

第四章 分布式数据库HBase

第四章考点

- HBase出现的必要性
- HBase与传统关系数据库的区别（数据、数据操作、存储模式、数据索引、数据维护，可伸缩性）
- HBase数据模型
- 数据模型相关概念：表、列族，列限定符，时间戳
- 数据坐标
- 行存储、列存储
- HBase组件
- Region的定位：三级寻址
- 系统架构
- Region服务器工作原理
- Store工作原理
- HLog工作原理
- HBase代码实践

一、HBase概述及访问接口

1. BigTable

BigTable是一个分布式文件存储系统，HBase是BigTable的开源实践，优点：可拓展性好，支持大规模数据

BigTable工作流程：

- 建立互联网索引
 - 通过爬虫持续不断地抓取新页面，将这些页面单独存储到BigTable里
 - MapReduce计算作业运行在整张表上，生成相关索引，为网络搜索应用阶段做准备
- 搜索互联网
 - 用户发起网络搜索请求
 - 网络搜索应用查询建立好的索引，从BigTable得到网页
 - 搜索结果交给用户

2. HBase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，利用廉价计算集群处理大量数据组成的数据表

HBase架构在底层HDFS上；

MapReduce可以对HBase中的数据进行处理

Pig和Hive都可以直接访问HBase中的数据，并且提供相应的支持

3. HBase和BigTable的底层技术之间的对应关系

	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

4. 关系数据库已经流行很多年，并且Hadoop已经有了HDFS和MapReduce但是为什么需要HBase (HBase必要性)

- HDFS主要解决大规模数据的离线批量处理问题，受限于Hadoop MapReduce编程框架的高延迟数据处理机制，使得HDFS**无法满足大规模数据实时处理应用的需求**
- 传统的通用**关系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**。 **HBase对于分表的水平切分操作是全自动实现的**，所以具有非常好的水平可扩展性
- 大数据时代数据的结构经常会发生变化，传统关系数据库在数据结构变化的时候一般需要停机维护，变更相应的模式，然后再上线
- 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、文档数据库和列组数据库

5. HBase与传统数据库的区别主要体现在以下六个方面

- **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(结构化/非结构化)存储为未经解释的字符串**，用户需要自己编写程序把字符串解析成不同的数据类型

- **数据操作：**

关系数据库包含了丰富的操作，其中会涉及插入、查询、更新等，其中会涉及复杂的多表连接(一对一、一对多、多对多)

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，通常只采用单表主键查询

- **存储模式：**

关系数据库是基于行模式存储的，元组或者行会连续地存储在磁盘页中。在读取数据的时候，需要顺序扫描每个元组，从中筛选出所需要的属性，如果每个元组只有少量的属性的值对于查询是有用的，那么基于行模式的存储就会浪费许多磁盘空间和内存带宽

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**。这样做的优点是可以降低I/O开销，**支持大量并发用户查询**，因为仅需要处理可以回答这写查询的列，而不需要处理与查询无关的大量数据行

- **数据索引：**

关系数据库可以针对不同的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。由于HBase位于Hadoop的框架之上，因此可以使用Hadoop MapReduce来快速高效地生成索引表

- **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中执行更新操作的时候，**并不会删除数据旧的版本，而是生成一个新的版本，旧的版本仍然保留**

- **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

6. Hbase访问接口的类型，特点和使用场景

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
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

二、HBase数据模型

1. HBase数据模型

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、行限定符和时间戳**（四维坐标）。



- 每个值都是**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。
- 由于同一张表里面的每一行数据都可以有截然不同的列，因此对于整个映射表的每行数据而言，有些列的值是空的，所以说HBase是稀疏的。
- 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）

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，列划分为若干列族
- 行：每个HBase表都由若干行组成，每个行由行键来标识。访问行的方式：通过单个行键访问，通过一个行键的区间来访问，全表扫描，设计行键的时候应该将经常一起读取的行存储在一起
- 列族：一个HBase表被分组成需要列族的集合，它是基本的访问控制单元，列族在表创建的时候就需要定义好，数量不能太多
- 列限定符：列族里的数据通过列限定符来定位
- 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[]。每个单元格中可以保存一个数据的多个版本，每个版本对应一个不同的时间戳。
- 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，并根据时间戳降序存储，时间戳较大的版本的数据是最新数据

3. 数据坐标

在HBase中数据定位可以理解为采用四维坐标[行键，列族，列限定符，时间戳]，HBase可以被视为一个键值数据库，四维坐标为键，存储的数据为值

4. 概念视图

在概念视图中，一个表可以视为一个稀疏、多维的映射关系

其中行键采用反向url存储这样可以将来自同一网站的数据划分到同一分区

HBase中每一行不需要在每个列族中存储数据，所以说HBase是一个稀疏的映射关系，即里面存在许多空的单元格

5. 物理视图

物理存储层面HBase的表是基于列的存储方式，并不是采用基于行的存储方式，在物理存储时候不同列族分开存放

行存储和列存储：

对于面向行存储的时候，当从磁盘中读取数据时，需要从磁盘顺序扫描每个元组的完整内容，然后从每个元组中筛选出查询所需要的属性。如果每个元组只有少量属性的值对于查询是有用的，那么就会浪费许多磁盘空间和内存带宽

列存储模型目的是最小化无用I/O，多个元组的同一属性值（同一列值）会被存储在一起，而一个元组中不同属性值通常会被分别存储在不同磁盘页中

面向列存储的优点：

- 适合批量数据处理和即席查询
- 可以降低I/O开销，支持大量并发用户查询

- 具有较高的数据压缩比
-

三、HBase实现原理

1. HBase的实现主要包括三个重要功能组件：

- 库函数：链接到每个客户端
- 一个**Master主服务器**
- 许多个**Region服务器**

2. 表和Region

Region概念

- 在HBase中，存出来许多表，对于每个表而言，表中的行数量可能特别多，无法存储在一台机器上，需要**分布存储到多台机器上**，因此需要根据行键的值对应的行进行**分区**，**每个行区间构成一个分区“Region”**。Region中包含了位于某个值域区间的所有数据，是**负载均衡和数据分发的基本单位**。这些Region会被分发到不同的Region服务器

- 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服务器的最佳大小建议为5GB到10GB
- 同一个Region不会被拆分到多个Region服务器
- 每个Region服务器存储10到1000个Region
- 开始每个表只有一个Region，随着数据的不断插入，Region会持续增大。当一个Region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Region。**随着行的数量继续增加，Region不断分裂**

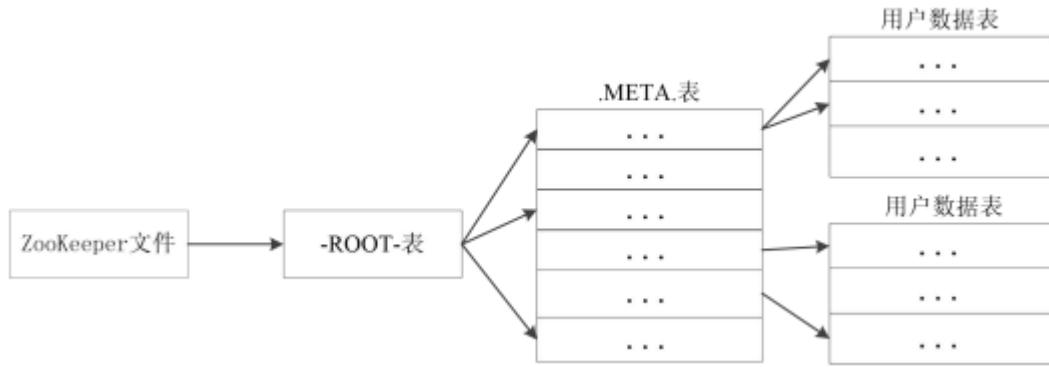
主服务器的作用

- 主服务器Master负责**管理和维护HBase表的分区信息**，比如一个表被分成了哪些Region，每个Region分别被存放在哪台服务器上，同时也**负责维护Region服务器列表**，因此如果Master主服务器死机，那么整个系统都会无效
- Master会**实时检测集群中的Region服务器**，把特定的Region分配到可用的Region服务器上，**确保这个集群内部不同Region服务器的负载均衡**
- 当某个Region服务器因为出现故障而失效的时候，Master会把该故障服务器上存储的Region重新分配给其他可用的Region服务器
- 除此之外Master还处理模式变化，例如表和列族的创建
- 客户端读取Region数据并不从主服务器读取
- Region服务器负责存储和维护自己的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**
- HBase客户端并不依赖Master而是**通过Zookeeper来获得Region位置信息**，所以大多数客户端甚至从来不和Master主服务器通信，通过这种设计减少Master的负载

3. Region的定位：

每个Region都有一个RegionID来标识它的唯一性，一个RegionID可以表示为：**表名+起始**

行键+RegionID



- **".META.表"：**

有了RegionID，就可以唯一地标识每个Region。为了定位每个Region所在的位置，可以构建一张映射表。映射表的每个条目（或每行）包含两项内容，一个是**RegionID**，另一个是**Region服务器标识**。这个条目表示Region和Region服务器之间的对应关系，从而可以知道某个Region被保存在哪个Region服务器中。这个映射表包含了关于Region的元数据（即Region和Region服务器之间的对应关系），因此也被称为“元数据表”，又名“**.META. 表**”。

- **-ROOT-表：**

- 当HBase表中的Region数量非常庞大的时候，“.META.表”的条目就会非常多，一个服务器保存不下，也需要分区存储到不同的服务器上。因此.META.表也会被分裂成多个Region。这时，为了定位这些Region，就需要构建一个新的映射表，记录所有元数据的具体位置，这个新的映射表就是“根数据表”，又名“**-ROOT-表**”。
- -ROOT-表是不能被分割的，永远只存在一个Region用于存放-ROOT-表。因此这个用来存放-ROOT-表的唯一一个Region，它的名字是在程序中被“写死”的，Master主服务器永远知道它的位置。

- **Zookeeper**文件记录了-ROOT-表的位置

为了加快访问速度，**.META.表的全部Region都会被保存在内存中**

4. 客户端访问数据时候“三级寻址”过程：

客户端访问用户数据之前首先需要访问Zookeeper，获取-ROOT表的位置信息，然后访问-ROOT表，获得.META表的信息，接着访问-META表，找到所需要的Region具体位于哪个服务器，最后才能到这个Region服务器中读取数据

- 该过程需要多次网络操作，为了加速寻址，客户端会将查询过的位置缓存起来，以后访问相同数据的时候就会直接从服务器缓存中获得Region的位置信息，并不会经历三级寻址过程
- 如果随着HBase中表的不断更新，缓存中的Region位置信息已经失效，访问数据的时候从缓存中获取Region位置发现不存在，判断缓存失效，随后则是要重新经过三级寻址过程来寻找最新的Region位置信息
- 客户端通过Zookeeper服务器来进行三级寻址来获取用户数据表所在Region并不需要连接Master服务器，这样就减少了Master的负担

四、HBase运行机制

1. 客户端

客户端汇总包含访问HBase的接口，同时缓存中也维护已经访问过的Region的位置信息，用来加快后续数据访问过程。HBase客户端使用HBase的RPC（远程过程调用）机制与Master和Region服务器进行通信，对于管理类操作则是客户端与Master进行RPC对于数据读写操作，客户端会与Region服务器进行RPC

2. Zookeeper服务器

- Zookeeper服务器并非一台单一的服务器，可能是由多个服务器构成的集群来提供稳定可靠的服务器进群，那么必须有一个“总管”知道当前集群每台机器的服务状态，一旦有某台机器不能提供服务，集群中其他机器必须知道，从而做出调整、重新分配服务策略
- 每个Region服务器都需要到Zookeeper服务器进行注册，Zookeeper服务器会实施检测每个Region服务器的状态，并且通知Master服务器，这样Master服务器就可以随时知道各个服务器的工作状态
- HBase可以启动多个Master但是Zookeeper可以帮助选举出一个Master主服务器作为集群的总管，并且保证每一时刻都有唯一一个Master在运行，避免单点失效的问题

3. Master

主服务器Master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分裂或者合并的时候调整Region的分布
- 对于发生故障失效的Region服务器上的Region进行转移

4. Region服务器

HBase一般采用HDFS作为底层存储的文件系统，因此Region服务器需要向HDFS读写数据，HBase本身并不具备数据复制和维护数据副本的功能，但是HDFS可以为HBase提供这些支持

Region服务器的工作原理

• 用户读写数据过程：

用户写入数据时，会被分配到对应的Region服务器去执行操作

用户数据首先写入MemStore和HLog中

只有当操作写入HLog之后，commit()调用才会将其返回到客户端

当用户读取数据的时候，Region服务器会首先访问MemStore缓存中，如果找不到，再去磁盘上的StoreFile中寻找

• 缓存的刷新

MemStore缓存的容量有限，系统会周期性将MemStore缓存里的内容写入磁盘的StoreFile文件，清空缓存，并且在HLog中写入一个标记

每次缓存刷新都会生成一个新的StoreFile文件，每个Store包含多个StoreFile文件

每个Region服务器都有自己的HLog文件，每次启动Region服务器都会检查该文件，确认最近一次执行刷新操作之前是否发生了新的写入操作，如果发生了就写入

MemStore，刷新缓存写入StoreFile，最后删除旧的HLog文件，开始为用户提供服务

- **StoreFile的合并**

每次MemStore刷新都会生成一个新的StoreFile，这样每个Store就会存在多个StoreFile文件

当需要访问某个Store的某个值的时候，必须查找所有的StoreFile，影响查找速度
为了节省时间成本，系统一般会调用Store.compact()，将多个StoreFile合并为一个
一般来说当数量达到某个阈值才会合并

Region服务器是HBase的核心模块，而Store是Region服务器的核心，Store是真正存数据的地方

5. Store工作原理

- **合并：** MemStore是排序的缓存区，用户写入数据的时候，系统会把数据放在MemStore中，当MemStore满了的时候就会刷新到磁盘中的一个StoreFile中，随着StoreFile文件不断增加，达到预先设定的数量的时候，会触发文件合并，多个合并为一个大的StoreFile文件
- **分裂：** StoreFile文件会越来越大，当单个StoreFile文件大小超过一定的阈值的时候，就会触发文件分裂操作，与此同时，当前的一个父Region会被分裂为两个子Region，父Region会下线，新分裂出两个子Region被Master服务器分配到对称的Region服务器中

6. HLog工作原理

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），换言之，**用户更新数据必须首先被记入日志才能写入MemStore缓存**，并且直到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。

故障处理

- 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主服务器。
- 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。

使用HLog的优缺点：

- **优点：**减少磁盘寻址次数，提高对表的写操作性能。由于每个Region服务器只需要维护一个HLog文件，所有Region对象共用一个HLog，而不是每个对象使用一个HLog因此多个Region对象的更新操作所发生的数据修改只需要不断追加到单个HLog中就可
- **缺点：**如果一个Region服务器发生故障，为了恢复其上面的Region对象，需要将Region服务器上的HLog按照所属的Region对象进行拆分，然后分发到其他服务器上执行恢复操作

五、HBase应用

1. HBase实际应用中的性能优化方法

- 行键：行键是按照字典序存储，因此可以利用这个排序特点，将经常在一起读取的数据存储到一块，例如可以使用**Long.MAX_VALUE-timestep**作为行键，这样就能保证写入的新数据可以被快速读取
- 最大版本：创建表的时候可以设置保存表中数据的最大版本数，例如只保存最新的版本，`setMaxVersion(1)`。这样设置可以节省存储空间
- 存活时间：创建表的时候可以设置表中数据的生存周期，过期数据自动删除，例如存储最近两天的数据，那么可以设置`setTimeToLive(2 24 60 60)`

2. 性能监视

Master-status可以查看HBase集群当前状态

Ambari的作用就是创建管理、监视Hadoop的集群

3. 在HBase上构建SQL引擎

优势：

易使用：用SQL这种语言更加轻松地使用HBase

减少编码：减少编写代码量

方案：

Hive整合HBase

Phoenix

六、HBase实践

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

- `create` 创建表：

```
create 'student', 'Sname', 'Ssex', 'Sage', 'Sdept', 'course'
```

此时创建了一个student表，属性为后面的'Sname','Ssex','Sage','Sdept','course'，HBase的表中有一个系统默认属性作为行键，无需自行创建，默认为put命令的表名后第一个数据

- `describe` 查看表

```
describe 'student'
```

- `put` 添加数据

```
put 'student', '95001', 'Sname', 'LiYing'  
put 'student', '95001', 'course:math', '80'
```

向student表添加了一行数据，行键为95001，名字为LiYing，对course列族的math列添加了一个数据

- `delete` 删除一个数据

```
delete 'student', '95001', 'Ssex'
```

删除行键为95001的Ssex列所有数据

- `deleteall` 删除一行数据

```
deleteall 'student', '95001'
```

删除student表中的行键95001的所有数据

- `get` 查看某一行数据

```
get 'student', '95001'
```

返回行键为95001对应行的所有数据

- `scan` 查看某个表的所有数据

```
scan 'student'
```

- `删除表`，第一步先让该表不可用，第二步删除表

```
disable 'student'  
drop 'student'
```

- `查看历史数据`

```
create 'teacher', {NAME=>'username', VERSIONS=>5}
```

创建表的时候指定保存的版本数，指定为5

随后使用 `put` 多次插入更新，最新一版是最近一次操作

查询数据的时候指定历史版本数，默认会查询出最新的数据

```
get 'teacher', '91001', {COLUMN=>'username', VERSION=>5}
```

使用`exit`退出HBase shell