基于HBase的支持频繁更新与多用户并发的R树

王波涛 梁 伟 赵凯利 钟汉辉 张玉圻
(东北大学计算机科学与工程学院 沈阳 110169)

摘 要 基于位置服务的应用已经进入大数据时代,传统基于位置服务的技术面临系统扩展性、性能等方面的挑战。云计算技术是大数据处理的基础,索引是优化查询的重要手段。尽管目前已存在大量的研究成果,但尚未有HBase上的支持频繁更新与多用户并发的R树索引。针对移动对象索引的频繁更新与多用户并发的需求,文中提出了基于HBase的支持频繁更新与多用户并发的R树索引,它只索引包含移动对象的网格,避免了频繁更新问题;进一步基于HBase的数据行与数据分区的组织与读写特性,对R树的节点进行重组,并对网格Z-order编码,从而减少了对HBase的读写操作,提高了查询效率;最后提出了基于ZooKeeper分布式读写锁的优化策略,提高了索引的吞吐量。实验结果表明,与网格索引相比,在数据非均匀的情况下,所提策略的查询吞吐量提高了25%～50%,更新吞吐量约在同一数量级;与分布式共享锁索引相比,分布式读写锁索引的吞吐量提高了近40%。

关键词 基于位置服务,R树,移动对象索引,HBase
中图法分类号 TP315 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2018.07.007

R-tree for Frequent Updates and Multi-user Concurrent Accesses Based on HBase

WANG Bo-tao LIANG Wei ZHAO Kai-li ZHONG Han-hui ZHANG Yu-qi
(School of Computer Science and Engineering,Northeastern University,Shenyang 110169,China)

Abstract Application based on location based service (LBS) has entered the era of big data. Traditional location based service techniques face new challenges such as scalability,performance,etc. Cloud computing technology is the basis of big data processing and index is an important way to optimize query. Although there exist a large number of research results,as far as we know,there is no R-tree index which supports frequent updates and multi-user concurrent accesses based on HBase. According to the above characteristics of moving objects,this paper proposed a new R-tree index which supports frequent updates and multi-user concurrent accesses based on HBase. In this new index,the R-tree only indexes the grid which contains the moving object to avoide the problem of frequent updating effectively. Furthermore,based on the organization of HBase data rows and I/O characteristics of data partitions,this paper reorganized the nodes and encoded the grid cells with z-order,which reduce the read and write operations of HBase and improve the query efficiency. Finally,it proposed an optimization strategy for distributed read and write locks based on Zookeeper,which improves the throughput of new indexes. The experimental results show that the query throughput of the proposed strategy is improved by 25%～50% and the update throughput is about the same level in the case of uneven data compared with the grid index. Compared with the index using distri-buted shared locks,the query throughput of the index using distributed read and write locks is increased by nearly 40%.

Keywords Location based service,R-tree,Moving object index,HBase

1 引言

移动大数据的高效处理是大数据时代基于位置服务(Location Based Service,LBS)研究中的关键问题之一。针对移动大数据的处理,传统存储与处理数据的技术手段遇到了瓶颈[1],海量数据与现有系统的数据处理能力之间存在着鸿沟[2]。

云计算具有扩展性强、处理速度快、可靠性高等优点,已经成为解决海量数据问题的有效方法[3]。移动对象索引是LBS应用中的关键技术之一,面临着高数据更新频率和高并

到稿日期:2017-07-15 返修日期:2017-08-20 本文受国家自然科学基金:面向动态位置服务的移动查询处理与优化技术(61173030)资助。
王波涛(1968－),男,博士,教授,CCF会员,主要研究领域为隐私保护、大数据、云计算、基于位置服务,E-mail:wangbotao@ise.neu.edu.cn(通信作者);**梁 伟**(1993－),女,硕士生,主要研究领域为机器学习、大数据,E-mail:1713900300@qq.com;**赵凯利**(1992－),女,硕士生,主要研究领域为移动大数据、时空索引,E-mail:1335888327@qq.com;**钟汉辉**(1993－),男,硕士生,主要研究领域为大数据、云计算,E-mail:1596694976@qq.com;**张玉圻**(1994－),女,硕士生,主要研究领域为大数据、云计算,E-mail:zhangyuqi1994@163.com。

发等多种挑战。尽管已经存在大量的相关研究成果,但由于体系架构与数据组织的差别,移动对象索引需要在云环境下重新实现。

HBase[4]是一个高可靠性、高性能、面向列、可伸缩的分布式存储系统,可以有效地支持大规模数据高吞吐的随机读写。许多传统的索引结构已在 HBase 上得到实现。

网格索引 Grid 简单易用,但在查询和更新数据量不断增加的过程中,查询点的位置不总是均匀分布的。此时,网格索引的性能急剧下降,查询效率降低。R 索引实现复杂,但在不均匀分布环境下的查询效率较高。针对 LBS 应用的多用户与位置更新频繁的需求,本文分析了网格索引、R 树索引与 HBase 的数据行的组织与读写特性,提出了一种结合 R 树索引和网格索引的节点重组 R 树索引以支持频繁更新,同时设计了一种基于 ZooKeeper 的优化策略来支持多用户并发。

尽管已存在大量的与 HBase 上的 R 树相关的研究成果[5-11],但还未有同时支持频繁更新与多用户并发的基于 Hbase 的 R 树索引。与现有的工作相比,本文的主要贡献如下:

1)针对读写效率问题,基于 HBase 的读写特性对 R 树节点进行重组,在一个 HBase 数据行中保存多个 R 树节点;进一步对网格节点进行 Z-order 编码,减少了对 HBase 的读写操作次数,提高了查询效率;给出了相应的数据结构与算法。

2)针对多用户并发问题,提出了基于 ZooKeeper 的读写锁优化策略,给出了相应的数据结构与流程。

3)在数据分布不均匀的情况下,验证了本文提案相比网格索引结构的有效性。

本文第 2 节对现有云平台上使用 HBase 和 R 树作为移动对象索引的相关研究成果做了简单介绍;第 3 节对提出的基于 HBase 的支持频繁更新的节点重组 R 树做了详细说明,并提供了相关查询和更新算法以及基于 Z-order 编码的优化策略;第 4 节提出了基于 ZooKeeper 分布式读写锁的优化策略;第 5 节对重组 R 树的性能进行分析;第 6 节基于仿真数据,测试了重组 R 树的性能;最后总结全文。

2　相关工作

随着相关技术的发展,云环境下的移动对象索引已经有了很多研究成果,下面对基于 HBase 的 R 树索引的相关技术进行说明。

Zhou 等[5]提出了分布式环境下的大数据管理多维索引 ZPR-tree,它将空间静态分区,用四分树索引针对每个四分树索引节点建立一棵 R 树,并利用 Z-order 曲线消除 R 树节点 MBR 的叠加。Takasu 等[6]提出了一种针对地理空间数据库的高效分布式索引,使用 Geohash 作为 HBase 的行键,R 树为二级索引。Zhou 等[7]提出了针对大量数据的分布式多维空间索引 ABR-Tree,根据 AB-Tree 中的递增属性将所有的数据划分成不同的间隔,并针对不同的间隔建立 R^*-Tree 来索引每个数据的其余属性,全局索引和本地索引分层存储在云数据库上。Huang 等[8]提出了 R-HBase,它具有索引层与存储层两层索引结构,存储层以 HBase 存储,支持高速的数据存取,索引层采用 R 树索引。进一步,Huang 等[9]提出支持高并发多用户的多维海量数据源上的样本提取方法,并提出基于 R 树与 HBase 的索引 HMVR-tree,该索引提供同步机制,可支持用户的并发读写访问。上述索引结构主要用于更新不频繁的海量多维空间数据查询,并不直接使用移动对象。

Xia 等[10]针对移动对象的过去、现在与未来建立了 PIPCF 索引。与 ZPR-tree[5]相似,它先将空间静态分区,并用四分树索引,而后针对每个四分树索引节点建一棵 R 树。该索引并没有对 R 树进行优化,且没有考虑多用户的并发操作问题。Du 等[11]首先将移动对象按时间段分组,然后基于时间与空间信息建立 R 树索引,并将 Hilbert 线性化应用于 R 树目标叶节点的映射中,以形成 HBase 的聚集特性。鉴于上述索引都需要对 HBase 进行频繁的访问,我们考虑将 R 树的多个节点存储在 HBase 的一行中,以减少 HBase 的访问次数,提高查询效率。

综上可知,目前还不存在同时支持频繁更新与多用户并发的基于 HBase 的 R 树索引。

3　基于 HBase 的支持频繁更新的节点重组 R 树

3.1　预备知识

HBase[4]是一个面向列的基于 HDFS 的分布式存储系统,具有高性能、高可靠性、可伸缩、面向列等特点,使用 ZooKeeper 作为协同服务。HBase 可以有效地支持大规模数据的高吞吐随机读写,许多传统的索引结构已在 HBase 上得到了实现。

HBase 的系统架构如图 1 所示,主要由 Client,ZooKeeper,HMaster 和 HRegionServer 这 4 个组件构成[12]。

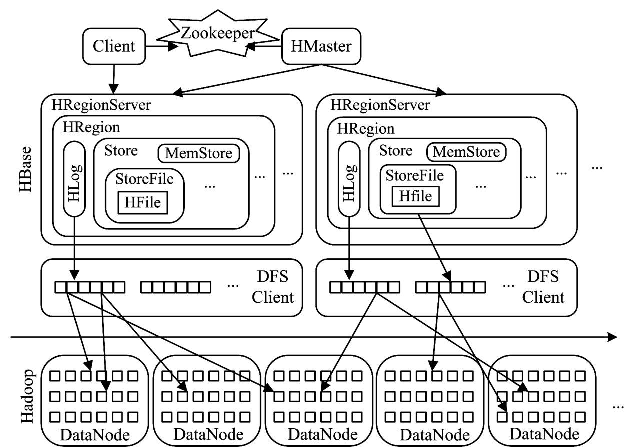

图 1　HBase 系统架构图

Fig. 1　System architecture of HBase

Client 通过 HBase 的 RPC 通信机制与 HMaster 和 HRegionServer 通信,其中,Client 与 HMaster 通信以处理管理类操作,Client 与 HRegionServer 通信以处理数据读写类操作。ZooKeeper 中存储了-ROOT-表和 HMaster 的地址,而 HRegionServer 也会将自己注册到 ZooKeeper 中,这样

HMaster 就能实时监控每个 HRegionServer 的健康状态。HBase 能启动多个 HMaster,通过 ZooKeeper 的领导者选举机制保证总会有一个 Master 在运行,因此成功避免了 HMaster 的单点故障问题。在功能上,HMaster 主要负责管理 Table 和 Region。HRegionServer 主要负责用户 I/O 请求的响应并读写 HDFS 文件系统中的数据,是 HBase 最核心的模块。HRegionServer 管理一系列的 HRegion 对象,每个 HRegion 与 Table 中的一个 Region 相对应,一个 HRegion 包含多个 HStore;每个 HStore 与 Table 中的一个 Column Family 相对应,每个 Column Family 实际上是一个集中的存储单元。

3.2 基于 HBase 的支持频繁更新的节点重组 R 树

3.2.1 基本思想

本索引的基本思想包含如下两个方面。

1)在用户数量不断增加的过程中,空间数据的聚集特性愈加明显,网格索引出现大量不包含移动对象的空网格。通过网格索引对地理空间进行划分非常简便,但在数据分布不均匀的情况下,查询效率较低。R 树索引实现复杂,但在不均匀分布环境下的查询效率较高。文中综合考虑网格索引和 R 树索引的优点,在 HBase 上设计了一种混合索引结构,即移动对象保存在网格索引中,用 R 树来索引网格单元。

2)如第 2 节所述,尽管目前已有大量的基于 HBase 的 R 树索引的研究成果,但在存储时,由于 HBase 数据表中的每一行存储 R 树的一个节点,当对索引进行频繁访问时,需要对 HBase 进行频繁访问。而 HBase 是一个面向列的分布式存储系统,对于相同数据量的读取,Scan 比多个 Get 更快。基于上述特点,本文重组 R 树索引,在 HBase 数据表中的每一行存储 R 树的当前节点及其孩子节点,这样只需访问一次 HBase 就可以获得这一行所有的 R 树节点,而不用逐个从 HBase 中读取,降低了磁盘访问的代价。

3.2.2 索引结构

索引结构分为两层:底层为网格索引,上层为 R 树索引。这样,构建好的上层的 R 树只索引包含移动对象的网格,解决了在网格索引中进行查询时须额外访问空网格的问题。此外,由于上层的 R 树索引的是不重叠的网格,在一定程度上减少了 R 树中的区域重叠。在自底向上划分 R 树的 MBR 时,将地理位置相邻的网格包含到相同或者邻近的 MBR 中,当 MBR 所包含的网格中的对象未移出该 MBR 时,只需更新其对应的叶子节点,降低了 R 树的更新代价。

图 2 给出了具体的实例,横轴和纵轴都被均分为 8 等份,这样就划分成了 64 个大小相同的网格。然后按照从左到右、由上到下的顺序从 1 开始对这 64 个网格进行编号,每个网格的编号都是唯一的。这 64 个网格各对应地存储着一个索引项,把每个移动对象添加到对应的索引项里。网格 3,20,62 和 63 中有 1 个移动对象;网格 2,61 和 64 中有 2 个移动对象;网格 21 中有 3 个移动对象;其他的 56 个网格均是空网格,不包含移动对象。然后,把这 8 个包含移动对象的网格插入到 R 树中。网格 2 和网格 3 位置最近,R3 恰好包含了这两个网格,是它们的 MBR。同理,R4 恰好包含了网格 20 和 21,R5 恰好包含了网格 61 和 62,R6 恰好包含了网格 63 和 64。接着,继续向上层构建 R 树,得到了 R1 和 R2 这两个 MBR 以及它们共同构成的 MBR。至此,便构建了一个指向图 2 中所有移动对象的基于网格和 R 树的两层索引结构。

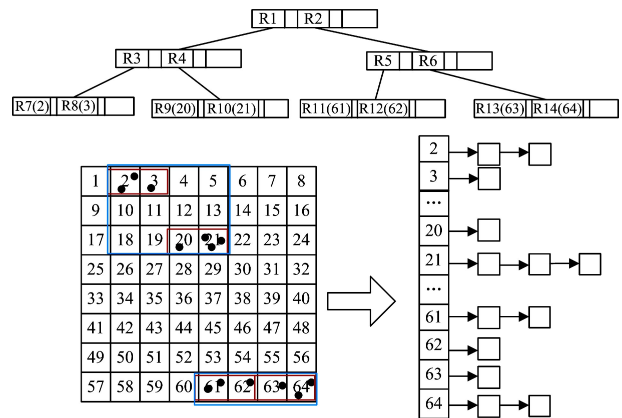

图 2 索引结构示意图

Fig. 2 Structure of index

3.2.3 数据结构

本文的数据存储采用的是分布式数据库 HBase,它是一个典型的 key-value 系统。HBase 把数据存储在表中,表中又包含许多行和列。作为一个 NoSQL 数据库,HBase 也是通过主键 RowKey 来检索记录。HBase 表中的列都属于一个列族(Column Family),并且列族是表模式的一部分,是基本的存储单元。

HBase 逻辑上以单元格的形式来存储数据,实际上则是以线性方式进行物理存储。如图 3 所示,用户把一个单元格的数据存放到一个表中,HBase 在底层实际存储时却是根据列族以线性方式来存储单元格的内容。

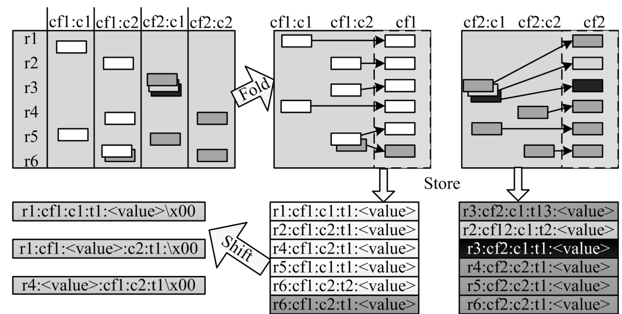

图 3 HBase 中逻辑记录到物理存储的转换

Fig. 3 Mapping from logical record to physical storage in HBase

根据优化索引的研究目的,本文用到的 HBase 数据库表包括 ObjectCell 表、HashCellTable 表和 RtreeHbase 表。

1)ObjectCell 表:存储网格内包含的移动对象;

2)HashCellTable 表:记录移动对象所在的网格;

3)RtreeHbase 表:存储 R 树索引的信息。

ObjectCell 表(见表 1)用来记录在底层的网格索引中,每个网格所包含的移动对象及其位置信息。在进行网格划分时,每个网格的编号 ID 是唯一的,使用网格的 ID 作为行键。ObjectCell 表只有一个列族 Object,用来存储移动对象的信息。Object 下的列是一系列动态列,列名为各个移动对象的 UserID,值是各个移动对象的位置。

表 1 ObjectCell 表的结构

Table 1 Structure of ObjectCell

类型	表中名称	说明
RowKey	CellID	网格在 HBase 中的行键
Column Family	Object	存储移动对象的信息

当移动对象进行位置更新操作时,经常会从一个网格更新到另外一个网格,此时必须记录移动对象在哪个网格。为此,本文创建了 HashCellTable 表(见表 2)来存储移动对象所在网格的 ID。HashCellTable 表使用移动对象的 UserID 作为行键,UserID 是移动对象的唯一标识。HashCellTable 表只有一个列族 Cell,用来存储移动对象所在网格的 ID。Cell 下只有一个固定列,列名为 cellID,值为具体的移动对象所在网格的 ID。

表 2 HashCellTable 表的结构

Table 2 Structure of HashCellTable

类型	表中名称	说明
RowKey	UserID	移动对象在 HBase 中的行键
Column Family	Cell	存储移动对象所在网格的 ID

RtreeHbase 表(见表 3)用来存储整个 R 树索引结构的信息,主要包括每个节点自身的信息以及指向其孩子节点的指针。在 R 树节点分裂时,会给节点生成一个唯一的 RtreeID。RtreeHbase 表使用 RtreeID 作为行键,并且只有一个列族 Data,用于存储编号为此 RtreeID 的节点及其孩子节点的相关信息。其列名是 R 树节点的 RtreeID,值为对应的 R 树节点的具体信息。

表 3 RtreeHbase 表的结构

Table 3 Structure of RtreeHbase

类型	表中名称	说明
RowKey	RtreeID	R 树节点在 HBase 中的行键
Column Family	Data	存储 R 树节点的信息

3.2.4 索引存储

在使用 R 树索引时,为了提高加载索引的效率,通常都选择在内存中建立 R 树索引,这样一旦服务器关闭,R 树空间索引就会随之消失。当用户再次进行查询时,需要重新构建 R 树索引,重新组织数据。随着用户查询次数的不断增多,重复开销必然越来越大,因此使 R 树持久化显得非常必要;同时,由于网格索引需要记录索引项,网格索引必须持久化。本文的网格索引和 R 树索引均存储在 HBase 上。

将网格索引和 R 树索引存储在 HBase 上的关键是 HBase 数据表的设计,首先需要确定具体需要哪些 HBase 数据表。由于本文的索引是一个结合网格索引和 R 树索引的双层索引结构,R 树索引的是网格,最终指向网格的索引项,因此在索引存储时将网格索引和 R 树索引分别存储在一个 HBase 数据表中,对应的索引表为 ObjectCell 表和 RtreeHbase 表。此外,当移动对象更新位置信息时,可能会从当前所在的网格移动到另一个网格,导致索引存在更新。因此,需要记录移动对象在哪个网格,此时便需要一个 HBase 数据表 HashCellTable 来保存移动对象和网格的对应关系。

在将底层网格索引存储到 ObjectCell 表的过程中,对网格进行编码,指定一个唯一的 ID 作为 ObjectCell 表的行键,列族 Object 存储移动对象的 ID 以及经纬度坐标,划分后的每个网格对应 ObjectCell 表中的一行。在 HashCellTable 表中,以移动对象的 UserID 为其行键,列族 Cell 存储移动对象所在网格的 ID,每个移动对象对应 HashCellTable 表中的一行。

3.2.5 索引节点的重组

R 树节点重组的基本思路是:R 树的奇数层(叶子节点为第 0 层)节点对应 HBase 的一行,并保存其孩子节点的信息。root 节点单独占一行。

在存储上层 R 树索引到 RtreeHbase 表的过程中,给每个 R 树节点生成一个唯一的 ID 作为 RtreeHbase 表,列族 Data 存储与这一行对应的 R 树节点的信息。R 树节点的 ID 是顺序递增的,起始值为 1,每当因为 R 树节点分裂而增加一个节点时,这个新节点的 ID 就为当前最大的 ID 值加 1。每个 R 树节点是具体的类的实例,包含节点的层数、节点的 MBR、其孩子节点的 ID 列表以及孩子节点对应的 MBR 的列表。R 树的叶子节点的层数为 0,R 树的索引节点的层数逐层向上加 1。在 R 树节点的存储过程中,层数为奇数层的节点对应 RtreeHbase 表中的一行,行键为此节点的 ID。列族 Data 下的列名是本节点及其所有孩子节点的 ID,对应的列值为表示这些节点的序列化后的实例对象。

R 树是自底向上分层的,根节点总是最后一层节点,无论根节点的高度是奇数还是偶数,根节点都对应 RtreeHbase 表中的一行。

每次查询都是从根节点开始向下进行,因此需要记录 R 树的根节点。在 RtreeHbase 表中额外有一行来记录 R 树的根节点 ID 和相关配置信息,其中的配置信息包括索引节点容量、叶子节点容量、维度、节点最少因子、节点最多因子、节点重新插入因子、节点分裂分布因子等。

结合网格索引和 R 树索引的基于 HBase 的支持频繁更新的节点重组 R 树的具体存储结构如 3.2.6 节图 4 所示。节点 R4 位于第一层,是奇数层;R10 与 R12 是其子节点,这 3 个节点存储在同一 HBase 行。

3.2.6 查询算法

算法 1 给出了通过 R 树索引获取查询范围内的网格 ID 列表的伪代码。其中,第 1 行和第 33 行分别对 R 树的根节点加读锁和解锁,防止在遍历 R 树索引时有进程更新索引。第 2 行初始化了保存查询结果的集合 grids、保存节点实例的栈 st 以及保存从 HBase 中读取的 R 树节点的集合 map,并且 map 的初始值为以根节点 root 为行键的 RtreeHbase 表中的所有列。第 4－6 行判断根节点是否有孩子节点并且其 MBR 是否与查询范围相交,如果满足这两者,将根节点压入栈 st。第 7－31 行是在栈 st 不为空时,进行循环查询操作。当栈 st 不为空时,在第 8 行栈 st 取出顶端节点 n;第 9－14 行处理了节点 n 是叶子节点的情况,此时遍历节点 n 的孩子节点,如果其 MBR 与查询范围相交,便将它的 ID 添加到结果集合中;第 15－30 行处理了节点 n 是索引节点的情况,此时先遍历节点 n 的孩子节点,对于其 MBR 与查询范围相交的节点,继续判断节点 n 的层数是否是偶数。

算法 1 范围查询

输入:查询范围

输出:查询范围内网格 ID 的列表

```
1.对 R 树根节点加读锁;
2.初始化 Set〈String〉集合 grids,Stack〈Node〉栈 st 和 HashMap〈Long,byte[]〉map;
3.将 map 中的 root 节点反序列化为节点实例;
4.IF(root 节点有孩子节点 &&root 节点的 MBR 与查询范围相交)
5.    root 入栈 st;
6.END IF
7.WHILE(st 不为空)
8.    栈 st 弹出节点 n;
9.    IF(n 是叶子节点)
10.       FOR(n 的每个孩子节点)
11.          IF(MBR 与查询范围相交)
12.             添加节点 ID 到 grids 中;
13.          END IF
14.       END FOR
15.    ELSE
16.       FOR(n 的每个孩子节点)
17.          IF(MBR 与查询范围相交)
18.             IF(n 的层数为偶数)
19.                获取以当前节点 ID 为行键的 RtreeHbase 表中的所有列;
20.                反序列化列中当前节点为节点实例;
21.                当前节点实例入栈 st;
22.                添加这些节点到 map 中;
23.             ELSE
24.                反序列化 map 中当前节点为节点实例;
25.                当前节点实例入栈 st;
26.             END IF
27.          END IF
28.       从 map 中移除当前节点
29.     END FOR
30.  END IF
31.END WHILE
32.返回 grids;
33.对 R 树根节点解锁;
```

若节点 n 的层数是偶数,则它的孩子节点层数一定是奇数,节点 ID 是 RtreeHbase 表中的行键,可以从 RtreeHbase 表中获取它及其所有的孩子节点。因此,在第 19 行获取了以当前节点 ID 为行键的 RtreeHbase 表中的所有列,然后在第 20-21 行反序列化,将结果列中的当前节点设为节点实例并添加到栈 st 中,之后在第 22 行将列中的节点添加到 map 中。如果节点 n 的层数是奇数,则其孩子节点已经从 RtreeHbase 表读取到 map 中。在第 24 行直接从 map 中获取当前节点并将其反序列化为节点实例,第 25 行将其添加节点到栈 st 中。最后,第 28 行将遍历过的节点 n 的孩子节点从 map 中移除。

图 4 展示了该范围查询算法的一个实例,图中的黑色框代表查询范围。查询的具体流程为:首先获取查询框左下角和右上角的点的坐标,然后根据查询区域从 1 号根节点向下查询,从 RtreeHbase 表中获取行键为 1 的那一行的 Data 列族下的所有列。由于这一行只有根节点,反序列化 R1,获取它的孩子节点信息,把孩子节点的 MBR 与查询范围进行比较,发现 4 号节点的 MBR 包含查询范围。接下来获取 RtreeHbase 表中行键为 4 的所有列,由于列数大于 1,这一行存储了 4 号节点以及它所有的孩子节点对象。比较 4 号节点的孩子节点的 MBR 是否与查询范围相交,得到包含查询范围的 12 号节点的 ID。根据节点 ID,可从已得到的列表中直接获得 12 号节点对象。由于 12 号节点的层数为 0,因此其为叶子节点。而叶子节点对象的孩子节点列表存储的是网格的范围以及网格 ID,把 R 树的 12 号节点存储的孩子节点 MBR 与查询范围进行比较,得到了 62,63 和 64 这 3 个网格 ID。之后从网格索引表 ObjectCell 中获取 62,63 和 64 号网格内的移动对象的坐标,再分别将获得的每个对象的坐标与查询范围进行比较,在查询范围内的这些移动对象就是用户想要得到的查询结果,从而最终得到了移动对象 O_{n-3},O_{n-2},O_{n-1} 和 O_n。

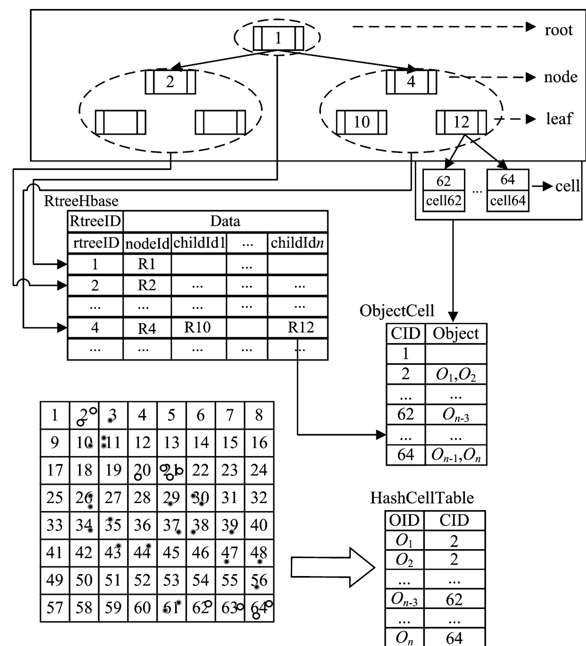

图 4 索引存储示意图

Fig. 4 Storage of index

3.2.7 索引的维护操作

索引的维护操作包含插入、删除与更新,其过程与传统的网格索引和 R 树索引相似,此处不再赘述。需要注意的内容如下:

1)如 3.2.2 节所述,本文的 R 树只索引网格,因此索引维护操作的数据单元是网格而不是移动对象。当一个对象移动但没有离开原来的网格时,只需改变对象本身的信息,不需要对 R 树索引进行修改,因此可以大大提升更新的吞吐率。

2)如 3.2.5 节所述,节点重组是对存储在 HBase 上的 R 树节点的重组,遵循 R 树的奇数层(叶子节点为第 0 层)节点对应 HBase 的一行,并保存其孩子节点的信息,root 节点不论奇偶都单独占一行的原则。当 R 树的内容发生变化时,只需基于上述原则,对需要维护的相关节点的内容进行 HBase 行读写操作即可。其中,R 树节点的访问方式与算法 1 中的第 18-26 行相似,其他操作与传统的 R 树维护算法一致,这里不再细述。

当 R 树的高度发生变化时,由于叶子节点为第 0 层,因此不需要对所有的 HBase 行的内容进行调整,只需按上述方式修改相关的节点即可。

3.3　基于 Z-order 编码的优化策略

3.3.1　问题描述

如 3.2.4 节所述,在建立网格索引时,需要对网格进行编号,给每个网格指定一个唯一的 ID。最简单直观的方式便是根据横坐标和纵坐标顺序地对网格编码,在横轴从左向右、纵轴从上到下地从 0 开始编号,如图 5 所示,将空间划分为 16 个网格。网格在 HBase 中存储时,按照网格的编号由小到大进行存储,获取指定网格时也是按照这个顺序进行查找。前文对每个网格分别进行查询,这样在获取网格时都是从 0 号开始扫描,造成了额外的访问代价。如图 5 所示,想要获取查询范围框中的网格 8,9,12 和 13,每次都要从 0 号网格向下扫描,一直扫描到所要访问的网格,此过程存在对 0－7 号网格的重复扫描,造成了额外的开销。针对这个问题,直接获取网格列表中的所有网格,这样只需对 0－13 号网格扫描一遍,减小了额外的扫描代价。但是,顺序地对网格编码不能保证空间相邻的网格在 HBase 中的存储也相邻,在进行范围查询,从 HBase 中获取网格列表时,会扫描多余的网格。在获取包含网格 8,9,12 和 13 的列表时,必须扫描网格 10 和 11。因此,在对网格编码的过程中需要保证空间相邻的网格编号也相邻。

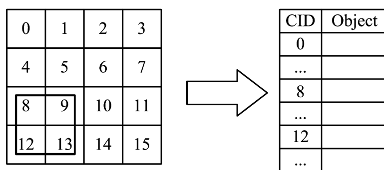

图 5　网格顺序编码示意图

Fig. 5　Example of grid coding sequentially

3.3.2　基于 Z-order 的编码

网格的编码实际上是一个降维的过程,将二维的空间坐标转换成一系列从小到大的一维编号,并且要求空间相邻的网格的编号也相邻。空间填充曲线就是为解决这些问题而被提出来的,是一种将高维空间问题转化为一维问题的映射。它通过空间聚类技术使得空间邻近的点逐一对应进行映射后在直线上也接近,从而避免了空间中两个不同的数据点被映射后在直线上重合的情况。

由于在应用过程中需要频繁地进行编码和解码操作,编码的效率非常重要。鉴于 Z-order 曲线的映射过程简单,没有复杂的反射和旋转操作,因此编码效率较高。本文在对网格编码时,采用了 Z-order 空间填充曲线来生成网格编号。

如图 6 所示,从左到右依次是对二维空间进行映射为 1 位、2 位和 3 位 Z-order 曲线的情况。在对网格编码时,用网格坐标经过 Z-order 曲线映射后的 Z 值来表示网格的 ID,由网格 ID 可以知道它在平面坐标系中的坐标。Z 值是通过对网格的二进制坐标的交叉操作生成的,位数为每维坐标二进制位数的和。对于图 6 中位数为 2 的情况,每维坐标都是由 2 位的二进制数表示,进行交叉操作后得到的二进制位数为 4。对于网格坐标(2,2),其中,$x=(2)_{10}=(10)_2$,$y=(2)_{10}=(10)_2$。先取 x 的最高位值 1,再取 y 的最高位值 1,之后取 x 的次高位值 0,最后取 y 的次高位值 0。对每次取到的值按取值顺序从左到右排列,最终得到网格 Z 值:$Z((10)_2,(10)_2)=(1100)_2=(12)_{10}$。

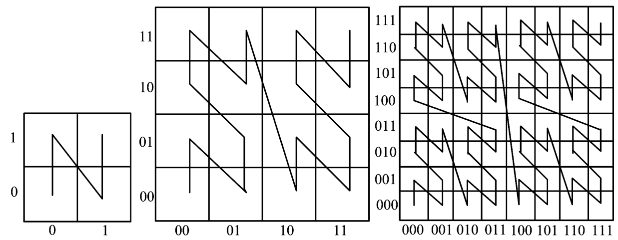

图 6　不同位数的 Z-order 曲线

Fig. 6　Z-order curve with different bits

4　基于 ZooKeeper 的分布式读写锁优化策略

4.1　预备知识

ZooKeeper[13]是一个高性能的开源分布式应用协调服务,是 Hadoop 的一个子项目。它主要用于解决分布式应用中一些常见的数据管理问题,包括集群管理、配置项管理、状态同步和统一命名服务等。ZooKeeper 中的数据都存储在内存中,保证了基于 ZooKeeper 实现的分布式锁的高效性。

ZooKeeper 使用了一个分层的类似文件系统目录的树结构,集群中的每个节点都维护着一棵相同的树结构。如图 7 所示,树中的所有节点都是 ZNode,/为树的根节点,可以从根节点往下逐层添加树节点,并且每个树节点都可以保存数据。每个 ZNode 节点存储节点自身的数据信息并维护其孩子节点的列表。例如,/app1 节点不仅存储了自身的数据,还要维护一个包含其孩子节点即/app1/_1,/app1/_2 和/app1/_3 这 3 个节点的列表。当/app1 节点的孩子节点增加或者减少时,列表会随之进行更新。

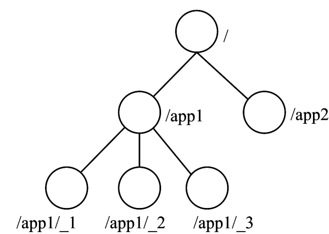

图 7　ZooKeeper 节点的结构

Fig. 7　Structure of ZooKeeper node

4.2　基于 ZooKeeper 的分布式读写锁优化策略

4.2.1　问题描述与基本思想

基于位置服务是面向多用户的,用户既是查询的发起者,又是被查询对象,同时用户的位置更新频繁。为了保证多用户并发查询结果的正确性,必须在索引操作时对节点加锁。文献[14]所介绍的分布式共享锁只有一个共享锁,按照 FIFO (First In First Out)的策略分配共享资源,进程请求锁的顺序和获得锁的顺序一致,不区分读锁与写锁,在多用户并发读取索引内容时,须进行不必要的等待。

针对这个问题,本文设计分布式读写锁,包括读锁和写锁两种。当多个进程同时访问一个资源时,多个进程可以同时

拥有读锁,但只有一个进程可以拥有写锁。读锁和写锁的冲突情况如表 4 所列,读锁与读锁之间不存在冲突,写锁与读锁以及写锁与写锁之间都存在冲突。

表 4 读/写锁冲突情况说明

Table 4 Description of conflicts between read lock and write lock

	读锁	写锁
读锁	否	是
写锁	是	是

本文实现的是基于 ZooKeeper 的公平分布式读写锁,客户端进程在获得读锁或者写锁时,依据的是进程获得读锁或写锁的顺序。

基于 ZooKeeper 实现的分布式读写锁通过 ZooKeeper 的数据节点来表示锁,有两种锁模式:读锁和写锁。在创建数据节点时,把节点分成读节点和写节点两类。读写锁的 ZooKeeper 节点如图 8 所示,本文中读节点和写节点通过不同的前缀来进行区分,读节点的前缀为 R_,写节点的前缀为 W_。例如,/rw_lock/R_1 节点代表一个读锁,/rw_lock/W_2 节点代表一个写锁。

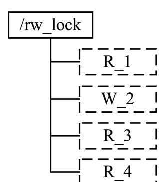

图 8 读写锁的 ZooKeeper 节点图

Fig. 8 ZooKeeper nodes of read-write lock

在请求读锁时,因为读锁与读锁之间不存在冲突,所以只要该读节点之前不存在写节点,就能成功获得读锁;即使该读节点之前存在读节点,读锁也可以与读锁相容,多个读节点能一起存在。如果该读节点之前有写节点,则请求读锁的这个进程监控该节点之前最大的写节点,这个写节点被删除之后该进程才能成功获得读锁。在请求写锁时,只有该写节点是最小的节点,才能成功获得写锁,并且请求写锁的这个进程要监控该节点的前一个节点,这个写节点被删除之后该进程才能成功获得写锁。

4.2.2 基于 ZooKeeper 的分布式读写锁

上文简要介绍了 ZooKeeper 实现的分布式读写锁的节点和读写锁的获取,下面将详细描述基于 ZooKeeper 的分布式读写锁的具体实现,流程如图 9 所示。

1)创建一个永久型(PERSISTENT)的 ZooKeeper 数据节点/rw_lock 作为读写锁的目录节点,用来表示客户端进程要访问的资源。本文需要对 R 树进行加锁,加锁的节点是 R 树的根节点。

2)所有需要获得读写锁的客户端进程都要调用 ZooKeeper 的 create( )方法在锁目录节点/rw_lock 下创建一个顺序临时节点(EPHEMERAL_SEQUENTIAL)作为其子节点。如果当前进程执行读请求,就创建以 R_为前缀的节点,例如图 8 中的/rw_lock/R_1,/rw_lock/R_3 和/rw_lock/R_4 节点;如果当前进程执行写请求,就创建以 W_为前缀的节点,例如图 8 中的/rw_lock/W_2 节点。

3)当前的客户端进程调用 ZooKeeper 的 getChildren()方法获得/rw_lock 目录下已经创建的所有子节点,然后判断当前的请求是否是写请求。如果是写请求,执行步骤 4);如果不是写请求,执行步骤 5)。

4)判断当前的节点是否是/rw_lock 目录下序号最小的节点,如果是,当前进程获得锁;如果不是,则调用 ZooKeeper 的 exists( )方法来对比自己的节点序号小的最后一个节点设置监听器。

5)如果当前的节点是/rw_lock 目录下序号最小的节点或者/rw_lock 目录下所有节点序号比自己小的都是读请求,则当前进程获得锁;否则,调用 ZooKeeper 的 exists( )方法来对比自己的节点序号小的最后一个写节点设置监听器。

6)一直等待监听器通知,一旦监控的节点的状态发生改变,就继续执行步骤 3),直到退出竞争。

7)当获得锁的客户端进程完成读/写操作或者此事务因故障中断时,当前进程通过删除它在/rw_lock 下创建的节点来释放锁。

由以上的描述可以看出,本文基于 ZooKeeper 实现的分布式读写锁是依据 FIFO 的策略来分配共享资源的,它实现了公平的分布式读写锁。

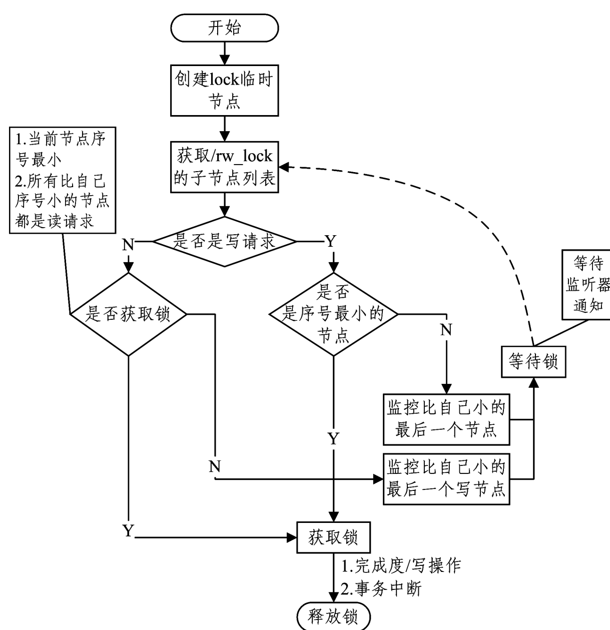

图 9 基于 ZooKeeper 的分布式读写锁算法的流程图

Fig. 9 Flowchart of read lock and write lock algorithm based on ZooKeeper

5 性能分析

基于 HBase 的 R 树索引在存储时,HBase 数据表中的每一行存储 R 树的一个节点。而本文提出的基于 HBase 的支持频繁更新的节点重组 R 树索引在 HBase 数据表中的每一行存储的是 R 树的当前节点及其孩子节点,这样只需访问一次 HBase 就可以获得这一行所有的 R 树节点,不需要逐个从 HBase 中读取,减小了磁盘访问的代价。

在索引进行更新时,由于索引的是网格,更新频率降低,

而且此索引更新时不需要额外的数据结构，因此与网格索引的更新效率相差不大。此外，随着查询范围的增大，查询的索引节点数目增多，索引的查询效率降低。

R 树索引的查询会受到进程和线程并行度的影响，并行度是影响本索引的一个因素，并行度越大，查询效率越高。

6　实验

6.1　实验环境与实验数据

实验使用 IBM X3650 M4 机架式服务器，集群由 5 台服务器组成。每台服务器的硬件配置如表 5 所列，软件环境如表 6 所列。本文分析对比了 4 种索引实现方案。

1）Grid：基于 HBase 的网格索引；

2）HR-tree：基于 HBase 的 R 树索引；

3）HNRSR-tree：本文提出的节点重组 R 树，网格顺序编码；

4）ZHNRSR-tree：本文提出的节点重组 R 树，网格 Z-order 编码。

表 5　单机硬件配置表

Table 5　Hardware configuration of single machine

名称	属性
处理器	2 * Xeon E5-2620 CPU(6 Cores 12 Thread)
内存	32 GB
硬盘	6 TB，10000rpm，raid5
显卡	8 MB 以上的 PCI 或 AGP 显卡
操作系统	CentOS 6.4 x86_64

表 6　软件环境配置表

Table 6　Configuration of software

名称	属性
开发工具	eclipse-kepler-SR2
开发语言	java1.7
云存储	hbase-0.96.0-cdh3u6
流处理系统	storm-0.10.0-wip16
资源协调	zookeeper-3.4.5-cdh5.0.2

本文使用仿真的移动对象数据集，表 7 列出了本节实验中用到的需要变化的数据参数。本节使用数据生成器生成的呈均匀分布和高斯分布这两种分布方式的合成数据进行实验。其中，查询范围是查询区域面积占整个地理空间的百分比；获取查询吞吐量时数据都是查询请求，获取更新吞吐量时数据都是更新请求；网格划分都统一默认为 256 * 256。另外，本文的索引支持多用户并发查询。该实验在 Storm[15] 环境下进行测试，Worker 与 Exectuor 是 Storm 相关的参数，详细架构请参见文献[13]。

表 7　实验数据参数表

Table 7　Experimental parameters

参数名称	变化范围	默认值
数据集	均匀分布，高斯分布	均匀分布
移动对象数目/k	50，100，150，200，250	50
查询范围/%	0.025，0.05，0.075，0.1，0.125	0.025
查询数据量/k	10，20，30，40，50	20
Worker 数目	4，8，16，32，64	64
Executor 数目	80，100，120，140，160	140

6.2　索引节点重组的实验结果

6.2.1　实验方案

本节主要是测试本文提出的基于 HBase 的支持频繁更新的节点重组 R 树索引的性能，并且与网格索引和基于 HBase 的 R 树索引作对比。此外，还分别测试了在不同参数设置下所提索引的性能。

6.2.2　实验结果及评价

1）均匀分布下索引的查询和更新效率

基于均匀分布的数据集，测试不同查询范围下 3 种索引的吞吐量，如图 10 所示。其中，HNRSR-tree 是本文提出的基于 HBase 的节点重组 R 树，Grid 是网格索引，HR-tree 是基于 HBase 的 R 树索引。实验查询数据量采用默认值 20 k，网格划分为 256 * 256，Worker 数目为 64，Executor 数目为 140。图 11 展示了均匀分布的数据集上不同数据量时 3 种索引的更新吞吐量。实验中的其他数据均采用默认值。

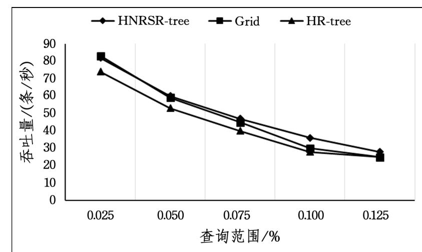

图 10　不同查询范围下 3 种索引的吞吐量

Fig. 10　Throughput of three kinds of indexes under different query ranges

如图 10 所示，随着查询范围的不断增大，3 种索引的查询吞吐量都逐渐下降。因为在查询范围增大的过程中，查询的网格数目增多。其中，HNRSR-tree 的查询吞吐量比 HR-tree 高，这是因为在存储时 R 树节点和它的孩子节点存储在同一行，减少了访问 HBase 的开销。此外，HNRSR-tree 的查询吞吐量比 Grid 高，这是因为随着查询范围的增大，查询区域内所包含的空网格的数量会增多，进而导致网格索引的查询性能下降。

如图 11 所示，随着数据量的增加，3 种索引的更新吞吐量变化不大，趋于稳定。其中，网格索引 Grid 的更新吞吐量最高，因为相对于 HNRSR-tree 和 HR-tree，网格索引更新更简单，不用更新 R 树索引。对于 HNRSR-tree 和 HR-tree，随着数据量的增大，更新的数据趋于稳定，向 R 树添加网格和从 R 树中删除网格的情况也较少。

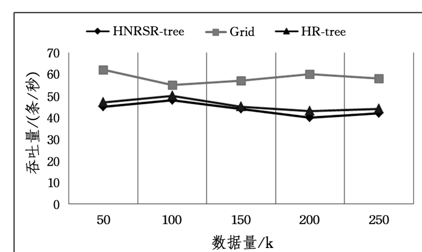

图 11　不同数据量下 3 种索引的更新吞吐量

Fig. 11　Update throughput of three kinds of indexes under different data sizes

2)高斯分布下索引的查询和更新效率

图 12 展示了高斯分布数据集上不同查询范围下 3 种索引的吞吐量。实验中查询的数据量为 20k,网格划分为 256 * 256,Worker 数目为 64,Executor 数目为 140。图 13 展示了高斯分布数据集上不同数据量下 3 种索引的更新吞吐量。实验中的其他数据均采用默认值。

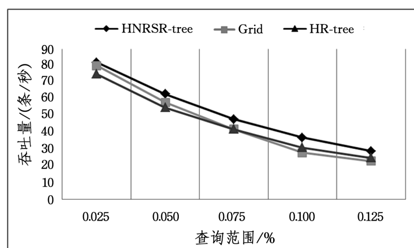

图 12 不同查询范围下 3 种索引的吞吐量

Fig. 12 Throughput of three kinds of indexes under different query ranges

如图 12 所示,随着查询范围的逐步增大,3 种索引的查询吞吐量均呈下降趋势。其中,Grid 的下降速度最快,在查询范围占比为 0.075%时,Grid 和 HR-tree 的查询吞吐量相等;当查询范围占比大于 0.075%时,HR-tree 的查询吞吐量大于 Grid;并且 HNRSR-tree 的查询吞吐量一直大于 Grid 和 HR-tree。在移动对象位置数据为高斯分布的情况下,划分后的网格中,空网格的数目增多,网格索引效率下降,而 HNRSR-tree 和 HR-tree 则不受空网格的影响。由于 HNRSR-tree 在查询时只访问一次 HBase 就能获取 R 树节点和它的孩子节点,减少了访问 HBase 的次数,降低了访问磁盘的开销,因此 HNRSR-tree 比 HR-tree 的查询效率高。

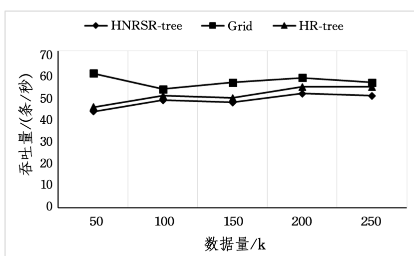

图 13 不同数据量下 3 种索引的更新吞吐量

Fig. 13 Update throughput of three kinds of indexes under different data sizes

由图 13 可以看出,随着更新数据量的增加,3 种索引的吞吐量都只有微小的变化。3 种索引在进行更新时,都相当于单线程在执行操作。因此,数据量的大小对更新的影响较小。在数据量增大的过程中,HNRSR-tree 和 HR-tree 的吞吐量有增加的趋势,这是因为数据量不断增大时,移动对象位置的更新趋于稳定,R 树更新的情况变少。

3)不同数据量下索引的查询效率

图 14 给出了不同数据量下 HNRSR-tree,Grid 和 HR-tree 这 3 种索引的查询吞吐量。该实验采用默认的均匀分布数据集,查询范围占比为 0.025%,网格划分为 256 * 256,Worker 数目为 64,Executor 数目为 140。

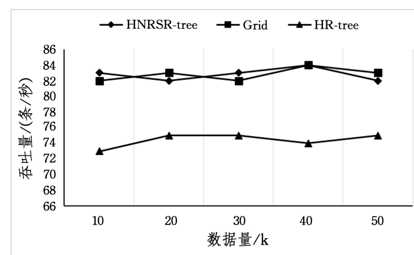

图 14 不同数据量下 3 种索引的查询吞吐量

Fig. 14 Query throughput of three kinds of indexes under different data sizes

如图 14 所示,随着查询数据量的增加,HNRSR-tree,Grid 和 HR-tree 索引的查询吞吐量都趋于稳定。这是因为查询数据都是先发送到消息队列中,然后从消息队列中依次拉取查询请求进行处理。由于发送消息的速度远超过消息处理的速度,因此消息队列一直有未处理的请求。这样,查询数据量对查询效率是没有影响的。

4)不同并行度下索引的查询效率

下面主要测试不同并行度下本文所提索引的性能。这里,并行度主要包括进程并行度和线程并行度。在 Storm 中,Worker 相当于进程,而 Executor 相当于线程,因此就是测试不同的 Worker 和 Executor 下索引的性能。

图 15 和图 16 分别描述了不同 Worker 和 Executor 数目下 HNRSR-tree 索引查询和更新的吞吐量。两个实验均采用默认的均匀分布数据集。图 15 的实验中 Executor 数目为 140,其他参数均为默认值。图 16 的实验中 Worker 数目为 64,其他参数均为默认值。

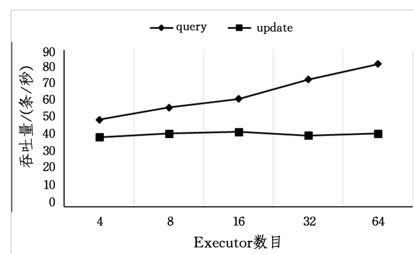

图 15 不同 Worker 数目下 HNRSR-tree 索引查询和更新的吞吐量

Fig. 15 Query and update throughput of HNRSR-tree index under different numbers of Worker

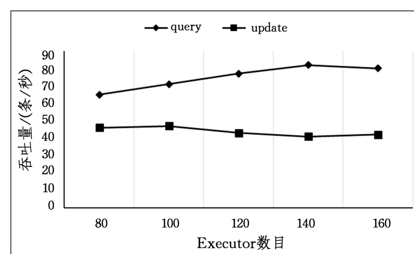

图 16 不同 Executor 数目下 HNRSR-tree 索引查询和更新的吞吐量

Fig. 16 Query and update throughput of HNRSR-tree index under different numbers of Executor

由图 15 可以看出,在 Executor 数目一定时,随着 Worker 数目的不断增加,HNRSR-tree 索引的查询吞吐量不断增大,HNRSR-tree 索引的更新吞吐量基本没有变化。Worker 数目

增大意味着进程并行度增大，更多进程并行提升了 R 树的查询效率。然而对于更新而言，无论进程数目多少都相当于单线程更新，因此更新吞吐量趋于稳定。

如图 16 所示，当 Worker 数目一定时，随着 Executor 数目的不断增加，HNRSR-tree 索引的查询吞吐量先不断增加，后逐渐下降，在 Executor 数目为 140 时达到最大值。这是因为在 Executor 数目较小时，并行效率没有得到充分利用，很多进程发生阻塞，查询效率降低；但是，当 Executor 数目过大时，会出现 Executor 空闲的情况，吞吐量也会降低。在 Executor 数目增加的过程中，HNRSR-tree 索引的更新吞吐量稳定。

6.3　Z-order 编码性能实验

6.3.1　实验方案

本实验主要测试采用 Z-order 对网格进行编码的性能。实验环境采用默认设置，实验使用了两种编码策略来进行对比，分别为 HNRSR-tree 和 ZHNRSR-tree。其中，HNRSR-tree 采用顺序编码方式，ZHNRSR-tree 采用 Z-order 编码方式。

6.3.2　实验结果及评价

图 17 描述了不同查询范围下两种编码策略和 Grid 的查询吞吐量。随着查询范围的增大，HNRSR-tree 和 ZHNRSR-tree 的吞吐量都不断减小，Grid 与 HNRSR-tree 的趋势与图 12 相同。在相同查询范围下，ZHNRSR-tree 的吞吐量提高幅度比 HNRSR-tree 快 25％～50％。采用 Z-order 编码策略后，空间相邻的区域在 HBase 表中的位置也相邻，查询时使用 getList()方法一次获取相邻的网格，减少了多次重复访问的冗余开销。此外，在查询范围增大的过程中，ZHNRSR-tree 的吞吐量提高幅度比 HNRSR-tree 的吞吐量提高幅度更大。这是因为查询范围增大后，范围内包含的网格数目增多，从而一次查询获取的网格数量也增多，减少了更多重复访问 HBase 的冗余磁盘开销。

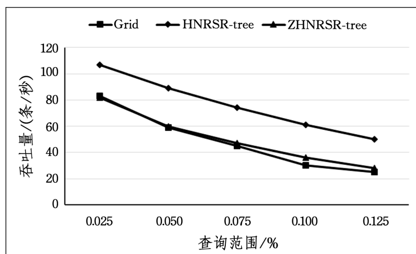

图 17　不同查询范围下不同编码策略的吞吐量

Fig. 17　Throughput of different code tragedies under different query ranges

6.4　基于 ZooKeeper 的分布式读写锁的性能实验

6.4.1　实验方案

本实验主要测试基于 ZooKeeper 的分布式读写锁的性能。实验环境采用默认设置，实验中使用了两种锁策略来进行对比，分别为 SLock 和 RWLock。其中，SLock 采用基于 ZooKeeper 的分布式共享锁，RWLock 采用本文实现的基于 ZooKeeper 的分布式读写锁。

6.4.2　实验结果及评价

图 18 给出了不同查询范围下两种锁的吞吐量。随着查询范围的不断增大，SLock 和 RWLock 的吞吐量都不断下降。但是，相同查询范围下，RWLock 的吞吐量大概是 SLock 的 4 倍。由于 R 树索引是对根节点进行加锁，SLock 不区分加锁的类型，只要有操作便独占整个索引，其他进程不能进行任何操作；RWLock 区分读写锁后，读操作可以同时访问共享索引，同时进行查询操作，从而在并行查询时明显提高了查询效率。但是，由于加锁和解锁操作本身也是有开销的，因此 RWLock 的吞吐量也只是 SLock 吞吐量的 4 倍左右。

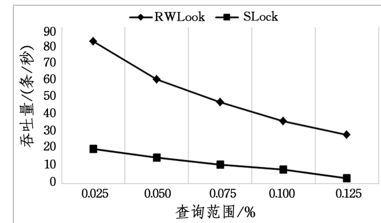

图 18　不同查询范围下两种锁粒度下的吞吐量

Fig. 18　Throughput of two lock granularity under different query ranges

结束语　移动大数据时代，传统基于位置服务的技术面临系统扩展性和性能等方面的挑战。云计算技术是大数据处理的基础，由于体系架构的不同，许多传统的处理技术需要在云环境下重新实现。

移动对象索引是基于位置服务中的关键技术之一，必须有效支持频繁更新与多用户并发操作。本文针对这一需求，在 HBase 上设计并实现一个支持频繁更新与多用户并发的 R 树索引。其主要思想是：1）基于 HBase 的数据行与数据分区的组织与读写特性，对 R 树的节点进行重组，并基于 Z-order 对网格编码，减少了对 HBase 的读写操作，提高了查询效率；2）提出了基于 Zookeeper 分布式读写锁的优化策略，提高了索引的吞吐量。实验结果表明，与网格索引相比，所提方案是有效的。本文方案是对现有研究成果的一个补充，同时也可以用于其他基于树的索引结构中。

未来工作包括两个方向：1）对节点重组策略进行深入的研究，如考虑对多层（大于两层）节点进行重组；2）对分布式锁机制进行深入研究，如利用 HBase 自身 put 与 get 操作的读写锁来提高吞吐率。

参 考 文 献

[1]　KHAN A U R，OTHMAN M，MADANI S A，et al. A Survey of Mobile Cloud Computing Application Models[J]. IEEE Communications Surveys & Tutorials，2014，16(1)：393-413.

[2]　WOLFSON O. Moving Objects Information Management：The Database Challenge[M]∥Next Generation Information Technologies and Systems. Springer Berlin Heidelberg，2002：75-89.

[3]　WANG S，WANG H J，QIN X P，et al. Architecture of Big Data：Challenges，Status and Prospects[J]. Chinese Journal of Computers，2011，34(10)：1741-1752. (in Chinese)
王珊，王会举，覃雄派，等. 架构大数据：挑战、现状与展望[J]. 计

算机学报，2011，34(10)：1741-1752.

[4] CARSTOIU D，LEPADATU E，GASPAR M. Hbase-non SQL Database，Performances Evaluation[J]. International Journal of Advancements in Computing Technology，2010，2(5)：42-52.

[5] ZHOU X，ZHANG X，WANG Y，et al. Efficient Distributed Multi-dimensional Index for Big Data Management[M]// Web-Age Information Management. Springer Berlin Heidelberg，2013：130-141.

[6] TAKASU A. An Efficient Distributed Index for Geospatial Databases[M] // Database and Expert Systems Applications. Springer International Publishing，2015：28-42.

[7] ZHOU X，LI H，ZHANG X，et al. ABR-Tree：An Efficient Distributed Multidimensional Indexing Approach for Massive Data [C]// International Conference on Algorithms and Architectures for Parallel Processing. Springer，Cham，2015：781-790.

[8] HUANG S，WANG B，ZHU J，et al. R-HBase：A Multi-dimensional Indexing Framework for Cloud Computing Environment [C] // IEEE International Conference on Data Mining Workshop. IEEE，2015：569-574.

[9] HUANG S，WANG B，DENG S，et al. HMVR-tree：A Multi-version R-tree Based on HBase for Concurrent Access[M]// Big Data Computing and Communications. Springer International Publishing，2016.

[10] XIA Y，HUANG Z，ZHANG X，et al. Parallel Indexing for Past，Current and Future Locations of Moving Objects[C]// International Conference on Service Sicence，Technology and Engineering. DEStech Publications，2016：20-27.

[11] DU N，ZHAN J，ZHAO M，et al. Spatio-temporal data index model of moving objects on fixed networks using hbase[C]// 2015 IEEE International Conference on Computational Intelligence & Communication Technology (CICT). IEEE，2015：247-251.

[12] GEORGE L. HBase：The Definitive Guide[M]. The People's Posts and Telecommunications Press，2013. (in Chinese)
GEORGE L. Hbase 权威指南[M]. 北京：人民邮电出版社，2013.

[13] Zookeeper[EB/OL]. https://zookeeper. apache. org.

[14] WANG B T，ZHAO K L，CHANG L D，et al. Optimization Technique fot Continuous Range Query Based on Storm[J]. Computer Science and Engineering，2017，39(1)：1-14. (in Chinese)
王波涛，赵凯利，常立东，等. 基于 Storm 的连续范围查询优化技术[J]. 计算机工程与科学，2017，39(1)：1-14.

[15] Apache Storm[EB/OL]. http://storm. apache. org.

(上接第 30 页)

[18] MCCLAIN J O，RAO V R. CLUSTISZ：A Program to Test for the Quality of Clustering of a Set of Objects[J]. Journal of Marketing Research，1975，12(4)：456-460.

[19] DAVIES D L，BOULDIN D W. A cluster separation measure [J]. IEEE Transactions on Pattern Analysis & Machine Intelligence，1979，PAMI-1(2)：224-227.

[20] INCORPORATED C S I. SAS - C Socket Library for TCP-IP，Release 5. 01：SAS Technical Report C-111[R]. SAS Publishing，1992.

[21] ROUSSEEUW P. Silhouettes：A graphical aid to the interpretation and validation of cluster analysis[J]. Journal of Computational & Applied Mathematics，1987，20(20)：53-65.

[22] KRZANOWSKI W J，LAI Y T. A Criterion for Determining the Number of Groups in a Data Set Using Sum-of-Squares Clustering[J]. Biometrics，1988，44(1)：23-34.

[23] XIE X L，BENI G. A Validity Measure for Fuzzy Clustering[J]. IEEE Transactions on Pattern Analysis & Machine Intelligence，1991，13(13)：841-847.

[24] HALKIDI M，VAZIRGIANNIS M，BATISTAKIS Y. Quality Scheme Assessment in the Clustering Process[M] // Principles of Data Mining and Knowledge Discovery. Springer Berlin Heidelberg，2000：265-276.

[25] HALKIDI M，VAZIRGIANNIS M. Clustering validity assessment：finding the optimal partitioning ofa data set[C] // IEEE International Conference on Data Mining. IEEE，2001：187-194.

[26] AMORIM R C D，HENNIG C. Recovering the number of clusters in data sets with noise features using feature rescaling factors[J]. Information Science，2015，324：126-145.

[27] CAMPO D N，STEGMAYER G，MILONE D H. A new index for clustering validation with overlapped clusters[J]. Expert Systems with Applications，2016，64(C)：549-556.

[28] FRIEDMAN H P，RUBIN J. On Some Invariant Criteria for Grouping Data[J]. Publications of the American Statistical Association，1967，62(320)：1159-1178.

[29] SCOTT A J，SYMONS M J. Clustering Methods Based on Likelihood Ratio Criteria[J]. Biometrics，1971，27(2)：387-397.

[30] HUBERT L J，LEVIN J R. A general statistical framework for assessing categorical clustering in free recall[J]. Psychological Bulletin，1975，83(6)：1072-1080.

[31] MILLIGAN G W. An examination of the effect of six types of error perturbation on fifteen clustering algorithms [J]. Psychometrika，1980，45(3)：325-342.

[32] JAIN A K，MURTY M N，FLYNN P J. Data clustering：a review[J]. Acm Computing Surveys，1999，31(3)：264-323.

[33] XU R，WUNSCH I D. Survey of clustering algorithms[J]. IEEE Transactions on Neural Networks，2005，16(3)：645-678.

[34] LAROSE D T. Introduction to Data Mining[M]. Boston：China Machine Press，2010.

[35] SALTON G，HARMAN D. Information retrieval[M]. Chichester：John Wiley and Sons Ltd. ，2003.

[36] MANNING C D，RAGHAVAN P，SCHÜTZE H. An Introduction to Information Retrieval[J]. Journal of the American Society for Information Science & Technology，2008，61(4)：852-853.

[37] WITTEN D M，TIBSHIRANI R. A framework for feature selection in clustering[J]. Publications of the American Statistical Association，2010，105(490)：713-726.

[38] SUN W，WANG J，FANG Y. Regularized k-means clustering of high-dimensional data and its asymptotic consistency[J]. Electronic Journal of Statistics，2012，6(2)：148-167.