Journal of Computer Applications
计算机应用,2018,38(6):1584 - 1590
ISSN 1001-9081
CODEN JYIIDU
2018-06-10
http://www.joca.cn

文章编号:1001-9081(2018)06-1584-07
DOI:10.11772/j.issn.1001-9081.2017112777

基于内存的 HBase 二级索引设计

崔　晨[1*],郑林江[1],韩凤萍[2],何牧君[1]
(1.重庆大学 计算机学院,重庆 400044;　2.重庆城市综合交通枢纽开发投资有限公司,重庆 401121)
(* 通信作者电子邮箱 15123034669@163.com)

摘　要: 在大数据时代,具有海量数据存储能力的 HBase 已被广泛应用。HBase 只对行键进行了索引优化,对非行键的列未建立索引,这严重影响了复杂条件查询的效率。针对此问题,提出了基于内存的 HBase 二级索引方案。该方案对需要查询的列建立了映射到行键的索引,并将索引存储在 Spark 搭建的内存环境中,在查询时先通过索引获取行键,然后利用行键在 HBase 中快速查找对应的记录。由于列的基数大小和是否涉及范围查询决定了建立索引的类型,故针对三种不同情况构建了不同类型的索引,并利用 Spark 内存计算、并行化的特点来提高索引的查询效率。实验结果表明,该二级索引具有较好的查询性能,查询时间小于基于 Solr 的二级索引,可以解决 HBase 中因非行键的列缺乏索引导致查询效率较低的问题,提高基于 HBase 存储的大数据分析的查询效率。

关键词: HBase; Spark; 二级索引; 内存索引 ; 并行化
中图分类号: TP311.133.1　　**文献标志码:** A

Design of secondary indexes in HBase based on memory

CUI Chen[1*], ZHENG Linjiang[1], HAN Fengping[2], HE Mujun[1]
(1. *College of Computer Science, Chongqing University, Chongqing* 400044, *China;*
2. *Chongqing Integrated Transport Hub Development Investment Company Limited, Chongqing* 401121, *China*)

Abstract: In the age of big data, HBase which can store massive data is widely used. HBase only can optimize index for the rowkey and donot create indexes to the columns of non-rowkey, which has a serious impact on the efficiency of complicated condition query. In order to solve the problem, a new scheme about secondary indexes in HBase based on memory was proposed. The indexes of mapping to rowkey for the columns which needed to be queried were established, and these indexes were stored in memory environment which was built by Spark. The rowkey was firstly got by index during query time, then the rowkey was used to find the corresponding record quickly in HBase. Due to the cardinality size of the column and whether or not the scope query determined the type of index, and different types of indexes were constructed to deal with three different situations. Meanwhile, the memory computation and parallelization were used in Spark to improve the query efficiency of indexes. The experimental results show that the proposed secondary indexes in HBase can gain better query performance, and the query time is less than the secondary indexes based on Solr. The proposed secondary indexes can solve the problem of low query efficiency, which is caused by the lack of indexes of non-rowkey columns in HBase, and improve the query efficiency for large data analysis based on HBase storage.

Key words: HBase; Spark; secondary index; memory index; parallelization

0　引言

物联网和大数据的广泛应用,产生了海量的多类型实时数据。传统的数据存储与管理方法已难以适应当前大规模数据管理对效率的需求,因此非关系型数据库(Not Only SQL,NoSQL)[1]得以迅速发展。HBase 作为 NoSQL 数据库的代表,已被广泛应用于各行各业的数据存储与管理中。索引是数据库中提高数据管理与查询效率的关键技术,HBase 在行键(Rowkey)上建立了类 B+树索引,可以高效地支持基于行键的快速数据查询[2],但对非行键的列没有建立索引,故进行非行键的列的查询时,需要对全表进行扫描,这样的查询效率十分低下,极大限制了 HBase 在复杂条件下的查询性能。为解决这一问题,现有研究提出建立非行键的列与行键的索引,从而构建起二级索引方案,方案在对非行键的列进行查询时,先通过二级索引获取与其对应的行键,再通过行键到 HBase 中进行查找。

目前,国内部分企业和个人基于第三方开源软件进行索引设计,使用搜索服务器 Solr[3] 来构建 HBase 二级索引。Solr 是一个基于 Lucene[4] 的企业级全文搜索服务器,用户可以通过超文本传输协议(HyperText Transfer Protocol, HTTP)请求向 Solr 提交 HBase 中的数据,建立非行键的列与行键的映射,生成索引;也可以通过 HTTP Get 操作提出查找请求返回索引

收稿日期:2017-11-27;修回日期:2018-02-04;录用日期:2018-02-05。　基金项目:国家 863 计划项目(2015AA015308);国家重点研发计划项目(2016YFC0801707);重庆市应用开发计划重点项目(cstc2014yykfB30003)。

作者简介:崔晨(1994—),男,安徽安庆人,硕士研究生,主要研究方向:智能交通系统、大数据;　郑林江(1983—),男,四川邻水人,副教授,博士,CCF 会员,主要研究方向:智能交通系统、大数据;　韩凤萍(1983 一),女,江苏扬州人,工程师,硕士,主要研究方向:交通工程;何牧君(1982 一),男,浙江慈溪人,博士研究生,主要研究方向:智能交通系统、大数据。

中的行键。华为基于协处理器(Coprocessors) 进行了 HBase 二级索引的实现,它利用 HBase 的原生代码即可获得索引,但这也带来插入数据速度变慢、升级维护困难以及索引无法动态修改等问题。葛微等[2] 提出了名为 HiBase 的二级索引,它是一种基于分层式索引的设计方案,其将热点索引进行缓存,并建立高效的缓存替换策略来提高二级索引的查询速度。Zou 等[5] 提出了互补聚簇式索引(Complemental Clustering Index, CCIndex) ,该方案把数据的详细信息也存放在索引表中,不需要通过获取的行键再到原表中去查找数据。

在索引实现方面,基于内存的索引相较于传统位于磁盘的索引在设计和架构上都大不相同,基于内存的索引在查询效率上得到了极大提升。广泛采用的内存索引有 T 树[6] 、基于缓存敏感的 CSS/CSB + 树和改进的 Hash 索引等。T 树,是针对 B + 树在内存中空间浪费的问题而提出的一种树型索引。T 树所有节点都直接指引数据项,节省了中间节点所占空间。而针对 Hash 索引空间利用问题,提出了最小完美 Hash 索引[7] ,其哈希函数不会产生冲突且函数的值域和参数域的大小完全相等,但这一哈希函数构造难度非常大。后来,研究发现 CPU 高速缓存对数据查询性能有着非常显著的影响[8] ,所以研究者在 B + 树的基础上对缓存作了优化,提出了 CSB + 树[9] 。在 Hash 索引的基础上,提出了可扩展 Hash、桶链 Hash 等缓存敏感的索引结构。到了 2007 年,Ross[10] 提出了基于现代处理器的 Hash 预取算法,将单指令多数据流(Single Instruction Multiple Data, SIMD) 指令集融入到 Hash 算法中,在内存环境中大大提高了索引的性能。文献[11] 基于有界障碍(Bounded Disorder, BD) 方法提出了一种 BD 树索引,其将树形结构与 Hash 表结构结合,上层是树结构,下层叶子节点是哈希表,每个哈希表中桶的大小等于 CPU 缓存块的大小。这样的索引结构既对缓存作了优化,又实现了针对 Hash 表的范围查找。文献[12] 中根据 BD 树索引的思想,实现了 B + 树索引与 Hash 索引的结合并进行了实验,结果展现 BD 树索引具有较好的性能。

本文提出一种基于内存的 HBase 二级索引的设计方案。为 HBase 中非行键的列建立索引并存储在 Spark 集群中。Spark 是一款基于内存的并行计算框架[13] ,其将数据加载到内存中,然后在内存中完成计算,且 Spark 集群能构造一个大的内存环境。由于是将索引建立在内存中,所以在索引的选择上希望得到节约存储空间、查询效率较高的索引。本文根据要建索引的列的基数大小以及是否涉及范围查询构建了三种基于内存的索引,分别是位图索引、分段 Hash 索引以及 BD 森林索引,并利用 Spark 并行化的特点来提高索引的查询效率。

1　基于内存的 HBase 二级索引总体设计

1.1　二级索引结构

HBase 对行键建立了索引,对非行键的列未建立索引,这严重影响了条件查询的效率。为此,针对 HBase 在进行非行键的列的查询和组合条件查询时效率较低的问题,本文对需要查询的列建立与行键对应的二级索引,建立的二级索引结构如图 1 所示。图 1 中对 HBase 的簇(Column Family, CF) 的 NAME 列和 ADDR. 列在 Spark 上建立了非行键的列的索引,索引可以通过 NAME 列或 ADDR. 列中的属性值映射到 HBase 中对应的行键。有时会对 NAME 列与 ADDR. 列进行组合条件查询,故针对这两列建立了组合条件的索引,可以通过索引映射到满足这一组合条件的行键。

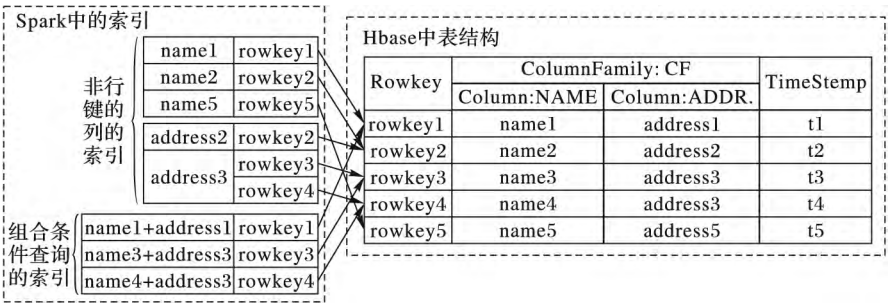

图 1　HBase 二级索引结构

Fig. 1　Secondary index structure in HBase

查询非行键的列或组合条件时,可以先通过位于 Spark 内存环境中的索引获取对应的行键,再利用行键在 HBase 中查询符合条件的记录。这样的二级索引能极大提高条件查询的效率。

1.2　二级索引查询框架

面对非行键的列或组合条件的查询,可以先建立上述结构的二级索引。该索引以弹性分布式数据集(Resilient Distributed Dataset, RDD) 的形式存储在 Spark 的内存环境中,RDD 是 Spark 内存计算中一种弹性分布式数据集,且在 Spark 上有许多内置操作,可以将其并行化地转换。为了防止断电内存数据的丢失,对主要的 RDD 进行磁盘持久化,也可从磁盘中读取历史数据的 RDD。查询时在 Spark 中对索引的 RDD 进行操作,获取对应的行键,再用行键在 HBase 中查找对应的记录并返回给查询的客户端。图 2 展示了二级索引的查询框架。

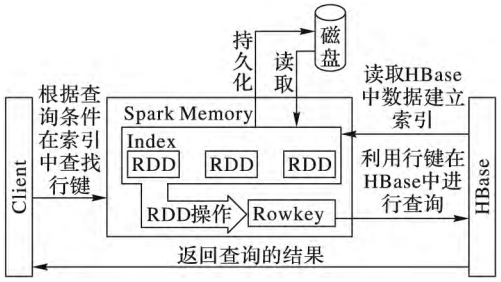

图 2　二级索引查询框架

Fig. 2　Secondary index query framework

这样的查询框架可以充分发挥 HBase 中行键的作用,同时利用了 Spark 内存计算和并行化的优势。为了提高框架的查询效率,本文要构建合适的内存索引用于该查询框架中,这

将在第 2 章节具体讨论。

1.3 表和二级索引的构造

使用上述的二级索引，首先要在 HBase 中构造存储数据的表，其中行键设计是构造表的关键。由于 HBase 是无模式的，事先不需要定义列限定符和数据类型，只需利用数据中的相关字段构建每条数据的行键。在构建行键时，要选择反映数据重要特征的字段组合成行键，并且这样的行键要具有唯一性。例如在后面的实验中针对基于射频识别(Radio Frequency IDentification，RFID)的机动车电子车牌数据，本文选择采集点 IP 和通行时间这两个重要的字段来建立行键，同时在其基础上加入随机数，使行键更加唯一和分散，避免热点问题。如果行键过长，为了在读写以及存储时具有好的性能，可以对行键进行压缩处理。

在二级索引的构造过程中，建立一个辅助的记录表。对 HBase 表中新插入的数据存储到辅助的记录表中，每当记录表中的数据个数达到一定的阈值时，就对记录表中的数据建立对应的索引并以 RDD 的形式存储(具体的索引构建过程见第二章)，故所建的索引都是分段的索引，这样有利于 Spark 的并行操作。与此同时清空辅助的记录表，用于记录下一批新插入的数据，从而对这部分新的数据建立二级索引。由于 HBase 中更新操作实际上是插入新的数据，而本文在行键的构造中加入随机数使其唯一化，不存在行键相同根据时间戳来更新数据的情况，故本文中二级索引不进行更新。当在 HBase 的表中删除数据时，则要将 RDD 形式的索引全部删除，重新分批读取 HBase 中的数据，每读取一定数量的数据就建立分段的索引，以保证索引内容的正确性。

2 内存索引构建

列的基数是指该列拥有的不重复数值的个数。例如：员工信息表中性别这列的基数是 2，只有“男”和“女”两个值。列的基数的大小直接影响索引的选择与构造。同时，是否涉及范围查询也决定了索引方法的选择。为此，根据各列基数大小以及是否涉及范围查询，本文构建了不同的内存索引。针对基数较小列，构建位图索引；针对基数较大但不涉及范围查询的列，构建分段 Hash 索引；针对基数较大且涉及范围查询的列，构建 BD 森林索引。整个索引构建的步骤流程如图 3 所示。

2.1 对基数较小列的索引

一般认为列基数小于 100 且小于 HBase 中的总列数的 0.1% 时，该列即为基数较小列。针对基数较小列，由于字段重复值较多，建立树形索引不能带来快速的查询响应，还会带来索引空间冗余，达不到“空间换取时间”的目的。因此，针对基数较小列，采用位图索引进行二级索引的构建。

位图索引是用二进制位 0、1 组成的位向量表示数据的索引信息，并通过将检索条件映射为二进制位的逻辑运算来实现高效的查询。设有一组数据，数据的总记录数为 N，数据中有一列为基数较小列。对基数较小列建立索引，针对基数较小列的每一属性值 a 建立一个位向量 X，X 的长度为 N，判断 X 的第 j 位 X_j 表示第 j 条记录在该列中的值是否为属性值 a，若是 a，则 $X_j = 1$；若不是 a，则 $X_j = 0$。员工信息表中性别这一列建立的位图索引示例如图 4 所示。

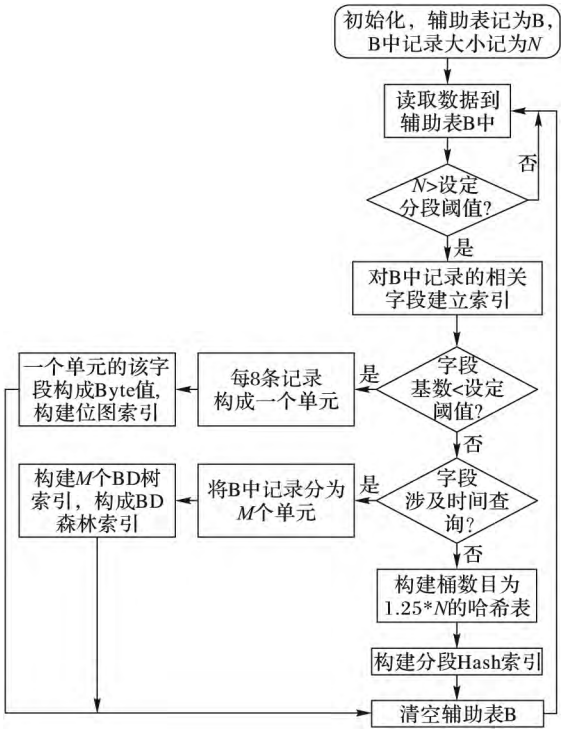

图 3 索引构建步骤流程

Fig. 3 Flowchart of index building step

ID	Name	Sex
Y1	name1	男
Y2	name2	女
Y3	name3	男
…	…	…
Yn	name$n$	女

Sex	Bitmap				
	Y1	Y2	Y3	…	Yn
男	1	0	1	…	0
女	0	1	0	…	1

图 4 员工信息表的性别位图索引

Fig. 4 Bitmap index of employee information about sex

位图索引采用 0、1 来存储信息，在内存中所占空间很小。内存中一个字节的空间可以存储 8 条某个属性值的取值情况，例如 800 万条员工信息对性别为“男”的情况建立索引，则只需消耗不到 1 MB 内存空间。位图索引的另一优势就是可以将组合查询转为高性能的位运算。例如已经得到 N 条记录其 A 列中属性值 a 的位向量 X 和 B 列中属性值 b 的位向量 W。若要查询的组合条件为 A 列中值为 a 且 B 列中值为 b 的记录时，只需将 X 与 W 进行按位与的运算得到位向量 Z，若 $Z_j = 1$，则第 j 条记录符合本文的查询条件。

在本文中，用字节数组来存储位图索引并将其建立在 Spark 构成的内存空间中。为了方便构造和提高构造速度，取每 N 条记录为一组构建分段的位图索引。每段位图索引在 Spark 内存中用 parallelize 方法转为 RDD 形式，RDD 在 Spark 的环境中可以被缓存且支持并行操作。为了防止断电内存数据丢失，将这些 RDD 进行 RDD. saveAsObjectFile 执行操作，从而把数据持久化到硬盘中。在查询时，由于需要利用数组下标和数组中字节数值转为对应记录的取值情况，故将这些分段的位图索引按记录顺序进行整合，用 Spark 中 RDD. collect 执行操作转为一个整体的位于本地的有序数组。对位图索引进行查询就是对这一内存中的数组进行遍历，内存中连续空间遍历的速度我们是可以接受的，故针对基数较少的列建立位图索引是有效的。位图索引在 Spark 中的操作如图 5 所示。

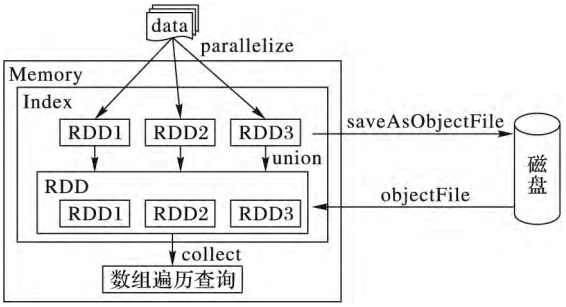

图 5　位图索引在 Spark 中操作过程

Fig. 5　Operation process of bitmap index in Spark

2.2　对基数较大但不涉及范围查询的列的索引

当 HBase 中某列的基数较大时,就可以使用 Hash 索引或树形索引。当该列不涉及范围查询时,即根据一个确定的值进行等值查询,选择能在 $O(1)$ 的时间复杂度下准确查找的 Hash 索引。

本文中对基数较大但不涉及范围查询的列构建链接桶哈希,其结构如图 6 所示。在构建 Hash 索引时,仍将数据分段,建立分段 Hash 索引,这样不仅能方便后面的并行化查找,同时可以根据分段的大小事先确定桶的数目,以防桶的数目过小带来查询效率下降以及桶的数目过大带来空间浪费的问题。为了减少哈希表所占用的内存空间,提高负载因子,通常的负载因子为 0.75,这里选用 0.8,即要对 N 条数据建立 Hash 索引,则先产生桶的数目为 $1.25*N$ 的哈希表。虽然提高负载因子会增加桶中链的长度从而增加查询数据的时间开销,但这是时间成本和空间成本上的一种折中,是可以接受的。

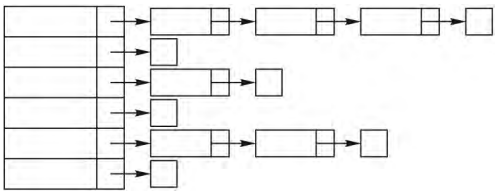

图 6　链接桶哈希结构

Fig. 6　Hash structure of link bucket

在查询时,对分段 Hash 索引进行并行的查找[14],这些索引转为 RDD 的形式存储在内存中,在硬盘上也有备份。利用 Spark 中 RDD.map 转化操作算子对每个分段 Hash 索引进行时间复杂度为 $O(1)$ 的快速查找。这样的查询速度非常理想,不会因数据量的增大产生巨大变化,这也是本文愿意提高负载因子的原因。索引在 Spark 中的操作如图 7 所示。

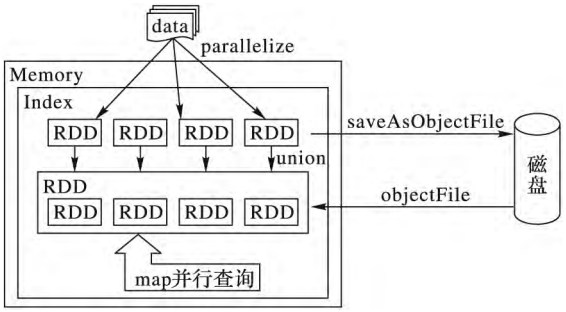

图 7　分段哈希索引在 Spark 中操作过程

Fig. 7　Operation process of segmented hash index in Spark

2.3　对基数较大且涉及范围查询的列的索引

2.3.1　BD 树索引的结构

Hash 索引不能进行范围查询,故面对 HBase 中某列基数较大且涉及范围查询时,本文采用 BD 树索引的思想,构建由多棵 BD 树索引组成的 BD 森林索引。其中 BD 树索引由 B + 树与哈希表结合得到,即索引的树结构是 B + 树,B + 树的叶子节点由 n 个哈希桶组成,且每个哈希桶的大小等于 CPU 缓存块的大小,故哈希桶中存放数据是固定的。同时为了能进行范围查找,各叶子节点按顺序连接起来,可以通过指针进行查询。图 8 是 BD 树索引的结构。

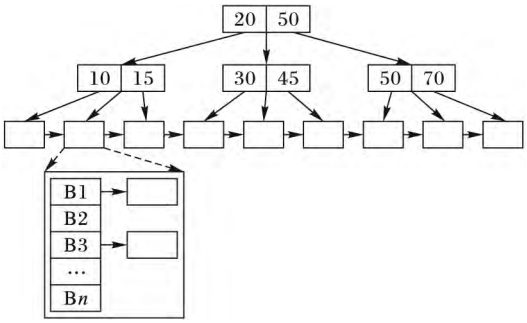

图 8　BD 树索引结构

Fig. 8　Index structure of BD tree

在构造 BD 树索引时,使用递归的方法将关键字不断插入。先查找关键字所在的哈希表,然后定位到其所在的桶中,若桶未满则插入关键字; 若桶已满则需分裂该节点,将原哈希表中所有关键字分配到两个新的哈希表中,分界可选原哈希表关键字的最大值和最小值的均值,同时对上层 B + 树进行修改,产生指向新的哈希表节点的指针。详细过程可以参考文献[11-12]。

在 BD 树索引的查询中,面对精确查询,根据关键字在 B + 树中找到对应的叶子节点,即哈希表,接着则是在哈希表中进行查询。BD 树索引关键在于可以进行范围查询,当查询的关键字范围为 $[k_s,k_e]$ 时,找到关键字 k_s 所在的哈希表记为 U,找到关键字 k_e 所在的哈希表记为 V,则 U、V 之间的哈希表的所有值(哈希表按顺序用指针串联在一起) 、U 中关键字大于 k_s 对应的值和 V 中关键字小于 k_e 对应的值是范围查询的结果。

2.3.2　BD 森树索引的结构

本文对基数较大且涉及范围查询的列构建 BD 森林索引,即多棵 BD 树索引,其结构如图 9 所示。

仍然将数据分段,对每一段建立 BD 树索引,并转为 RDD 的形式存储在内存中,利用 Spark 中操作对每个 BD 树索引进行查询,操作过程同图 7 的分段 Hash 索引。但这样在面对范围查询时,要对每棵 BD 树进行查询,最后将查询结果进行汇总,算法 1 描述了 BD 森林索引范围查询算法的过程。

算法 1　BD 森林索引范围查询算法。

输入　查询范围 $[k_s,k_e]$ 的关键字 k_s,k_e。

输出　行键集合 RQ。

```
RQ ← ∅                                  // 结果集合设为空
for each tree T in T_s do               // 遍历每棵 BD 树索引
  H_s ← T.GetHash(k_s)                  // 通过树形索引查找 k_s 所在的哈希表
  H_e ← T.GetHash(k_e)
  H ← H_s
  while next[H] ≠ H_s do
    H ← next[H]
```

```
    RQ ∪ allValue [H]      // 将哈希表中所有值并入结果集合中
  end while
  for each map M in H_s do                    // 遍历哈希表
    if key [M] > k_s then
                  // 当遍历映射的键大于范围开始的关键字时
      RQ ∪ value [M]              // 将映射的值并入结果集合中
    end if
  end for
  for each map M in H_e do
    if key [M] < k_e then
                  // 当遍历映射的键小于范围结束的关键字时
      RQ ∪ value [M]
    end if
  end for
end for
```

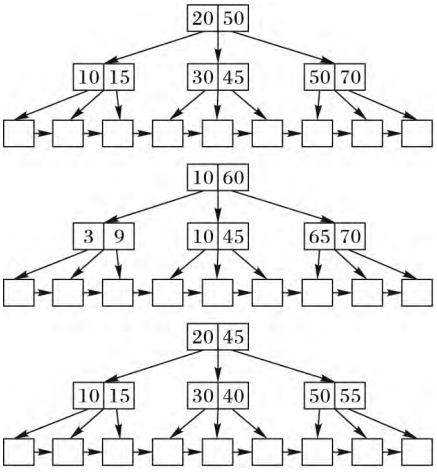

图 9 BD 森林索引结构

Fig. 9 Index structure of BD forest

构建 BD 森林索引是因为一次性将所有数据构造成 BD 树索引，在转化 RDD 时会因为树的递归构造次数[15]太多带来 StackOverflowError 问题，无法转为 RDD 存储在 Spark 内存环境中。构建 BD 森林索引，降低每棵树的高度，减少树的递归构造深度，将其成功转为 RDD。范围查询时虽然要对森林中的每棵树进行操作，但可以利用 Spark 中 RDD. map 转化操作算子进行并行操作，故效果是可以接受的。

3 实验与结果分析

3.1 实验环境与数据

本文的实验环境是在三台虚拟机下搭建的 HBase 集群与 Spark 集群，为了进行对比实验还搭建了 Solr 集群。每台虚拟机的环境是 1 核 CPU，2 GB 内存，60 GB 硬盘，操作系统是 CentOS7。

本文的实验数据是基于 RFID 电子车牌的交通数据，其来源于重庆市智能交通系统，其数据模型如表 1 所示。数据模型中各字段的含义分别是：记录在表中的 id 号、采集点名称、行驶方向、采集点的 IP 标识、车辆的电子车牌号、车辆通过时间、车辆类型、号牌的种类以及车辆使用性质。数据存入 HBase 中，用 01 到 99 之间的随机数加上采集点 IP 标识加上通行时间的月日时分秒构成每条原始记录的行键(Rowkey)，例如表 1 中的第一条记录的行键为“19101112010229012445”(产生的随机数是 19)。这样的行键具有唯一性和分散性。

在后面的实验中，主要对三种索引的检索效率进行了实验对比，期望基于这样的二级索引能在数据查询上带来效率的提高。另外对索引的大小也十分关注，期望索引所占空间合理，以便能在内存中存储，故在实验中对索引所占的空间大小也进行了实验测试。

表 1 原始数据模型

Tab. 1 Original data model

LL_id	rfid_name	direction	rfid_ip	EID	Passtime	car_type	use_type	property
1	菜袁路	菜园坝	10.11.12.1	3245	2016-02-29 01:24:45	K33	02	D
2	陈庹路	大渡口	10.11.13.30	1155	2016-02-29 01:24:49	K31	02	A
3	金开大道	人和	10.11.12.79	4879	2016-02-29 01:24:51	K33	02	D

3.2 结果分析

3.2.1 二级索引实验

在 HBase 中进行查询时通常需要利用行键，在不知道行键的情况下或通过过滤器扫描进行查找，但这样的查询速度非常慢。在本实验中分别从 1 000 万、2 000 万、3 000 万和 5 000 万数量级的数据中查找行键为“34101079500229080703”，内容为“金开大道，两路方向，10.10.79.50，1410080，2016-02-29 08:07:03，K33，02，D”的记录，分别使用行键查询和过滤器扫描两种方法进行查找，实验结果如图 10 所示。由图 10 可以看出，通过行键进行数据查询的时间是毫秒级的，而利用过滤器进行扫描查询的时间是秒级的，所以建立二级索引，在查询前事先获取行键是十分有必要的。后面将分别讨论前面的三种索引获取行键的性能，根据列的基数大小以及是否涉及范围查询选择位图索引、分段 Hash 索引和 BD 森林索引进行实验，并将实验结果与基于 Solr 建立的二级索引获取行键的性能进行对比。

3.2.2 位图索引实验

在实验数据中，车辆使用性质这一列有“非运营”“租赁”“警用”“工程抢险”等值，其基数为 16；车辆类型这一列有“大型普通客车”“大型卧铺客车”“轿车”“中型罐式货车”等值，其基数为 52。故车辆使用性质和车辆类型这两列都属于基数较小的列，在这两列上进行组合条件查询时，可以使用位图索引。例如，要查询车辆类型为“H37”，车辆使用性质为“K”的记录，即查找类型为轻型自卸货车的工程抢险车辆的过车记录。对这两列建立位图索引，将属性值为“H37”的位向量与属性值为“K”的位向量进行按位与操作，得到这一组合条件查询的位图索引。对索引的位向量进行遍历得到所需的行键。在 1 000 万、2 000 万、3 000 万和 5 000 万数量级的数据中利用位图索引查找类型为轻型自卸货车的工程抢险车辆的过车记录行键的时间和基于 Solr 获取行键的时间如图 11 所示。由图 11 可以发现，位图索引查找时间是毫秒级的且优于基于 Solr 的二级索引。得到这些行键后，就可以在 HBase 中快速查到相应的记录。

另外，位图索引所占的存储空间要远小于 Solr 中的索引，如图 12 所示，故位图索引适用于在内存中使用。

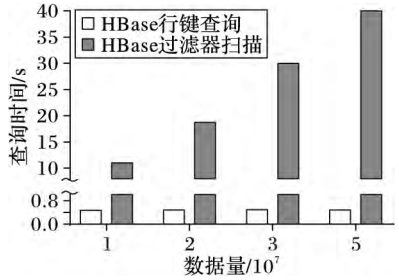

图 10　使用行键查询和过滤器扫描查询效率比较

Fig. 10　Query efficiency comparison of rowkey query and filter scan

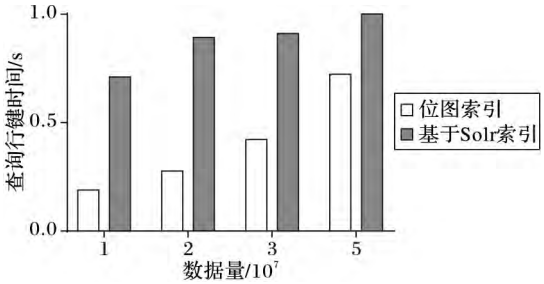

图 11　位图索引和基于 Solar 索引查询行键时间比较

Fig. 11　Time comparison of querying rowkey by bitmap index and Solar-based index

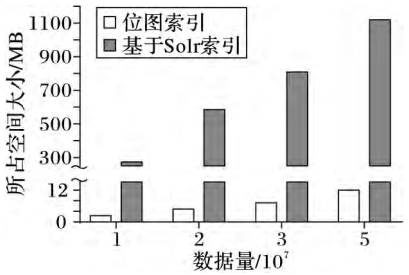

图 12　位图索引和基于 Solar 索引存储空间大小比较

Fig. 12　Storage space size comparison of bitmap index and Solar-based index

3.2.3　分段 Hash 索引实验

当要根据车辆的电子车牌号来查询表中的记录时，可以在该列建立分段 Hash 索引来获取相应的行键。在 1 000 万、2 000 万、3 000 万和 5 000 万数量级的数据中利用分段 Hash 索引获取电子车牌号为“1410080”的过车记录行键的时间和基于 Solr 获取行键的时间如图 13 所示，可以发现分段 Hash 索引的效果是较好的。由于使用了分段 Hash 索引进行了并行化的查找，故查找时间并没有因为数据量的增加而发生陡增，基本维持在 100 ms 左右获取行键，大大提高了查询效率。

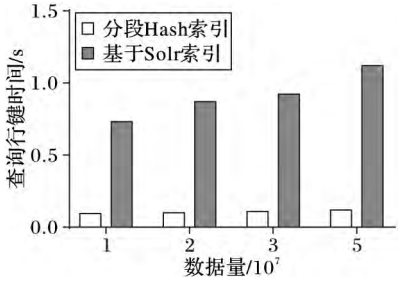

图 13　分段 Hash 索引和基于 Solar 索引查询行键时间比较

Fig. 13　Time comparison of querying rowkey by segment Hash index and Solar-based index

3.2.4　BD 森林索引实验

有时要对某辆车查询其在一段时间内的记录，这时涉及到时间的范围查询，故选择将电子车牌号与通行时间这两列结合在一起建立 BD 森林索引。在 1 000 万、2 000 万、3 000 万和 5 000 万数量级的数据中利用 BD 森林索引获取电子车牌号为“1410080”在“2016-02-29 08:00:00”至“2016-02-29 12:00:00”期间的过车记录行键的时间和基于 Solr 获取行键的时间如图 14 所示。由图 14 可以看出，当数据量增大时，可以适当提高哈希表中桶的个数来维持树的高度，但桶的个数过大也会影响范围查询中首尾两哈希表的查找性能。BD 森林索引由于数据结构的复杂性，故查询时间稍长但获取行键后进行查询的综合时间仍比通过 HBase 过滤器扫描的时间要短不少，获取行键的时间也少于基于 Solr 获取行键的时间。

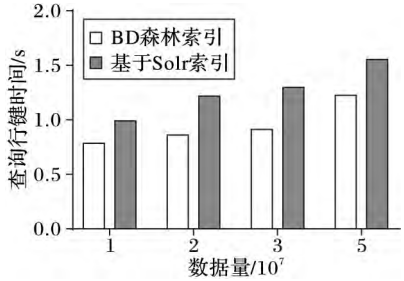

图 14　BD 森林索引和基于 Solar 索引查询行键时间比较

Fig. 14　Time comparison of querying rowkey by BD forest index and Solar-based index

3.2.5　二级索引所占空间实验

整个二级索引以 RDD 的形式存储在 Spark 构建的内存环境中，故期望索引能够在内存中存储。在 1 000 万、2 000 万、3 000 万和 5 000 万数量级的数据中建立整体的二级索引(包含关于车辆使用性质、车辆类型的位图索引、关于电子车牌号的分段 Hash 索引以及关于电子车牌号和通行时间的 BD 森林索引)的空间大小如图 15 所示。在当前的数据规模下索引可以在集群的 2 560 MB 执行内存中存储。

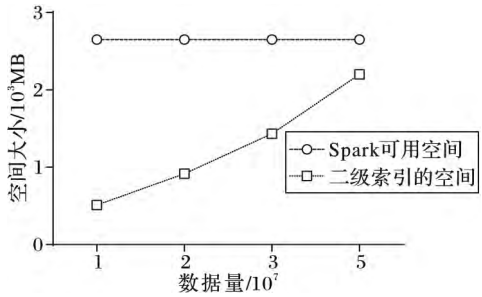

图 15　二级索引空间大小

Fig. 15　Space size of secondary index

4　结语

本文针对 HBase 无法在非行键的列上建立索引导致条件查询效率低下的问题建立了基于内存的二级索引。根据每列基数大小以及是否涉及范围查询构建不同类型的索引并进行改进，使其在 Spark 搭建的内存环境中进行高效的检索。实验结果表明这样的二级索引大大提高了查询效率，值得构建。对此下一步准备对索引的结构进行再次的改进，希望减小 Hash 索引的空间大小以及提高树形索引的并行度，并考虑分布式索引的改进，从而继续提高二级索引的整体性能并将其运用在实时数据处理中。

参考文献(References)

[1] 申德荣，于戈，王习特，等. 支持大数据管理的 NoSQL 系统研究综述[J]. 软件学报，2013，24(8)：1786－1803. (SHEN D R，YU G，WANG X T，et al. Survey on NoSQL for management of big data

[J]. Journal of Software, 2013, 24(8): 1786 - 1803.)
[2] 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(1): 140 - 153. (GE W, LUO S M, ZHOU W H, et al. HiBase: a hierarchical indexing mechanism and system for efficient HBase query [J]. Chinese Journal of Computers, 2016(1): 140 - 153.)
[3] MANGHI P, ARTINI M, BARDI A, et al. High-performance annotation tagging over solr full-text indexes [J]. Information Technology and Libraries, 2014, 33(3): 22 - 44.
[4] 李永春,丁华福. Lucene的全文检索的研究与应用[J]. 计算机技术与发展,2010,20(2): 12 - 15. (LI Y C, DING H F. Research and application of full text search based on Lucene [J]. Computer Technology and Development, 2010, 20(2): 12 - 15.)
[5] ZOU Y Q, LIU J, WANG S C, et al. CCIndex: a complemental clustering index on distributed ordered tables for multi-dimensional range queries [C]// Proceedings of the 2010 IFIP International Conference on Network and Parallel Computing, LNCS 6289. Berlin: Springer, 2010: 247 - 261.
[6] LU H J, NG Y Y, TIAN Z P. T-Tree or B-Tree: main memory database index structure revisited [C]// Proceedings of the 2000 11th Australasian Database Conference. Piscataway, NJ: IEEE, 2000: 65 - 73.
[7] BOTELHO F C, KOHAYAKAWA Y, ZIVIANI N. A practical minimal perfect hashing method [C]// Proceedings of the 2005 4th International Conference on Experimental and Efficient Algorithms, LNCS 3503. Berlin: Springer, 2005: 488 - 500.
[8] AILARNAKI A G, DEWITT D J, HILL M D, et al. DBMSs on modern processors: where does time go? [EB/OL]. [2017-10-16]. http://pdfs.semanticscholar.org/16c6/f35d6ba45134743142 2e5c37590170198949.pdf.
[9] RAO J, ROSS K A. Making B+-trees cache conscious in main memory [J]. ACM Sigmod Record, 2000, 29(2): 475 - 486.
[10] ROSS K A. Efficient hash probes on modern processors [C]// Proceedings of the 2007 IEEE 23rd International Conference on Data Engineering. Piscataway, NJ: IEEE, 2007: 1297 - 1301.
[11] CUI B, OOI B C, SU J, et al. Main memory indexing: the case for BD-Tree [J]. IEEE Transactions on Knowledge and Data Engineering, 2004, 16(7): 870 - 874.
[12] 肖富平,罗军. HT树:缓存敏感的内存数据库索引[J]. 计算机工程,2009,35(16): 68 - 70. (XIAO F P, LUO J. Hash & Tree Tree: cache conscious index of memory database [J]. Computer Engineering, 2009, 35(16): 68 - 70.)
[13] 闫梦洁,罗军,刘建英,等. IABS: 一个基于Spark的Apriori改进算法[J]. 计算机应用研究,2017,34(8): 2274 - 2277. (YAN M J, LUO J, LIU J Y, et al. IABS: parallel improved Apriori algorithm based on Spark [J]. Application Research of Computers, 2017, 34(8): 2274 - 2277.)
[14] CHOU J, HOWISON M, AUSTIN B, et al. Parallel index and query for large scale data analysis [C]// Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. New York, NY: ACM, 2011: Article No. 30.
[15] DUAN M X, LI K L, TANG Z, et al. Selection and replacement algorithms for memory performance improvement in Spark [J]. Concurrency and Computation: Practice and Experience, 2016, 28(8): 2473 - 2486.

This work is partially supported by the National High Technology R&D Program of China (2015AA015308), the National Key R&D Program of China (2016YFC0801707), the Key Project of Chongqing Application Development Plan (cstc2014yykfB30003).

CUI Chen, born in 1994, M. S. candidate. His research interests include intelligent transportation system, big data.

ZHENG Linjiang, born in 1983, Ph. D., associate professor. His research interests include intelligent transportation system, big data.

HAN Fengping, born in 1983, M. S., engineer. Her research interests include traffic engineering.

HE Mujun, born in 1982, Ph. D. candidate. His research interests include intelligent transportation system, big data.

(上接第1583页)

[9] 李萍萍. 时空数据库中高维数据的降维方法[D]. 哈尔滨: 哈尔滨理工大学,2009: 7 - 18. (LI P P. Dimensionality reduction of higher dimensional data in spatio-temporal databases [D]. Harbin: Harbin University of Science and Technology, 2009: 7 - 18.)
[10] KARYPIS G, KUMAR V. Analysis of multilevel graph partitioning [C]// Proceedings of the 1995 ACM/IEEE Conference on Supercomputing. New York: ACM, 1995: Article No. 29.
[11] ZHONG R C, LI G L, TAN K-L, et al. G-Tree: an efficient and scalable index for spatial search on road networks [J]. IEEE Transactions on Knowledge & Data Engineering, 2015, 27(8): 2175 - 2189.
[12] SAHU P K, MANNA K, SHAH N, et al. Extending Kernighan-Lin partitioning heuristic for application mapping onto Network-on-Chip [J]. Journal of Systems Architecture, 2014, 60(7): 562 - 578.
[13] DÜNTGEN C, BEHR T, GÜTING R H. BerlinMOD: a benchmark for moving object databases [J]. VLDB Journal, 2009, 18(6): 1335 - 1368.
[14] KOMAI Y, NGUYEN D H, HARA T, et al. KNN search utilizing index of the minimum road travel time in time-dependent road networks [C]// Proceedings of the 2014 IEEE 33rd International Symposium on Reliable Distributed Systems Workshops. Piscataway, NJ: IEEE, 2014: 131 - 137.
[15] KE S N, GONG J, LI S N, et al. A hybrid spatio-temporal data indexing method for trajectory databases [J]. Sensors, 2014, 14(7): 12990 - 13005.

This work is partially supported by the National Natural Science Foundation of China (61602151, 61370091), the National Key Research and Development Program of China (2017YFC0405806), the Key R & D Program of Jiangsu Province (Social Development) Project (BE2015707).

FENG Jun, born in 1969, Ph. D., professor. Her research interests include spatio-temporal data management, intelligent data processing, data mining, water conservancy informatization.

LI Dingsheng, born in 1993, M. S. candidate. His research interests include spatiotemporal data index, intelligent transport system, big data Hadoop.

LU Jiamin, born in 1983, Ph. D., lecturer. His research interests include mobile object data management, distributed data processing, water conservancy informatization.

ZHANG Lixia, born in 1993, M. S. candidate. His research interests include spatiotemporal data index, intelligent transport system, big data Hadoop.