北京交通大学
BEIJING JIAOTONG UNIVERSITY




《大数据概论》
大数据存储与管理

鲍鹏
软件学院

本章内容

- 分布式文件系统
- 分布式数据库
- 非关系型数据库

大数据存储

- 结构化数据
  - 利用传统的关系型数据库表示和存储。
  - 二维数据：以行为单位，每一行数据属性相同。
- 半结构化数据
  - 非关系模型，有基本固定的结构模式。
  - 自描述数据：XML、JSON、Email。
- 非结构化数据
  - 数据结构不规则或不完整，没有预定义的数据模式。
  - 多样性数据：WORD、PPT、文本、图片、音频和视频。

非关系型数据库

- 关系型数据库存在的问题：
  - 无法有效处理大量的半结构化和非结构化数据
  - 无法保证数据并发性、可扩展性和可用性
  - 完善的事务机制和高效的查询机制成为负担
- 非关系型数据库
  - 特点：分布式、非关系型、不保证遵循ACID原则
  - 典型代表：HBase、MongoDB、Cassandra、Redis。

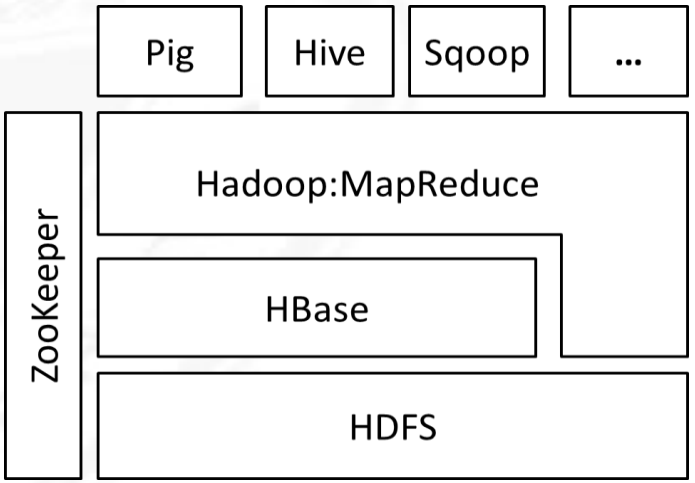
HBase

- HBase(Hadoop Database)，是一个能提供高可靠性、高性能、面向列、可伸缩、实时读写的分布式数据库，主要用来存储半结构化和非结构化的数据。

- HBase是Hadoop的子项目，建立在HDFS之上，原型为Google的分布式存储系统BigTable。HBase目标为依靠水平扩展的方式，不断增加廉价的商用服务器集群实现处理非常庞大的数据表，增加计算和存储能力。

HBase

- HBase与其他构件的关系：
  - HBase： 利用Hadoop MapReduce实现海量数据计算
  - ZoopKeeper： 协同服务，实现稳定服务和失败恢复
  - HDFS： 作为高可靠的底层存储，利用廉价集群提供海量数据存储能力
  - Sqoop ： 为HBase提供了高效、便捷的关系数据导入功能
  - Pig 和 Hive： 为HBase提供了高层SQL语言支持
  - …



Hadoop生态系统

HBase数据模型

- HBase本质上是一个稀疏、多维度、排序的数据映射表。
  - 表的索引是行键、列族、列限定符和时间戳，HBase中的数据均为字符串，没有特定类型。
  - 用户在表格中存储数据，每一行都有一个可排序的行键和任意多数量的列。由于是稀疏存储，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>:<qualifier>”，均由字符串组成，每一张表有一个列族集合，该集合固定不变，只能通过改变表结构来改变，qulifier值相对于每一行可以改变。

HBase数据模型

- 表 (Table)
  - 与关系数据模型类似，HBase也采用表来组织数据。
  - 表由行和列组成，列划分为若干个列族。
  - 由于HBase的表需映射成HDFS上面的文件，表名必须为能用在文件路径里的合法名字。

HBase数据模型

- 行 (Row)
  - 每个HBase表由若干行组成，每个行由行键来标识。
  - 行键是HBase表的主键，数据按照行键的字典序存储。
  - 行键可以为任意字符串，内部保存为字节数组，最大长度为64KB，实际应用中长度一般为10~100B。
  - 设计行键的原则：充分考虑行键的字典序存储特性，使经常一起读取的行的行键尽量集中，并在物理上能够临近存储。

HBase数据模型

- 列族(Column Family)
  - 一个HBase表在水平方向由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。
  - 列族需要在表创建时就定义好，数量不能太多，且不应频繁修改。
  - 存储在一个列族中的所有数据通常属于同一种数据类型，courses：history和courses：math两个列都属于courses列族。
  - HBase中，访问控制、磁盘和内存的使用统计都是在列族层面上进行的。

HBase数据模型

- 列限定符(Column Qualifier)
  - 列族里的数据通过列限定符（或列）来定位。
  - HBase支持动态扩展，无需预先定义列的数量以及类型，也无需在不同行之间保持一致。
  - 列限定符没有数据类型，使用字节数组存储。

HBase数据模型

- 单元格(Cell)
  - 在HBase表中，通过行、列族和列限定符即可定位一个单元格。
  - 单元格中存储的数据没有特定的数据类型，使用字节数组存储。
  - 每个单元格中可保存一个数据的多个版本，每个版本对应了一个时间戳。

HBase数据模型

- 时间戳(Timestamp)
  - 每次对一个单元格执行操作时，会隐式地自动生成并存储一个时间戳。
  - 每个单元格中数据的不同版本可利用时间戳进行索引。
  - 时间戳一般为64位整型，可由用户赋值，也可由HBase在数据写入时自动赋值。
  - 数据的不同版本按照时间戳降序进行存储，无指定时间戳访问时，最新的版本会被最先读取和返回。

HBase数据模型

- HBase物理模型
  - 关系数据库中，数据定位可理解为根据行和列确定关系表中一个具体的值。
  - HBase则需要根据行键、列族、列限定符和时间戳来确定一个单元格的数据值。
  - 将坐标 [行键，列族，列限定符，时间戳] 看作一个整体的“键”，单元格中的数据视为“值”，则HBase可被视为一个键值数据库。

HBase概念视图

Row Key	Time Stamp	Column Family:c1		Column Family:c2	
		列	值	列	值
r1	t7	c1:1	value1-1/1		
	t6	c1:2	value1-1/2		
	t5	c1:3	value1-1/3		
	t4			c2:1	value1-2/1
	t3			c2:2	value1-2/2
r2	t2	c1:1	value2-1/1		
	t1			c2:1	value2-1/1

- 从上表可以看出，test表有r1和r2两行数据，c1和c2两个列族，在r1中，列族c1有三条数据，列族c2有两条数据；在r2中，列族c1有一条数据，列族c2有一条数据，每一条数据对应的时间戳都用数字来表示。

HBase物理视图

- 从概念层面上看，HBase每个表格中存在很多“空”的单元格。
- 但从物理存储层面上看，HBase是采用了基于列的存储方式，这些空的单元格根本就不存储，当请求这些空白的单元格的时候，直接返回null 值。

Row Key	Time Stamp	Column Family c1	
		列	值
r1	t7	c1:1	value1-1/1
	t6	c1:2	value1-1/2
	t5	c1:3	value1-1/3

Row Key	Time Stamp	Column Family c2	
		列	值
r1	t4	c2:1	value1-2/1
	t3	c2:2	value1-2/2

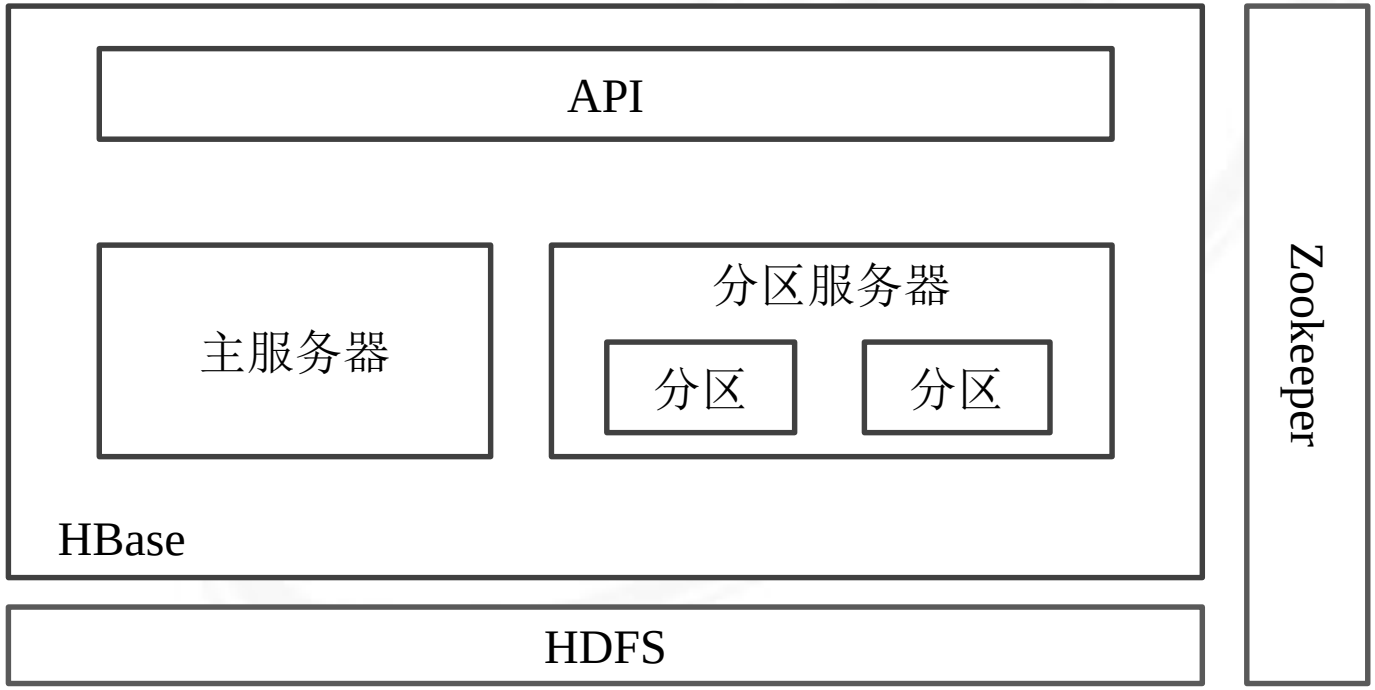
HBase实现模块

- HBase的实现包括三个主要的功能模块：
  - 链接到每个客户端的库函数
  - 一个主（Master）服务器
  - 多个分区（Region）服务器

- 分区服务器
  - 负责存储和维护分配给自己的分区，处理来自客户端的读写请求。
- 主服务器
  - 负责管理和维护HBase表的分区信息，监测集群中的分区服务器，确保负载均衡，处理表和列族的创建等。

HBase实现模块

- 客户端并不是直接从主服务器上读取数据，而是在获得分区的存储位置信息后，直接从分区服务器上读取数据。
- HBase客户端并不依赖于主服务器，而是借助于Zookeeper来获得分区的位置信息。因此，大多数客户端从不和主服务器通信，减小了主服务器的负载。



HBase的实现模块

HBase实现模块

- 表和分区
  - 根据行键对表中的行进行分区，每个行区间构成一个分区，这些分区会被分发存储到不同的分区服务器上。
  - 位于某个行键取值区间内的所有数据，其默认大小是100MB到200MB，构成了负载均衡和数据分发的基本单位。
  - 初始时，每个表只包含一个分区，随着数据的不断插入，分区会不断增大，当一个分区中包含的行数量达到一个阈值时，就会被自动等分成两个新的分区。
  - 主服务器会把不同的分区分配到不同的分区服务器上，但是同一个分区不会被拆分到多个分区服务器上。每个分区服务器负责管理一个分区集合，通常是10~1000个分区。

HBase实现模块

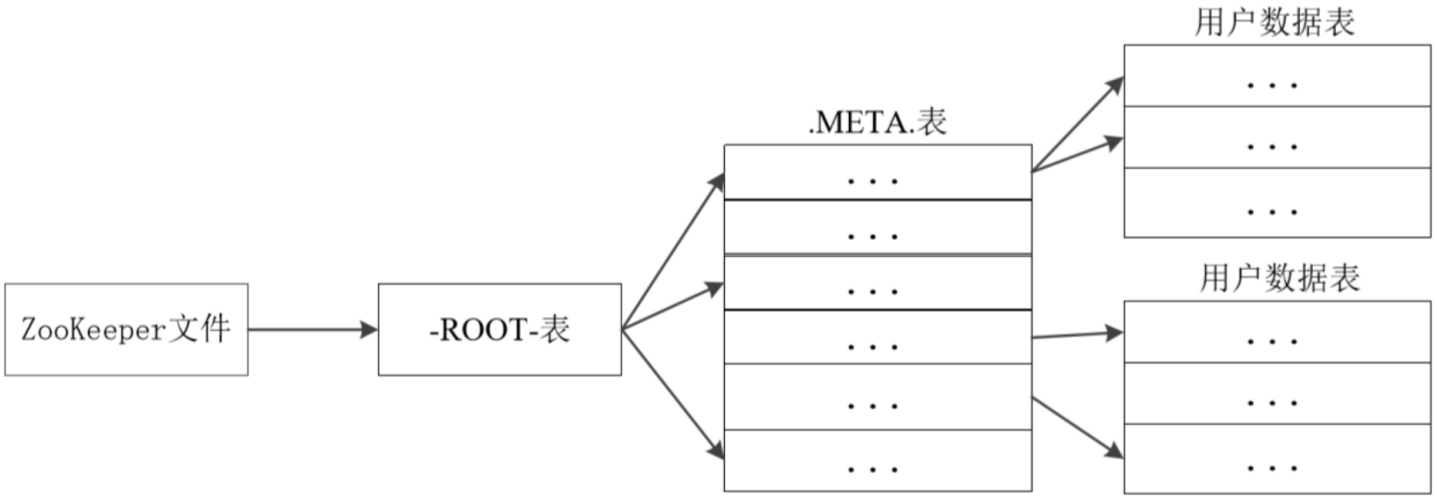
- 分区定位机制
  - 每个分区都有一个RegionId作为唯一标识。
  - 一个分区标识符可表示为“表名+开始主键+RegionId”。
  - 为定位每个分区的位置，需要构建一张位置映射表，表示分区和分区服务器之间的对应关系。

HBase实现模块

- 分区定位机制
  - 元数据表(.META.表)，每行包含两项：分区标识符和分区服务器标识符，当条目过多时分裂成多个分区。
  - 根数据表(-ROOT-表)，作为.META.表的分区映射表，记录所有元数据的具体位置，-ROOT-表是不能被再分割的，存在-ROOT-表的唯一分区在程序中被写死。
  - ZooKeeper 中记录 -ROOT- 表的存储位置。

HBase实现模块

- 分区定位机制
  - HBase使用类似B+树索引的三层结构来保存分区位置信息。



HBase三层结构

HBase实现模块

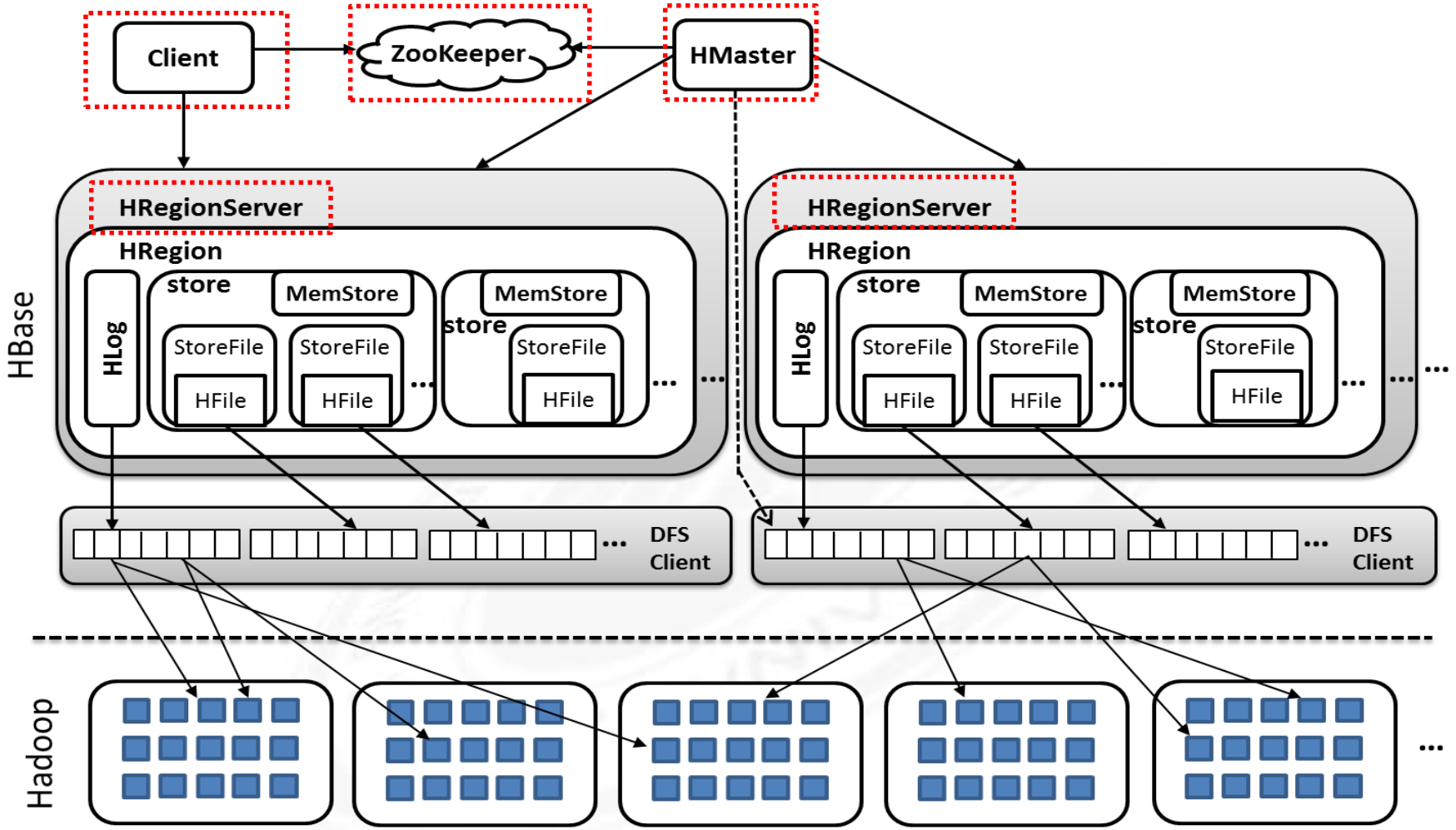
- “三级寻址”过程
  1. 客户端访问用户数据之前，需要先访问Zookeeper获取-ROOT-表的位置信息。
  2. 通过访问-ROOT-表获取所请求行所在范围所属的.META.表的位置。
  3. 访问.META.表，获取所需的数据分区的具体位置。
  4. 客户端直接与管理该分区的分区服务器进行交互。

- 缓存机制
  - 在客户端缓存查询过的分区的位置信息。
  - 当缓存失效时，重新发起“三级寻址”过程，并更新缓存。

HBase体系结构

- HBase的体系结构包含四个功能组件:
  - 客户端（Client）
  - ZooKeeper服务器
  - 主服务器（HMaster）
  - 分区服务器（HRegionServer）

HBase体系结构



HBase体系结构

HBase体系结构

- 客户端（Client）
  - 客户端是HBase功能的使用者，包含访问HBase的接口，同时在缓存中维护着已经访问过的分区位置信息。
  - 客户端使用HBase的RPC机制与主服务器和分区服务器进行通信。
  - 对于管理类操作，客户端与主服务器进行RPC。
  - 对于数据读写类操作，客户端与分区服务器进行RPC。

HBase体系结构

- ZooKeeper服务器
  - Zookeeper服务器并非一台单一的机器，可能是由多台机器构成的集群来提供稳定可靠的协同服务。
  - 每个分区服务器都需要到Zookeeper中注册，Zookeeper实时监控每个分区服务器的状态并通知给主服务器。
  - HBase中可以启动多个主服务器，Zookeeper可以帮助选举出一个主服务器作为集群的总管，并保证在任何时刻总有唯一一个主服务器在运行，从而避免Master的“单点失效”问题。
  - ZooKeeper存储-ROOT-表的地址、HMaster的地址、HBase表、列族等信息。

HBase体系结构

- 主服务器（HMaster）
  - 每台HRegion服务器均会和HMaster服务器通信，HMaster的主要任务是通知每台HRegion服务器它要维护哪些HRegion。
  - HMaster在功能上主要负责表和HRegion的管理工作。

（1）管理用户对表的增加、删除、修改、查询等操作。

（2）实现不同分区服务器之间的负载均衡。

（3）在分区分裂或合并后负责重新调整分区的分布。

（4）对发生故障失效的分区服务器上的分区进行迁移。

HBase体系结构

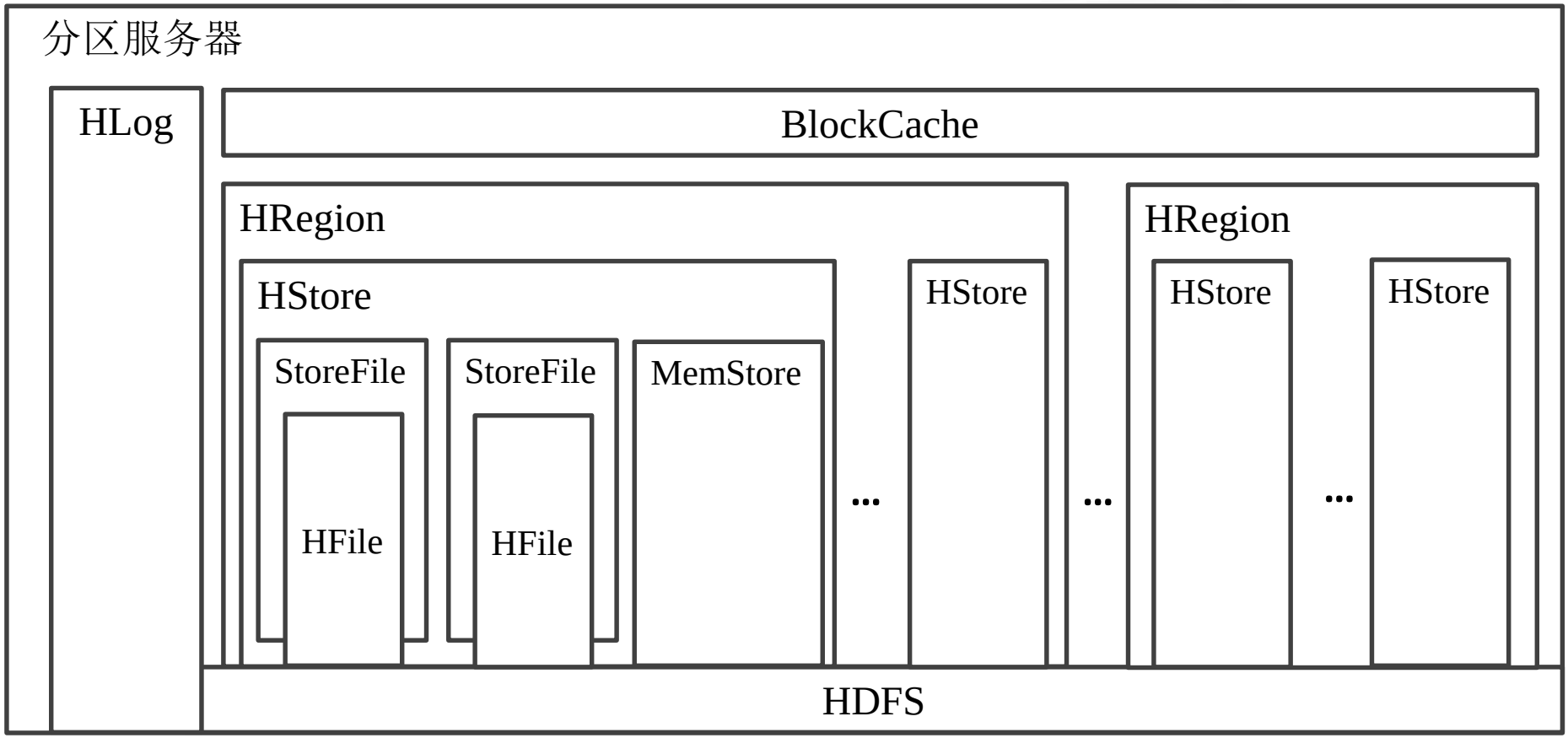
- 分区服务器（HRegionServer）
  - 分区服务器是HBase中最核心的模块，负责维护位于本机的所有分区，并响应用户的读写请求。
  - HBase通常采用HDFS作为底层存储文件系统，来保证可靠稳定的数据存储，因此，分区服务器需要向HDFS读写数据。
  - 用户通过一系列HRegion服务器来获取这些数据，一台机器上一般只运行一个HRegion服务器，且每一个区段的HRegion也只会被一个HRegion服务器维护。

HBase体系结构

- 分区服务器（HRegionServer）
  - HRegionServer 内部管理了一系列分区对象HRegion 和一个HLog文件，HLog是磁盘上面的记录文件，它记录着所有的更新操作。
  - 每个 HRegion 由多个 HStore 组成，每个 HStore 对应表中的一个列族的存储。
  - HStore是分区服务器的核心，每个HStore包含一个MemStore和若干个StoreFile。MemStore是在内存中的缓存，保存最近更新的数据，StoreFile是磁盘中的文件。
  - StoreFile在底层的实现方式是HDFS文件系统的HFile，HFile 的数据块通常采用压缩方式存储，以减少网络和磁盘I/O。

HBase体系结构

- 分区服务器（HRegionServer）



分区服务器工作原理

HBase体系结构

- HRegionServer工作原理——读写数据过程

（1）“写”数据

– 当客户端写入数据时，被分配到相应的分区服务器去执行操作。
– 数据首先被写入到MemStore和HLog中。
– 当操作写入HLog之后，commit()调用才会将其返回给客户端。

（2）“读”数据

– 当客户端读取数据时，分区服务器会首先访问MemStore缓存。
– 若数据不在缓存中，则去磁盘上的StoreFile中去寻找。

HBase体系结构

- HRegionServer工作原理——缓存的刷新
  - MemStore缓存的容量有限，系统会周期性地调用Region.flushcache() 把MemStore缓存里面的内容写到磁盘的StoreFile 文件中，清空缓存，并在HLog文件中写入一个标记，表示缓存中的内容已被写入StoreFile文件。
  - 每次缓存刷新操作，都会在磁盘上生成一个新的StoreFile文件，因此，每个Store包含多个 StoreFile文件。

HBase体系结构

- HRegionServer工作原理——缓存的刷新
  - 每个分区服务器在启动的时候会检查自己的HLog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。
  - 若没有更新，表明所有数据已经被永久保存到磁盘的StoreFile文件中。
  - 若发现更新，先将这些更新写入MemStore，然后再刷新缓存并写入到磁盘的StoreFile 文件中，最后删除旧的HLog文件，并开始提供数据访问服务。

HBase体系结构

- HRegionServer工作原理——StoreFile合并与分区
  - 当访问某个Store中的某个值时，必须查找多个StoreFile，非常耗时，影响查找速度。
  - 系统一般会调用Store.compact()将多个StoreFile合并成一个大文件，但由于合并操作比较耗资源，系统只会在StoreFile文件的数量达到一个阈值时才会触发该操作。
  - 当多个StoreFile合并后，会逐步形成越来越大的StoreFile文件，当单个StoreFile文件大小超过一定阈值时，就会触发文件分裂操作。当前的分区会分裂成两个子分区然后下线，新分裂出的两个子分区被主服务器分配到相应的分区服务器上。

HBase体系结构

- HRegionServer工作原理——故障处理与数据恢复
  - HLog文件是一种预写式日志（Write Ahead Log），更新数据必须首先被记入日志后才能写入MemStore缓存，直到MemStore缓存内容对应的日志已经被写入磁盘之后，该缓存内容才会被刷新写入磁盘。
  - Zookeeper 会实时监测每个分区服务器的状态，当某个分区服务器发生故障时，Zookeeper会通知主服务器。
  - 主服务器首先会处理该故障分区服务器上遗留的HLog文件。由于一个Region服务器上面可能会维护着多个分区对象，它们共用一个HLog文件，因此该遗留的HLog文件中包含了来自多个分区对象的日志。

HBase体系结构

- HRegionServer工作原理——故障处理与数据恢复
  - 主服务器根据每条日志记录所属的分区对象对HLog数据进行拆分，拆分结果分别放到相应分区对象的目录下。
  - 将失效的分区重新分配到可用的分区服务器中，并将与该分区对象相关的部分HLog日志发送给新的分区服务器。
  - 分区服务器领取到分配给自己的分区对象及日志后，重做日志中的所有操作，并将日志中的数据写入MemStore缓存，然后刷新到磁盘的StoreFile文件中，完成分区数据恢复。

HBase体系结构

- HRegionServer工作原理——HLog文件设计原则
  - HBase中设计的每个分区服务器只需要维护一个多分区对象共享的HLog文件。
  - 优点：多个分区对象的更新操作所发生的日志修改，只需不断把日志记录追加到单个日志文件中，无需同时打开、写入到多个日志文件，可减少磁盘寻址次数，提高对表的写操作性能。
  - 缺点：若分区服务器发生故障，为实现数据恢复，需要将分区服务器上的HLog按照分区对象进行拆分后再分发到新的分区服务器上。

分布式数据仓库Hive

- Hive是一款基于Hadoop的数据仓库工具，可以用于对HDFS中的数据集进行数据整理、查询和分析。
- Hive 定义了简单的类SQL 查询语言，称为Hive QL。Hive QL允许熟悉 SQL的用户查询数据。同时，Hive 也允许熟悉 MapReduce 的开发者自定义Mapper和Reducer操作，从而支持MapReduce框架。

分布式数据仓库Hive

- Hive的设计特点如下：
  - 支持不同的存储类型，如纯文本文件、HBase中的文件。
  - 可将元数据保存在关系数据库中，减少了在查询过程中执行语义检查的时间。
  - 可直接使用存储在Hadoop文件系统中的数据。
  - 内置大量用户函数UDF来操作时间、字符串和其他的数据挖掘工具，支持用户扩展UDF函数来完成内置函数无法实现的操作。
  - 采用类SQL的查询方式，可将SQL查询转换为MapReduce的job在Hadoop集群上执行。

Hive与HBase对比

特性	HBase	Hive
适用查询	大数据的实时查询	一段时间内数据的分析查询
部署方式	需要Zookeeper辅助	只需Hadoop MapReduce 即可工作
SQL查询	不支持	支持
数据操作	搭配使用Apache Phonenix 使用SQL	类SQL语言Hive QL
存储层	默认HDFS，可选本地 文件系统等解决方案	默认HDFS
应用场景	Facebook用HBase进行 消息的实时分析	计算未来趋势或者网站的日志