro

软件运行数据和软件质量数据，不像监控指标数据那么简单。例如，软件崩溃报告是有用的软件运行数据，经常用来探究软件质量和规划软件开发路线图。HBase 可以成功地用来捕获和存储用户计算机上生成的软件崩溃报告。

Mozilla 基金会负责 FireFox 网络浏览器和 Thunderbird 电子邮件客户端两个产品。这些工具安装在全世界数百万台计算机上，支持各种操作系统。当这些工具崩溃时，会以 Bug 报告的形式返回一个软件崩溃报告给 Mozilla。Mozilla 如何收集这些数据？收集后又是怎么使用的呢？实际情况是这样的，一个叫做 Socorro 的系统收集了这些报告，用来指导研发部门研制更稳定的产品。Socorro 系统的数据存储和分析建构在 HBase 上。

使用 HBase，基本分析可以用到比以前多得多的数据。这种分析用来指导 Mozilla 的开发人员，使其更为专注，研制出 Bug 最少的版本。

Trend Micro 为企业客户提供互联网安全和入侵管理服务。安全的重要环节是感知，日志收集和分析对于提供这种感知能力是至关重要的。Trend Micro 使用 HBase 来管理网络信誉数据库，该数据库需要行级更新和支持 MapReduce 批处理。有点像 Mozilla 的 Socorro 系统，HBase 也用来收集和分析日志活动，每天收集数十亿条记录。HBase 中灵活的数据模式允许数据结构出现变化，当分析流程重新调整时，Trend Micro 可以增加新属性。

4．广告效果和点击流

过去十来年，在线广告成为互联网产品的一个主要收入来源。先提供免费服务给用户，在用户使用服务的时侯投放广告给目标用户。这种精准投放需要针对用户交互数据做详细的捕获和分析，以便理解用户的特征。基于这种特征，选择并投放广告。精细的用户交互数据会带来更好的模型，进而导致更好的广告投放效果，并获得更多的收入。但是，这类数据有两个特点：它以连续流的形式出现，它很容易按用户划分。理想情况下，这种数据一旦产生就能够马上使用，用户特征模型可以没有延迟地持续优化，也就是说，以在线方式使用。

HBase 非常适合收集这种用户交互数据，HBase 已经成功地应用在这种场合，它可以存

储第一手点击流和用户交互数据，然后用不同的处理方式（MapReduce 是其中一种）来处理数据（清理、丰富和使用数据）。在这类公司，你会发现很多 HBase 案例。

7.2.3 内容服务

传统数据库最主要的使用场合之一是为用户提供内容服务。各种各样的数据库支撑着提供各种内容服务的应用系统。多年来，这些应用一直在发展，因此，它们所依赖的数据库也在发展。用户希望使用和交互的内容种类越来越多。此外，由于互联网迅猛的增长以及终端设备更加迅猛的增长，对这些应用的接入方式提出了更高的要求。各种各样的终端设备带来了另一个挑战：不同的设备需要以不同的格式使用同样的内容。

上面说的是用户消费内容（user consuming content），另外一个完全不同的使用场景是用户生成内容（user generate content）。Twitter 帖子、Facebook 帖子、Instagram 图片和微博等都是这样的例子。

它们的相同之处是使用和生成了许多内容。大量用户通过应用系统来使用和生成内容，而这些应用系统需要 HBase 作为基础。

内容管理系统（Content Management System，CMS）可以集中管理一切，可以用来存储内容和提供内容服务。但是，当用户越来越多，生成的内容越来越多的时候，就需要一个更具可扩展性的 CMS 解决方案。可扩展的 Lily CMS 使用 HBase 作为基础，加上其他开源框架，如 Solr，构成了一个完整的功能组合。

Salesforce 提供托管 CRM 产品，这个产品通过网络浏览器界面提交给用户使用，显示出丰富的关系型数据库功能。在 Google 发表 NoSQL 原型概念论文之前很长一段时间，在生产环境中使用的大型关键数据库最合理的选择就是商用关系型数据库管理系统。多年来，Salesforce 通过数据库分库和尖端性能优化手段的结合扩展了系统处理能力，达到每天处理数亿事务的能力。

当 Salesforce 把分布式数据库系统列入选择范围后，他们评测了所有 NoSQL 技术产品，最后决定部署 HBase。一致性的需求是这个决定的主要原因。BigTable 类型的系统是唯一的架构方式，结合了无缝水平扩展能力和行级强一致性能力。此外，Salesforce 已经在使用 Hadoop 完成大型离线批处理任务，他们可以继续沿用 Hadoop 上面积累的宝贵经验。

1．URL 短链接

最近一段时间 URL 短链接非常流行，许多类似产品破土而出。StumbleUpon 使用名字为 su.pr.的短链接产品，这个产品以 HBase 为基础。这个产品用来缩短 URL，存储大量的短链接以及和原始长链接的映射关系，HBase 帮助这个产品实现扩展能力。

2．用户模型服务

经 HBase 处理过的内容往往并不直接提交给用户使用，而是用来决定应该提交给用户什么内容。这种中间处理数据用来丰富用户的交互。

还记得前面提到的广告服务场景里的用户特征吗？用户特征（或者说模型）就是来自 HBase。这种模型多种多样，可以用于多种不同场景。例如，针对特定用户投放什么广告的决定，用户在电商网站购物时实时报价的决定，用户在搜索引擎检索时增加背景信息和关联内容，等等。很多这种使用案例可能不便于公开讨论，说多了我们就有麻烦了。

当用户在电商网站上发生交易时，Runa 用户模型服务可以用来实时报价。这种模型需要基于不断产生的新用户数据持续调优。

7.2.4 信息交换

各种社交网络破土而出，世界变得越来越小。社交网站的一个重要作用就是帮助人们进行互动。有时互动在群组内发生（小规模和大规模），有时互动在两个个人之间发生。想想看，数亿人通过社交网络进行对话的场面。单单和远处的人对话还不足以让人满意，人们还想看看和其他人对话的历史记录。让社交网络公司感到幸运的是，保存这些历史记录很廉价，大数据领域的创新可以帮助他们充分利用廉价的存储。

在这方面，Facebook 短信系统经常被公开讨论，它也可能极大地推动了 HBase 的发展。当你使用 Facebook 时，某个时候你可能会收到或者发送短信给你的朋友。Facebook 的这个特性完全依赖于 HBase。用户读写的所有短信都存储在 HBase 里。Facebook 短信系统要求：高的写吞吐量，极大的表，数据中心内的强一致性。除了短信系统之外，其他应用系统要求：高的读吞吐量，计数器吞吐量，自动分库。工程师们发现 HBase 是一个理想的解决方案，因为它支持所有这些特性，它拥有一个活跃的用户社区，Facebook 运营团队在 Hadoop 部署上有丰富经验等等。在“Hadoop goes realtime at Facebook”这篇文章里，Facebook 工程师解释了这个决定背后的逻辑并展示了他们使用 Hadoop 和 HBase 构建在线系统的经验。

Facebook 工程师在 HBaseCon 2012 大会上分享了一些有趣的数据。在这个平台上每天

交换数十亿条短信，每天带来大约 750 亿次操作。尖峰时刻，Facebook 的 HBase 集群每秒发生 150 万次操作。从数据规模角度看，Facebook 的集群每月增加 250TB 的新数据，这可能是已知的最大的 HBase 部署，无论是服务器的数量还是服务器所承载的用户量。

上述一些示例，解释了 HBase 如何解决一些有趣的老问题和新问题。你可能注意到一个共同点：HBase 可以用来对相同数据进行在线服务和离线处理。这正是 HBase 的独到之处。

7.3HBase 和传统关系数据库的对比分析

HBase 与以前的关系数据库存在很大的区别，它是按照 BigTable 来开发的，是一个稀疏的、分布的、持续多维度的排序映射数组。

HBase 就是这样一个基于列模式的映射数据库，它只能表示很简单的“键-数据”的映射关系，它大大简化了传统的关系数据库。二者具体区别如下：

- **数据类型**：HBase 只有简单的字符串类型，所有类型都是交由用户自己处理，它只保存字符串。而关系数据库有丰富的类型选择和存储方式。
- **数据操作**：HBase 操作只有很简单的插入、查询、删除、清空等，表和表之间是分离的，没有复杂的表和表之间的关系，所以，不能也没有必要实现表和表之间的关联等操作。而传统的关系数据通常有各种各样的函数、连接操作。
- **存储模式**：HBase 是基于列存储的，每个列族都有几个文件保存，不同列族的文件是分离的。传统的关系数据库是基于表格结构和行模式保存的。
- **数据维护**：HBase 的更新，确切地说，应该不叫更新，而是一个主键或者列对应的新的版本，而它旧有的版本仍然会保留，所以，它实际上是插入了新的数据，而不是传统关系数据库里面的替换修改。
- **可伸缩性**：HBase 和 BigTable 这类分布式数据库就是直接为了这个目的开发出来的，能够轻易地增加或者减少（在硬件错误的时候）硬件数量，而且对错误的兼容性较高。而传统的关系数据库通常需要增加中间层才能实现类似的功能。

当前的关系数据库基本都是从上世纪 70 年代发展而来的，它们基本都有以下的体系特点：

- 面向磁盘存储和索引结构；
- 多线程访问；

- 基于锁的同步访问机制；
- 基于日志记录的恢复机制。

而 BigTable 和 HBase 之类基于列模式的分布式数据库，更适应海量存储和互联网应用的需求，灵活的分布式架构可以使其利用廉价的硬件设备组建一个大的数据仓库。互联网应用是以字符为基础的，BigTable 和 HBase 就是针对这些应用而开发出来的数据库。由于其中的时间戳特性，BigTable 和 HBase 与生俱来就特别适合于开发 wiki、archive.org 之类的服务，而 HBase 直接就是作为一个搜索引擎的一部分被开发出来的。

7.4HBase 访问接口

- **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**：支持 REST 风格的 Http API 访问 HBase，解除了语言限制；
- **Pig**：可以使用 Pig Latin 流式编程语言来操作 HBase 中的数据，和 Hive 类似，最终也是编译成 MapReduce 作业来处理 HBase 表数据，适合做数据统计；
- **Hive**：可以使用类似 SQL 语言来访问 HBase。

7.5 HBase 数据模型

7.5.1 概述

HBase 是一个类似 BigTable 的分布式数据库，大部分特性和 BigTable 一样，是一个稀疏的、长期存储的（存在硬盘上）、多维度的、排序的映射表。这张表的索引是行关键字、列关键字和时间戳。每个值是一个不解释的字符数组，数据都是字符串，没有类型。

用户在表中存储数据（如表 7-1 所示），每一行都有一个可排序的主键和任意多的列。由于是稀疏存储的，所以，同一张表里面的每一行数据都可以有截然不同的列。

列名字的格式是“<family>:<label>”，都是由字符串组成，每一张表有一个 family 集合，这个集合是固定不变的，相当于表的结构，只能通过改变表结构来改变。但是，label 值相

对于每一行来说都是可以改变的。

HBase 把同一个 family 里面的数据存储在同一个目录底下，而 HBase 的写操作是锁行的，每一行都是一个原子元素，都可以加锁。

所有数据库的更新都有一个时间戳标记，每个更新都是一个新的版本，而 HBase 会保留一定数量的版本，这个值是可以设定的。客户端可以选择获取距离某个时间最近的版本，或者一次获取所有版本。

表 7-1 HBase 数据实例

Row Key	Timestamp	Column Family	
		URI	Parser
r1	t3	url=http://www.taobao.com	title=天天特价
	t2	host=taobao.com	
	t1		
r2	t5	url=http://www.alibaba.com	content=每天…
	t4	host=alibaba.com	

7.5.2 数据模型相关概念

在 HBase 数据模型中，包括如下三个重要概念：

- 行键（**Row Key**）：HBase 表的主键，表中的记录按照行键排序；
- 时间戳（**Timestamp**）：每次数据操作对应的时间戳，可以看作是数据的版本号；
- 列族（**Column Family**）：表在水平方向有一个或者多个列族组成，一个列族中可以由任意多个列组成，即列族支持动态扩展，无需预先定义列的数量以及类型，所有列均以二进制格式存储，用户需要自行进行类型转换。

1. 行键

行键是用来检索记录的主键。访问 HBase 表中的行，只有三种方式：

- 通过单个行键访问
- 通过行键的区间范围

- 全表扫描

行键可以是任意字符串(最大长度是 64KB，实际应用中长度一般为 10-100bytes)，在 HBase 内部，行键保存为字节数组。存储时，数据按照行键的字典序(byte order)排序存储。设计键时，要充分考虑这个特性，将经常一起读取的行存储放到一起 (位置相关性) 。注意：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)。这个设计决策能够使用户很容易地理解程序在对同一个行进行并发更新操作时的行为。

2. 列族

HBase 表中的每个列都归属于某个列族。列族是表的模式的一部分(而列不是表的模式的一部分)，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我们管理不同类型的应用：我们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）。

3. TimeStamp

HBase 中通过行和列确定的一个存储单元称为 cell。每个 cell 都保存着同一份数据的多个版本。不同的版本是通过时间戳来进行索引的，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。cell 中的数据是没有类型的，全部是字节码形式存储。

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。

7.5.3 概念视图

一个表可以想象成一个大的映射关系，通过主键，或者主键+时间戳，可以定位一行数据，由于是稀疏数据，所以某些列可以是空白的，表 7-2 就是 HBase 存储数据的概念视图。

表 7-2 HBase 数据的概念视图

Row Key	**Time Stamp**	**Column** *"contents:"*	**Column** *"anchor:"*		**Column** *"mime:"*
"com.cnn.www"	t9		"anchor:cnnsi.com"	"CNN"	
	t8		"anchor:my.look.ca"	"CNN.com"	
	t6	"<html>..."			"text/html"
	t5	"<html>..."			
	t3	"<html>..."			

表 7-2 是一个存储 Web 网页的范例列表片断。行名是一个反向 URL（即 com.cnn.www）。contents 列族用来存放网页内容，anchor 列族存放引用该网页的锚链接文本。CNN 的主页被 Sports Illustrater（即所谓 SI，CNN 的王牌体育节目）和 MY-look 的主页引用，因此，该行包含了名叫“anchor:cnnsi.com”和 “anchhor:my.look.ca”的列。每个锚链接只有一个版本（由时间戳标识，如 t9，t8）；而 contents 列则有三个版本，分别由时间戳 t3，t5 和 t6 标识。

7.5.4 物理视图

虽然从概念视图来看，HBase 中的每个表格是由很多行组成的，但是，在物理存储上面，它是按照列来保存的，这点在数据设计和程序开发的时候必须牢记。表 7-2 的概念视图在物理存储的时候应该表现成类似表 7-3 的样子。

表 7-3 HBase 数据的物理视图

Row Key	**Time Stamp**	**Column** *"contents:"*
"com.cnn.www"	t6	"<html>..."
	t5	"<html>..."

	t3	"<html>..."

Row Key	**Time Stamp**	**Column** *"anchor:"*	
"com.cnn.www"	t9	"anchor:cnnsi.com"	"CNN"
	t8	"anchor:my.look.ca"	"CNN.com"

Row Key	**Time Stamp**	**Column** *"mime:"*
"com.cnn.www"	t6	"text/html"

需要注意的是，在概念视图（见表 7-2）上面有些列是空白的，这样的列实际上并不会被存储，当请求这些空白的单元格的时候，会返回 null 值。

如果在查询的时候不提供时间戳，那么会返回距离现在最近的那一个版本的数据。因为在存储的时候，数据会按照时间戳排序。

例子 1：一个程序写 10 行数据，row[0-9]，先写入 anchor:foo 列，再写入 anchor:bar 列，最后重复写入 anchor:foo 列，由于是同一个列族，写到同一个映射文件里面，最后写到文件里面是下面这个样子的：

```
row=row0， column=anchor:bar， timestamp=1174184619081
row=row0， column=anchor:foo， timestamp=1174184620720
row=row0， column=anchor:foo， timestamp=1174184617161
row=row1， column=anchor:bar， timestamp=1174184619081
row=row1， column=anchor:foo， timestamp=1174184620721
row=row1， column=anchor:foo， timestamp=1174184617167
row=row2， column=anchor:bar， timestamp=1174184619081
row=row2， column=anchor:foo， timestamp=1174184620724
row=row2， column=anchor:foo， timestamp=1174184617167
row=row3， column=anchor:bar， timestamp=1174184619081
```

row=row3， column=anchor:foo， timestamp=1174184620724

row=row3， column=anchor:foo， timestamp=1174184617168

row=row4， column=anchor:bar， timestamp=1174184619081

row=row4， column=anchor:foo， timestamp=1174184620724

row=row4， column=anchor:foo， timestamp=1174184617168

row=row5， column=anchor:bar， timestamp=1174184619082

row=row5， column=anchor:foo， timestamp=1174184620725

row=row5， column=anchor:foo， timestamp=1174184617168

row=row6， column=anchor:bar， timestamp=1174184619082

row=row6， column=anchor:foo， timestamp=1174184620725

row=row6， column=anchor:foo， timestamp=1174184617168

row=row7， column=anchor:bar， timestamp=1174184619082

row=row7， column=anchor:foo， timestamp=1174184620725

row=row7， column=anchor:foo， timestamp=1174184617168

row=row8， column=anchor:bar， timestamp=1174184619082

row=row8， column=anchor:foo， timestamp=1174184620725

row=row8， column=anchor:foo， timestamp=1174184617169

row=row9， column=anchor:bar， timestamp=1174184619083

row=row9， column=anchor:foo， timestamp=1174184620725

row=row9， column=anchor:foo， timestamp=1174184617169

其中 anchor:foo 被保存了两次，由于时间戳不同，是两个不同的版本，而最新的数据排在前面，所以最新那次更新会先被找到。

7.6HBase 的实现

7.6.1 表和 HRegion

HBase 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 HMaster 主服务器；(3) 许多个 HRegion 服务器。HRegion 服务器可以根据工作负载的变化，从一个

簇中动态地增加或删除。主服务器 HMaster 负责把 HRegion（类似于 BigTable 中的 Tablet）分配到 HRegion 服务器，探测 HRegion 服务器的增加和过期，进行 HRegion 服务器的负载均衡，以及 HDFS 文件系统中的垃圾收集。除此以外，它还处理模式变化，比如表和列族的创建。

每个 HRegion 服务器管理一个 HRegion 集合，通常在每个 HRegion 服务器上，会放置 10 到 1000 个 HRegion。HRegion 服务器处理针对那些已经加载的 HRegion 而提出的读写请求，并且会对过大的 HRegion 进行划分。

就像许多单服务器分布式存储系统一样，客户端并不是直接从主服务器读取数据，而是直接从 HRegion 服务器上读取数据。因为 HBase 客户端并不依赖于主服务器 HMaster 来获得 HRegion 的位置信息，所以，大多数客户端从来不和主服务器 HMaster 通信，从而使得在实际应用中，主服务器负载很小。

一个 HBase 中存储了许多表。每个表都是一个 HRegion 集合，每个 HRegion 包含了位于某个值域区间内的所有数据。在最初阶段，每个表只包含一个 HRegion。随着表的增长，它会被自动分解成许多 HRegion，每个 HRegion 默认尺寸大约是 100 到 200MB。

注：本章内容是林子雨老师根据网络资料整理编写的，对于这部分内容，网络资料的描述并不规范。比如，网络资料中，某个地方正文文字描述用 HRegion，而图片中文字或其他地方的文字描述却用没有“H”前缀的 Region，类似的情形包括 HStore 和 Store，HRegionServer 和 RegionServer，HMaster 和 Master 等，其实二者是等价的，即 HRegion 和 Region 是同一个概念。为了规范，本章内容在正文描述部分，全部统一成 HRegion、HStore 和 HRegionServer 这种格式，但是，由于图片无法修改（重新画图工作量比较大，暂时没有时间做这个工作），因此，图片中可能仍然出现 Region、Store、RegionServer 等概念。

关于 HBase 中的表和 HRegion 的概念，总结如下：

- 表中的所有行都按照行键的字典序排列；
- 表在行的方向上分割为多个 HRegion（如图 7-3 所示）；

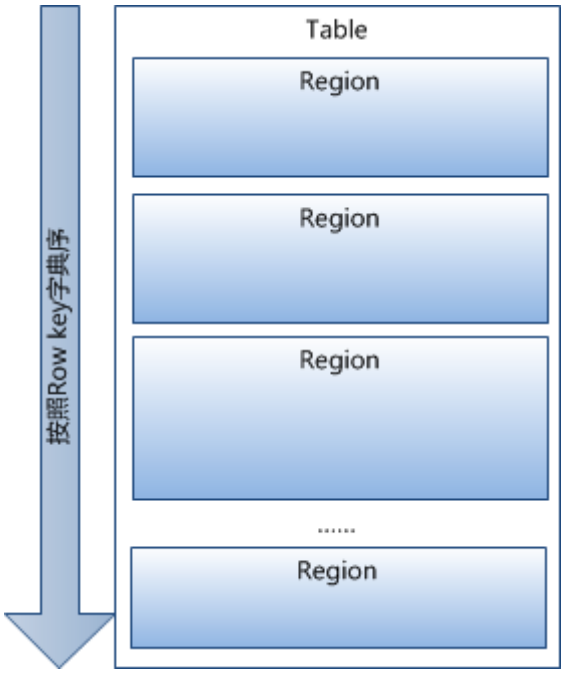

图 7-3 HBase 中的一个表包含多个 HRegion

- HRegion 会按照大小进行分割，每个表一开始只有一个 HRegion，随着数据不断插入表，HRegion 不断增大，当增大到一个阀值的时候，HRegion 就会等分成两个新的 HRegion（如图 7-4 所示）。表中的行不断增多，就会有越来越多的 HRegion。一个 HRegion 由[startkey，endkey)表示，不同的 HRegion 会被 HMaster 分配给相应的 HRegionServer 进行管理。

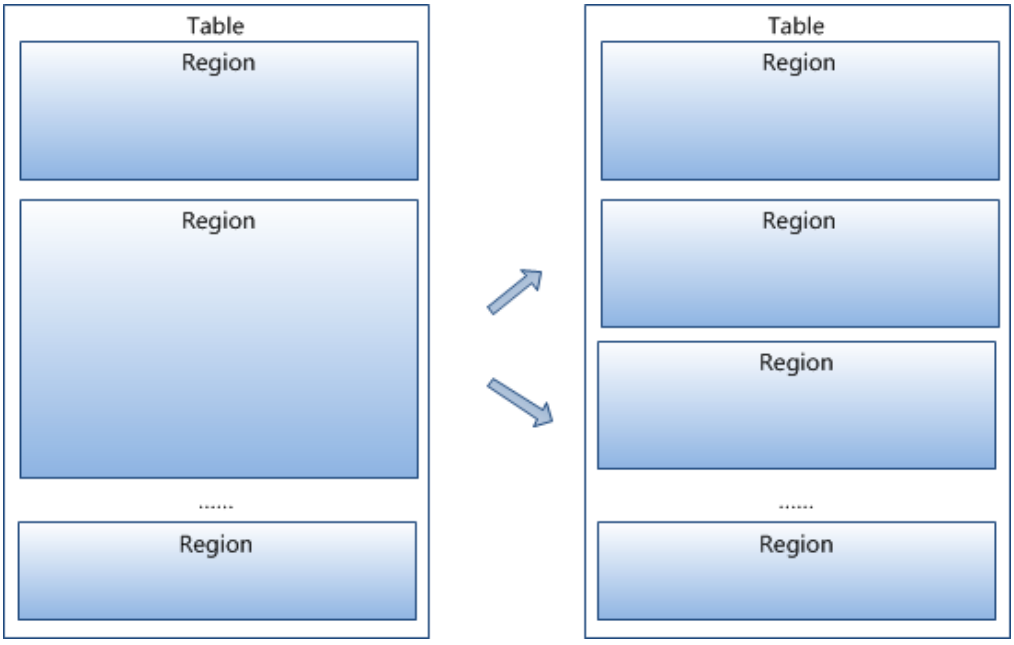

图 7-4 HBase 中的一个 HRegion 会分裂成多个新的 HRegion

- HRegion 是 HBase 中分布式存储和负载均衡的最小单元。最小单元就表示不同的

Hregion 可以分布在不同的 HRegionServer 上，但是，同一个 HRegion 是不会拆分到多个 HRegionServer 上的（如图 7-5 所示）。

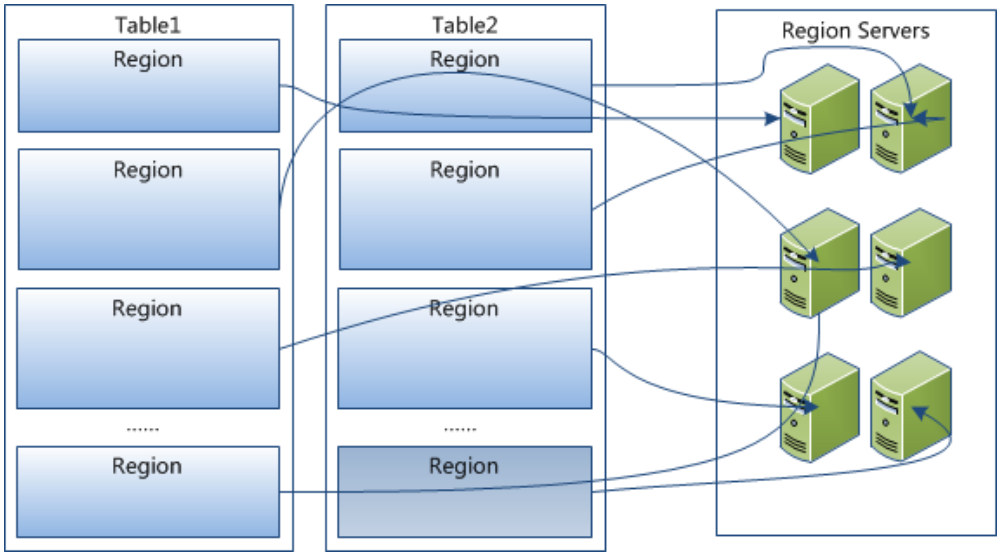

图 7-5 不同的 HRegion 可以分布在不同的 HRegionServer 上

- HRegion 虽然是分布式存储的最小单元，但并不是底层存储的最小单元。事实上，HRegion 由一个或者多个 HStore 组成，每个 HStore 保存一个列族。每个 HStrore 又由一个 memStore 和 0 至多个 HStoreFile 组成。HStoreFile 以 HFile 格式保存在 HDFS 上（如图 7-6 所示）。

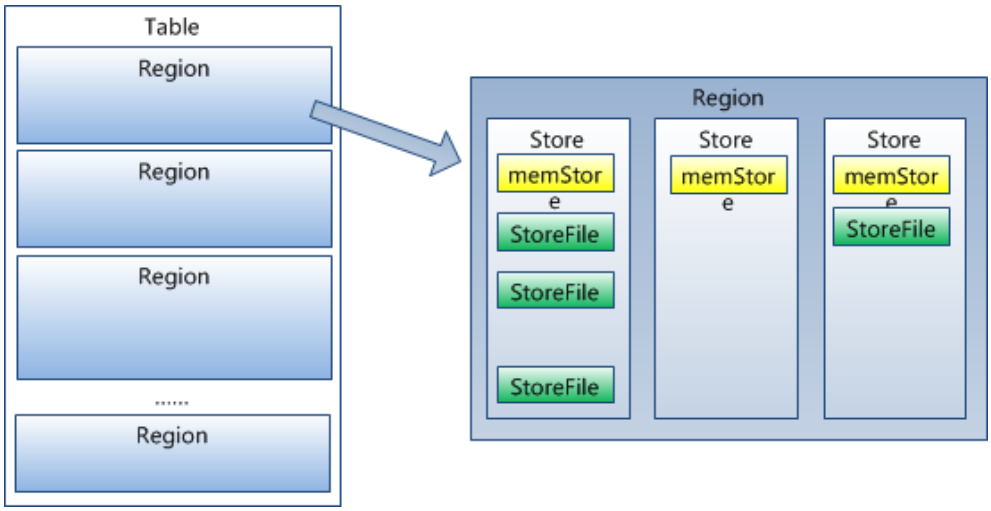

图 7-6 每个 HStore 由一个 memStore 和 0 至多个 HStoreFile 组成

7.6.2 HRegion 的定位

7.6.2.1 HBase 三层结构

HBase 使用类似 B+树的三层结构来保存 HRegion 位置信息：

- 第一层是 Zookeeper 文件：它记录了-ROOT-表的位置信息，即 root region 的位置信息；
- 第二层是-ROOT-表：只包含一个 root region，记录了.META.表中的 region 信息。通过 root region，我们就可以访问.META.表的数据。
- 第三层是.META.表：是一个特殊的表，记录了用户表的 HRegion 信息，.META.表可以有多个 HRegion，保存了 HBase 中所有数据表的 HRegion 位置信息。

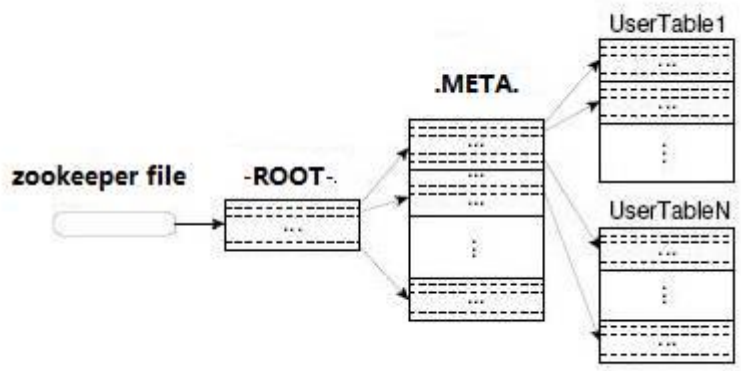

图 7-7 HBase 的三层结构

Client 访问用户数据之前，需要首先访问 Zookeeper，然后访问-ROOT-表，接着访问.META.表，最后才能找到用户数据的位置去访问，中间需要多次网络操作，不过 client 端会做 cache 缓存（如图 7-8 所示）。

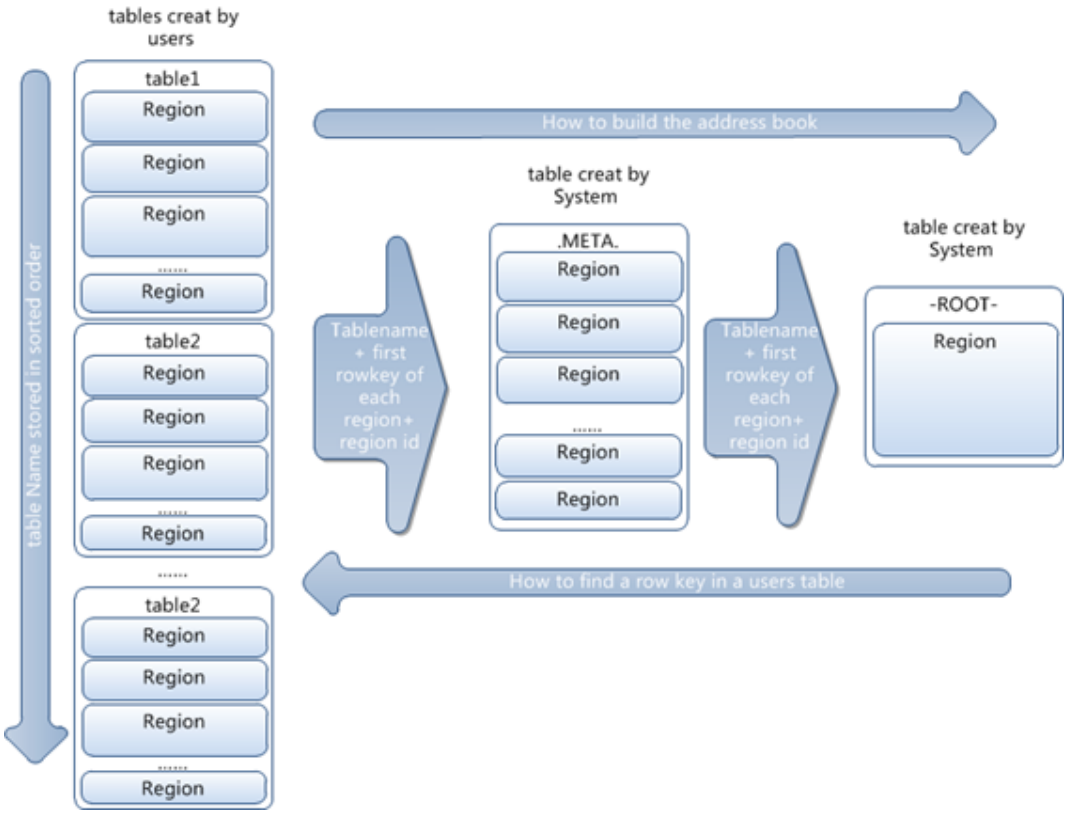

图 7-8 Client 访问用户数据过程

关于 HRegion 定位，需要进行几点说明：

- root region 永远不会被分裂，保证了最多需要三次跳转，就能定位到任意 HRegion；
- .META.表每行保存一个 HRegion 的位置信息，行键采用“表名+HRegion 的第一个行键+HRegion 的 ID”编码而成；
- 为了加快访问，.META.表的全部 HRegion 都保存在内存中；假设.META.表的一行在内存中大约占用 1KB，并且每个 HRegion 限制为 128MB。那么，上面的三层结构可以保存的 HRegion 数目为：(128MB/1KB) * (128MB/1KB) = 2^{34} 个 HRegion；
- client 会将查询过的位置信息保存缓存起来，缓存不会主动失效，因此，如果 client 上的缓存全部失效，则需要进行 6 次网络来回，才能定位到正确的 HRegion(其中三次用来发现缓存失效，另外三次用来获取位置信息)。

7.6.2.2 关于.META.表

之前我们说过，HRegion 是按照表名和主键范围区分的，由于主键范围是连续的，所以一般用开始主键就可以表达出来。

但是，如果只有开始主键还是不够的，因为我们有合并和分割操作，如果正好在执行这

些操作的过程中出现死机，那么就可能存在多份表名和开始主键一样的数据，这个就要通过 HBase 的元数据信息来区分哪一份才是正确的数据文件了，为了区分这样的情况，每个 HRegion 都有一个 HRegionId 来标识它的唯一性。

所以，一个 HRegion 的表达式最后是：表名+开始主键+唯一 id（tablename + startkey + regionId）。

例子：hbaserepository，w-nk5YNZ8TBb2uWFIRJo7V==，6890601455914043877

我们可以用这个识别符来区分不同的 HRegion，这些数据就称为“元数据”。而元数据本身也是被保存在 HRegion 里面的，我们称呼这个表为“元数据表”(.META.表)，里面保存的就是 HRegion 标识符和实际 HRegion 服务器的映射关系。

元数据表本身也会增长，并且可能被分割为几个 HRegion，为了定位这些 HRegion，有一个根数据表（ROOT table），保存了所有元数据表的位置，而根数据表是不能被分割的，永远之存在一个 HRegion。

在 HBase 启动的时候，主服务器先去扫描根数据表，因为这个表只会有一个 HRegion，所以这个 HRegion 的名字是被写死的。当然要把根数据表分配到一个 HRegion 服务器需要一定的时间。

当根数据表被分配好之后，主服务器就会去扫描根数据表，获取元数据表的名字和位置，然后把元数据表分配到不同的 HRegion 服务器。

最后就是扫描元数据表，找到所有 HRegion 区域的信息，然后把它们分配给不同的 HRegion 服务器。

主服务器在内存中保存着当前活跃的 HRegion 服务器的数据，因此，如果主服务器死机的话，整个系统也就无法访问了，而服务器的信息也没有必要保存到文件里面。

元数据表和根数据表的每一行都包含一个列族——info 列族：

- info:regioninfo：包含了一个串行化的 HRegionInfo 对象。
- info:server：保存了一个字符串，是服务器地址 HServerAddress.toString()。
- info:startcode：一个长整型的数字的字符串，是 HRegion 服务器启动的时候传给主服务器的，让主服务器决定这个 Hregion 服务器的信息有没有更改。

因此，当一个客户端从 Zookeeper 服务器上拿到根数据表地址以后，就可以直接访问相应的 HRegion 服务器获得数据，没有必要再连接主服务器。因此，主服务器的负载相对就小了很多，它只会处理超时的 HRegion 服务器，在启动的时候扫描根数据表和元数据表，和返回根数据表的 HRegion 服务器地址。

因此，HBase 的客户端是十分复杂的，它经常需要浏览元数据表和根数据表，在查询表的时候，如果一个 HRegion 服务器死机或者它上面的数据更改了，客户端就会继续重试，客户端保留的映射关系并不会一直正确的。这里的机制还需要进一步完善。

7.6.2.3 总结

关于 HRegion 的定位，总结如下：

- HRegion 服务器提供对 HRegion 的访问，一个 HRegion 只会保存在一个 HRegion 服务器上面。
- HRegion 会注册到主服务器上面。
- 如果主服务器死机，那么整个系统都会无效。
- 当前的 HRegion 服务器列表只有主服务器知道。
- HRegion 区域和 HRegion 服务器的对应关系保存在两个特别的 HRegion 里面（根数据表和元数据表），它们像其它 HRegion 一样被分配到不同的服务器。
- 根数据表是最特别的一个表，主服务器永远知道它的位置（在程序中写死）。
- 客户端需要自己浏览这些表，来找到数据在哪里。

7.7HBase 系统架构

图 7-9 给出了 HBase 的系统架构，围绕该架构，下面我们将分别介绍 Client、Zookeeper、HMaster、HRegionServer、HStore、HRegion 分配、HRegionServer 上线、HRegionServer 下线、HMaster 上线和 HMaster 下线。

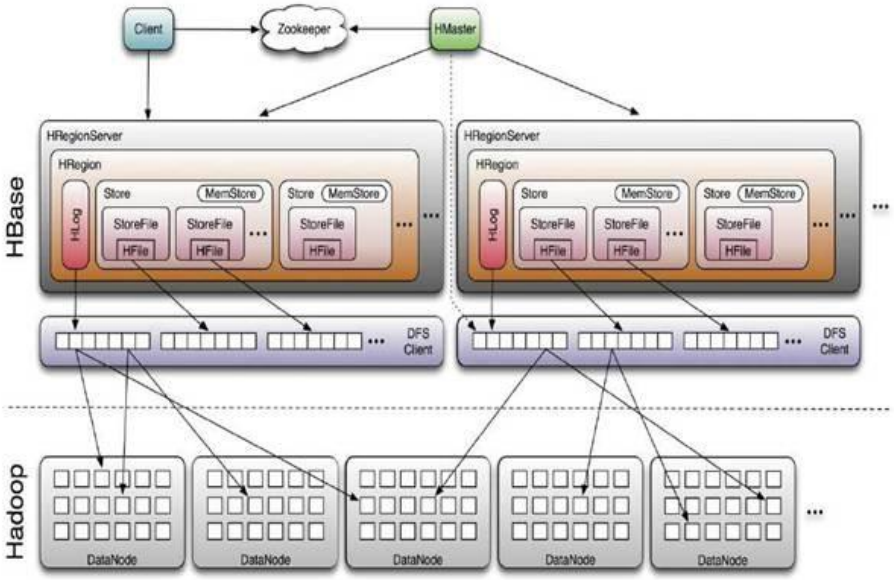

图 7-9 HBase 的系统架构

7.7.1 Client

Client 包含访问 HBase 的接口，client 维护着一些缓存来加快对 HBase 的访问，比如 HRegion 的位置信息。

HBase Client 使用 HBase 的 RPC 机制与 HMaster 和 HRegionServer 进行通信，对于管理类操作，Client 与 HMaster 进行 RPC；对于数据读写类操作，Client 与 HRegionServer 进行 RPC。

7.7.2 Zookeeper

Zookeeper 中除了存储-ROOT-表的地址和 HMaster 的地址，HRegionServer 也会把自己以“Ephemeral”方式注册到 Zookeeper 中，使得 HMaster 可以随时感知到各个 HRegionServer 的健康状态。此外，Zookeeper 也避免了 HMaster 的单点问题。关于 Zookeeper 的作用，这里总结说明如下：

- 保证任何时候，集群中只有一个 HMaster；
- 存储所有 HRegion 的寻址入口；
- 实时监控 HRegionServer 的状态，将 HRegionServer 的上线和下线信息实时通知给 HMaster；
- 存储 HBase 的 schema，包括有哪些表，每个表有哪些列族。

7.7.3 HMaster

HMaster 没有单点问题，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，HMaster 在功能上主要负责表和 HRegion 的管理工作：

- 管理用户对表的增、删、改、查操作；
- 管理 HRegionServer 的负载均衡，调整 HRegion 分布；
- 在 HRegion 分裂后，负责新 HRegion 的分配；
- 在 HRegionServer 停机后，负责失效 HRegionServer 上的 HRegion 的迁移。

每个 HRegion 服务器都会和 HMaster 服务器通讯，HMaster 的主要任务就是要告诉每个 HRegion 服务器它要维护哪些 HRegion。

HMaster 服务器会和每个 HRegion 服务器保持一个长连接。如果这个连接超时或者断开，会导致：

- HRegion 服务器自动重启。
- HMaster 认为 HRegion 已经死机，同时把它负责的 HRegion 分配到其它 HRegion 服务器。

和 Google 的 BigTable 不同的是，当 BigTable 的 TabletServer 和主服务器通讯中断的情况下，它仍然能提供服务。而 HBase 不能这么做，因为，HBase 没有 BigTable 那样额外的加锁系统，BigTable 是由主服务器管理 TabletServer，同时加锁服务器提供数据访问的，而 HBase 只有唯一一个接入点，就是 HMaster 服务器。

当一个新的 HRegion 服务器登陆到 HMaster 服务器，HMaster 会告诉它先等待分配数据。而当一个 HRegion 死机的时候，HMaster 会把它负责的 HRegion 标记为未分配，然后把它们分配到其它 HRegion 服务器。

7.7.4 HRegionServer

HRegionServer 主要负责响应用户 I/O 请求，向 HDFS 文件系统中读写数据，是 HBase 中最核心的模块（如图 7-10 所示）。

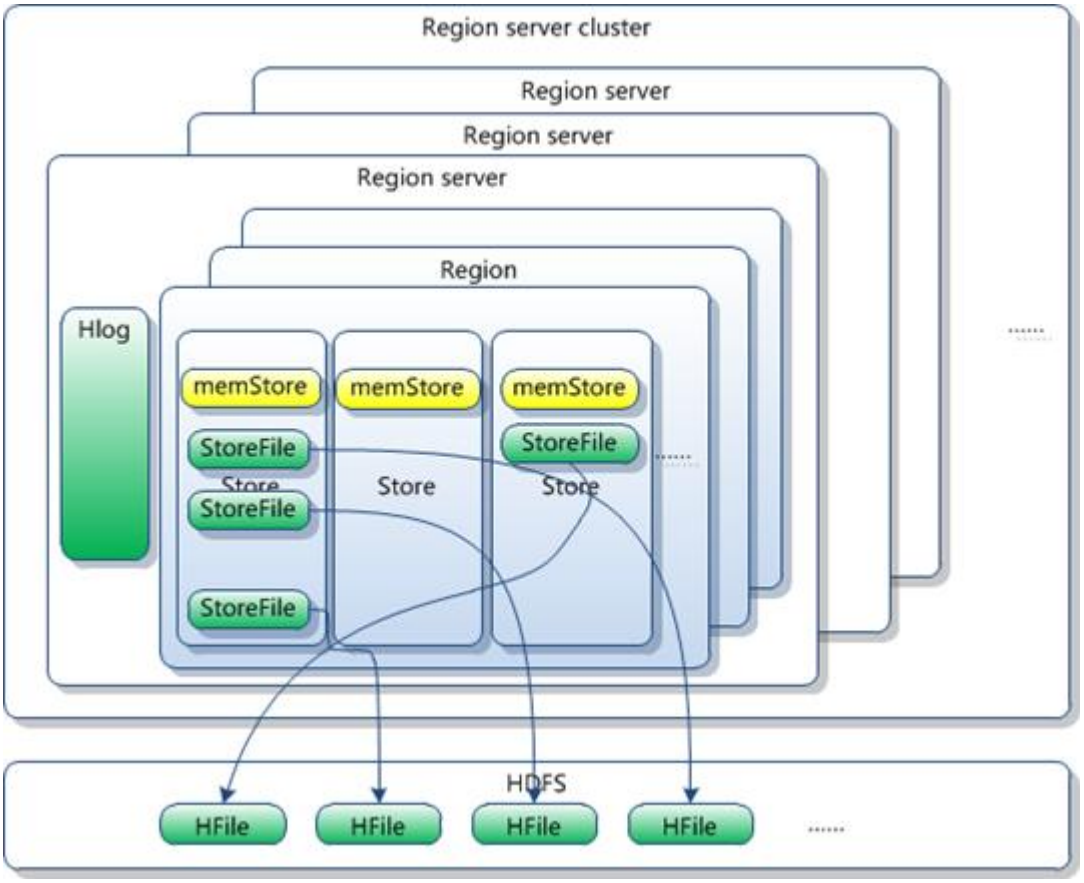

图 7-10 HRegionServer 向 HDFS 文件系统中读写数据

HRegionServer 内部管理了一系列 HRegion 对象，每个 HRegion 对应着表（Table）中的一个 HRegion，HRegion 由多个 HStore 组成。每个 HStore 对应了表中的一个列族的存储，可以看出，每个列族其实就是一个集中的存储单元，因此，最好将具备共同 IO 特性的列放在一个列族中，这样最高效。

Client 访问 HBase 上数据的过程并不需要 HMaster 参与（寻址访问 Zookeeper 和 HRegionServer，数据读写访问 HRegionServer），HMaster 仅仅维护着表和 HRegion 的元数据信息，负载很低。

对于用户来说，每个表是一堆数据的集合，靠主键来区分。物理上，一张表是被拆分成多个块，每一块就称为一个 HRegion。用“表名+开始/结束主键”，来区分一个 HRegion，一个 HRegion 会保存一个表里面某段连续的数据，从开始主键到结束主键，一张完整的表是被分开保存在多个 HRegion 上面的。

所有的 HBase 数据库数据一般是保存在 Hadoop 分布式文件系统 HDFS 上面，用户通过一系列 HRegion 服务器获取这些数据，一般而言，一台机器上面运行一个 HRegion 服务器，而每一个区段 HRegion 只会被一个 HRegion 服务器维护。

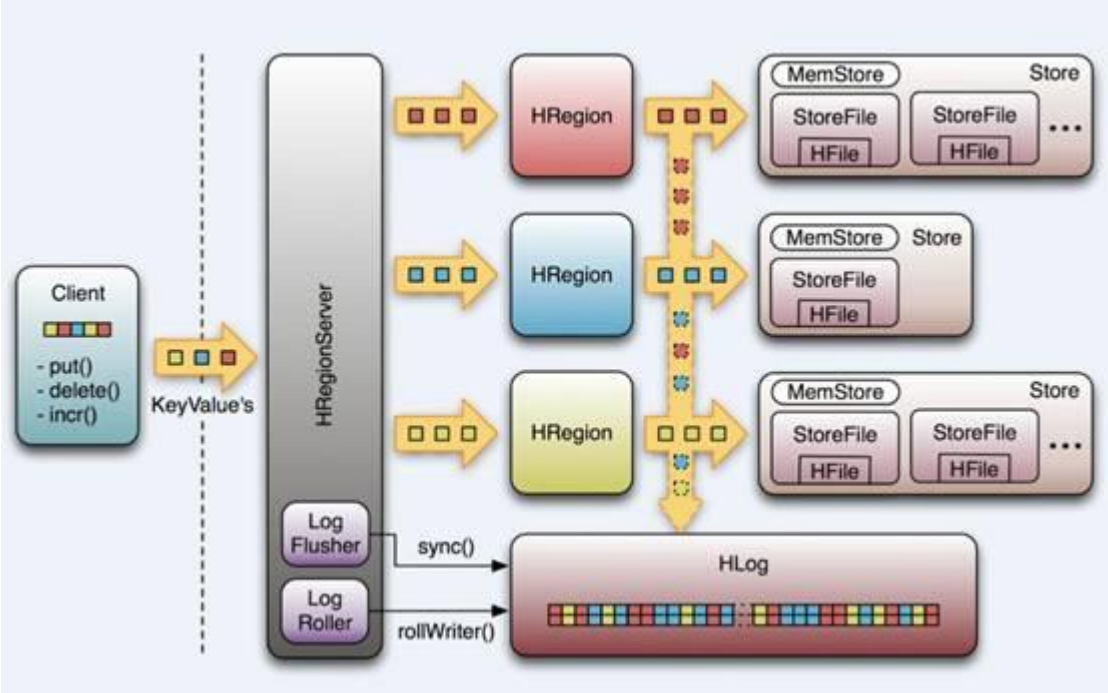

图 7-11 HRegionServer 和 HMemStore 缓存、Hlog 文件

图 7-11 给出了更新和读取数据的过程。当用户需要更新数据的时候，他会被分配到对应的 HRegion 服务器提交修改，这些修改先是被写到 HMemStore 缓存和服务器的 HLog 文件里面，HMemStore 是在内存中的缓存，保存最近更新的数据，HLog 是磁盘上面的记录文件，它记录着所有的更新操作，当操作写入 HLog 之后，commit()调用才会返回给客户端。

当读取数据的时候，HRegion 服务器会先访问 HMemStore 缓存，如果缓存里面没有该数据，才会到 HStore 磁盘上面寻找，每一个列族都会有一个 HStore，每个 HStore 包含很多 HStoreFiles 具体文件，这些文件都是 B 树结构的，方便快速读取。

系统会定期调用 HRegion.flushcache() 把 HMemStore 缓存里面的内容写到文件中，一般这样会增加一个新的 HStoreFile 文件，而此时高速缓存就会被清空，并且写入一个标记到 HLog，表示上面的内容已经被写入到文件中保存。

在启动的时候，每个 HRegion 服务器都会检查自己的 HLog 文件，看看最近一次执行 flushcache 之后有没有新的更新写入操作。如果没有更新，就表示所有数据都已经更新到文件中了；如果有更新，服务器就会先把这些更新写入高速缓存，然后调用 flushcache 写入到文件。最后，服务器会删除旧的 HLog 文件，并开始给用户访问数据。

因此，为了节省时间，可以少去调用 flushcache，但是，这样会增加内存占用，而且，在服务器重启的时候会延长很多时间。如果可以定期调用 flushcache，缓存大小会控制在一个较低的水平，而且，HLog 文件也会很快地重构，但是，调用 flushcache 的时候会造成系

统负载瞬间增加。

HLog 会被定期回滚，回滚的时候是按照时间备份文件，每当回滚的时候，系统会删除那些已经被写到文件中的更新，回滚 HLog 只会占用很少的时间，建议经常回滚以减少文件尺寸。

每一次调用 flushcache 会生成一个新的 HStoreFile 文件，从一个 HStore 里面获取一个值都需要访问所有的 HStoreFile 文件，这样十分耗时，所以，我们要定期把这些分散的文件合并到一个大文件里面，HStore.compact()就可以完成这样的工作。这样的合并工作是十分占用资源的，当 HStoreFile 文件的数量超过一个设定值的时候才会触发。

Google 的 BigTable 有高级合并和低级合并的区别，但是，HBase 没有这个概念，只要记住下面两点就可以了：

（1）flushcache 会建立一个新的 HStoreFile 文件，并把缓存中所有需要更新的数据写到文件里面，flushcache 之后，HLog 的重建次数会清零。

（2）compact 会把所有 HstoreFile 文件合并成一个大文件。

和 BigTable 不同的是，HBase 每个更新如果是被正确提交了并且没有返回错误的话，它就一定是被写到记录文件里面了，这样不会造成数据丢失。

两个 HRegion 可以通过调用 HRegion.closeAndMerge()合并成一个新的 HRegion，当前版本这个操作是需要两台 HRegion 都停机才能操作。

当一个 HRegion 变得太过巨大的时候，超过了设定的阈值，HRegion 服务器会调用 HRegion.closeAndSplit()，这个 HRegion 会被拆分为两个，并且报告给主服务器让它决定由哪个 HRegion 服务器来存放新的 HRegion。这个拆分过程是十分迅速的，因为，两个新的 HRegion 最初只是保留原来 HRegionFile 文件的引用，而这个时候旧的 HRegion 会处于停止服务的状态，当新的 HRegion 构建完成并且把引用删除了以后，旧的 HRegion 才会删除。

最后总结几点：

（1）客户端以表的形式读取数据；

（2）一张表是被划分成多个 HRegion 区域；

（3）HRegion 是被 HRegionServer 管理的，当客户端需要访问某行数据的时候，需要访问对应的 HRegionServer。

（4）HRegionServer 里面有三种方式保存数据：

A、 HMemStore 高速缓存，保留最新写入的数据；

B、 HLog 记录文件，保留的是提交成功了、但未被写入文件的数据；

C、 HStore 文件，数据的物理存放形式。

7.7.5 HStore

HStore 存储是 HBase 存储的核心了，由两部分组成，一部分是 HMemStore，一部分是 HStoreFile。HMemStore 是排序的内存缓冲区，用户写入的数据首先会放入 HMemStore，当 HMemStore 满了以后会 Flush 成一个 HStoreFile（底层实现是 HFile），当 HStoreFile 文件数量增长到一定阈值，会触发合并操作，将多个 HStoreFile 合并成一个 HStoreFile，合并过程中会进行版本合并和数据删除，因此可以看出 HBase 其实只有增加数据，所有的更新和删除操作都是在后续的 compact 过程中进行的，这使得用户的写操作只要进入内存中就可以立即返回，保证了 HBase I/O 的高性能。当 HStoreFile 合并后，会逐步形成越来越大的 HStoreFile，当单个 HStoreFile 大小超过一定阈值后，会触发分裂操作，同时把当前 HRegion 分裂成 2 个 HRegion，父 HRegion 会下线，新分裂出的 2 个孩子 HRegion 会被 HMaster 分配到相应的 HRegionServer 上，使得原先 1 个 HRegion 的压力得以分流到 2 个 HRegion 上。图 7-12 描述了合并和分裂的过程。

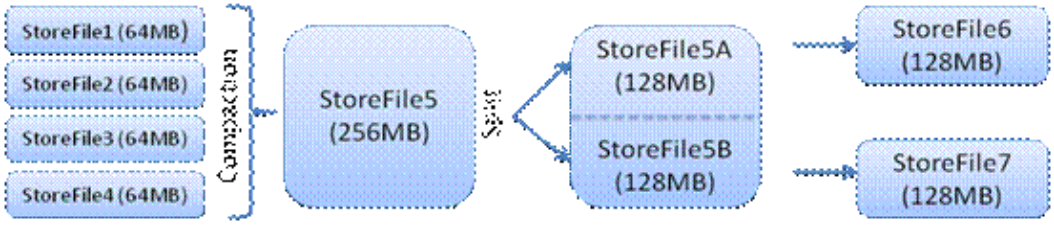

图 7-12 HStoreFile 的合并和分裂过程

在理解了上述 HStore 的基本原理后，还必须了解一下 HLog 的功能，因为，上述的 HStore 在系统正常工作的前提下是没有问题的，但是，在分布式系统环境中，无法避免系统出错或者宕机，因此，一旦 HRegionServer 意外退出，HMemStore 中的内存数据将会丢失，这就需要引入 HLog 了。每个 HRegionServer 中都有一个 HLog 对象，HLog 是一个实现写前日志（Write Ahead Log）的类，在每次用户操作写入 HMemStore 的同时，也会写一份数据到 HLog 文件中（HLog 文件格式见后续），HLog 文件定期会滚动出新的，并删除旧的文件（已持久化到 HStoreFile 中的数据）。当 HRegionServer 意外终止后，HMaster 会通过 Zookeeper 感知到，HMaster 首先会处理遗留的 HLog 文件，将其中不同 HRegion 的 HLog 数据进行拆分，分别放到相应 HRegion 的目录下，然后再将失效的 HRegion 重新分配，领取到这些 HRegion 的 HRegionServer 在加载 HRegion 的过程中，会发现有历史 HLog 需要处理，因此会 Replay

HLog 中的数据到 HMemStore 中，然后刷新到 HStoreFiles，完成数据恢复。

7.7.6 HRegion 分配

任何时刻，一个 HRegion 只能分配给一个 HRegionServer。HMaster 记录了当前有哪些可用的 HRegionServer，以及当前哪些 HRegion 分配给了哪些 HRegionServer，哪些 HRegion 还没有分配。当存在未分配的 HRegion 时，并且有一个 HRegionServer 上有可用空间时，HMaster 就给这个 HRegionServer 发送一个装载请求，把 HRegion 分配给这个 HRegionServer。HRegionServer 得到请求后，就开始对此 HRegion 提供服务。

7.7.7 HRegionServer 上线

HMaster 使用 Zookeeper 来跟踪 HRegionServer 的状态。当某个 HRegionServer 启动时，会首先在 Zookeeper 上的 server 目录下建立代表自己的文件，并获得该文件的独占锁。由于 HMaster 订阅了 server 目录上的变更消息，当 server 目录下的文件出现新增或删除操作时，HMaster 可以得到来自 Zookeeper 的实时通知。因此，一旦 HRegionServer 上线，HMaster 能马上得到消息。

7.7.8 HRegionServer 下线

当 HRegionServer 下线时，它和 Zookeeper 的会话断开，Zookeeper 而自动释放代表这台 server 的文件上的独占锁。而 HMaster 不断轮询 server 目录下文件的锁状态。如果 HMaster 发现某个 HRegionServer 丢失了它自己的独占锁(或者 HMaster 连续几次和 HRegionServer 通信都无法成功)，HMaster 就尝试去获取代表这个 HRegionServer 的读写锁，一旦获取成功，就可以确定下面两种情形中的一种发生了：

- HRegionServer 和 Zookeeper 之间的网络断开了；
- HRegionServer 挂了。

无论哪种情况，HRegionServer 都无法继续为它的 HRegion 提供服务了，此时 HMaster 会删除 server 目录下代表这台 HRegionServer 的文件，并将这台 HRegionServer 的 HRegion 分配给其它还活着的 HRegionServer。

如果网络短暂出现问题导致 HRegionServer 丢失了它的锁，那么 HRegionServer 重新连

接到 Zookeeper 之后，只要代表它的文件还在，它就会不断尝试获取这个文件上的锁，一旦获取到了，就可以继续提供服务。

7.7.9 HMaster 上线

HMaster 启动时，需要执行以下步骤：

- 第一步：从 Zookeeper 上获取唯一一个代表该 HMaster 的锁，用来阻止其它 HMaster 成为主服务器；
- 第二步：扫描 Zookeeper 上的 server 目录，获得当前可用的 HRegionServer 列表；
- 第三步：和第二步中的每个 HRegionServer 通信，获得当前已分配的 HRegion 和 HRegionServer 的对应关系；
- 第四步：扫描.META.中 HRegion 的集合，计算得到当前还未分配的 HRegion，将他们放入待分配 HRegion 列表。

7.7.10 HMaster 下线

由于 HMaster 只维护表和 HRegion 的元数据，而不参与表数据 IO 的过程，HMaster 下线，仅导致所有元数据的修改被冻结(无法创建删除表，无法修改表的 schema，无法进行 HRegion 的负载均衡，无法处理 HRegion 上下线，无法进行 HRegion 的合并，唯一例外的是 HRegion 的分裂可以正常进行，因为只有 HRegionServer 参与)，表的数据读写还可以正常进行。因此，HMaster 下线短时间内对整个 HBase 集群没有影响。从上线过程可以看到，HMaster 保存的信息全是可以冗余信息（都可以从系统其它地方收集到或者计算出来），因此，一般 HBase 集群中总是有一个 HMaster 在提供服务，还有一个以上的 HMaster 在等待时机抢占它的位置。

7.8 HBase 存储格式

HBase 中的所有数据文件都存储在 Hadoop 分布式文件系统 HDFS 上，主要包括上述提出的两种文件类型：

- **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 HStoreFile 就是对 HFile 做了轻量级包装，即 HStoreFile 底层就是 HFile。

- **HLog File**：HBase 中 WAL（Write Ahead Log）的存储格式，物理上是 Hadoop 的顺序文件。

7.8.1 HFile

图 7-13 描述了 HFile 的存储格式。

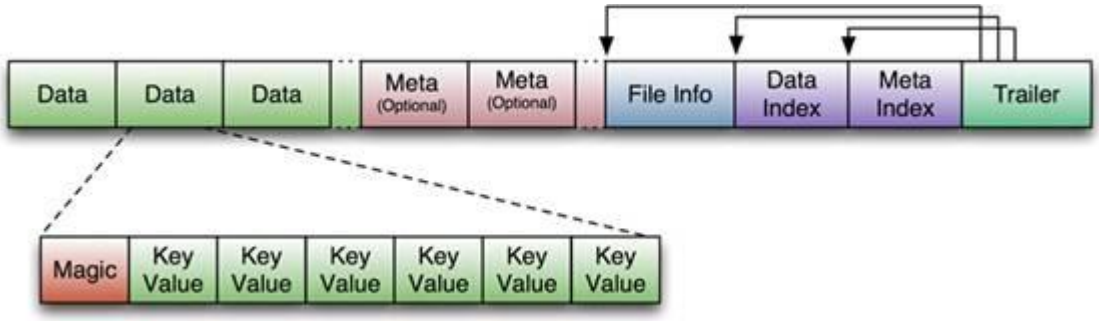

图 7-13 HFile 的存储格式

HFile 分为六个部分：

- 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 key/value 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 的 Data Block 和 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 和 LZO。

HFile 文件是不定长的，长度固定的只有其中的两块：Trailer 和 FileInfo。正如图 7-13 中所示的，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表的时候通过参数指定，大号的 Block 有利于顺序扫描，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 Key/Value 对的内部构造。

HFile 里面的每个 Key/Value 对就是一个简单的 byte 数组。但是这个 byte 数组里面包含了很多项，并且有固定的结构（如图 7-14 所示）。

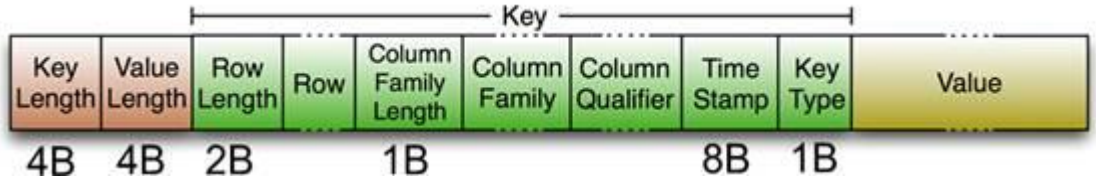

图 7-14 Hfile 里面的 Key/Value 的具体结构

开始是两个固定长度的数值，分别表示 Key 的长度和 Value 的长度。紧接着是 Key，开始是固定长度的数值，表示 RowKey 的长度，紧接着是 RowKey，然后是固定长度的数值，表示 Family 的长度，然后是 Family，接着是 Qualifier，然后是两个固定长度的数值，表示 Time Stamp 和 Key Type（Put/Delete）。Value 部分没有这么复杂的结构，就是纯粹的二进制数据了。

7.8.2 HLogFile

HLog 又称 WAL。WAL 意为 Write Ahead Log，类似 Mysql 中的 binlog，用来做灾难恢复，HLog 记录数据的所有变更，一旦数据修改，就可以从 HLog 中进行恢复。

每个 HRegion Server 维护一个 HLog，而不是每个 HRegion 一个。这样不同 HRegion(来自不同表)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，可以提高对表的写性能。带来的麻烦是，如果一台

HRegionServer 下线，为了恢复其上的 HRegion，需要将 HRegionServer 上的 HLog 进行拆分，然后分发到其它 HRegionServer 上进行恢复。

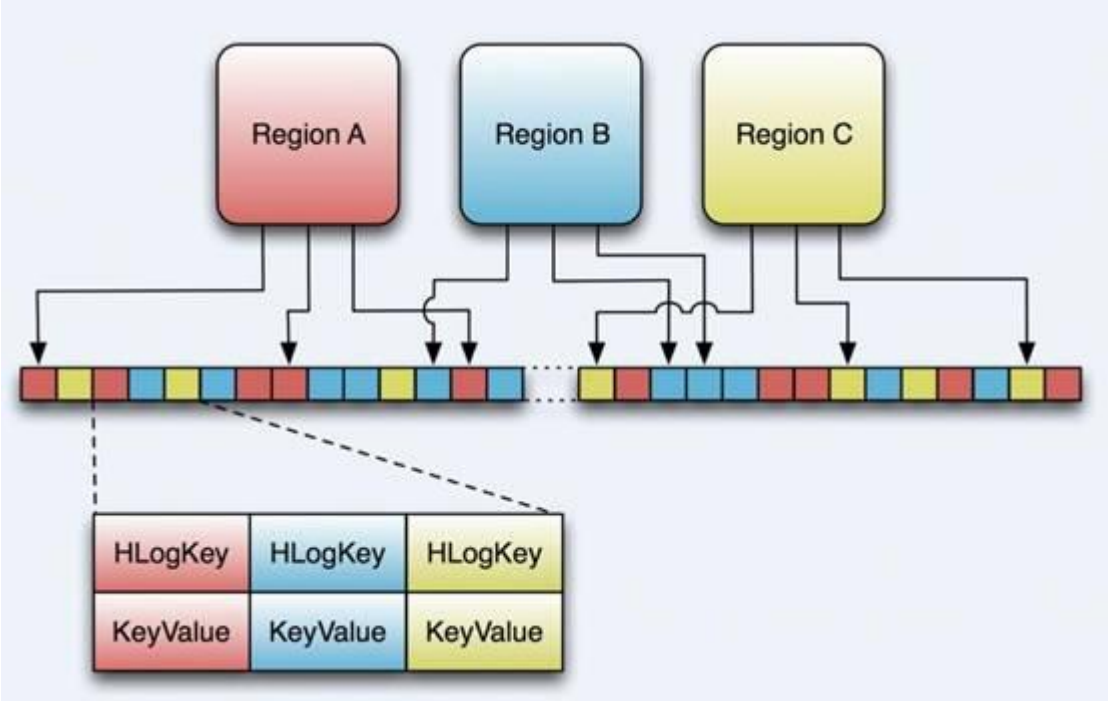

图 7-15 Hlog 文件的结构

图 7-15 给出了 HLog 文件的结构，其实，HLog 文件就是一个普通的 Hadoop 顺序文件（Sequence File），顺序文件的 Key 是 HLogKey 对象，HLogKey 中记录了写入数据的归属信息，除了表和 HRegion 名字外，同时还包括顺序号和时间戳，时间戳是“写入时间”，顺序号的起始值为 0，或者是最近一次存入文件系统中顺序号。HLog 顺序文件的 Value 是 HBase 的 Key/Value 对象，即对应 HFile 中的 Key/Value。

7.9读写数据

HBase 使用 HMemStore 和 HStoreFile 存储对表的更新。

数据在更新时，首先写入 HLog 和内存(HMemStore)中，HMemStore 中的数据是排序的，当 HMemStore 累计到一定阈值时，就会创建一个新的 HMemStore，并且将老的 HMemStore 添加到 flush 队列，由单独的线程 flush 到磁盘上，成为一个 HStoreFile。与此同时，系统会在 Zookeeper 中记录一个检查点，表示这个时刻之前的变更已经持久化了。

当系统出现意外时，可能导致内存(HMemStore)中的数据丢失，此时使用 HLog 来恢复

检查点之后的数据。

HStoreFile 是只读的，一旦创建后就不可以再修改。因此，HBase 的更新其实是不断追加的操作。当一个 HStore 中的 HStoreFile 达到一定的阈值后，就会进行一次合并，将对同一个 key 的修改合并到一起，形成一个大的 HStoreFile，当 HStoreFile 的大小达到一定阈值后，又会对 HStoreFile 进行分裂，等分为两个 HStoreFile。

由于对表的更新是不断追加的，处理读请求时，需要访问 HStore 中全部的 HStoreFile 和 HMemStore，将它们按照行键进行合并，由于 HStoreFile 和 HMemStore 都是经过排序的，并且 HStoreFile 带有内存中索引，合并的过程还是比较快的。

写请求处理过程具体如下：

- client 向 HRegionServer 提交写请求；
- HRegionServer 找到目标 HRegion；
- HRegion 检查数据是否与 schema 一致；
- 如果客户端没有指定版本，则获取当前系统时间作为数据版本；
- 将更新写入 HLog；
- 将更新写入 HMemstore；
- 判断 HMemStore 是否需要 flush 为 HStore 文件。

7.10 MapReduce on HBase

在 HBase 系统上运行批处理运算，最方便和实用的模型依然是 MapReduce，如图 7-16 所示。

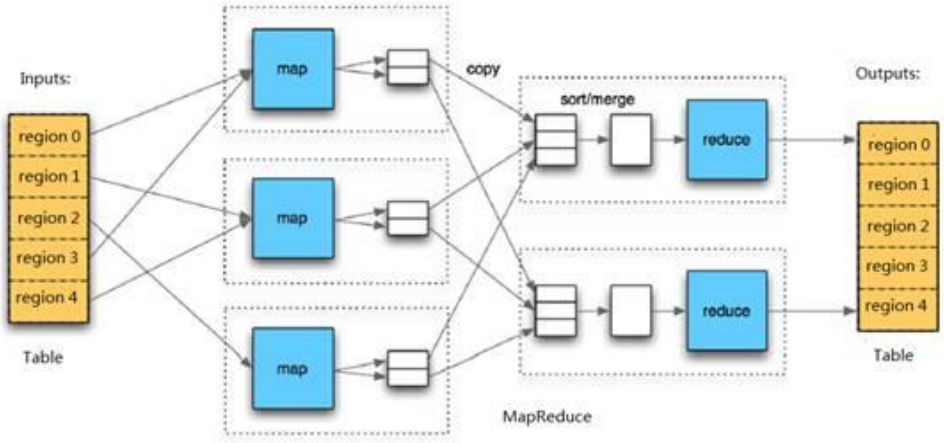

图 7-16 MapReduce 过程示意图

HBase Table 和 HRegion 的关系，比较类似 HDFS File 和 Block 的关系，HBase 提供了配套的 TableInputFormat 和 TableOutputFormat API，可以方便地将 HBase Table 作为 Hadoop MapReduce 的 Source 和 Sink，对于 MapReduce Job 应用开发人员来说，基本不需要关注 HBase 系统自身的细节。

本章小结

本章介绍了 HBase 的相关知识、使用场景和成功案例，并介绍了 HBase 和传统关系数据库的对比分析；接下来介绍了 HBase 访问接口、数据模型、实现方法、系统架构、存储格式、读写数据等；最后，简单介绍了 MapReduce 是在 HBase 系统上运行批处理运算的最方便和实用的模型。

参考文献

[1] HBase. 百度百科.

[2] HBase 分析报告. 百度文库. http://wenku.baidu.com/view/cf4b96c4bb4cf7ec4afed07d.html

附录 1:任课教师介绍

林子雨(1978－),男,博士,厦门大学计算机科学系助理教授,主要研究领域为数据库,数据仓库,数据挖掘.

主讲课程：《大数据技术基础》

办公地点：厦门大学海韵园科研 2 号楼

E-mail: ziyulin@xmu.edu.cn

个人网页：http://www.cs.xmu.edu.cn/linziyu