分片位图索引:一种适用于云数据管理的辅助索引机制

孟必平[1),3)] 王腾蛟[1),3)] 李红燕[2),3)] 杨冬青[1),3)]

[1)](高可信软件技术教育部重点实验室(北京大学) 北京 100871)

[2)](机器感知与智能教育部重点实验室(北京大学) 北京 100871)

[3)](北京大学信息科学技术学院 北京 100871)

摘 要 云计算技术的快速发展为海量数据的存储和管理提供了可能.然而,由于存储模型的根本改变,传统关系数据库管理系统中成熟的索引技术既不能直接应用于海量数据的处理,也无法被简单地迁移到云计算环境中.通过分析对比辅助索引在云环境中的两种截然不同的基本逻辑结构,即集中式方案与分布式方案,在吸收两者的优势并规避其弱点的基础上,提出了具有良好可扩展性的分片位图索引机制,从而对云环境中海量数据的检索任务提供高效的支持.通过充分利用云环境中的并行计算资源,使单条查询的响应速度得到提升;与此同时,局部节点根据其所掌握的全局信息规避了不必要的检索开销从而使大量请求并发到达时的查询吞吐量得以保证.在真实数据上进行实验的结果表明,分片位图索引的查询性能大大优于其它方法.

关键词 云计算环境;辅助索引;集中式方案;分布式方案;分片位图索引

中图法分类号 TP311 **DOI号**: 10.3724/SP.J.1016.2012.02306

Regional Bitmap Index: A Secondary Index for Data Management in Could Computing Environment

MENG Bi-Ping[1),3)] WANG Teng-Jiao[1),3)] LI Hong-Yan[2),3)] YANG Dong-Qing[1),3)]

[1)](*Key Laboratory of High Confidence Software Technologies (Peking University) of Ministry of Education, Beijing* 100871)

[2)](*Key Laboratory of Machine Perception (Peking University) of Ministry of Education, Beijing* 100871)

[3)](*School of Electronics Engineering and Computer Science, Peking University, Beijing* 100871)

Abstract The fast development of Cloud Computing technologies has brought new dawns to the storage and management of massive data. Nevertheless, due to the essential changes in the storage model, the matured indexing techniques used in traditional relational data management systems can neither be directly applied to massive data, nor be migrated to Cloud environment in an easy way. Based on comparisons between two basic approaches to secondary indexing, i.e. centralized and distributed approaches, the Regional Bitmap Index (RBI) is proposed to combine the advantages of both approaches and provide efficient supports to various queries against massive data in the Cloud. By means of fully utilizing the parallel computing resources provided by the Cloud, the query efficiency is dramatically improved. Meanwhile, based on global distribution information, RBI can avoid the unnecessary computing expenses on local nodes; therefore query

收稿日期:2012-06-05;最终修改稿收到日期:2012-08-14.本课题得到国家"八六三"高技术研究发展计划项目基金(2012AA011002,2011AA010706)、核高基重大专项(2010ZX01042-002-002-02,2010ZX01042-001-003-05)、国家自然科学基金(60973002,61170003,61073018)和深港创新圈项目(JSE201007160004A)资助.**孟必平**,男,1986年生,硕士,主要研究方向为云存储系统.E-mail: mengbiping@gmail.com. **王腾蛟**,男,1973年生,博士,教授,博士生导师,主要研究领域为互联网舆情分析与大型数据库管理系统实现技术等. **李红燕**,女,1970年生,博士,教授,博士生导师,主要研究领域为云环境下的数据管理、数据仓库和数据挖掘等. **杨冬青**,女,1945年生,教授,博士生导师,主要研究领域为数据库系统实现技术、Web环境下的信息集成与共享、数据仓库和数据挖掘等.

throughputs can keep steady even if concurrency of the incoming queries increases. Experiments on real dataset show that the Regional Bitmap Index can significantly outperform other methods.

Keywords　cloud computing; secondary index; global approach; distributed approach; regional bitmap index

1　引　言

现今，随着万维网(World Wide Web)的快速发展，网络数据大量涌现. 丰富的网络活动产生了规模异常庞大的数据①：2009 年全年共有 90 万亿封电子邮件被发出，平均每天发出的电子邮件数目就高达 2.47 千亿封，每一封电子邮件都拥有发送时间、发送方地址、接收方地址和投递服务器等属性；Google 搜索引擎每天收到的检索请求数以十亿计，每个检索任务都将产生若干条日志记录，包括检索时间、检索关键字、响应时间、命中页面数以及用户 Cookie 等属性；来自世界各地的用户每天在 Twitter 网站上发布的微博数量超过三千万条，每条微博都拥有微博标识符、发布者标识符、发布时间、微博内容甚至转发关系等属性. 如此大规模数据的涌现是互联网时代所特有的. 传统关系型数据管理系统已无法胜任海量数据管理任务.

以 Web 为主导的信息时代的繁荣和演进正在向数据管理系统提出前所未有的新挑战. 所幸，云计算技术的快速发展为海量数据的存储和管理提供了可能. 相比传统的单机计算环境，云环境可以有效地利用分布式集群的庞大计算资源来满足海量数据管理对计算资源和存储资源的需求，并且拥有易于维护、易于扩展和易于管理等优良特性. 面对快速增长的数据规模，云计算技术能够快速调整并分配所需资源以适应数据的疯狂膨胀；同时能够提供具有弹性的、组织松散的存储模式以及建立在这种存储模式之上的可配置的分布式并行计算资源[1]. 云计算技术的运用正逐渐成为海量数据管理任务中的新趋势.

但是，配置于云环境中的分布式存储系统并不能够像传统关系数据库管理系统那样对数据的多样化查询提供高效支持[2]. 在数据存储和管理领域，索引被证明是能使存储系统支持高效检索的有效手段. 为了支持在存储于大规模的分布式集群上的海量数据中快速找到满足条件的元组的操作，云环境中的数据管理系统也需要配备高效的索引机制.

作为基本的索引类型，聚集索引仅能够服务于针对元组主键的查询请求. 在普遍情况下，云环境中数据表的主键往往仅用于区别不同的元组，而并不具有实际意义. 相比之下，在更多的实际应用场景中，针对非主键属性上的查询请求常常被更加频繁地应用于各种各样的检索任务中. 此类查询请求的处理无法使用聚集索引，而只能借助于辅助索引.

然而，由于存储模型的根本改变，传统关系数据库管理系统中成熟的辅助索引技术，既不能直接用于处理海量数据，也无法被简单地迁移到云计算环境中. 为云环境中的数据管理系统设计全新的辅助索引结构并非易事，其难点主要包括：

(1) 海量数据上的辅助索引结构势必拥有与数据本身规模相仿的索引项数目. 如此庞大的索引结构的管理及其高效访问算法的设计面临着传统数据库管理系统中未曾遇到的新问题.

(2) 云环境中并行计算资源对于加快大规模数据上的检索速度是有利的. 但是，以粗暴的方式滥用并行资源将会导致系统的服务租赁费用急剧升高. 而且在多客户端并发访问的情况下，将极大地影响系统能够承受的查询吞吐量.

(3) 一般地，服务提供商隐藏了云环境中各计算节点的物理拓扑结构[3]. 因此，检索算法对网络的访问接口往往会受到限制. 依赖于网络通信协议的索引实现方案将会引起难以评估的巨大的潜在网路通信开销.

(4) 由多个检索条件复合而成的复杂查询请求将引起多个检索结果集的求交操作. 在大规模数据集上执行求交运算将引起较高的时间复杂度[4]，且难以被并行化.

作为辅助索引的一种成熟实现，位图索引在云数据管理任务中能够体现出其独特的优势：对于很多常见属性，其基数值受数据增长的影响很小，例如性别、生日、商品类别等；而多个属性上的复合检索条件是常常被用到的，例如在电子商务应用中检索

① 网络流量数据来自于 Netcraft 网站的统计信息；电子邮件的统计信息来自于 Radicati Group；社交网络的相关统计信息来自于 Twittercounter 和 GigaOm.

出生于1990年之后的购买过电子产品的女性用户.相比其它类型的索引,位图索引能够非常高效地获得多个检索条件复合之后的检索结果[5-6].

基于以上考虑,本文通过分析对比云环境中的辅助索引的两种截然不同的基本逻辑结构,即集中式方案与分布式方案,在吸收二者的优势并规避其缺陷的基础上,提出了一种具有良好可扩展性的辅助索引机制:分片位图索引(Regional Bitmap Index).该索引结构能够充分发挥现代计算机执行位图逻辑运算的优势.通过使用分布式的索引存储方法,大规模索引结构得到了有效管理.属性值的全局排序使得在索引上进行并发检索的代价得到了降低,从而有利于查询吞吐量的提升.另外,本方法不依赖于计算节点的网络组织结构,因而拥有较强的适用性.

本文第2节将分类介绍相关的研究工作;第3节将陈述分片位图索引的核心技术及其生成、维护与检索算法;在真实数据上进行对比实验的结果将于第4节中进行汇总分析;最后,第5节对本文贡献进行归纳和总结.

2 相关工作

传统数据管理系统已经拥有非常成熟的索引技术,相比之下,在云环境中,索引技术的研究工作尚不充分.本文将在云环境中实现辅助索引机制的方案按照其的基本逻辑结构分为两类,集中式方案与分布式方案.以下对它们分别进行介绍.特别地,针对基于分布式方案的扩展工作,本节做出了单独的分析和说明.

2.1 集中式方案

在集中式方案中,被索引字段上的所有值被全局排序并集中管理.具体而言,每条元组对应的索引项将包含被索引的字段的值以及该记录对应的主键值.在索引结构中,这些索引项按照被索引字段的值全局排序.系统处理被索引字段上的检索请求的过程分为两个步骤.首先在全局排序的索引结构中找到符合条件的索引项,从而得知相应记录的主键值.然后,依据主键值访问聚集索引从而定位完整的记录.

集中式方案的最大挑战莫过于全局排序了的索引项的管理.一个最直接的方法是将这些索引项如同其它一般数据一样分布式地存储在数据管理系统中.以Bigtable[7]的开源实现HBase[8]的存储系统为例,图1展示了一个基于HBase的集中式方案的辅助索引的实现(Transactional Index[9])及其查询流程.

图1中ROOT表记录了目标元组应当对应于哪一个META表,而META表则记录了目标元组的真实位置,二者共同构成了一个多级的聚集索引结构.检索过程中,首先通过第1,2,3步获得目标元组所对应的主键值,然后经过第5,6,7步根据主键值找到目标元组.通过这种方式,庞大的索引结构也可以享受到海量数据管理系统提供的可靠性、可扩展性和易管理性.但是,整个访问过程中的每个步骤都仅有一个单独的数据节点参与完成,这并未有效利用分布式计算资源带来的优势.从而,检索任务的响应时间将会非常长.区别于全局排序的方式,文献[10]在分布式环境中实现了B树索引[11],该方法将完整数据上的B树分拆为多个分支,并将各个分支分布存储于各个数据节点上.各个数据节点均记录有分支的分布情况.检索B树需要在向下搜索的过程中定位目标分支的宿主节点并在宿主节点中获取命中的元组.该方法为大规模B树的分布式存储和统一管理提供了指导.文献[12]将计算节点的点对点组织形式抽象为凯莱图(Cayley Graph)[13],并在此基础之上集中管理全局的索引结构.

此类方法的特点在于索引结构在逻辑上是被集

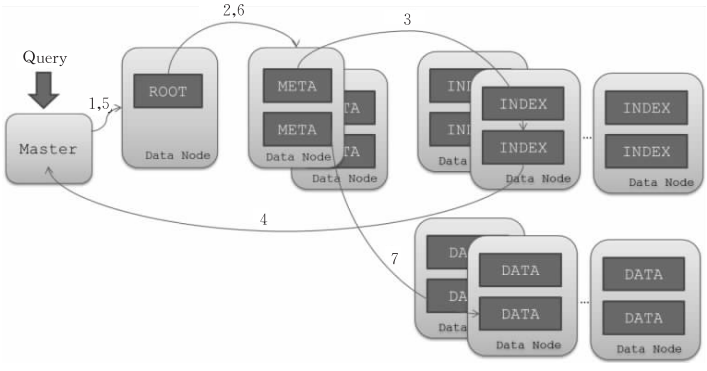


图1 辅助索引的集中式方案

中管理的，各种具体方法均是从为大规模数据上完整的大型索引结构设计高效的管理机制入手来实现索引的. 而索引结构本身在逻辑上与传统关系数据库中的集中管理模式无异. 此类方法的优点在于它们是传统单节点数据管理系统中索引结构的直观扩展，其实现不需了解数据的真实分布情况；但是正如传统索引并未对分布式环境进行特别设计，此类方法也往往难以有效利用云环境所提供的分布式计算资源来提高索引的检索性能.

2.2　分布式方案

在分布式方案中，各数据节点独立建立了各自管理的局部数据上的索引. 分布式方案没有维护索引值的全局排序，而是将其局部化到每个单独的计算节点上. 从而，计算节点相互之间不存在依赖关系，这为检索请求的并发执行带来了便利. 当拥有索引的属性上的检索请求到达时，检索任务将被分发到所有的计算节点上并以并发的方式执行. 最终的检索结果将是所有数据节点上返回结果的并集. 每个计算节点上数据的索引将被独立维护，因此其局部的索引结构具有很强的灵活性：各节点使用的索引技术可以是同构的，例如均使用 B^+ 树索引；也可以是异构的[14]，例如有的节点上使用 B^+ 树索引，而其它的节点上则使用位图索引等等. 异构的局部索引允许各计算节点依据自身的计算资源来选择所使用的索引技术，例如，主存资源不充足的节点可以使用 B^+ 树索引并仅在主存中维护 B^+ 树靠近根部两层的节点. 而 CPU 计算能力较差的节点则可以使用位图索引，从而利用位图上的逻辑操作来回避计算开销. Indexed HBase[15] 是基于 HBase 的分布式方案辅助索引的一个代表性的朴素实现. 图 2 展示了分布式方案中索引的组织方式.

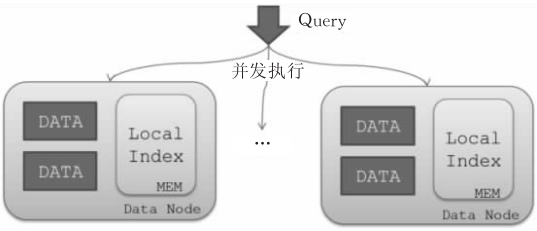


图 2　辅助索引的分布式方案

在此类方法中，索引结构依附于数据本身而被分散到各个数据节点中. 节点之间具有独立性. 检索任务也分配到各节点中独立执行，从而，并行计算资源能够得到很好地利用. 但是，特别以最常用到的等值条件为代表，由于绝大多数检索任务的目标记录数量相对较少，在分布式集群中并行地执行该任务往往造成很多未存储任何目标记录的计算节点也触发了检索过程，而最终将返回空集. 在检索任务频繁的情况下，这一并行执行过程将会耗费大量的不必要的计算资源，最终将会降低系统的吞吐量.

2.3　分布式方案的扩展

也应当注意到，一些研究工作在采用分布式方案来组织局部索引结构的同时，使用了特殊的全局结构来加快对有效数据节点的定位. 文献[16]在云环境中设计了全新的通信协议 RT-CAN，用于管理一个全局索引结构和各数据节点上的局部多维数据 R 树索引. 首先检索全局索引，然后定位到含有目标元组的局部 R 树索引上继续检索. 文献[17]与文献[16]拥有相同的项目背景，其作者以分布式方式为各数据节点上的本地数据构建局部的 B^+ 树索引，另外，各局部 B^+ 树上的某些结点被公开出来并通过 BATON[18] 协议统一组织成为一个全局结构 CG-index. 检索请求到达时，首先 CG-index 被检索，定位到局部 B^+ 树上的某些结点之后继而转入相应的局部数据节点继续搜索.

在这类方法中，通过维护一种特殊的全局索引结构，分布式方案的基本思路得以扩展，从而可以在充分并行的同时，避免了一些不必要的检索代价. 然而，该类方法往往需要通过较为复杂的通信协议来组织全局索引结构的各个部分. 因此，访问全局结构的过程将引发大量的网络通信开销. 在复杂的拓扑网络中，这一过程将带来严重的响应延迟. 另外，并不是所有的云环境都向开发人员暴露了完整的网络通信接口，在通信接口被高度抽象了的云环境中，复杂的网络通信协议的实现将受到致命的限制.

3　分片位图索引

集中式方案和分布式方案各有优势. 集中式方案拥有被索引字段的全局信息，可以有效地避免不必要的查询开销，但是，全局信息的维护与访问也带来了额外的代价，致使查询的响应速度受到影响. 分布式方案能够充分利用并行计算资源的优势，但是由于缺乏被索引字段上值的整体分布信息，即使对于目标值仅局部分布在少数几个数据节点上的检索请求，也会导致触发整个集群上的并行检索. 而事实上，其中大部分数据节点上返回的命中集合都将是空集. 因而，大量的并行计算资源被白白浪费了. 本节将介绍一种折中的解决方案：分片位图索引 (regional bitmap index). 该方法既具有分布式方案

的高并行度特点,从而带来了快速的请求响应,同时又拥有如同在集中式方案中那样了解全局的数据分布信息的优势,进而尽最大努力避免了不必要的检索代价,提高了系统的查询吞吐量.相比使用特殊的通信协议而实现的分布式方案的诸扩展方法,本方法通过引入属性值的全局排序机制来彻底避免不同数据节点之间的通信.从而,使得开发人员能够从设计复杂通信协议的任务中解放出来.这一特点降低了本方法对网络拓扑和通信系统的依赖,从而提高了查询效率.另外,通过这种方式,本方法将无需访问目标云环境所暴露的网络通信接口,因此能够适用于更广泛的云存储环境.

下面将在 3.1 节中介绍分片位图索引的核心技术及其构建方法;3.2 节集中介绍各类数据操纵操作所引起的索引的维护算法;3.3 节阐述使用本索引结构进行检索的算法流程.

3.1 索引结构

本节首先介绍分片位图索引的核心技术:字段值上的全局排序机制以及位图索引的分片机制(全局排序机制能够为各数据节点提供一定的全局信息,即局部数据在全局值域中的分布情况;而位图索引的分片机制使得索引结构局部化,从而使不同数据节点上的检索任务充分独立,以便并发执行).然后综合运用上述技术给出了分片位图索引技术的构造方法及其存储结构.

3.1.1 字段值的全局排序

考虑在数值属性 $A_1,A_2,A_3,\cdots,A_f$ 共 f 个属性上建立的索引,首先将各属性 A_i 的值域切割为 c_i 个子域.如果属性取值为离散值,则可将每一个取值单独划分为一个子域;如果为其它类型,则可将其二进制表示作为其位串值,例如字符串中各字符的 ASCII 码组成的二进制位串;并按照二进制位串的数值大小对属性值域进行划分①.设值域切割所得子域所构成的集合为 C_i,那么子域的笛卡尔积 $Des_{1\cdots f}=C_1\times C_2\times C_3\times\cdots\times C_f$ 的大小为 $B=\prod_{i=1}^{f}c_i$.使用一个长为 $\sum_{i=1}^{f}c_i$ 的位串来表示每个元组.设元组 t 的属性 A_i 的第 j 个子域对应的位为 $b_{i,j}$,那么该元组所对应的位串可表示为

$$b_{1,1}b_{1,2}b_{1,3}\cdots b_{1,c_1},b_{2,1}b_{2,2}b_{2,3}\cdots b_{2,c_2},\cdots,b_{f,1}b_{f,2}b_{f,3}\cdots b_{f,c_f}.$$

给定任意元组 t 及其所对应的位串,$\forall i\in[1,f]$ 有且仅有唯一的一个 $j\in[1,c_i]$ 使得 $b_{i,j}=1$.因此该位串的所有合法取值可一一对应于上述子域笛卡尔积 $Des_{1\cdots f}$ 中的元素.从而,任意元组所对应的位串将拥有 $B=\prod_{i=1}^{f}c_i$ 个可能的取值.在给定属性列及其上的子域划分的情况下,将元组对应的位串的所有可能取值按照从小到大的顺序排序.位串的所有可能的取值可一一对应到一个排序值 $r\in[1,B]$ 上.这样,任意元组在被考察的数据字段上的取值均对应于一个唯一的排序值 r.至此,完成了对所有元组上的索引值的全局排序.下面以为某公司员工信息表中的两个属性建立索引为例,介绍从元组生成对应的位串,进而计算对应的排序值的过程.

例 1. 给定某公司的员工信息表,设该表属性 A_1 指示员工性别,包含 male 和 female 两个取值;属性 A_2 记录员工薪水,取值为[0,3000]范围内的整数.首先,属性 A_1 的值域被分割为两个子域,分别仅包括取值 male 和 female;属性 A_2 的值域被分割为 3 个子域:[0,1000],(1000,2000]和(2000,3000].考虑员工 Emp_1,设其性别为男性,薪水为 1300,那么该员工对应的位串为'10010'.其中前两位'10'表示属性 A_1 上取值为 male,后三位'010'表示其薪水在范围(1000,2000]内.再考虑员工 Emp_2,设其性别为女性,薪水为 2600,那么该员工对应的位串为'01001'.其中前两位'01'表示属性 A_1 上取值为 famale,后三位'001'表示其薪水在范围(2000,3000]内.根据以上对属性值域的划分,可知属性子域的笛卡尔积的大小为 $B=2\times3=6$.任意元组所对应的位串可能的取值有 6 个,将其按照从小到大的顺序排列可得'01001'、'01010'、'01100'、'10001'、'10010'和'10100'.对比可知,员工 Emp_1 对应的位串排在第 5 位,而员工 Emp_2 对应的位串排在第 1 位.因此,员工 Emp_1 和 Emp_2 对应的位串的排序值分别为 5 和 1.

3.1.2 索引的分片

类似于上述分布式方案,本方法同样采用各数据节点独立管理局部数据上索引的方式,以便提高检索执行时的并行度.在每个数据节点上,对该节点上存在的元组所对应的位串的全局排序值建立 B^+ 树.树的叶结点中的每个键都对应于一个排序值,并记录一个长度为本数据节点所管理的元组总数的位

① 值域的划分规则对利用本文所介绍索引进行查询的结果的正确性没有影响,但是会在一定程度上影响其查询性能.该现象对于位图索引而言是普遍存在的,对该影响的讨论超出了文本范畴.读者可参考文献[19]进一步了解相关内容.在本文中,统一按照数据的分布进行均匀划分,即各个子域中所包含的元组数量相等.

图，其中拥有该排序值的元组所对应的位被置 1，其它位被置 0. 这里将其称为元组位图. 易见，单个数据节点上的 B^+ 树的所有叶子结点上最多拥有 B 个键，因此最多存在 B 个不同的元组位图. 继续例 1 中的假设，例 2 介绍了某个数据节点上的局部位图索引的生成过程.

例 2.　关于员工信息表的假设同例 1. 如表 1 所示，设该表在某个数据节点上共有 7 条记录. 其中 Employee ID 是该表的主键，Bit String 和 Rank 不是原表中的属性，而是由性别和薪水两个属性上的值生成的位串及其对应的排序值. 依据各记录的排序值建立的 B^+ 树结构如图 3 所示（该 B^+ 树中的每个结点最多含有 3 个孩子结点）.

表 1　某数据节点上的初始元组及其位串值与排序值

Employee ID	Gender	Salary	Bit String	Rank
Emp_1	Male	1300	10010	5
Emp_2	Female	2600	01001	1
Emp_3	Female	2100	01010	1
Emp_4	Male	1900	10010	5
Emp_5	Male	2300	10001	4
Emp_6	Male	1100	10010	5
Emp_7	Female	900	01100	3

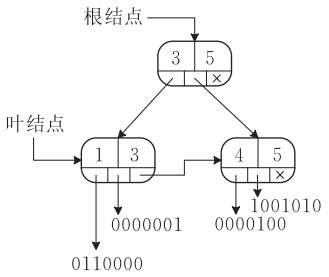


图 3　初始的局部 B^+ 树结构和各元组位图

3.1.3　索引的创建

综合运用以上两项技术，本节给出在云环境中分片位图索引的各组成部分以及其创建流程. 本方法的动机来源于吸取并综合集中式方案与分布式方案的优点. 首先对于给定的被索引属性上的取值，其对应位串存在唯一的全局排序值. 据此，利用局部存在于各数据节点上的数据所对应的排序值为其构造局部的指示位图. 该指示位图的长度为子域笛卡尔积的大小 B. 如果该数据节点上存在排序值为 r 的元组，则该数据节点上指示位图的第 r 位为 1，反之则为 0.

例 3.　关于员工信息表的假设同例 1. 考虑例 2 中所述数据节点. 由于 $B=6$，因此该数据节点上的指示位图长度为 6. 又由于该数据节点仅包含排序值为 1、3、4 和 5 的元组，因此其指示位图应为 101110.

指示位图记录了局部属性值的存在情况. 检索请求到达各数据节点时，首先通过比对指示位图来确定本数据节点是否包含目标元组，如果不存在，则直接返回空值，而不执行检索任务.

例 4.　关于员工信息表的假设同例 1. 考虑例 2 中所述数据节点. 由例 3 可知其指示位图为 101110. 假设用户发出了检索性别为女性且薪水范围在 (1000，2000] 内的员工的请求，据此可生成目标记录所对应的位串为 01010，其排序值为 2. 从而，构造位串 01000 与指示位图进行逻辑按位与 01000&101110，结果为全零. 因此该节点上不存在目标元组，可直接返回空集作为本节点的检索结果.

除了为每个数据节点构造指示位图之外，还应当建立局部数据上的位图索引. 该局部位图索引由两部分构成：以全局排序值为键的 B^+ 树以及 B^+ 树叶子结点上所附着的相应字段值上的局部元组位图. 在执行检索任务时，如果与指示位图进行比对的结果不为全零串，则需要触发 B^+ 树上的检索操作.

根据以上分析可知，指示位图的创建过程中需要扫描局部节点上所存在的所有元组，此过程的计算代价正比于局部数据节点上元组的数量；对于局部位图索引，B^+ 树的规模正比于子域笛卡尔积的大小 B，与元组数量无关，元组位图的创建过程也需要扫描局部节点上所存在的所有元组，因此其代价亦正比于节点所管理的元组数量. 通常情况下子域笛卡尔积的大小远小于元组数量，因此整个分片位图索引的创建代价正比于局部数据节点上所管理元组数量的最大值.

3.2　索引的维护

以下各小节将依次介绍插入新元组、删除或更新已存在的元组以及数据发生迁移时，分片位图索引的维护算法.

3.2.1　元组删除

当元组被删除时，需要对相应的指示位图和局部位图索引进行更新：首先根据元组的主键查找该元组所在的数据节点及其在该数据节点中的相对位置，依据该元组上的被索引属性的值所对应的排序值搜索该数据节点的局部位图索引 B^+ 树，设查找到的元组位图为 rb，生成长度为该数据节点上存放的所有元组总数的全 1 位串 tb，并置该元组对应位为 0. 然后，检查按位逻辑与 $rb\&tb$ 的计算结果是否为 0，并按照计算结果对以下两种情况分别进行处理：

(1) 如果为 0,则表示该元组是所找到的叶子结点上唯一的对应元组,因此元组删除后,该叶子结点及其对应的元组位图也应当被删除,B^+ 树的更新操作可在 $O(\log_m(B/m))$ 时间代价内完成,其中 m 是每个 B^+ 树结点的孩子结点数目的最大值,其具体算法步骤请读者参考文献[9],此处不再重复;同时更新指示位图,通过逻辑与操作,置以该元组对应的排序值为下标的位为 0.

(2) 否则,表示该元组不是拥有该排序值的唯一元组,因此无需更新 B^+ 树结构和指示位图.

最后,从 B^+ 树叶子结点首指针开始扫描所有元组位图,通过向左移位,将各元组位图中被删除元组对应的位删除. 例 5 展示了删除元组时更新索引的过程.

例 5. 关于员工信息表的假设同例 1. 假设用户发出删除元组 Emp_5 的请求. 首先找到该元组所在的数据节点,即例 2 所示的数据节点,依据元组 Emp_5 在属性性别和薪水上的值计算得到其对应的位图为 10001,排序值为 4,搜索该数据节点上的 B^+ 树得到对应的元组位图为 $rb=0000100$. 为元组 Emp_5 生成位串 $tb=1111011$. 由于 $rb\&tb=0000000$. 因此元组 Emp_5 是该元组位图上记录着的唯一元组,删除该元组之后应当随之删除该叶子结点;同时需要更新该数据节点上的指示位图,即将指示位图的第 4 位置为 0. 最后通过向左移位操作删除 B^+ 树上各元组位图的第 5 位. 元组删除之后的局部位图索引如图 4 所示,更新后的指示位图为 101010.

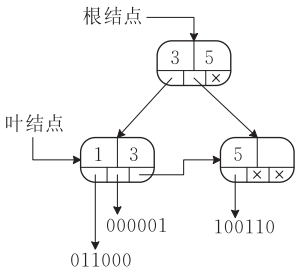


图 4 删除元组 Emp_5 后的局部 B^+ 树结构以及各元组位图

类似的,假设用户发出了删除元组 Emp_6 的请求. 注意由于元组 Emp_5 已被删除,此时该数据节点上记录着 6 条元组,其中元组 Emp_6 处于该局部数据节点的第 5 位. 由计算可知 $rb=100110$ 且 $tb=111101$. 从而,得知 $rb\&tb$ 不为 0,因而不需更新 B^+ 树的结构. 只需通过向左移位操作删除元组位图中的第 5 位. 删除后的 B^+ 树的结构如图 5 所示.

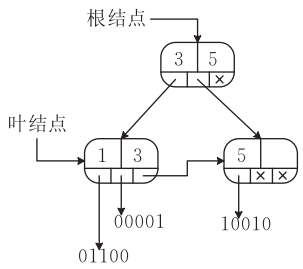


图 5 删除元组 Emp_6 后的局部 B^+ 树结构以及各元组位图

3.2.2 元组插入

当新元组被插入时,同样需要对相应的指示位图和局部位图索引进行更新:首先根据元组的主键查找该元组应当插入的数据节点,并计算新元组在该数据节点中所对应的相对位置. 从该数据节点上的局部位图索引 B^+ 树的叶子结点首指针开始扫描所有元组位图,通过向右移位操作,在各元组位图中新插入元组所对应的位置插入新的位,并置其值为 0. 然后,获取该数据节点上的指示位图 ib. 生成长度为 B 的全 0 位串 cb,并按照该元组相应属性的值计算其排序值 r,置第 r 位为 1. 最后,检查按位逻辑与 $eb\&cb$ 的计算结果是否为 0,并按照计算结果分两种情况进行处理:

(1) 如果为 0,则表示该元组上被索引的属性值在该数据节点上为首次出现. 首先更新指示位图为原指示位图 ib 和位串 tb 的按位逻辑或:$ib\leftarrow ib|tb$. 然后将排序值 r 插入到该数据节点的局部 B^+ 树中,并为排序值 r 构造长度等于该数据节点上所管理元组总数的全 0 元组位图,并将被插入元组对应的位置为 1. 同时,置指示位图的第 r 位为 1.

(2) 否则,表示该叶子结点上已存在拥有相同的属性排序值的其它元组,因此不需要更新指示位图和 B^+ 树结构. 而需要按照排序值搜索该数据节点上的局部 B^+ 树,并设置所找到的叶子结点上所附着的元组位图中新插入元组所对应的位为 1.

例 6. 关于员工信息表的假设同例 1. 根据例 5,元组 Emp_5 和 Emp_6 已被删除. 假设用户发出插入元组 $Emp_8=\{Gender=\text{female},\ Salary=1700\}$ 的请求. 首先找到该元组应当被插入的目标数据节点,假设存储引擎决定将其插入到例 2 所述的数据节点中. 由主键值 Emp_8 可知,该元组应当位于本数据节点所维护的局部数据表的末端. 遍历如图 5 所示的 B^+ 树的所有叶子结点上附着的元组位图,并将它们在末尾增加一位. 依据元组 Emp_8 在属性性别和薪水上的值计算得到其对应的位图为 01010,其排序值为 2. 生成长度为 6 的条件位串 $cb=010000$,由例 5 可知其指示位图为 $ib=101010$. 计算按位逻辑与

$ib\&\sim cb=000000$，可知排序值 2 在该数据节点上为首次出现. 因此更新指示位图 $ib\leftarrow ib|cb=111010$，并将排序值 2 插入到局部 B^+ 树(如图 5 所示)中，为其构造元组位图：010000. 至此，更新后的 B^+ 树如图 6 所示.

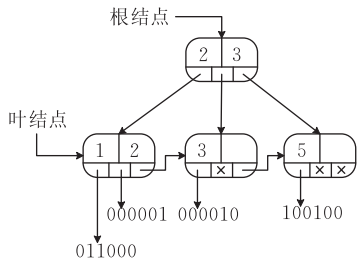


图 6　插入元组 Emp_8 后的局部 B^+ 树结构以及各元组位图

3.2.3　元组更新

如果该元组被更新的域包括主键，则执行一次该元组的删除操作，随后执行一次新元组的插入操作. 否则，元组的主键未被更新，可以通过一次原地更新操作来完成索引的维护. 原地更新的执行流程如下所述：首先根据元组的主键查找该元组所在的数据节点. 然后依照旧元组的删除操作中的步骤依次对局部 B^+ 树的结构和指示位图进行更新；依照插入新元组操作中的步骤依次对局部 B^+ 树的结构和指示位图进行更新. 最后由存储引擎更新该元组上的相应属性值. 注意，由于被更新的元组在局部数据节点中的相对位置没有发生变化，因此无需扫描并更新所有的元组位图.

3.2.4　数据迁移

在云环境中，为了平衡负载和保证充足的计算资源，数据的动态扩张将导致数据迁移操作被触发. 随着元组的宿主节点的改变，参与迁移的数据节点上的索引结构也将发生变化. 本文使用元组的删除和插入操作来完成数据迁移时索引结构的更新. 例如欲将元组 t 从数据节点 $Node_1$ 迁移到 $Node_2$ 上，需要在 $Node_1$ 上执行元组 t 的删除操作并更新 $Node_1$ 上的指示位图和局部位图索引，然后在 $Node_2$ 上执行元组 t 的插入操作并更新 $Node_2$ 上的指示位图和局部位图索引.

3.2.5　维护效率

从以上维护算法中可以看出，单条元组的插入、删除和更新所引起的索引维护代价相对于系统所管理的数据规模是常数级别的. 因此即便在数据规模异常庞大情况下，面对频繁的元组插入、删除和更新请求，索引维护所带来的额外开销并不会引起数据管理系统计算代价的显著升高.

3.3　检索算法

本节介绍当集群收到带有条件的查询请求时，利用索引完成查询的步骤. 首先介绍单个检索条件下的检索算法. 然后介绍对于复合的检索条件利用位图的逻辑运算在单个条件的检索算法基础上快速获得最终计算结果的方法.

3.3.1　单个索引上的检索条件

对于查询请求中的单个索引上检索条件，主节点首先将检索条件转换为条件位串. 例如，检索女性薪水为 1500 到 1800 之间的员工将得到位串 01010；又如，检索条件薪水低于 1300 的男性员工将被转换为位串 10110. 注意，生成的条件位串应当覆盖检索条件所包含的所有可能性. 然后主节点将条件位串并行地发送到所有的数据节点上，各数据节点分别将条件位串转换为对应的排序值的范围，接着生成长度等于 B 的全 0 位串 cb，并将属于排序值范围内的位置为 1. 检查按位逻辑与 $eb\&cb$ 的计算结果是否为 0. 如果为 0 则在该数据节点上直接返回空集作为计算结果；否则，在该数据节点的局部 B^+ 树上依次搜索各符合条件的排序值，将找到元组位图进行按位逻辑与，一一检查其计算结果中被置为 1 的位所对应的元组是否满足检索条件，最后返回所有满足条件的结果作为计算结果. 算法 1 展示了分片位图索引上对于单个检索条件的检索算法的伪代码.

算法 1.　分片位图索引在单一检索条件下的检索算法.

输入：给定查询属性 A_j 上的划分 $A_{j_1}, A_{j_2}, \cdots, A_{j_f}$ 以及其上的查询条件：$\mathrm{Cond}(A_j)$

输出：满足查询条件的元组的主键

算法步骤：

1. 主节点收到查询请求.
2. 基于 RBI 索引在属性 A_j 上的分片规则，由查询条件 $\mathrm{Cond}(A_j)$ 生成条件位图 cb，并计算其相应的排序值集合 r.
3. 将条件位图 cb 和排序值集合 r 发送至所有计算节点.
4. 各计算节点并行执行步 5～13：
5. 　取得局部指示位图 ib；
6. 　计算逻辑按位与 $ib\&\sim cb$；
7. 　如果计算结果为全 0 串则该节点直接返回空集 $\varnothing$；
8. 　否则，顺序执行步 9～13：
9. 　　在局部 B^+ 树上搜索 r 中的每一个排序值；
10. 　　获取所有的命中叶结点上的元组位图 $rb_{1\cdots t}$；
11. 　　计算 $rb\leftarrow\bigcup_{i=1}^{t} rb_i$；
12. 　　对于 rb 中每个 1 位：
13. 　　　如果该位所对应的元组 rec 满足条件 $\mathrm{Cond}(A_j)$ 则返回其主键 $rec.pk$；
14. 主节点向用户返回所有计算节点返回结果的并集

3.3.2　检索条件的复合

通过将在同一数据节点上的不同属性上检索得

到的元组位图进行相应的按位逻辑运算,可以实现多个检索条件的快速复合.这充分利用了现代计算机能够对位图快速操作的计算优势.具体而言,其执行步骤为:首先主节点将各简单检索条件分别转换为条件位串,并将这些条件位串以及条件之间的复合关系分发到各个数据节点.然后,各数据节点分别依次执行各简单检索条件上的检索流程,从而查找到各简单条件所对应的元组位图,并将这些元组位图依照检索条件复合方法执行按位逻辑运算,得到最终的元组结果位图.最后,扫描该位图中为被置为 1 的位,并向主节点返回相应的满足检索条件的元组.

4 实 验

本节给出在云环境中,本文所提出的分片位图索引与现有基本算法在查询效率和查询吞吐量上的测试结果,以验证本文所介绍算法的有效性.

4.1 实验环境与设置

我们在一个拥有 4 台分布式计算节点的集群上部署了一个小型的云环境.4 台服务器位于同一个局域网中,相互之间通过千兆以太网连接.各计算节点的配置及其在集群中的角色如表 2 所示.

表 2 对比实验所使用的集群环境

主机名	CPU	内存	磁盘空间	操作系统	数据节点	控制节点
Master	两颗 Quad-Core AMD Opteron™ Processor 2378 频率 2.4GHz	16GB	1.4TB	Ubuntu Server 64bit 10.04.4 LTS	是	是
Slaver1	四颗 Dual-Core AMD Opteron™ Processor 865 频率 1.8GHz	32GB	900GB	Ubuntu Server 64bit 10.04.4 LTS	是	否
Slave2	两颗 Quad-Core AMD Opteron™ Processor 2378 频率 2.4GHz	16GB	1.4TB	Ubuntu Server 64bit 10.04.4 LTS	是	否
Slave3	四颗 Dual-Core AMD Opteron™ Processor 865 频率 1.8GHz	4GB	160GB	Ubuntu Server 64bit 10.04.4 LTS	是	否

另外,Master 主机上部署了 MySQL 5.0 系统以便于与传统关系型数据库管理系统进行对比.

本文采用了如表 3 所示的 2 个数据集作为实验数据.各数据集均以 *RowKey* 字段的值的大小顺序来组织数据,并且在 *RowKey* 字段上创建有聚集索引.数据集 *A* 中的 *fromUser*、*toUser* 字段和数据集 *B* 中的 *ref* 字段均创建有分片位图索引.实验采用以下 4 种算法与本文所提出的分片位图索引(RBI)进行对比:

(1)相同数据内容存储于 MySQL 中,相应属性列无索引(MySQL Query,MQ).

(2)并发的全库扫描方式(Full Scan,FS).

(3)如第 2.1 节所述的辅助索引的集中式方案(Global Approach,GA)的基本实现.

(4)如第 2.2 节所述的辅助索引的分布式方案(Distributed Approach,DA)的基本实现.

表 3 对比实验所使用的数据集

数据集	元组数目	大小	数据内容	属性数目	模式
A	4.5 亿	52GB	Twitter 微博转发关系	3	*RowKey*:整型 ID 编号,主键 *fromUser*:转发用户名,字符串类型 *toUser*:被转发用户名,字符串类型
B	162 万	1TB	自动生成的多媒体数据	4	*RowKey*:整型 ID 编号,主键 *text*:描述信息,字符串类型 *data*:多媒体数据,二进制数据类型 *ref*:引用次数,整型数值

对于各对比算法,本实验使用了相同的查询集.查询集按照命中与不命中数量比例 1∶1 的原则随机生成.对不同的目标属性生成了不同的查询集.各项实验向系统连续发起查询集中的随机挑选出的查询请求.对于同一组对比实验中不同的检索算法,各查询请求按照相同的顺序发出.

4.2 分片位图索引的查询性能

实验中分别为 *A*、*B* 两个数据集手工挑选了等值条件、非等值条件、复合条件各 10 条查询来组成查询请求集.以下图中记录了处理单条查询的平均响应时间.实验中以 10 个客户端同时连续地发起查询请求的方式对各算法的查询吞吐量进行测试.吞吐量以每分钟处理的查询数量来表示.

图 7 和图 8 分别列出了以数据集 *A* 的 *fromUser* 属性上值的比较为条件的查询请求以及以 *fromUser* 与 *toUser* 两个属性上的复合检索条件构成的查询请求的检索效率与查询吞吐量.可以看到,云环境中各种索引的检索方法(GA、DA 和 RBI)都拥有优秀的

单条查询响应速度.其中，能够以并行方式执行检索任务的 DA 和 RBI 算法表现更加出色.在密集的查询请求并发地、连续地到达时，无索引的检索方法(MQ 与 FS)效率不佳.集中式方法(GA)的查询吞吐量也表现较差.这一现象在复合检索条件情况下尤其突出.对此，不难推断其原因在于集中式方法中检索算法无法被并行化且检索过程会触发较多的外存 I/O.

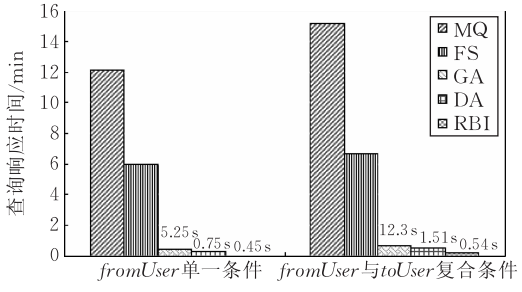


图 7　数据集 A 上的检索效率

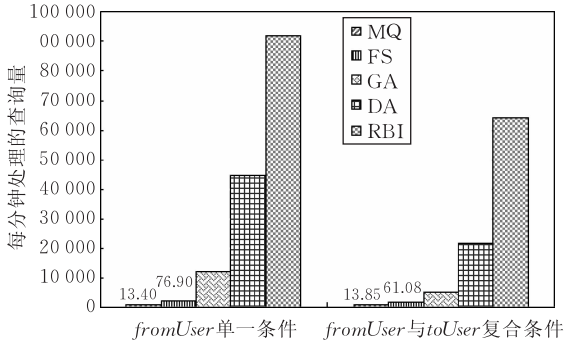


图 8　数据集 A 上的查询吞吐量

图 9 和图 10 分别列出了以数据集 B 的 ref 属性上值的比较为条件的查询请求的检索效率与查询吞吐量.由于数据集 B 数据规模较大，向 MySQL 导入数据的过程消耗了超过一周的时间，因此本文未对数据集 B 在 MySQL 上进行查询实验.总体而言在数据集 B 上各方法的性能的相对关系与数据集 A 上获得的结果类似.通过对比，尽管数据集 B 拥有的元组数目少于数据集 A，但是由于数据总规模增大了数十倍，数据访问所带来的开销显著升高，全

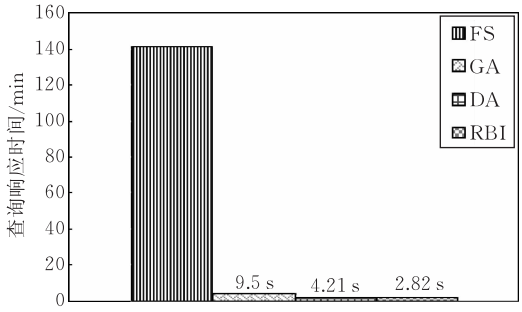


图 9　数据集 B 的 ref 属性上的检索效率

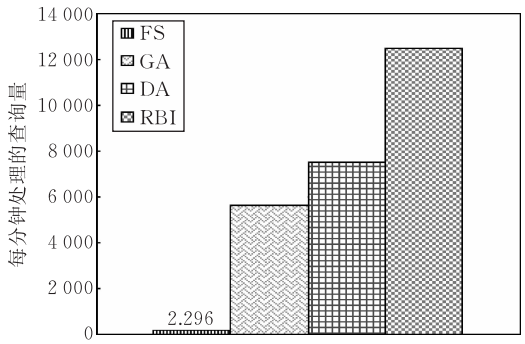


图 10　数据集 B 的 ref 属性上的查询吞吐量

库扫描方法(FS)的检索效率被成倍地降低了，而对于其它使用索引来进行检索方法而言，所受到的影响相对较小.相比之下，分片位图索引(RBI)依然保持着显著的优势.

概括而言，两组数据集上所得到的实验结果是基本一致的：得益于云环境的强大计算资源，即便是全表扫描算法(FS)，相对于传统单节点数据库上的粗暴检索算法(MQ)也具有非常明显的优势.对比云环境中各种索引集中上的检索算法，可以发现，本文提出的分片位图索引技术(RBI)在查询的响应时间和吞吐量上都大大优于其竞争对手，这一优势在复合检索条件情况下尤其突出.

5　总　结

本文以云环境的成熟发展为契机，着眼于云环境中大规模数据上的辅助索引机制的设计与实现.通过分析对比了两种截然不同的云环境中辅助索引的基本逻辑结构，即集中式方案与分布式方案，在吸收二者的优势并规避其缺陷的基础上，提出了具有良好可扩展性的分片位图索引技术来对云环境中海量数据的检索任务提供高效支持.其特点在于充分发挥两种基本方案中的优势，既能够使云环境中并行计算资源得到很好的利用，从而提高了单条查询的响应速度，又能够通过局部节点所掌握的全局信息而规避不必要的检索代价从而使在大量检索请求并发到达时查询吞吐量得以保证.通过在真实数据上进行实验测试，分片位图索引被证实拥有优于其它方法的性能表现.

参　考　文　献

[1] Armbrust Michael, Fox Armando, Griffith Rean et al. A view of cloud computing. Communications of the ACM, 2010, 53(4): 50-58

[2] Yang H-C, Dasdan A, Hsiao R-L, Parker D S. Map-reduce-

merge: Simplified relational data processing on large clusters//Proceedings of the SIGMOD 2007. Beijing, China, 2007: 1029-1040

[3] Chowdhury N M Mosharaf Kabir, Boutaba Raouf. A survey of network virtualization. Computer Networks, 2010, 54 (5): 862-876

[4] Seshadri P, Pirahesh H, Leung T Y C. Complex query decorrelation//Proceedings of the ICDE. New Orleans, LA, 1996: 450-458

[5] Canahuate Guadalupe, Apaydin Tan, Sacan Ahmet, Ferhatosmanoglu Hakan. Secondary bitmap indexes with vertical and horizontal partitioning//Proceedings of the EDBT. Saint Petersburg, Russia, 2009: 600-611

[6] Sadoghi Mohammad, Jacobsen Hans-Arno. Be-tree: An index structure to efficiently match boolean expressions over high-dimensional discrete space//Proceedings of the SIGMOD Conference. Athens, Greece, 2011: 637-648

[7] Chang Fay, Dean Jerey, Ghemawat Sanjay et al. Bigtable: A distributed storage system for structured data//Proceedings of the OSDI. Seattle, Washington, USA, 2006: 205-218

[8] Apache HBase Project. http: //hbase. apache. org/

[9] HBase Transactional Index. https: //github. com/hbase-trx/hbase-transactional-tableindexed

[10] Aguilera Marcos Kawazoe, Golab Wojciech M, Shah Mehul A. A practical scalable distributed B-tree//Proceedings of the VLDB. Auckland, New Zealand, 2008: 598-609

[11] Goetz Graefe, Kuno Harumi A. Modern B-tree techniques//Proceedings of the ICDE. Hanover , Germany, 2011: 1370-1373

[12] Chen Gang, Vo Hoang Tam, Wu Sai, Ooi Beng Chin, Ozsu M Tamer. A framework for supporting DBMS-like indexes in the cloud//Proceedings of the VLDB. Seattle, Washington, USA, 2011: 702-713

[13] Levandoski J J, Lomet D B, Mokbel M F, Zhao K. Deuteronomy: Transaction support for cloud data//Proceedings of the CIDR. Asilomar, California, 2011: 123-133

[14] Zhang Xiangyu, Ai Jing, Wang Zhongyuan, Lu Jiaheng, Meng Xiaofeng. An efficient multi-dimensional index for cloud data management//Proceedings of the CloudDB. Beijng, China, 2009: 17-24

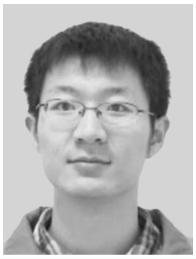
[15] Indexed HBase Prjoect. http: //svn. apache. org/repos/asf/hbase/branches/0. 20 _ on _ hadoop-0. 18. 3/src/contrib/indexed/src/java/org/apache/hadoop/hbase/client/idx/package. html

[16] Wang Jinbao, Wu Sai, Gao Hong, Li Jianzhong, Ooi Beng Chin. Indexing multi-dimensional data in a cloud system//Proceedings of the SIGMOD Conference. Indianapolis, Indiana, 2010: 591-602

[17] Wu Sai, Jiang Dawei, Ooi Beng Chin, Wu Kun-Lung. Efficient B-tree based indexing for cloud data processing//Proceedings of the VLDB. Singapore, 2010: 1207-1218

[18] Jagadish H V, Ooi B C, Vu Q H. Baton: A balanced tree structure for peer-to-peer networks//Proceedings of the VLDB. Trondheim, Norway, 2005

[19] Rotem Doron, Stockinger Kurt, Wu Kesheng. Efficient binning for bitmap indices on high-cardinality attributes. Lawrence Berkeley National Laboratory, U. S. Department of Energy, Berkeley, California, USA, 2004



MENG Bi-Ping, born in 1986, M. S. . His research interest is cloud storage system.

WANG Teng-Jiao, born in 1973, Ph. D. , professor, Ph. D. supervisor. His research interests include public opinion analysis, large database managment system.

LI Hongyan, born in 1970, Ph. D. , professor, Ph. D. supervisor. Her research interests include auto-construction environment for Web information system, domain specific information system and integration, intelligent information system and object-relational database technology and applications.

YANG Dong-Qing, born in 1945, Ph. D. , professor, Ph. D. supervisor. Her research interests include database system implementation techniques, information integration and sharing in Web environment, data warehousing and data mining.

Background

In recent years, the rapid development of World Wide Web has caused dramatic increment of user generated data. The demand of large-scale data processing is posing unprecedented tough challenges to data management systems. Traditional data management systems fail to handle the new problems raised by massive data processing tasks. Fortunately, the fast development of Cloud Computing technologies has brought new dawns to the storage and management of massive data. Nevertheless, due to the essential changes in the storage model, the matured indexing techniques used in traditional relational data management systems can neither be directly applied to massive data, nor be migrated to Cloud environment in an easy way. This paper focuses on the design of a secondary indexing mechanism aiming to facilitate the process of storing, querying and mining of large-scale data. The proposed method, i. e. the regional bitmap index is able to make the best of cloud computing resource and proven to scale well in face of cloud data against various queries. By comparing to the state-of-art methods in experiments on real dataset, our method is reported to outperform its competitors in terms of both query response time and throughputs. This work is supported by National High Technology Research and Development Program (863 Program) of China (Grant Nos. 2012AA011002, 2011AA010706), National Science and Technology Major Program (Grant Nos. 2010ZX01042-002-002-02, 2010ZX01042-001-003-05), Natural Science Foundation of China (Grant Nos. 60973002, 61170003, 61073018) and Shenzhen-Hong Kong Innovation Cooperation Project (Grant No. JSE201007160004A).