第5章 分布式数据库HBase

高琰

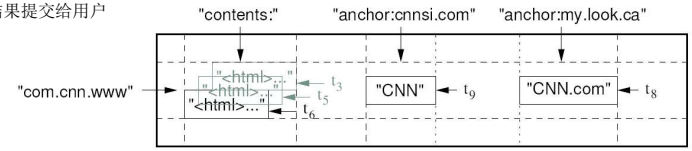
目录

从BigTable说起

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
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



网页在BigTable中的存储样例

从BigTable说起

- BigTable是一个分布式存储系统
- 利用谷歌提出的MapReduce分布式并行计算模型来处理海量数据
- 使用谷歌分布式文件系统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
- 谷歌的许多项目都存储在BigTable中，包括搜索、地图、财经、打印、社交网站Orkut、视频共享网站YouTube和博客网站Blogger等

APACHE HBASE

HBase简介

表5-1 HBase和BigTable的底层技术对应关系

	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

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

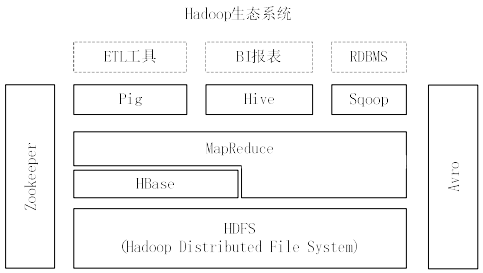


图4-1 Hadoop生态系统中HBase与其他部分的关系

HBase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（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，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）.目前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式数据分析处理系统中

HBase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

HBase访问接口

表5-2 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

HBase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：

- （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

02

HBASE数据模型

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）

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”

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

概念视图

表5-4 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>..."	

物理视图

表5-5 HBase数据的物理视图

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

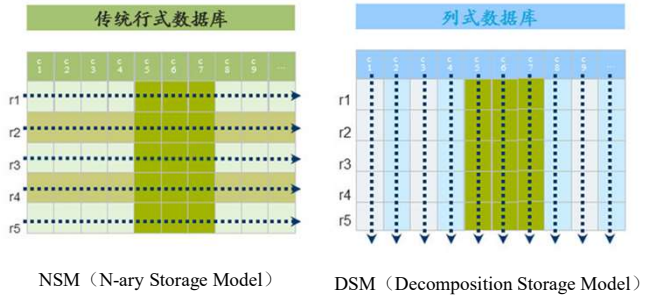
行键	时间戳	列族anchor
"com.cnn.www"	t5	anchor:cnnsi.com="CNN"
	t4	anchor:my.look.ca="CNN.com"

面向列的存储(1)

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

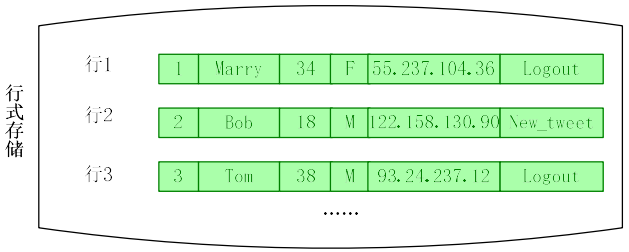
面向列的存储



NSM（N-ary Storage Model）　　DSM（Decomposition Storage Model）

图5-3 行式数据库和列式数据库示意图

面向列的存储(2)



面向列的存储(3)

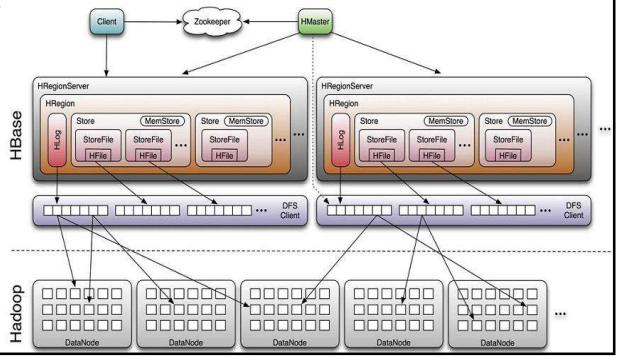
列式存储

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

HBase功能组件

•HBase的实现包括三个主要的功能组件：

- （1）库函数-链接到客户端
- （2）一个Master主服务器
- （3）许多个Region服务器



03

HBASE 的实现原理

HBase功能组件

- 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

Region

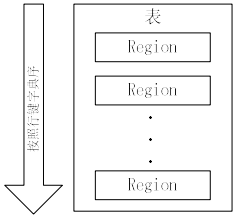


图4-5一个HBase表被划分成多个Region

- 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

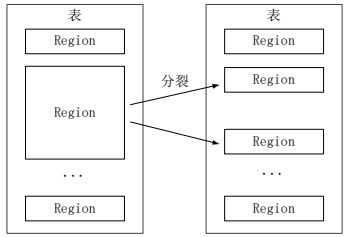


图4-6 一个Region会分裂成多个新的Region

Region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

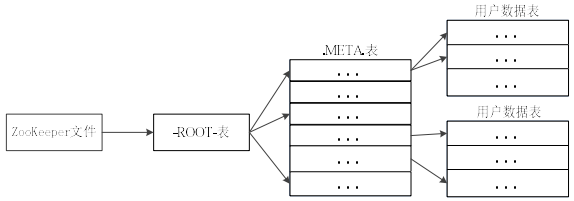


图4-8 HBase的三层结构

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

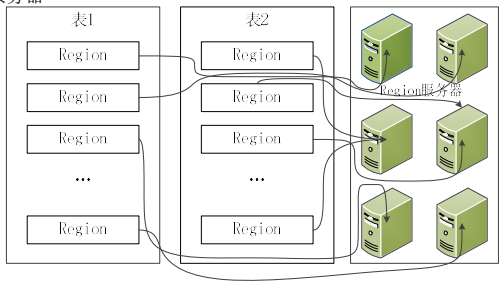


图5-7 不同的Region可以分布在不同的Region服务器上

Region的定位

表5-6 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

4.4.3 Region的定位

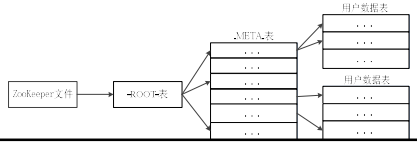
•假设.META.表的每行（一个映射条目）在内存中大约占用1KB，并且每个Region限制为128MB，那么，上面的三层结构可以保存的用户数据表的Region数目的计算方法是：

（-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



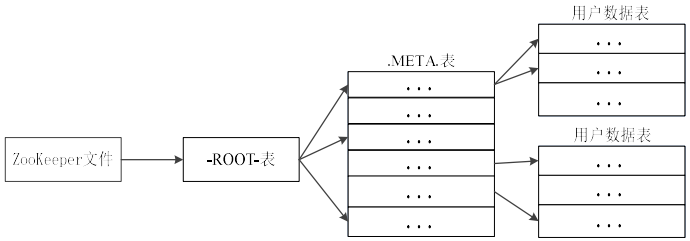
04 HBASE 运行机制

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



HBase系统架构

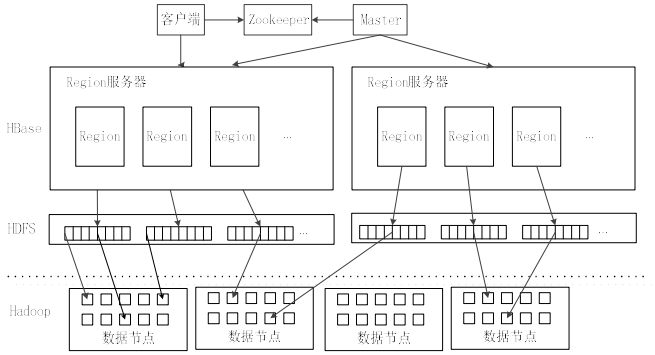
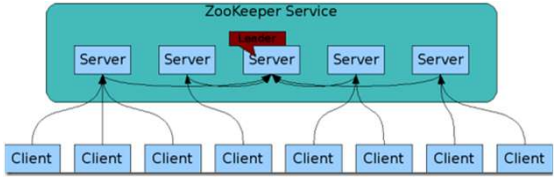


图4-9 HBase的系统架构

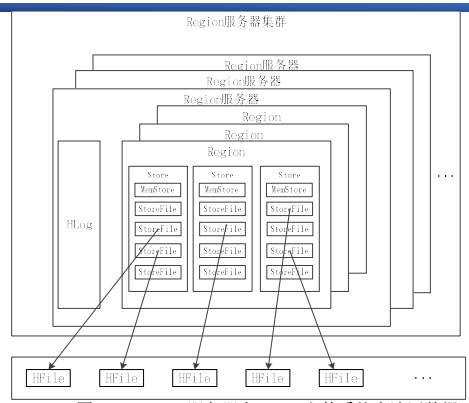
HBase系统架构

- 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。



Region服务器工作原理



1. 用户读写数据过程
2. 缓存的刷新
3. StoreFile的合并

图5-10 Region服务器向HDFS文件系统中读写数据

HBase系统架构

- 3. Master主服务器，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

Region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

Region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

Store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

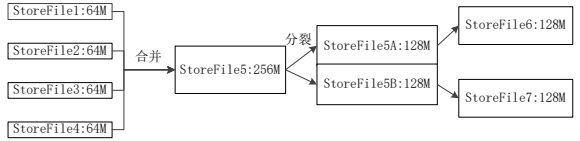


图4-11 StoreFile的合并和分裂过程

Region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

HLog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

HLog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

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。

举个例子：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。

05

HBASE 应用方案

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，可以通过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)。

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，可以通过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)。

HBase性能监视

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

HBase实际应用中的性能优化方法

- **写表操作中的HTable参数设置**
- **Auto Flush**

  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，可以通过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**

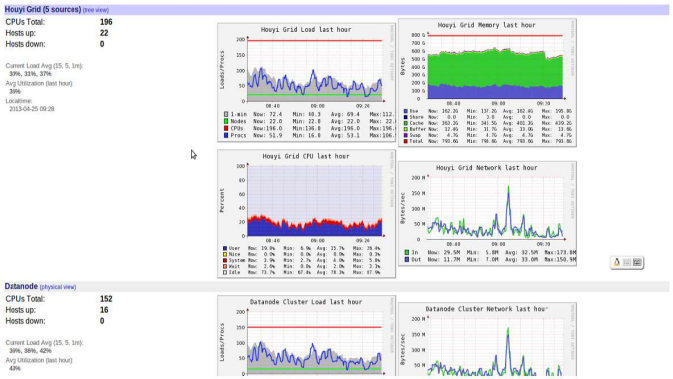
  创建表的时候，可以通过HColumnDescriptor.setTimeToLive(int timeToLive)设置表中数据的存储生命期，过期数据将自动被删除，例如如果只需要存储最近两天的数据，那么可以设置setTimeToLive(2 * 24 * 60 * 60)。

Master-status

- 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

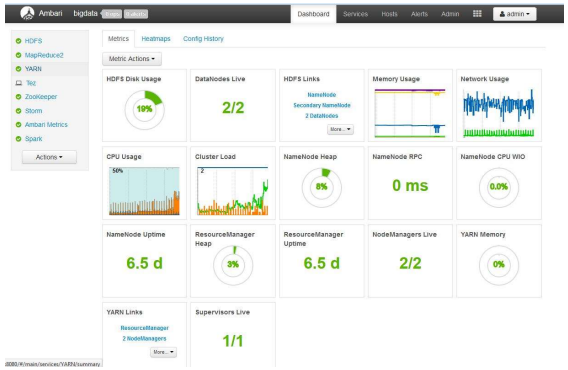
Ganglia

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



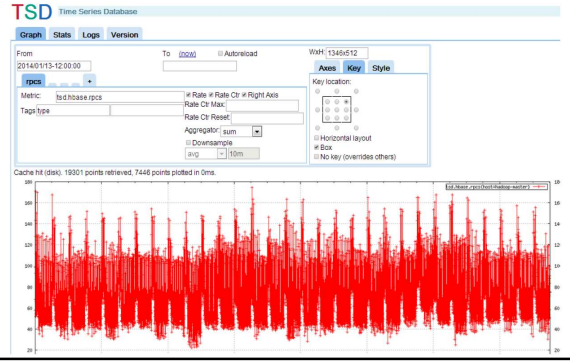
Ambari

Ambari 的作用就是创建、管理、监视 Hadoop 的集群



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



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

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

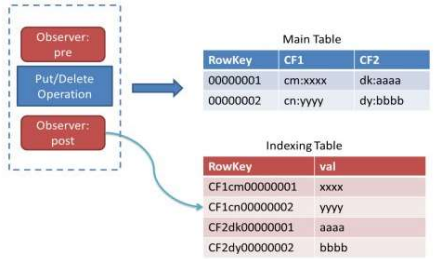
Phoenix由Salesforce.com开源，是构建在Apache HBase之上的一个SQL中间层，可以让开发者在HBase上执行SQL查询。

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

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

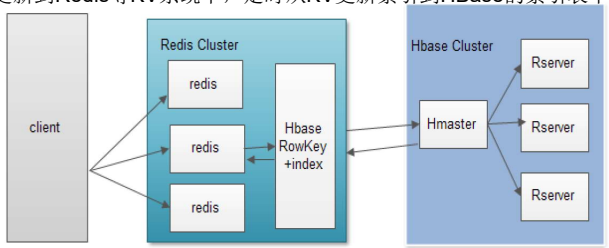
HBase作为列族数据库最经常被人诟病的特性包括：**无法轻易建立“二级索引”**，难以执行求和、计数、排序等操作

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

原理：采用HBase0.92版本之后引入的Coprocessor特性（协处理器）

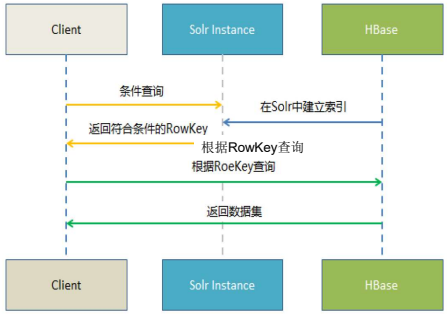
HBase+Redis

•Redis+HBase方案
•Coprocessor构建二级索引
•Redis做客户端缓存
•将索引实时更新到Redis等KV系统中，定时从KV更新索引到HBase的索引表中



Solr+HBase

Solr是一个高性能，采用Java5开发，基于Lucene的全文搜索服务器。同时对其进行了扩展，提供了比Lucene更为丰富的查询语言，同时实现了可配置、可扩展并对查询性能进行了优化，并且提供了一个完善的功能管理界面，是一款非常优秀的全文搜索引擎。



- Solr+HBase
- Solr保存索引

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：第2章安装完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，则由HBase自己管理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

05

HBASE 实践

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

```
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
```

备注：后面的例子都在此基础上继续操作

HBase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”

```
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

HBase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

```
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
```

HBase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

作业

•安装HBASE

•利用Hadoop 提供的Shell 命令完成下列人物

- •列出HABSE中所以表的相关信息，如：创建时间
- •设计表student，并建立表
- •先student表添加和删除指定的列族或列


Thank You!