doi:10.14132/j.cnki.1673-5439.2019.05.009

HOS:一种基于HBase的分布式存储系统设计与实现

季一木[1,2,3,4],张 宁[1],尧海昌[1],李 奎[1],李 航[1],刘尚东[1,2,3,4],王汝传[1,2,3,4]

(1.南京邮电大学 计算机学院,江苏 南京 210023
2.南京邮电大学 江苏省无线传感网高技术研究重点实验室,江苏 南京 210023
3.南京邮电大学 高性能计算与大数据处理研究所,江苏 南京 210023
4.南京邮电大学 高性能计算与智能处理工程研究中心,江苏 南京 210023)

摘要:大数据时代,数据快速增长,迫切需要寻找有效的数据存储方案,HBase系统具有分布式、列式存储的特点,为大数据的存储管理提供了一种高效的解决方案。由于HBase只支持主键索引,对于非主键查询效率低下,难以满足实时需求。为此,提出一种分层式索引查询模型,该模型基于HBase建立持久性索引层、基于Redis建立分布式热点索引缓存层,前者为存储在HBase中的数据建立索引表,提高查询效率,后者基于Redis在内存中存储热点索引,降低磁盘访问开销,进一步提高查询效率。最终,依据此模型实现了分层式索引查询系统HOS,基于Imagenet图片数据集对HOS进行实验,实验结果表明,HOS数据查询性能优于标准HBase。

关键词:HBase;查询处理;分层式索引;分布式存储
中图分类号:TP399 **文献标志码**:A **文章编号**:1673-5439(2019)05-0063-09

HOS:design and implementation of distributed storage system based on HBase

JI Yimu[1,2,3,4],ZHANG Ning[1],YAO Haichang[1],LI Kui[1],LI Hang[1]

LIU Shangdong[1,2,3,4],WANG Ruchuan[1,2,3,4]

(1. School of Computer Science,Nanjing University of Posts and Telecommunications,Nanjing 210023,China
2. Jiangsu High Technology,Nanjing Research Key Laboratory for Wireless Sensor Networks,Nanjing University of Posts and Telecommunications,Nanjing 210023,China
3. Institute of HPC and Bigdata,Nanjing University of Posts and Telecommunications,Nanjing 210023,China
4. Jiangsu Research Engineering of HPC and Intelligent Processing,Nanjing University of Posts and Telecommunications,Nanjing 210023,China)

Abstract:The data is expected to grow rapidly in the era of big data. It is urgently need to find an effective data storage scheme. HBase system has the characteristics of distributed and column storage,and provide an efficient solution for the storage and the management of big data. Because HBase only supports the primary key index and is inefficient for non-primary key queries,thus it is difficult to meet real-time requirements. Therefore,a hierarchical index query model is proposed. The model establishes a persistent index layer based on HBase and a distributed hot index cache layer based on Redis. The former establishes index tables for data stored in HBase to improve query efficiency,while the latter stores the hot index in

收稿日期:2019-03-06 **本刊网址**:http://nyzr.njupt.edu.cn
基金项目:国家重点研发计划专项(2017YFB1401300,2017YFB1401302)、国家自然基金(61572260,61872196)、江苏省重点研发计划(BE2017166)和江苏省自然科学基金优秀青年基金(BK20170100)资助项目
作者简介:季一木,男,博士,教授,jiym@njupt.edu.cn
引用本文:季一木,张宁,尧海昌,等. HOS:一种基于HBase的分布式存储系统设计与实现[J]. 南京邮电大学学报(自然科学版),2019,39(5):63-71.

memory based on Redis to reduce disk access overhead and further improve the query efficiency. Finally, the hierarchical index query system HOS is implemented based on the model. The experimental result on HOS based on Imagenet image dataset shows that the query performance of data of HOS is better than that of standard HBase.

Keywords: HBase; query processing; hierarchical index; distributed storage

随着云计算、物联网、社交网络等技术的发展,数据快速积累,大数据时代已经到来[1]。数据量的快速增长,传统的数据处理、存储和分析技术存在查询效率低,数据维护困难等问题[2]。例如,根据 eMarketer(市场研究公司)的研究数据显示,新浪微博 2018 年拥有 3.41 亿用户,同比增长 17%,微博每天新增 25 亿条分享内容,32 亿条评论,使用传统关系型数据表已经难以支撑大数据背景下的业务需求。

为解决关系型数据库难以存储、查询大数据的问题,非关系型分布式存储系统应运而生,例如,Apache 社区的顶级项目 HBase,Facebook 的 Cassandra 系统[3]以及高效内存数据存储系统 Redis 等等[4]。在这些非关系型存储系统中,HBase 的应用最为广泛[5]。HBase (Hadoop Database)底层使用类 B + 树索引结构,在面对基于主键的数据查询时可以达到毫秒级响应,查询性能非常高效。但是因为缺少非主键索引,面对复杂的非主键查询,必须对全表进行扫描操作,耗时较高,无法满足需要快速响应的数据查询场景[5]。另外,HBase 中的 Region 服务器存储数据时,先把数据存储到内存,在内存到达阈值后 Region 服务器会将数据 flush 到磁盘,而存储文件过大会较频繁触发 compact 和 split 此类耗时操作,对性能影响较大。

本文针对 HBase 对于非主键查询效率较低的问题提出一种分层式索引查询模型,并基于该模型实现了分层式索引查询系统 HOS(HBase Object Storage):(1) 持久化存储层,基于 HBase 将数据存储分为目录表和文件表,其中文件表存储数据文件,目录表存储文件表数据的索引信息;(2) 分布式热点索引缓存层,在 Redis 中对 HBase 文件表中涉及的非主键字段建立索引,利用 Redis 的多条件查询快速获得 Rowkey,实现非主键快速查询,同时基于 Redis 缓存目录表中热点索引数据。通常,在数据的查询分析过程中,数据有着"冷热"之分:其中"冷数据"代表很少被访问的数据,"热数据"代表被频繁访问的数据。HOS 采用基于内存的 Redis 数据库缓存"热数据"的索引。另外,为解决 HBase 大文件存储效率问题,本文提出一种基于 HBase 的大文件存储方案 H-FS(HBase-File Storage),H-FS 会设定一个数据量判定标准,HOS 会根据用户上传文件的大小进行判断,大文件会被 HOS 直接存储在 HDFS 中,小文件直接存储在 HBase 中。

1 相关工作

通常,在 HBase 中检索数据有 2 种方式:基于行键查询和扫描。基于行键查询又分为 2 种情况:基于单个行键查询和基于一定范围的行键查询。由于底层采用了类 B + 树的索引结构,HBase 基于行键查询数据效率非常高效,时间复杂度可以达到 0(logN),如果使用布隆过滤器(Bloom Filter)[6],时间复杂度甚至可以达到 0(1)。但是,HBase 在面对复杂非主键查询条件时,必须使用扫描操作,扫描操作需要对全表数据进行扫描,这导致 HBase 在面对非主键查询时效率较低。大数据背景下,各个行业应用的数据记录可以达到亿级以上,如果每次非主键查询都需要对全表进行扫描,延时过高,无法满足各个行业应用的需求。为解决上述问题,一些研究工作已经展开。

华为公司基于 HBase 实现了 Hindex 系统,其通过在每个 Region 服务器中添加索引从而提高 Region 服务器查询性能。由于 Hindex 在每个 Region 中都放置了索引表,在进行数据查询过程中,Hindex 系统需要访问全部的 Region,如果集群中只有少量的节点放置了待查询数据,而集群中大部分节点并没有存储查询数据的目标记录,此时访问所有的 Region 服务器会导致集群中大部分节点返回空集,浪费不必要的计算资源。

ElasticSearch 和 Solr[7]基于第三方独立引擎 Lucene[8]建立二级索引提高 HBase 非主键查询能力。通过在 ElasticSearch 或者 Solr 中建立索引关联 HBase 表中需要查询的字段,从而提高查询效率。但是这种方式需要维护一套索引集群,造成额外开销。

希腊 Patras 大学基于线段树结构提出了 HBase 非主键查询模型 Interval Index[9]。通过 MapReduce

在内存中建立线段树索引结构,提高 HBase 非主键查询效率。但是 Interval Index 模型面对单点查询,线段树索引结构会退化成一个二叉树,导致索引空间消耗增加。

Yoram Kulbak 和 Dan Washusen 基于内存拦截实现了 IHBASE[10] 系统。HBase Region 服务器将数据存储在内存,在内存达到阈值后把数据 flush 到磁盘。IHBASE 系统在 Region 服务器把数据从内存刷入磁盘前,提前为数据建立索引,从而提高 HBase 非主键查询效率。但是这个方法需要重构 HBase,并且 IHBASE 社区在近几年已经不再更新了。

中国科学院研究团队提出了一种非主键索引方案—CCIndex[11]。通过在每个数据的副本为非主键属性列族建立索引,在面对非主键查询时基于索引表主键进行顺序扫描,从而提高 HBase 非主键查询能力。CCIndex 提升了非主键查询时的性能,但其索引维护困难且由于底层禁止了 HDFS 副本机制,导致数据可靠性较低。

为优化 HBase 非主键查询性能,本文提出了一种基于 HBase 和 Redis 的分层式索引方案。第一层为持久化索引存储层,数据分为 2 个表存储在 HBase 中:目录表和文件表,其中目录表即为文件表中数据文件的索引信息。第二层为分布式内存索引存储层,为文件表中非主键字段建立索引并存储持久化索引存储层中的热点索引。

2　关键技术

2.1　分层式索引存储设计

分层式索引存储模型如图 1 所示。第一层为基于 HBase 的持久化索引存储层,文件表存储数据,目录表存储文件表中数据的索引信息,文件表存储实际数据。第二层为基于 Redis 的分布式热点索引缓存层,目录表索引信息存放在 HBase 中,每次访问都会造成磁盘开销,为进一步提升查询效率,引入内存数据库 Redis,构造分布式内存热点索引数据缓存层,将频繁访问的索引数据存储在 Redis 中,提高查询速度;同时,为提高复杂非主键查询的速度,基于文件表中关键字段(非主键)建立索引信息。

HOS 系统进行数据查询流程如下:

Step1:请求到达 redis 分布式内存索引缓存层,查询热点索引数据,如果命中缓存,根据索引信息直接在 HBase 文件表中查询数据。如果没有命中缓存,跳转 step2。

Step2:查询请求转发到 HBase 目录表中,查询索引数据,根据索引信息在文件表中查询数据。

加入 Redis 缓存后,对于一些热点数据的访问,HOS 系统直接从内存查询到索引,不需要对磁盘进行访问,对于存在数据偏向访问的应用场景,HOS 提高了查询性能。

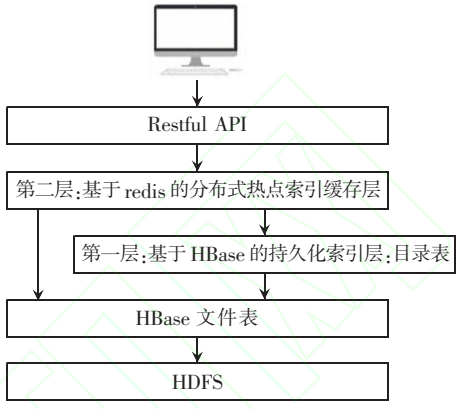

图 1　HOS 分层索引模型

2.1.1　基于 HBase 的持久化索引存储层

传统关系型数据库通常会使用 id 作为索引查询数据,与关系型数据库不同,HBase 基于主键(Rowkey)查询数据,且由于底层采用类 B + 树索引结构,基于 Rowkey 的查询效率非常高效。因此,合理的 Rowkey 设计变得至关重要。第一种方案考虑使用文件 id 作为 Rowkey,此时会面临 2 个问题:(1) 用户如何知道文件 id;(2) 如果用户只知道文件名字,检索该文件需要对整个 HBase 进行扫描操作,无法满足快速读取的要求。第二种方案使用文件名作为 Rowkey,把文件数据作为列族,这种方案同样存在一个问题:不同的文件夹下可能会存在相同的文件名,违背了 Rowkey 的唯一性原则。第三种方案使用文件的全路径作为 Rowkey,这种方案基本可以满足要求,但是却存在一个较大的缺点:如果在一段时间内对某个目录下任意子文件夹进行大量读写,会造成 HBase 的 Region 服务器的热点问题。

为提高 HBase 非主键查询能力,本文设计的基于 HBase 的持久化索引层将数据存储分为 2 个表:(1) 目录表,用来存储管理文件表中的索引信息。(2) 文件表,用来存储数据。我们分别为文件表和目录表设计了表的 Rowkey 使其支持范围文件检索、文件前缀匹配等复杂检索,同时又避免了一部分的热点问题。

目录表是持久化索引存储层的核心,通过对目录表 Rowkey 进行合理的设计,使其能够提高复杂检索的响应速度。对于目录表,我们定义了如下格式的结构:

<文件路径:sub:子目录:t:<value>>
<文件路径:cf:creator:t:<value>>
<文件路径:cf:seqid:t:<value>>

将目录表的 Rowkey 设置为文件的路径,并创建了 2 个列族:sub、cf,如表 1 所示。其中,sub 列族下存储的是当前目录下所有的子文件夹,例如,sub:dir2 = 1,sub:dir3 = 1,这表示在 dir1 下有 2 个子文件夹;cf 列族下存储的是当前文件的一些基础属性,它有 2 列:creator 和 seqid。其中,creator 列是文件的创建者的相关信息,而 seqid 列是一个标识,它将和文件名组合起来作为数据表的 Rowkey。用户可以根据目录的绝对路径来找到一条记录,然后根据其下的 sub 列族找到这个目录下的所有子目录。

表 1 目录表结构

Rowkey	Column:Qualifier
/dir1	sub:dir2 = 1 sub:dir3 = 1 cf:creator = zhangning cf:seqid = 0001
/dir1/dir2	sub:dir4 = 1 cf:creator = zhangning cf:seqid = 0002
/dir1/dir2/dir4	sub:dir5 = 1 cf:creator = zhangning

文件表在 HBase 中存储数据文件,对于文件表,我们定义了如下格式的结构:

<seqid_文件名:c:content:t:<value>>
<seqid_文件名:cf:creator:t:<value>>
<seqid_文件名:cf:size:t:<value>>
<seqid_文件名:cf:type:t:<value>>

将目录表中的 seqid 与文件名组合起来作为文件表的 Rowkey,并创建了 2 个列族:c 和 cf,如表 2 所示。其中,c 列族下的 content 列存储了文件的内容;cf 列族下存储的是当前文件的一些基础属性,它有 3 列:filename、size 和 type。其中,文件名称的相关信息存储在 filename 列,文件大小存储在 size 列,文件类型存储在 type 列。文件表的设计以 seqid 与文件名的组合做为 Rowkey,如果用户对某个文件夹进行频繁的读写,那么所有文件的前缀 seqid 是相同的,这同样存在着热点问题,但是对比上述三种方案,这种设计方案减少了 Region 服务器的热点问题。HOS 通过 seqid 实现文件的查找和过滤,如果用户想查找某个文件,只需要获取到父目录的 seqid,然后拼接成文件的 Rowkey 就可以实现随机的读取。另外,HBase 本身支持字典排序,在加了 seqid 后,文件名依然有序,可以通过起止文件名对文件进行过滤操作。文件表中文件名的选取是依据存储数据的不同而选取,本文因为采用 Imagenet 图片数据集,所以选取了图片名称作为文件名。

表 2 文件表结构

Rowkey	Column:Qualifier
0001_file1	c:content = bytes cf:filename cf:size cf:type
0001_file2	c:content = bytes cf:filename cf:size cf:type
0002_file3	c:content = bytes cf:filename cf:size cf:type

2.1.2 基于 Redis 的分布式缓存索引存储层

为降低查询数据时磁盘开销并增加 HOS 非主键查询能力,引入基于 Redis 的分布式热点索引缓存层,为文件表中关键字段建立索引信息和存储目录表中热点索引数据。

对于非主键查询,基于 Redis 建立索引信息,存储结构为 key-value 模式,如表 3 所示。其中,key 为非主键字段的列名,value 为 Rowkey。非主键查询流程如图 2 所示,客户端从 Redis 上查询到非主键字段对应的 Rowkey,然后基于 Rowkey 在 HBase 中快速查到数据。

表 3 非主键索引结构

key	Value
Filename1	Rowkey1
Filename2	Rowkey2
Filename3	Rowkey3

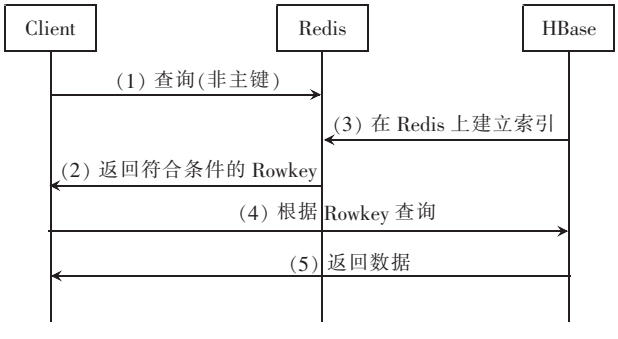

图 2 非主键查询流程

热点数据的索引主键作为 key,而索引集合作为 Redis Set 的 value 保存在内存缓存中。具体存储结构如图 3 所示。

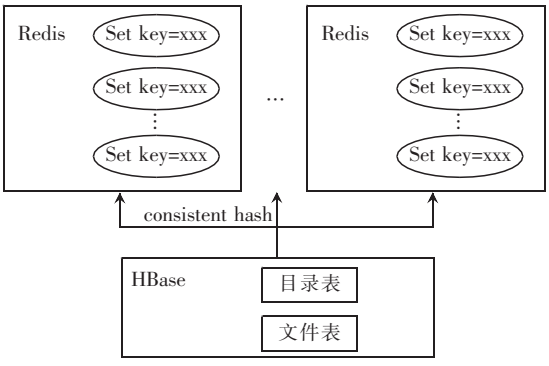

图 3　HOS 分层式数据存储

HOS 系统在 HBase 服务器节点的分布式内存上采用一致性哈希算法管理所有的索引热点数据。一致性哈希的基本原理如图 4 所示[12]:(1) 构造环形 hash 空间。使用哈希函数将 value 映射到 $0-2^{32}$ 的圆(continuum)上。(2) 将数据对象映射到 hash 空间。通过哈希函数计算 hash 值 key,key 值必定分布在环形 hash 空间上。(3) 将 cache 映射到 hash 空间。使用相同的哈希算法将对象和 cache 都映射到同一个 hash 数值空间中。(4) 把数据对象映射到 cache 上。从环形 hash 空间上该数据对象的映射点开始,沿顺时针方向,找到的第一个 cache 节点即为该数据对象的存储位置。

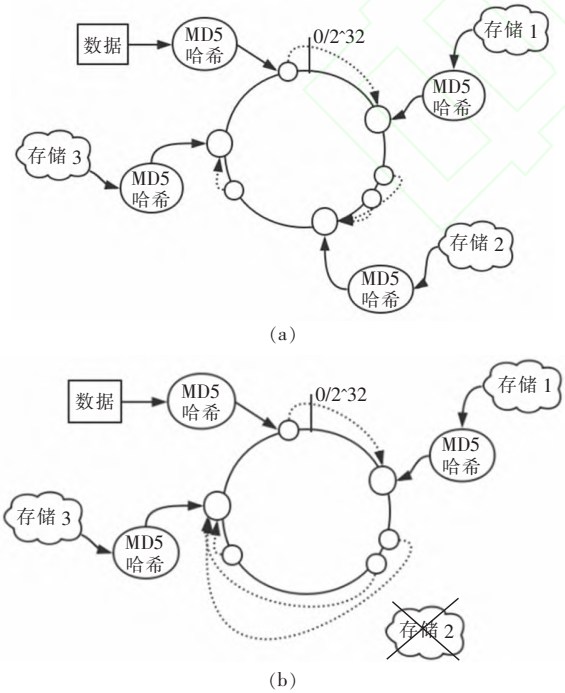

图 4　一致性哈希的存储节点映射

一致性哈希降低了 cache 节点变化(增加或减少)带来的数据传输开销,例如,当 cache 节点出现故障时,只需要迁移该 cache 节点与上一个 cache 节点中间的数据对象,而无需迁移所有已存储的数据对象。如图 4(a)中存储节点 cache2 出现故障时,原先映射到 cache2 中的数据对象会继续沿顺时针方向映射到节点 cache3 上。

一致性哈希通过将数据 hash 到不同的存储节点上,从而保证了各个存储节点的平衡。当需要查询数据的索引信息时,HOS 系统会通过 2 个步骤找到数据的索引信息:(1) 对数据进行一致性哈希,如图 3 所示,找到数据索引信息所在的存储节点;(2) 哈希机制(Redis),找到节点内的索引数据地址。

2.2　大文件存储模型 H-FS 设计

HBase 在存储数据时,其 Region 服务器先将数据存储在内存中,在内存到达阈值后将数据刷到磁盘上。因此,写入的数据文件过大会导致 Region 服务器的 Split 过程和 Compact 过程频繁触发,在一定程度上影响客户端的写入,降低 HBase 的写入性能。为解决 HBase 存储大文件的效率问题,可以使用 2 种方法[13-14]:(1) 对大文件进行分片处理,在 HBase 中存储多个分片。这个方案提高了 HBase 系统对大文件存储的效率,但是需要额外维护数据分片的顺序信息。(2) 大、小文件分开存储,大文件存储到 HDFS 中,其索引信息存储在 HBase 表中,小文件直接存储在 HBase 中。

与第 1 种方法相比,第 2 种方法避免了对数据文件进行分片,从而也就不需要维护数据分片的顺序信息。并且由于把大、小文件分开存储,不需要对分片后的文件进行 compact 操作。提高了 HBase 系统对大文件的存储效率。

本文基于第 2 种方法设计了大文件存储模型 H-FS。如图 5 所示,H-FS 模型流程如下:

Step1:设定一个基准的文件大小(该值根据服务器的配置设置,本系统设置为 20 M)。

Step2:判断上传文件的大小,如果小于基准值,将该文件存储在 HBase 中,否则跳转 step3。

Step3:将文件存储在 HDFS 上,同时把文件索引信息存储 HBase 表中。

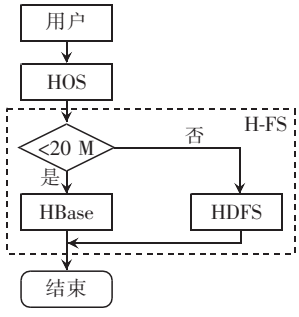

图 5　大文件存储模型 H-FS

3 HOS 系统实现

HOS 系统的功能架构图如图 6 所示,由 3 层构成。最底层为数据持久化模块,这部分主要是基于 Redis、MySQL、HBase、HDFS 对数据进行存储管理。持久化模块的上层是项目的三大核心模块:用户管理、权限管理和文件管理。其中,用户管理实现对 HOS 系统中所有用户的添加、删除和修改等功能;权限管理模块实现对系统所有用户的授权操作功能,用户根据权限获取文件;而文件管理模块为 HOS 系统的核心,其实现了文件的上传、下载和删除等操作。最后,系统的最上层就是直接面向用户的接口服务和 SDK 模块。接口与 SDK 模块实现 Restful API 和 SDK 功能,使系统用户可以更加方便地使用 HOS 系统服务。

图 6 HOS 系统功能架构

HOS 系统的实现如图 7 所示。图中为 HOS 的核心模块——文件管理,用户可以通过 Hbase、Redis 实现文件的上传、下载、查询和过滤等功能。

图 7 HOS 文件管理模块

3.1 系统部署设计

为方便用户远程使用,本系统采用 B/S 架构,通过 Apache Tomcat 容器将应用部署在云服务器上,用户通过浏览器可直接访问。服务端采取 Spring boot、Mybatis 和 HBase 架构。其中,Mybatis 用来完成数据的持久化操作和读取操作,项目采用 Mybatis 实现数据持久层的操作;采用 Spring Boot 简单快速搭建 web 项目;HBase 和 Redis 用来存储数据和热点索引,是 HOS 系统实现分层式索引存储的核心。最后,HOS 系统的部署设计如图 8 所示。

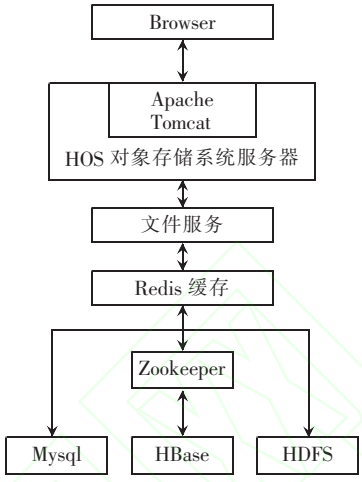

图 8 HOS 系统部署

3.2 数据库设计

在 HOS 系统中,底层的对象存储服务是基于 HBase 和 HDFS,而系统本身的数据库使用的是 Mysql。在 Mysql 中,建立了 4 张互相关联的数据库表管理全局数据:user 表、Token 表、授权表和 Bucket 表。其中,user 表主要存放用户的基本信息,其 user_id 是用户的 Token 信息,用户可以通过 Token 访问 HOS 服务;Token 表存放的是用户对其所属 Bucket 文件的授权信息,通过对某个 Bucket 创建 Token 可以使得非本系统用户也可以访问 HOS 服务;授权表存放的是被授权的 Bucket、用户创建的被授权 Token 信息;Bucket 表存放的是 Bucket 文件信息。表关系如图 9 所示。

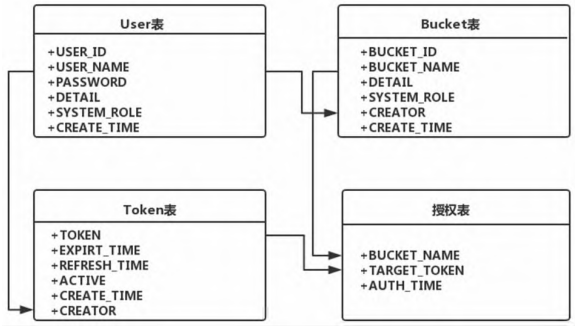

图 9 HOS 系统表关系

3.3 HOS 详细功能设计

根据系统 4 个模块的作用划分,HOS 系统具有以下主要功能:用户管理模块实现用户信息管理,包括系统用户的登录、删除和注册等功能;权限管理模块主要分为 Token 管理及 Token 授权 2 部分,包括 Token 的创建、删除和授权等功能;文件管理模块为本系统的核心,负责对海量非结构化数据的存储,包括文件的上传、下载、删除和过滤等功能;接口与

SDK 模块直接面向用户提供系统接口与 SDK,方便用户访问系统服务。系统整个功能流程如图 10 所示。

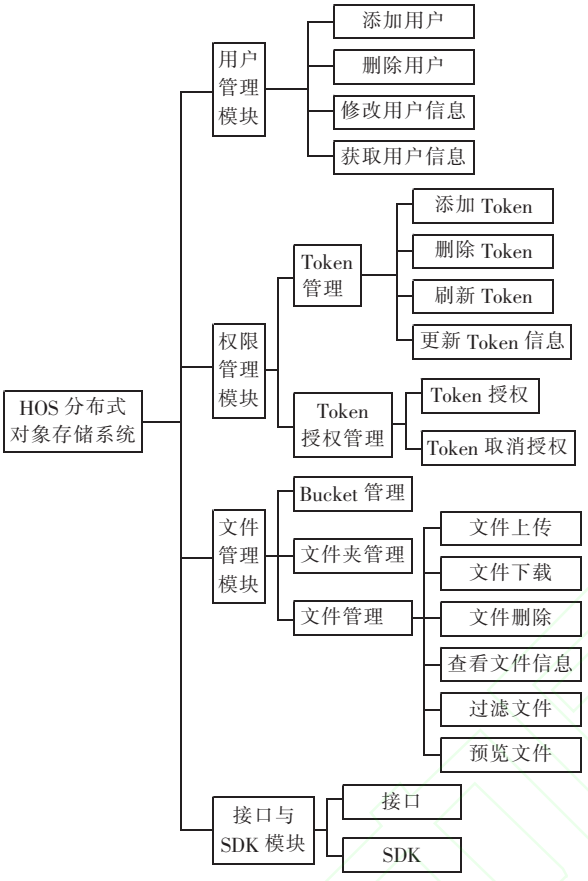

图 10　HOS 系统功能

3.3.1　用户管理

用户管理模块负责 HOS 系统所有用户的登录、退出和鉴权等操作管理。此外,因为本系统的权限管理是跟 Token 关联的,所以在新建用户时系统默认会把用户的 user_id 作为该用户的 Token,并且设置成永不过期,只要用户存在,那么该 Token 就不会过期。所以,在系统新建用户时,不仅需要在用户表添加一条记录,同时需要在 Token 表中添加一条永久 Token 值;同样,在删除用户信息时,需要删除用户表中的记录、Token 表中的对应值及其 Token 与 Bucket 文件的权限映射。

3.3.2　权限管理

在本系统中,权限管理模块主要由 2 个子模块构成:Token 管理、Token 授权管理。用户可以进入 Token 管理子模块实现 Token 的添加、删除、刷新及更新操作。此外,系统中不同的用户对文件拥有不同的操作权限,每个用户只能访问由自己创建的文件,但是用户可以通过 Token 授权管理子模块,把新建的 Token 赋予别的用户,这样,被赋予 Token 的用户就有了访问该用户文件的权限。系统通过引入 Token 值,能够较为灵活方便地管理权限。

3.3.3　文件管理

文件管理模块是整个 HOS 系统服务的核心,负责对整个系统的数据进行管理维护。该模块主要由 3 个子模块构成:Bucket 管理、文件夹管理和文件管理。在 Bucket 管理子模块中,用户可以创建新的 Bucket,此时系统会默认创建 2 个 HBase 表,一个为目录表,另一个为文件表,并且会在 HDFS 上创建对应的目录;在删除 Bucket 的时候也要删除对应的 HBase 及 HDFS 文件夹;访问相关 Bucket 时,系统会验证访问者的 Token 权限,防止没有权限的用户访问 Bucket,保证文件的安全性。同时,用户可以在文件管理子模块中对文件进行上传和下载等操作。在对文件进行上传时,系统可为用户自动创建多级目录,且文件的存储被划分为 2 个步骤:小文件存储到 HBase,大文件直接存储到 HDFS 中。同时,为避免出现用户 A 删除目录 1 时,用户 B 正在向目录 1 上传文件等系统并发情况,系统通过 zookeeper 实现分布式锁。用户可以通过文件唯一标识 seqid 对文件进行过滤操作。

3.3.4　接口与 SDK

接口与 SDK 模块支持用户登陆或者 Header 头中带有 Token 权限的访问,并且提供 restful api 响应用户的请求。

4　实验与性能分析评估

4.1　实验环境

本文选择了 4 台服务器搭建了完全分布式的 HBase 的集群环境,在该集群中,有 1 台 master 节点,3 台 slave 节点。其中,每个节点的详细配置信息如表 4 所示,集群拓扑结构如图 11 所示。

表 4　计算节点配置信息

参数	配置
CPU	Intel(R) Core(TM) i5-7300HQ 2.50 GHz
Memory	16 GB
Disk	2TB 7200RPM SATA II
OS	Ubuntu 14.04
JVM Version	Java 1.8.0
Hadoop Version	Hadoop 2.7.3
HBase Version	HBase 1.2.4

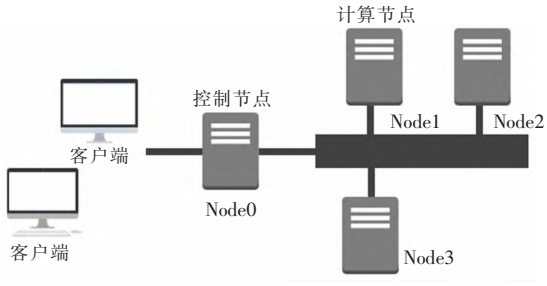

图 11 集群拓扑图

4.2 实验内容

本文对 HOS 系统与标准 HBase 在插入和读取两方面进行性能对比。测试选取了 Imagenet 图片数据集(IMAGENET Large Scale Visual Recognition Challenge),大小为 1 TB,其中有 1400 多万幅图片,涵盖 2 万多个类别。

(1) 数据插入性能对比

测试目的:测试比较标准 HBase 和改进后的 HOS 系统在集群下的插入性能。

测试步骤:

① 在客户端开启 30 个线程进行数据插入;

② 在标准 HBase 系统下循环插入图片数据;

③ 在 HOS 系统下同样循环插入图片数据;

④ 统计标准 HBase 和 HOS 插入数据的吞吐量(5 s 一次);

⑤ 计算每秒插入多少条记录。

结果对比分析,如图 12 所示:在刚开始进行大对象插入操作时,HOS 系统和标准 HBase 的插入性能很接近。随着时间的增长,标准 HBase 的性能在下降。从图中可以看出,标准 HBase 对大对象的存储存在很大波动,在存储过程中会出现延时,严重影响其写入性能;而 HOS 系统在存储大对象时,随着时间的增长,其插入性能也在逐渐的提升,并且逐渐走向稳定。可以看出,改进后的 HOS 系统平均每秒可以存储 200 ~ 240 张图片数据,并且在存储大对象时,与标准 HBase 相比,HOS 降低了存储过程的延时,增加了存储性能的实时性。通过两者的比较,HOS 系统插入数据的效率要优于标准 HBase。

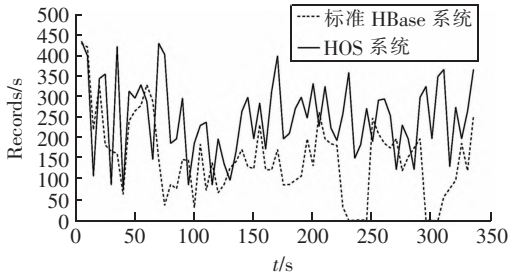

图 12 插入性能对比

(2) 查询性能对比

测试目的:测试比较标准 HBase 和改进后的 HOS 系统在集群下的查询性能。

测试步骤:

① 在标准 HBase 下查询目标数据,记录查询所用时间;

② 在 HOS 系统下查询目标数据,记录查询所用时间;

③ 多次测试,取平均值,根据结果绘图并比较在 2 个系统下的查询时间。

结果对比分析,如图 13 所示,从图 13 中可以看出,无论查询多少数据,标准 HBase 系统因为需要对全表进行扫描操作,其最后的结果都为 800 s 左右,这显然是极为耗时的。而改进后的 HOS 系统在面对复杂非主键查询时有了巨大的性能提升,从上面的结果可以看出,查询 100 张图片只需 15 s 左右,查询 1000 张图片需要 67 s 左右,查询 1 百万张图片,才需要 200 s 左右。并且由于有 Redis 热点索引缓存层的存在,在后面的查询中,HOS 系统的查询性能会进一步提升。

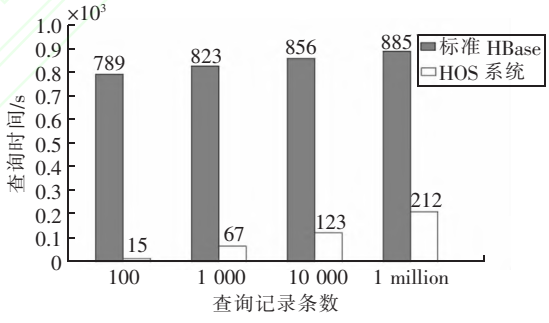

图 13 查询性能对比

5 结束语

为提高 HBase 插入数据的效率和面对非主键查询时的检索效率,本文提出了大文件存储方案 H-FS 和分层式索引存储模型,并基于其实现了分层式索引存储系统 HOS。HOS 借助 HDFS 实现了大文件与小文件的分离存储,提升了整个系统写入大文件的效率。同时,在 HBase 上搭建了持久化的索引存储层,在 Redis 上搭建了基于内存的热点索引缓存层,为持久索引存储层的非主键建立索引并将热点数据缓存在内存中,这使 HOS 系统获得了比标准 HBase 更优的存储和查询效率。

后续我们将针对分布式内存热点数据的缓存替换策略进行研究改进,考虑数据访问的累积热度,保证缓存空间得到充分利用,从而进一步提高数据访

问的速度。

参考文献:

[1] 蒋晨晨,季一木,孙雁飞,等. 基于 Storm 的面向大数据实时流查询系统设计研究[J]. 南京邮电大学学报(自然科学版),2016,36(3):100 - 105,111.
JIANG Chenchen,JI Yimu,SUN Yanfei,et al. Design of real-time steam query system oriented to big data based on Storm[J]. Journal of Nanjing University of Posts and Telecommunications (Natural Science Edition),2016,36(3):100 - 105,111. (in Chinese)

[2] 陈国良,尧海昌,李航,等. 大数据一体机关键技术及应用研究[J]. 南京邮电大学学报(自然科学版),2018,38(1):1 - 19.
CHEN Guoliang,YAO Haichang,LI Hang,et al. Key technology and application of big data machine[J]. Journal of Nanjing University of Posts and Telecommunications (Natural Science Edition),2018,38(1):1 - 19. (in Chinese)

[3] REDFORD K,SANJAYAN M A. Retiring Cassandra[J]. Conservation Biology,2003,17(6):1473 - 1474.

[4] ZAKI A K,INDIRAMMA M. A novel redis security extension for NoSQL database using authentication and encryption[C]// IEEE International Conference on Electrical. 2015.

[5] 葛微,罗圣美,周文辉,等. HiBase:一种基于分层式索引的高效 HBase 查询技术与系统[J]. 计算机学报,2016,39(1):140 - 153.
GE Wei,LUO Shengmei,ZHOU Wenhui,et al. HiBase:a hierarchical indexing mechanism and system for efficient HBase query[J]. Chinese Journal of Computers,2016,39(1):140 - 153. (in Chinese)

[6] HOLLEY G,WITTLER R,STOYE J. Bloom filter trie:an alignment-free and reference-free data structure for pan-genome storage[J]. Algorithms for Molecular Biology,2016,11(1):3.

[7] MANGHI P,ARTINI M,BARDI A,et al. High-performance annotation tagging over Solr full-text indexes[J]. Information Technology & Libraries,2014,33(3):22 - 44.

[8] YANG P,FANG H,LIN J. Anserini:enabling the use of Lucene for information retrieval research[C]// International ACM SIGIR Conference on Research & Development in Information Retrieval. 2017:1253 - 1256.

[9] SFAKIANAKIS G,PATLAKAS I,NTARMOS N,et al. Interval indexing and querying on key-value cloud stores[C]// IEEE International Conference on Data Engineering. 2013.

[10] 王文贤,陈兴蜀,王海舟,等. 一种基于 Solr 的 HBase 海量数据二级索引方案[J]. 信息网络安全,2017,(8):39 - 44.
WANG Wenxian,CHEN Xingshu,WANG Haizhou,et al. A secondary index scheme of big data in HBase Based on Solr[J]. Information Network Security,2017,(8):39 - 44. (in Chinese)

[11] ZOU Y,LIU J,WANG S,et al. CCIndex:a complemental clustering index on distributed ordered tables for multi-dimensional range queries[C]// Seventh International Conference on Semantics Knowledge & Grid. IEEE,2011.

[12] LIU Q,CAI W,SHEN J,et al. VPCH:a consistent hashing algorithm for better load balancing in a hadoop environment[C]// Third International Conference on Advanced Cloud & Big Data. IEEE,2015.

[13] 马振,哈力旦·阿布都热依木,李希彤. 海量样本数据集中小文件的存取优化研究[J]. 计算机工程与应用,2018,54(22):80 - 84,98.
MA Zhen,HALIDAN Abudureyimu,LI Xitong. Research on access optimization of small files in massive sample data sets[J]. Computer Engineering and Applications,2018,54(22):80 - 84,98. (in Chinese)

[14] SHEORAN S,SETHIA D,SARAN H. Optimized MapFile based storage of small files in Hadoop[C]// 17th IEEE/ACM International Symposium on Cluster,Cloud and Grid Computing (CCGRID). 2017.