软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(12):3067−3084 [doi: 10.13328/j.cnki.jos.005013]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

一种基于共享执行策略的间隔查询优化技术*

周 新[1,2], 张 孝[1,2], 薛忠斌[1,3], 王 珊[1,2]

[1](数据工程与知识工程教育部重点实验室(中国人民大学),北京 100872)

[2](中国人民大学 信息学院,北京 100872)

[3](神华国华(北京)电力研究院有限公司,北京 100069)

通讯作者: 张孝, E-mail: zhangxiao@ruc.edu.cn

摘 要: 间隔查询作为重要的查询类型,广泛应用在社交网络、信息检索和数据库领域.为了支持高效的间隔查询,涌现出多种优化技术.尽管已有方法能够快速响应单个间隔查询,然而当查询负载超过服务器的处理能力时,70%的查询均不能在期望时间内得到响应.针对这一问题,提出采用共享执行策略优化间隔查询的方法 SESIQ(shared execution strategy for interval queries).SESIQ 对间隔查询进行批处理,分析一组间隔查询间可共享的操作,减少重复数据的访问,从而降低磁盘 I/O 和网络传输代价,提高检索性能.理论分析并实验验证了 SESIQ 的可行性,基于两种真实数据集的大量实验结果表明,SESIQ 是有效的,间隔查询的检索性能可提升数十倍.

关键词: 间隔查询;优化;共享执行

中图法分类号: TP311

中文引用格式: 周新,张孝,薛忠斌,王珊.一种基于共享执行策略的间隔查询优化技术.软件学报,2016,27(12):3067−3084. http://www.jos.org.cn/1000-9825/5013.htm

英文引用格式: Zhou X, Zhang X, Xue ZB, Wang S. Technique based on shared execution strategy for optimizing interval query. Ruan Jian Xue Bao/Journal of Software, 2016,27(12):3067−3084 (in Chinese). http://www.jos.org.cn/1000-9825/5013.htm

Technique Based on Shared Execution Strategy for Optimizing Interval Query

ZHOU Xin[1,2], ZHANG Xiao[1,2], XUE Zhong-Bin[1,3], WANG Shan[1,2]

[1](Key Laboratory of Data Engineering and Knowledge Engineering of the Ministry of Education (Renmin University of China), Beijing 100872, China)

[2](School of Information, Renmin University of China, Beijing 100872, China)

[3](Guohua (Beijing) Electric Power Research Institute Co. Ltd., Beijing 100069, China)

Abstract: As an important query type, interval query is widely used in social networks, information retrieval and database domain. Many kinds of optimization techniques have sprung up to support effective interval query. Although existing methods are efficient to handle single query, they all suffer from performance problem when the concurrent query loads exceed the processing capacity of the server such that more than 70% queries couldn't receive the results in the expected time. To solve this problem, this paper presents a method named SESIQ (shared execution strategy for interval queries). SESIQ batches interval queries, analyzes common operations among a group of interval queries and reduces duplicate data access to lower the cost of disk I/O and network transmission. The paper theoretically studies and analyzes SESIQ, and demonstrates the feasibility by large number of experiments based on two types of real datasets. Results show that SESIQ improves the performance of interval query by several ten folds.

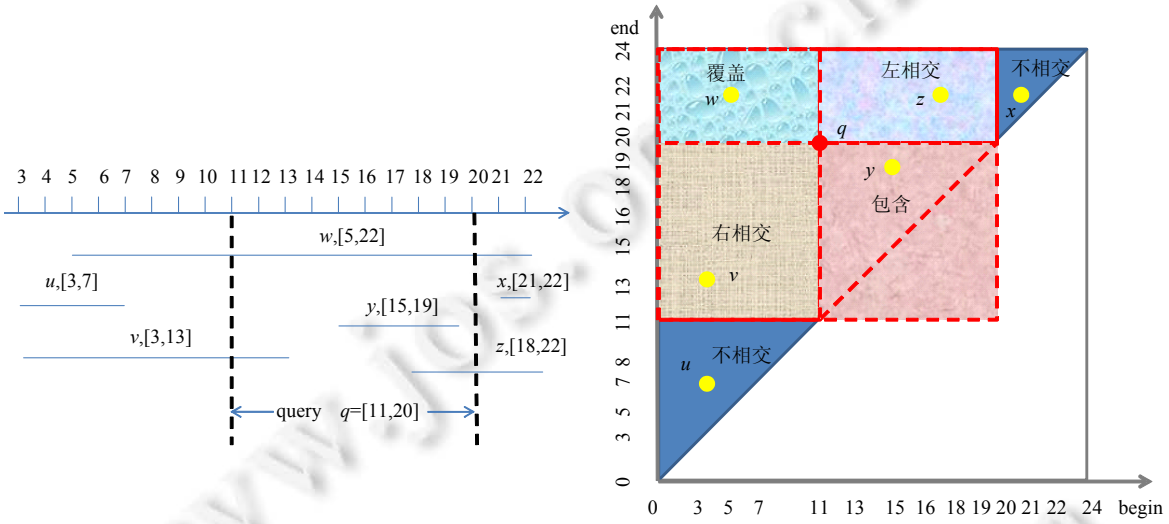
Key words: interval query; optimization; shared execution

* 基金项目: 国家自然科学基金(61432006); 中国人民大学科学研究基金(中央高校基本科研业务费专项资金)(10XNI018)

Foundation item: National Natural Science Foundation of China (61432006); Fundamental Research Funds for the Central Universities, and the Research Funds of Renmin University of China (10XNI018)

收稿时间: 2015-09-21; 修改时间: 2015-11-16; 采用时间: 2015-12-14

间隔查询作为重要的查询类型,在多个领域得到广泛应用,如信息检索领域查找某个时间内所有 Web 文档的所有版本.Web 文档在不断更新,因而网络爬虫在不同时刻抓取到的 Web 文档不同,相邻的 2 次抓取时间形成一个间隔,构成文档的一个版本.图 1 给出仅有 1 个版本的 6 篇文档,现在查找“时间间隔 q 内的所有 Web 文档的所有版本”.由于网络爬虫仅记录 Web 文档的抓取时间,间隔信息是隐含的,且间隔查询有别于范围查询,研究间隔查询优化技术是一件有趣且富有挑战的事情.而随着间隔数据增多,间隔查询负载增大,实时地响应间隔查询的任务更加艰巨.



(a) 线段表示的一维间隔查询示例 (b) 二维点表示的一维间隔查询示例

Fig.1 Sample of interval queries

图 1 间隔查询示例

为了高效处理间隔查询,多种集中式/分布式索引和检索算法被提出.现代技术的发展使得集中式索引已不能满足快速增长的间隔数据的存储和检索需求,基于分布式系统(如 Bigtable[1],HBase[2],PNUTS[3],Cassandra[4],Dynamo[5]等)的分布式间隔索引能够适应海量实时间隔数据的需求而成为研究热点.尽管分布式间隔索引能够高效地响应单个间隔查询,但是当海量间隔查询急速到来时,超过 70%[6]的用户不能在期望时间内得到反馈,这正是本文所关注的问题.

图 2 展示了单台服务器中查询负载对查询响应时间的影响[6].

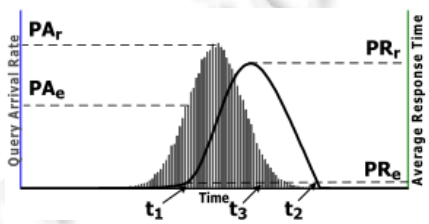


Fig.2 Respond time varies with the query load on single server

图 2 单台服务器的查询响应时间随查询负载的变化[6]

图 2 中,PA_r 为真实情况下查询负载的峰值,PA_e 为服务器处理能力的峰值,PR_r 为实际最长响应时间,PR_e 为用户期望的最长响应时间.t_1 时刻前,服务器的处理能力大于查询到来的速度,服务器能够很快响应查询;t_1 时刻之后到来的查询数超过服务器的处理能力,此时查询在队列中累积,t_3 时刻,队列中累积的查询最多,查询的平均响应时间最长,远远超过用户期望的响应时间 PR_e;t_3 之后到来的查询数小于服务器的处理能力,查询的响应时间

下降,直到 t_2 时刻查询的响应时间才符合用户期望.t_1 到 t_2 范围内提交的查询均不能在期望时间内得到反馈.

产生这种现象的主要原因是,服务器采用每次处理一个(one-by-one)的方式处理间隔查询.当大量查询到来时,查询需要排队以等待服务器响应查询,查询等待时间在查询响应时间占更大比例.同时,间隔查询属于数据密集型任务,涉及频繁的数据访问和网络通信.One-by-One 的处理模式需要多次访问相同的数据并传递到客户端,浪费磁盘 I/O 和网络资源,严重影响查询的检索性能.

为了减少查询等待时间、避免重复访问和传递相同的数据,本文提出共享执行策略优化间隔查询的方法 SESIQ(shared execution strategy for interval queries),解决分布式间隔索引不能适应单点上的高并发间隔查询的问题.SESIQ 的思想是:服务器采用每次处理一组(group-by-group)的方式处理接收到的查询,通过复用一组查询间的公共操作,从而减少查询的响应时间,提高服务器响应查询的能力.SESIQ 面临的挑战在于分析一组间隔查询间可共享的操作,生成全局访问计划并执行.本文主要有 3 点贡献.

(1) 首次采用共享执行策略优化间隔查询.本文从查询特性出发,首次提出基于共享执行策略的间隔查询优化技术 SESIQ.SESIQ 和基于索引技术优化间隔查询的方法是正交的,以已有的索引结构为基础,进一步优化间隔查询;

(2) 设计并实现了 SESIQ 引擎,给出关键技术的算法实现和时空开销分析.SESIQ 引擎能够分析一组查询间的特性,为该组查询建立全局访问计划并执行,从而减少重复数据的访问和网络通信开销,降低响应时间.本文给出了详细的算法实现阐述 SESIQ 批量处理间隔查询的过程.同时,提供了形式化方法分析 SESIQ 的性能收益以及空间开销;

(3) 实验证明共享执行策略的有效性和高效性.基于真实数据集的大量实验评估表明:SESIQ 是有效且高效的,共享执行策略能够显著提升检索性能,间隔查询的响应时间降低数十倍.

1 问题阐述

以一维间隔查询为例,阐述本文要解决的问题,多维间隔查询可以从一维间隔查询扩展后获得.本节给出间隔的表示方法并探索两个间隔间的关系,提出间隔查询、范围查询和覆盖查询的形式化描述,分析三者之间的关联,最后定义间隔查询间的共享关系.

定义 1(间隔). 对于一维间隔,本文给出两种不同表示.间隔 I 可以由起始点 $begin$ 和终止点 end 确定($begin \leqslant end$),标记 $I=[begin,end]$,如图 1(a)所示;也可以由二维空间中的二维点 $I(begin,end)$表示,这个二维空间由起始维和终止维构成,$begin$ 位于起始维,end 位于终止维,如图 1(b)所示.由于 $begin \leqslant end$,间隔数据仅分布在有颜色填充的等腰直角三角形区域.

定义 2(间隔间的关系). 依据间隔端点之间的关系,定义间隔 $I=[begin,end]$和间隔 $q=[a,b]$的 5 种关系,分别为覆盖、左相交、右相交、包含和不相交,I 和 q 具有前 4 种关系的任一种,均表明 I 和 q 是相交的.

(1) 覆盖:如果 $begin \leqslant a$ 且 $end \geqslant b$,则称 I 覆盖 q;

(2) 左相交:如果 $begin \geqslant a$ 且 $begin \leqslant b$,即 I 的左边部分在 q 范围内,则称 I 左相交 q;

(3) 右相交:如果 $end \geqslant a$ 且 $end \leqslant b$,即 I 的右边部分在 q 范围内,则称 I 右相交 q;

(4) 包含:如果 $a \leqslant begin \leqslant end \leqslant b$,则称 I 被 q 包含,即 q 覆盖 I;

(5) 不相交:如果 $b<begin$ 或 $end<a$,则称 I 和 q 不相交.

定义 3(间隔查询). 间隔查询可以用三元组(q,D,O)形式化表示,其中:q 表示查询间隔;D 表示检索目标,由多个对象组成,D 中的每个对象 o 由二元组(id,I)构成,id 标识对象 o,I 是对象 o 的间隔;O 为检索结果集合,对于 O 中的任意对象 $o \in O$,$o.I$ 和 q 相交.

当查询间隔 q 的起点等于终点,即 $a=b$ 时,这样的查询又称为点查询.

定义 4(范围查询). 给定一个查询间隔 q,范围查询从对象集合 D 中返回所有在查询范围内出现的对象构成集合 R,对于 R 中的任意对象 $o \in R$,$o.I$ 和 q 满足定义 2 中第(2)种~第(4)种这 3 种关系的任意一种.类似于间隔查询,范围查询可用三元组(q,D,R)表示.

定义 5(覆盖查询). 给定一个查询间隔 q,覆盖查询从对象集合 D 中返回所有间隔 I 覆盖 q 的对象形成结果集 C,使用三元组标记为(q,D,C).

定义 6(间隔查询的一维转换). 依据范围查询、覆盖查询以及间隔查询的定义可知:间隔查询可以分解为范围查询和覆盖查询,2 个子查询可并行执行.间隔查询的一维转换形式化描述为$(q,D,O)\rightarrow(q,D,R)+(q,D,C)$.

定义 7(间隔查询的二维转换). 对于二维点 $q(a,b)$表示的任意间隔查询 $q=[a,b]$,其检索空间由左下角$(begin,a)$、右上角(b,end)定义的矩形框确定,其中,$begin/end$ 为数据阈值的最小值/最大值.一维间隔查询 $q(a,b)$和二维范围查询 $q'=\langle(begin,a),(b,end)\rangle$的检索空间是一致的.间隔查询的二维转换形式化描述为

$$(q,D,O)\rightarrow(q'=\langle(begin,a),(b,end)\rangle,D,R).$$

例如,间隔查询 $q(11,20)$的检索空间如图 1(b)红色矩形框所示,其检索空间和范围查询 $q'=\langle(0,11),(20,24)\rangle$的检索空间一致.

定义 8(间隔查询间的共享). 当间隔查询 $q_1=[a,b]$和 $q_2=[c,d]$相交时,q_1 和 q_2 部分检索结果相同,则称二者是可共享的.q_1 和 q_2 进行一维转换后,产生范围查询 $q_{1R}=[a,b],q_{2R}=[c,d]$以及覆盖查询 $q_{1C}=[a,b],q_{2C}=[c,d]$.当 q_{1R} 和 q_{2R} 相交时,q_{1R} 和 q_{2R} 是可共享的;当 q_{1C} 和 q_{2C} 存在覆盖关系,q_{1C} 和 q_{2C} 是可共享的.q_1 和 q_2 进行二维转换后,产生范围查询 $q_1'=\langle(begin,a),(b,end)\rangle,q_2'=\langle(begin,c),(d,end)\rangle$.$q_1'$ 和 q_2' 相交时,q_1 和 q_2 是可共享的.

例 1:如图 1 所示,数据集合 D 中包含 u,v,w,x,y,z 共 6 个间隔对象,对于查询间隔 $q=[11,20]$,w 覆盖 q,z 左相交 q,v 右相交 q,y 被 q 包含,最终结果集 $O=\{v,w,y,z\}$.对于范围查询 $q=[11,20]$和覆盖查询 $q=[11,20]$,检索结果集分别为 $R=\{v,y,z\},O=\{w\}$.图 1 的数据集合 D 将作为间隔查询的检索目标贯穿全文实例,例 2 将详细阐述间隔间的共享关系.

2 相关工作

间隔查询的执行经过解析查询、生成访问计划、执行访问计划这 3 个步骤,如后文图 3 所示.已有工作通过设计合理的索引结构提高数据访问速度和分析查询特性共享公共操作两种途径提高间隔查询的检索性能.下面将分别考察基于索引和查询特性的优化技术.

2.1 基于索引的优化技术

依据数据存储在单台机器还是分布式平台,将支持间隔查询的索引分为集中式和分布式两种.

(1) 集中式索引

早期的数据量小,索引存储在单台机器中,这种索引称为集中式索引.支持间隔查询的集中式索引本主要分为以下 3 类,文献[7]对这些索引结构和数据访问方法进行了详细对比.

- 基于 B+-tree 的索引.典型的扩展 B+-tree 的间隔索引是 Time Index[8].Time Index 是为了检索某个间隔内有效的所有对象版本而设计的间隔索引.Time Index+[9]作为 Time Index 的改良版降低了其 $O(n^2)$(n 为间隔数)的空间复杂度.Interval B-tree[10](IB-tree)进一步解决 Time Index 和 Time Index+空间开销大的问题,提出新的结构 augmented B+-tree 代替二叉树结构.TD-tree[11]为了提高开区间的间隔查询的检索性能而提出的新结构,它用三角形二分的方式分解间隔查询的检索空间,并编码路径信息提高检索效率;
- 基于 R-tree 的索引.间隔查询可以转换为多维范围查询,因此,多维索引支持间隔查询,R-tree 和 R-tree 变种(Segment R-tree[12],4R-tree[13]等)主要支持多维时间数据的间隔查询,可用在双时态数据库[14]中.由于时间数据自身的特点导致时间间隔的重叠性较大,R-tree 索引的检索性能较低,不适合间隔查询;
- 基于 Segment-tree 的索引.Segment-tree 是一种高效且通用的间隔索引.它不仅支持一维数据的检索还适合多维数据.Segment-tree 是二叉树,结点采用间隔作为 key,每个结点拥有 $list$ 属性,用来存储覆盖间隔 key 的所有对象.为了节省存储空间,覆盖父节点的对象不会在子节点重复存储.Segment R-tree 和 Time Index+都利用了 Segment tree 的特性来降低空间复杂度.

(2) 分布式索引

当间隔数据的数据量逐渐增大、单机不能存储海量数据时,索引分布式在多台机器.这种索引称为分布式索引.

- 基于 P2P 架构的分布式索引.文献[15]将节点组织成 DHT 网络覆盖,提出分布式线段树 Distributed Segment Tree(DST),主要支持范围查询和覆盖查询.除了明确说明支持间隔查询的分布式索引外,由于一维间隔查询可以转换为二维范围查询,支持范围查询的分布式索引能够支持间隔查询,如 RT-CAN[16],CGIndex[17]等基于 P2P 架构的分布式多维索引;
- 基于 Master-Slave 架构的分布式索引.EPI+MRST[18]是首个分布式间隔索引.为了高效响应间隔查询,它将间隔查询进行一维转换,采用索引结构 EPI 和 MRST 分别处理范围查询和覆盖查询.此外,支持多维范围查询的分布式多维索引也支持间隔索引,如 MD-HBase[19],KR$^+$-tree[20]等.

间隔查询因重要的理论意义和实践价值而受到持续关注,从 20 世纪 80 年代至今,优化间隔查询的大量索引技术涌现.然而这些技术仅优化单个查询的数据访问速度,而没有考虑查询本身的特性,当单点上海量查询请求急速到来时,70%的查询不能在期望时间内得到响应.

2.2 基于查询特性的间隔查询优化技术

1) 缓存技术

数据缓存是数据库和信息检索系统优化检索性能的常用方法.依据二八原则,80%的查询是频繁访问的,因此将频繁查询和检索结果缓存起来,当缓存命中时,可以直接从内存获取结果,不需要访问磁盘,从而提高查询的响应时间.缓存技术要求查询或子查询必须和缓存的查询相同,没有探索不相同的查询间是否存在部分结果复用,并且缓存技术有很强的时效性,设计合理的缓存粒度[21]和淘汰算法[22,23]是缓存技术的关键,该方法通过提高缓存命中率来提升检索性能.缓存技术中查询的执行依然采用 one-by-one 的方式,不同于本文实施的共享执行策略.

2) 共享执行策略

共享执行策略先将接收到的查询进行缓存,当查询累计到一定量或一定时间时,对这组查询进行批处理.共享执行策略的关键技术是探索查询间的相似性,分析可以共享执行的操作.共享执行的操作可以是计算能力或访问数据,可以共享部分操作也可以共享全部操作,共享的粒度更细致更灵活.共享执行策略的有效性和高效性,使得其在多个应用中均取得显著效果.文献[6]首次关注高并发情况下查询的等待时间对查询的响应时间影响,探索路网中 KNN 查询间的同质性,将一个查询的中间结果给多个查询进行共享,从而减少计算代价.S*e*TPR*-tree[24]将共享执行策略应用到时空范围查询,对一批范围查询进行处理,通过调整查询的执行顺序提高 buffer 的命中率,从而提升检索性能.SharedDB[25]是一次处理多个查询的内存数据库,为一批查询生成一个全局查询计划,仅执行该全局查询计划,从而节省计算资源和数据访问代价.TOF[26]将共享执行策略应用到移动对象中,对一组范围查询进行批处理,从而保证所有高并发的查询请求均能在用户期望时间内得到响应.

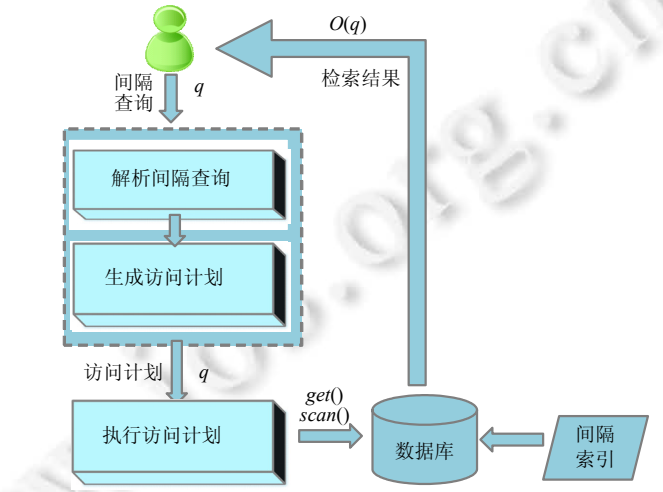
本文首次将共享执行策略应用到间隔查询中,针对目前日益增长的间隔数据和相应的应用需求优化间隔查询,解决基于索引的优化技术不能满足海量高并发查询的性能要求问题.

3 基于共享执行策略优化间隔查询(SESIQ)

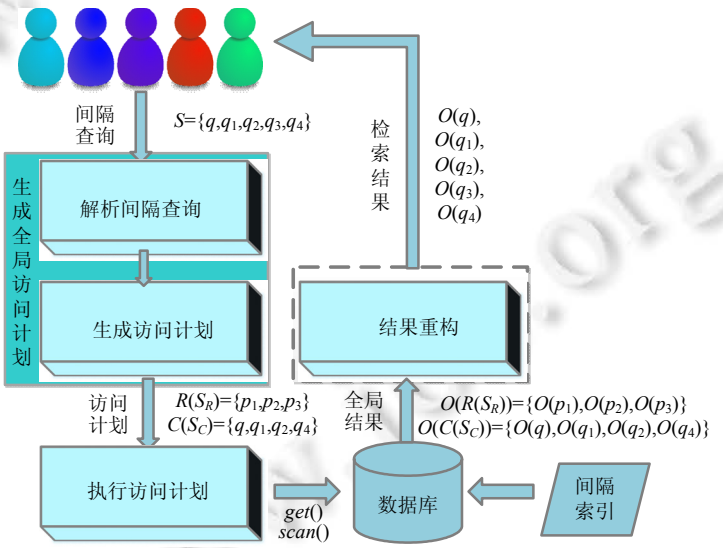
相交的两个间隔查询含有相同的检索结果,而相同结果可以仅访问一次,从而减少磁盘 I/O 和网络传输代价,提高检索性能.本节将介绍基于此思想所设计的 SESIQ 的引擎架构及其关键实现技术.

SESIQ 的引擎包括计划生成器 Planner、执行器 Executor 和结果重组器 Reconstructor 这 3 个核心组件,关键执行流程如图 3(b)所示.3 个核心组件分别完成生成全局访问计划(generating global access plan,简称 GAP)、执行访问计划(executing access plan,简称 EAP)、重构检索结果(reconstructing search result,简称 RSR)这 3 个任务:(1) GAP 负责分析一组间隔查询间是否含有重复访问的数据,通过查询分解或原子化将原查询转换为子查询集合,形成全局访问计划;(2) EAP 负责执行全局访问计划,得到每个子查询的检索结果;(3) RSR 需要对子查询的检索结果进行重构,以满足用户提交的原查询的要求.

SESIQ 的主要贡献在于对查询进行批处理,为一组间隔查询生成全局访问计划代替多个查询的局部访问计划,通过查找间隔查询间的共享操作减少重复数据访问,提升检索性能.对比图 3(a)和图 3(b)可知:图 3(a)中主要通过间隔索引优化间隔查询;图 3(b)从查询特性的角度出发,在间隔索引基础上进一步优化间隔查询的检索性能.由于篇幅受限,本文主要介绍间隔查询进行一维转换的 SESIQ 实现技术,未来详细探讨间隔查询进行二维转换的 SESIQ 方法.



(a) 单个间隔查询执行过程



(b) 多个查询的共享执行过程

Fig.3 The execute process of interval query

图 3 间隔查询的执行过程

3.1 生成全局访问计划(GAP)

GAP 是 SESIQ 的计划生成器中最关键的过程,它需要解决两个技术难点.

a) 分析一组间隔查询间是否含有重复访问的数据;

b) 尽可能地减少重复数据的访问.

对于问题 a),我们可以两两比较间隔查询,判断是否满足相交关系.然而这种方法的时间复杂度 $O(M^2 \cdot T)(M$

为查询的个数,T 为判断是否相交的时间),效率低下;对于问题 b),需要获得一组间隔查询最少访问数据量.为此,本文为一组间隔查询创建内存线段树,借助线段树回答问题 a)和问题 b).假设服务器接收到的一组间隔查询 $S=\{q,q_1,q_2,q_3,q_4\}$ 如图 4 所示,对应图 4 查询示例的内存线段树 ST 如图 5 所示.由于对间隔查询进行了一维转换,S 分解为一组范围查询 $S_R=\{q,q_1,q_2,q_3,q_4\}$ 和一组覆盖查询 $S_C=\{q,q_1,q_2,q_3,q_4\}$,问题 a)和问题 b)转化为分析 S_R/S_C 元素间是否相交以及 S_R/S_C 尽可能地减少重复数据访问的问题.下面介绍 GAP 处理 S_R 和 S_C 的详细过程.

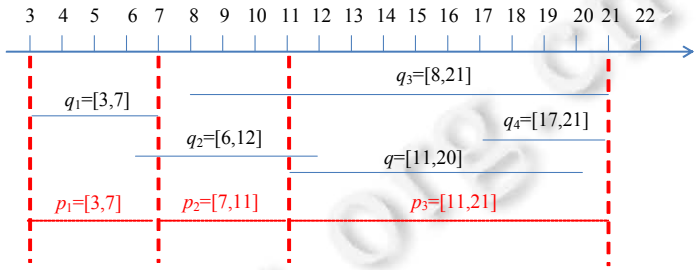


Fig.4 A group interval queries

图 4 一组间隔查询示例

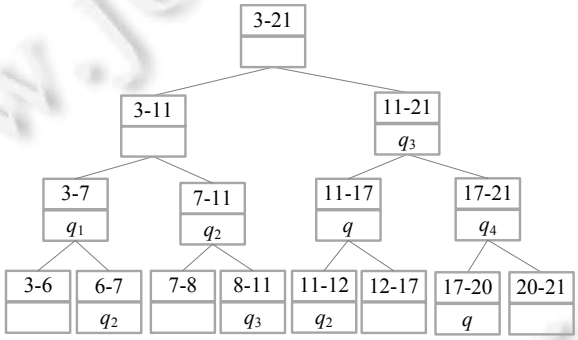


Fig.5 The corresponding segment tree for a interval query set S

图 5 间隔查询集合 S 对应的线段树

GAP 处理 S_R 主要思想是:分解 S_R 所有元素构成的范围区间,得到一组子查询集合 $R(S_R)$,对于任意范围查询 $p\in R(S_R)$,必有 $q\in S_R$,使得 p 被 q 包含,同时,$R(S_R)$满足:

a) 如果 S_R 不存在相交的元素,则 $R(S_R)$和 S_R 相等;否则,S_R 中存在相交的元素;

b) $R(S_R)$的检索结果能够还原 S_R 的检索结果,且 $R(S_R)$重复访问的数据最少.

GAP 处理 S_C 的关键,是判断覆盖集合是否存在互相覆盖的查询:如果两个覆盖查询相交,即二者满足定义 2 中的覆盖关系,则二者的检索结果可共享.例如间隔 q 覆盖间隔 p,则覆盖查询 p 的检索结果中必然包含覆盖查询 q 的结果,服务器仅需要响应 p 而省略对 q 结果的访问,p 成为向服务器提交的子查询.

给定一个间隔查询集合 S,算法 1 描述了借助线段树 ST 生成 S 的全局执行计划的过程.

算法 1. 生成间隔查询集合 S 的全局执行计划.

输入:间隔查询集合 S,内存线段树 ST;

输出:子查询集合 $R(S_R)$和 $C(S_c)$,子查询和原查询的映射关系 $RMap$ 和 $Cmap$.

1: 初始化集合 $R(S_R)$和 NS、队列 $queue$、哈希表 $RMap$ 和 $CMap$ 为空,集合 $C(S_c)=S$;

2: 将 ST 的根结点 $node$ 入队列 $queue$;

3: WHILE $queue$ 不为空

4: $queue$ 移出队头元素 $node$;

5: 获取 $node$ 的对象列表 $list$;

```
6:      IF list 不为空 THEN
7:        NS.add(node);
8:      ELSE        //list 为空
9:        将 node 的左右孩子结点分别入队列 queue;
10: END WHILE
11: FOR node: NS
12:    subquery←node 的间隔;
13:    list←node 的对象列表;
14:    R(S_R).add(SubQuery);
15:    清空 queue;
16:    将 node 加入队列 queue;
17:    WHILE queue 不为空
18:      queue 移出队头元素 subnode;
19:      获取 subnode 的对象列表 sublist;
20:      将 subnode 的左右孩子结点入队列 queue;
21:      IF sublist 不为空 THEN
22:      FOR o:o∈sublist and q:q∈list
23:           RMap.put(subQuery,o);
24:           IF q∈C(S_c) and q 覆盖 o THEN
25:              CMap.Put(o,q);
26:              从 C(S_c)中删除 q;
27:           END IF
28:        END FOR
29:      END IF
30:    END WHILE
31: END FOR
32: 返回 R(S_R),C(S_c),RMap,CMap;
```

算法 1 从根结点开始层次遍历线段树 *ST*(第 2 行),如果遇到结点 *node* 的对象列表 *list* 不为空(第 4 行~第 6 行),将 *node* 加入到集合 *NS*(第 7 行),处理队列 *queue* 中下一个 *node*,直到 *queue* 为空;否则,将 *node* 的孩子结点加入 *queue*,需要遍历孩子结点(第 8 行、第 9 行).通过步骤 3~步骤 10,算法 1 能够找到 *ST* 所有路径中首个对象列表不为空的结点,这样的结点为关键结点.依据关键结点,算法 1 可以获得 S_R 和 S_C 子查询集合 $R(S_R)$和 $C(S_C)$以及子查询和原查询的映射关系 *RMap* 和 *CMap* (第 11 行~第 31 行).对于 S_R,任意关键结点 *node*,其间隔构成 *list* 中原查询的子查询 *subQuery*(第 12 行),*subQuery* 作为 $R(S_R)$的元素加入到 $R(S_R)$(第 14 行),而以 *node* 为根的子树中包含的对象(即原查询)和 *subQuery* 存在映射关系(第 17 行~第 30 行),添加他们的映射关系到 *RMap*(第 23 行).对于 S_c,任意关键结点 *node*,其对象列表 *list*(第 13 行)中的对象 *q*(即原查询)为覆盖范围最广的查询,若 *q* 覆盖其他的查询 *o*(第 24 行),则 *o* 存在于以 *node* 为根的子树中(第 17 行~第 30 行).最终返回 $R(S_R)$,$C(S_C)$,*Rmap*,*CMap*(第 32 行).

若算法 1 中得到的 $R(S_R)=S_R$,*RMap* 为空,或 $C(S_C)=S_C$,*CMap* 为空,则 S_R/S_C 元素间不存在共享的操作,此时 SESIQ 没有优化 S_R/S_C 的检索性能;当 $R(S_R)\neq S_R$ 时,$R(S_R)$中的每个元素都是 S_R 中一个查询的子查询,$R(S_R)$元素之间互不相交,不存在重复访问的数据,服务器仅响应 $R(S_R)$中的查询,使得能够获得 S_R 全部检索结果且重复访问数据最少;当 $C(S_C)$为 S_C 的真子集时,此时服务器少响应部分覆盖查询,减少了数据访问,SESIQ 能够优化 S_C.

例 2:对于 $S=\{q,q_1,q_2,q_3,q_4\}$,采用算法 1 层次遍历 *ST*,依次得到对象列表不为空的结点为 11-21,3-7,7-11,覆盖

能力最强的原查询为 q_3,q_1,q_2,因此,

$$R(S_R)=\{p_3=[11,21],p_1=[3,7],p_2=[7,11]\},$$

$$RMap=\{\langle p_3,\{q,q_2,q_3,q_4\}\rangle,\langle p_1,\{q_1,q_2\}\rangle,\langle p_2,\{q_2,q_3\}\rangle\},$$

$$C(S_C)=\{q,q_1,q_2,q_4\},$$

$$CMap=\{\langle q,q_3\rangle\}.$$

由于 $R(S_R)\neq S_R,C(S_c)\neq S_c$,*RMap* 和 *CMap* 不为空,可知 S_R/S_C 存在相交的元素,SESIQ 可以优化 S_R/S_C 的性能.

3.2 执行访问计划(EAP)

间隔查询进行一维转换后,生成范围查询和覆盖查询 2 个子查询,因而支持范围查询和覆盖查询的索引均可以用来执行 GAP 中生成的访问计划.EPI+MRST[18]中的 EPI 结构可以用来响应范围查询,MRST 结构用来响应覆盖查询,且 EPI+MRST 是明确指出支持间隔查询的首个分布式索引,因而更具借鉴和比较价值,EAP 采用 EPI+MRST 执行 GAP 产生的子查询集合.

GAP 产生 S_R 和 S_C 的子查询集合分别为 $R(S_R)$和 $C(S_C)$.对于 $R(S_R)$中的任意查询 $p=[a,b]$,EPI 使用 HBase 的 *scan* 操作获取 *Rowkey* 在$[a,b]$之间的对象,取得范围查询的结果;对于 $C(S_C)$中的任意查询 $p'=[c,d]$,MRST 处理点查询 $p''=[c,c]$,使用 HBase 的 *get* 操作遍历 MRST 来获得覆盖查询 p'的结果.EPI 和 MRST 处理查询的算法描述详见文献[18].

例 3:对应例 3 产生的子查询集合 $R(S_R)=\{p_3=[11,21],p_1=[3,7],p_2=[7,11]\}$和 $C(S_C)=\{q_1=[3,7],q_2=[6,12],q=[11,20],q_4=[17,21]\}$,其检索结果见表 1.为了对比,表 1 还给出集合 S 中元素的检索结果.

Table 1 The results of the queries responded by EPI+MRST

表 1 EPI+MRST 响应查询获得的检索结果

查询	所属集合	查询类型	检索结果	查询	所属集合	查询类型	检索结果
p_1=[3,7]	$R(S_R)$	范围查询	{*u*,*v*,*w*}	q_3=[8,21]	*S*	覆盖查询	{*w*}
p_2=[7,11]	$R(S_R)$	范围查询	{*u*}	*q*=[11,20]	*S*	范围查询	{*v*,*y*,*z*}
p_3=[11,21]	$R(S_R)$	范围查询	{*v*,*x*,*y*,*z*}	q_1=[3,7]	*S*	范围查询	{*u*,*v*,*w*}
q=[11,20]	$C(S_C)$, *S*	覆盖查询	{*w*}	q_2=[6,12]	*S*	范围查询	{*u*}
q_1=[3,7]	$C(S_C)$, *S*	覆盖查询	{*u*,*v*}	q_4=[17,21]	*S*	范围查询	{*x*,*y*,*z*}
q_2=[6,12]	$C(S_C)$, *S*	覆盖查询	{*v*,*w*}	q_3=[8,21]	*S*	范围查询	{*v*,*x*,*y*,*z*}
q_4=[17,21]	$C(S_C)$, *S*	覆盖查询	{*w*}				

3.3 重构检索结果(RSR)

GAP 创建了子查询和原查询的映射关系 *RMap* 和 *CMap*,用于重构检索结果.对应例 1,下面详细阐述怎样将 EAP 执行的结果映射到原查询中,即 RSR 的执行过程.因算法较简单,略去算法描述.

例 4:对于间隔查询集合 $S=\{q,q_1,q_2,q_3,q_4\}$,需要合并 S_R 和 S_C 的结果形成最终结果,即需要建立 $R(S_R)$,$C(S_C)$和 *S* 的关系,对 $R(S_R)$和 $C(S_C)$的检索结果进行分配.表 2 展示了原查询和子查询的关系,并指出得到正确结果要执行的操作.

Table 2 The mapping relationship between interval queries and subqueries, and the final retrieval results

表 2 间隔查询和子查询的映射关系及最终检索结果

间隔查询	范围查询	覆盖查询	检索结果
q=[11,20]	p_3,过滤结果	*q*	{*v*,*w*,*y*,*z*}
q_1=[3,7]	p_1	q_1	{*u*,*v*,*w*}
q_2=[6,12]	p_1,p_2,p_3,过滤结果	q_2	{*u*,*v*,*w*}
q_3=[8,21]	p_2,p_3,过滤结果	q_4,过滤结果	{*v*,*w*,*y*,*x*,*z*}
q_4=[17,21]	p_3,过滤结果	q_4	{*w*,*y*,*x*,*z*}

如:范围查询 *q* 和范围查询 p_3 部分相交,因此需要精炼 p_3 的执行结果以过滤不在查询间隔 *q* 内的对象 *x*,*q* 的结果为{*v*,*y*,*z*},而覆盖查询 *q* 的结果为{*v*,*w*},间隔查询 *q* 的最终结果为{*v*,*w*,*y*,*z*}.类似可获得其他间隔查询的

检索结果,见表 2.

了解 SESIQ 方法的执行流程后,第 3.4 节和第 3.5 节将定量分析 SESIQ 的时空开销.

3.4 性能分析

SESIQ 能够有效执行的条件是,一组查询中存在共享执行的操作.当 S_R/S_C 中没有共享的操作时,即 $R(S_R)=S_R, C(S_C)=S_C$,此时 SESIQ 和 NSESIQ(没有采用 SESIQ 的方法)的性能是相当的,间隔查询没有得到优化;当 S_R/S_C 存在可共享的操作,减少数据访问的收益大于内存计算开销,SESIQ 的性能优于 NSESIQ.

下面分别讨论当 S_R/S_C 存在可共享的操作时,SESIQ 和 NSESIQ 的性能差异.

(1) 范围查询

范围查询的检索结果常通过顺序扫描获得,如 EPI+MRST 中采用 *scan* 操作访问 EPI,其消耗的时间和访问对象数成正比,SESIQ 和 NSESIQ 的性能比 λ_R 可用公式(1)描述:

$$\lambda_R \propto O(S_R)/O(R(S_R)) \tag{1}$$

其中,$O(S_R)$和 $O(R(S_R))$分别表示 NSESIQ 和 SESIQ 访问的对象总数,通过累积 S_R 和 $R(S_R)$中的每一个范围查询 I 访问的对象数 $O^R(I)$而获得.由算法 1 可知,ST 中所有对象列表不为空的结点 LN 构成的集合 SLN 能够还原 S_R.S_R 访问的对象总数可用公式(2)计算:

$$O(S_R)=\sum_{i\in SIN(ST)} O^R(I_i)\times OL(i,ST) \tag{2}$$

其中,I_i 表示结点 i 的间隔,$OL(i,ST)$表示 ST 中覆盖结点 i 的对象数.$R(\mathrm{S_R})$访问的对象总数和 $ULN(ST$ 所有从根开始的路径中首个 LN 形成的集合)相等,见公式(3):

$$O(R(S_R))=\sum_{i\in UIN(ST)} O^R(I_i) \tag{3}$$

而 O^R(I)可通过公式(4)或公式(5)进行估算或精算.

- 仅知道查询信息时,用公式(4)估算 $O^R(I)$,假设 N 个间隔数据均匀地分布在范围为[min,max]的一维空间内:

$$O^R(I)=\lceil N\times||I||/||\max-\min||\rceil \tag{4}$$

- 索引元数据信息和查询信息都获取后,可用公式(5)精算 $O^R(I)$,其中,df_t 表示间隔端点为 t 的对象总数:

$$O^R(I=[a,b])=\sum_{t=a}^{b} df_t \tag{5}$$

从以上分析可知:范围查询中 SESIQ 访问的对象总数是 NSESIQ 的一部分,利用公式(1)~公式(5)可以粗略估算 SESIQ 在范围查询中获得的性能收益.

(2) 覆盖查询

支持覆盖查询的索引常为树形结构,覆盖查询的检索结果通过随机访问获得.如 EPI+MRST 中,执行 *get* 操作访问 MRST,其消耗的时间和访问结点数成正比,因此,访问结点数成为 SESIQ 和 NSESIQ 在覆盖查询中性能差异的度量,SESIQ 和 NSESIQ 的性能比可用公式(6)描述:

$$\lambda_C \propto N(S_C)/N(C(S_C)) \tag{6}$$

其中,NSESIQ 和 SESIQ 访问的结点总数用 $N(S_C)$,$O(C(S_C))$表示,可由公式(7)和公式(8)获得,通过累积 S_C 和 $C(S_C)$中的每一个覆盖查询 p 访问的结点数 $N^C(p)$而获得:

$$N(S_C)=\sum_{p\in S_C} N^C(p) \tag{7}$$

$$N(C(S_C))=\sum_{p\in C(S_C)} N^C(p) \tag{8}$$

在公式(7)和公式(8)中,$N^C(p)$由索引信息确定:

- 仅知道树形索引的树高 h 时,可用公式(9)估算:

$$N^C(p)\approx h=1+\lceil \log_2 m\rceil \tag{9}$$

- 获得索引从根到叶每条路径的长度和叶结点信息,可用公式(10)计算覆盖查询 p 所访问的结点数:

$$N^C(p)=Lpath(p) \tag{10}$$

覆盖查询中,SESIQ 的性能收益来自于 S_C 中存在具有覆盖关系的查询,使得 SESIQ 向后台提交的查询请求数减少,降低了访问结点数.基于公式(6)~公式(10),我们可以估算 SESIQ 在覆盖查询中的性能收益.

例 5:假设我们不知道索引的具体信息,采用第 3.4 节介绍的公式估算 SESIQ 和 NSESIQ 响应图 4 示例查询的性能差异,数据集为图 1 所示的 6 个间隔对象,因此,N=6,m=8,h=3,max=22,min=3.使用公式(4)可得范围查询访问对象数,如 $O^R(q)$=3,公式(2)和公式(3)估算范围查询集合访问对象总数可得 $O(S_R)$=14,$O(R(S_R))$=8.公式(9)估算覆盖查询访问结点数,如 $N^C(q)$=4,公式(7)和公式(8)估算覆盖查询集合访问结点总数可得 $N(S_C)$=20,$N(C(S_C))$=16.而真实值 $O(S_R)$=14,$O(R(S_R))$=8,$N(S_C)$=20,$N(C(S_C))$=16.对比估算值和真实值可知,以上公式可以作为性能评估的参考.对比 SESIQ 和 NSESIQ 的性能可知:SESIQ 在图 4 示例查询中是有效的,其范围查询的性能约为 NSESIQ 的 1.5 倍,覆盖查询的性能约为 NSESIQ 的 1.25 倍.

3.5 空间开销分析

SESIQ 批处理一组(M 个)间隔查询,并为该组间隔查询建立了内存线段树,且 SESIQ 在重构检索结果阶段需要缓存部分检索结果,下面定量分析 SESIQ 和 NSESIQ 的空间开销.

SESIQ 的空间开销来源于两部分:

a) 维护 M 个间隔查询的内存线段树消耗的空间——S_{ST};

b) 缓存 M 个间隔查询的检索结果消耗的空间——S_{RSR}.

SESIQ 的空间开销为这两部分的最大值,其计算方法见公式(11):

$$S_{SESIQ}=\max(S_{ST},S_{RSR}) \tag{11}$$

而 NSESIQ 的空间开销主要用于缓存单个查询的检索结果 S_r,其计算方法见公式(12):

$$S_{NSESIQ}=S_r \tag{12}$$

S_{ST},S_{RSR} 的计算方法分别见公式(13)、公式(14):

$$S_{ST}=(2M-1)\times S_{node}+\sum_{i\in SIN(ST)} OL(i,ST)\times S_q \approx (M\times 2\mu+M\times\log M)\times S_q \tag{13}$$

$$S_{RSR}=\begin{cases} M\times S_r & \text{(a)} \\ S_r & \text{(b)} \end{cases} \tag{14}$$

公式(13)计算 S_{ST} 时,需要统计(2M−1)个结点(不包含对象列表)的空间开销和 ST 中对象间隔的空间开销,其中,单个结点的开销 S_{node} 约等于μ个 S_q 的代价(公式(15)):

$$S_r=r_q\times S_q \tag{15}$$

而最终检索结果的形成,需要合并每个公共子查询的检索结果.当服务器多次提交子查询的检索结果到客户端,即,检索结果的合并在客户端完成时,S_{RSR} 的计算方法见公式(14)(b);当服务器合并所有子查询的检索结果形成最终结果提交到客户端时,S_{RSR} 的计算方法见公式(14)(a),其中,S_r 表示单个查询的检索结果占用的最大内存,主要由检索结果数 r_q 确定,详见公式(16):

$$S_{node}=\mu\times S_q \tag{16}$$

对比 SESIQ 和 NSESIQ 的空间开销可知:当检索结果缓存在客户端时,SESIQ 的空间开销和 NSESIQ 相差不多,主要是增加了内存线段树的开销,其空间复杂度为 $O(M\times\log M)$;当检索结果缓存在服务器端时,SESIQ 的空间开销主要用于缓存检索结果,空间复杂度为 $O(M\times r_q)$,其代价是 NSESIQ 的 M 倍.

例 6:下面依据例 5 计算 SESIQ 和 NSESIQ 的空间开销,公式中的参数设置为μ=2,r_q=5,N=5,m=8.M=5.NSESIQ 的空间开销依据公式(12)可得 $S_{NSESIQ}=5S_q$;由公式(13)、公式(14)分别可得 $S_{ST}=15\times S_{node}+8\times S_q=38S_q$,$S_{RSR}=25S_q$.依据公式(11)可知:SESIQ 的空间开销为 $S_{SESIQ}=38S_q$,是 NSESIQ 的 7.6 倍.

4 实验评估

本文使用 2 种真实数据集评估 SESIQ 的有效性,底层索引为 EPI+MRST.实验过程中,将采用共享策略的方法 EPIMRST_SQ 和没有使用共享策略的方法 EPIMRST 进行对比,评估多种影响因子在两种不同方法中对检索性能的影响.程序实现均采用 Java 完成,EPI+MRST 的实现代码来自原作者.所有的实验在一个 8 台虚拟机组成的分布式集群上完成,网络带宽为 1Gbps.每台虚拟机拥有 8GB 内存、500GB 磁盘空间.EPI+MRST 采用分布式数据库 HBase-0.94.2 做存储系统,底层分布式文件基于 Hadoop-1.0.4.

4.1 数据集

本文选择 2 个真实数据集 DMOZ 和 Wiki 进行实验评估.DMOZ 数据集来自网站 dmoz.org,人工记录了不同类别多个网站的版本变更信息,本文从 2004 年~2011 年的类别变更数据中(约 426GB)抽取 1 200 万个间隔(86 751 个间隔端点)组成 DMOZ 数据集,其时间间隔以小时为单位;Wiki 数据集来自 wikimedia.org,记录 Web 文档的版本变更信息,本文从 2001 年~2015 年的 Web 文档版本变更元数据(约 10TB)抽取 3.2 亿个间隔(120 735 322 个间隔端点)形成 Wiki 数据集,其时间间隔以 ms 为单位.

4.2 实验结果及分析

本节评估查询数、选择率、数据量(间隔数)这 3 种影响因子对间隔查询响应时间的影响,验证 SESIQ 的有效性.为了使结果更为可靠,每个实验重复 10 次,并取平均值作为最终结果.表 3 给出实验用的参数设置,缺省值加粗显示.为了测试数据量对检索性能的影响,选择 Wiki 数据集 20M,40M,80M,160M 个间隔评估实验.模拟真实应用中,服务器接收到的查询是随机的,本文从原始数据中随机抽取选择率不确定的间隔作为查询用例.除评估选择率对检索性能的影响使用固定选择率的查询,其他情况均使用随机查询.

Table 3 The parameter setting of experiments

表 3 实验的参数设置

参数	含义	值
查询数	批处理的查询数	100, 200, 400, 800
选择率	检索到的间隔/总间隔	0.01,0.1,1,10,随机选择率
数据量	Wiki 数据集的子集,不同数量构成的间隔	20M, 40M, 80M, 160M, 320M
数据集	两种真实数据集	DMOZ, Wiki

1) 查询数对间隔查询检索性能的影响

SESIQ 批量处理的查询数是影响 SESIQ 性能的重要因素,本节重点评估查询数对间隔查询检索性能的影响.由于本文的间隔查询进行了一维转换,分解为范围查询和覆盖查询,本文评估查询数对范围查询和覆盖查询检索性能的影响,分析访问对象数、访问结点数、响应时间随查询数的变化.

- 访问对象数随范围查询数变化

EPI+MRST 中,EPI 索引负责处理范围查询,因此,EPI-SQ 区分 EPI 表示采用共享执行策略的方法.对应于第 3.4 节的分析,本文给出 EPI 和 EPI-SQ 的估算值,用~EPI 和~EPI-SQ 表示.

从图 6 展示的 DMOZ 数据集中访问对象数随范围查询数的变化可知:查询数对 EPI 检索性能几乎没有影响,其访问对象数在小范围内波动;而随着查询数增大,EPI-SQ 访问对象数减少,检索性能线性提升.主要原因是:4 组查询的平均选择率相同,查询数不影响 EPI 的检索性能;EPI-SQ 中增加查询数增大查询间可共享性,从而访问的对象总数减少.由于实验所用服务器的物理配置受限,本文仅评估了 100 个~800 个查询数的性能.而查询数不是越多越好,有些情况下,检索性能反而下降.原因在于:分析查询间可共享性会增加内存计算开销,当增加查询数带来的收益小于开销,此时检索性能下降.总体来说,EPI 访问对象数远大于 EPI-SQ,是 EPI-SQ 的 50 倍~500 倍.图 6 还给出了 EPI 和 EPI-SQ 访问对象数的估算值,对比实测值可知,估算值和实测值相近,表明公式(2)~公式(4)是有效的.

类似地,图 7 给出了 Wiki 数据集中,访问对象数随范围查询数的变化.Wiki 数据集中,EPI-SQ 访问对象数随

着查询数的增加而下降,而 EPI 出现先下降后增加的趋势,但 EPI-SQ 的性能依然优于 EPI,EPI 访问对象数是 EPI-SQ 的 5 倍~30 倍.从 EPI-SQ 表现可知:增加范围查询数,通常会增多范围查询间可共享的操作.EPI 访问对象数出现先升后降的原因是:4 组查询的平均选择率不同,100 个~400 个随机查询中,选择率小的查询所占比重越来越多,800 个随机查询选择率大的查询的比重更大;同时也可以看出:相较 Wiki 以 ms 为单位,DMOZ 中的间隔以小时为单位,数据分布相对较均匀.图 7 给出了 EPI 和 EPI-SQ 访问对象数的实测值和估算值,也验证了公式(2)~公式(4)是有效的.

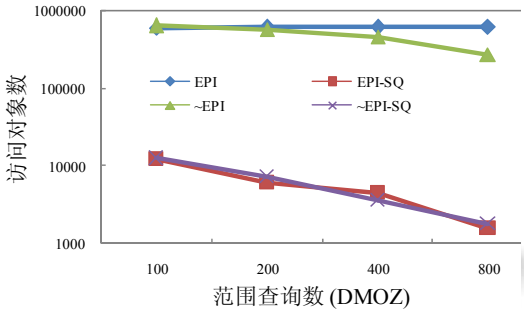


Fig.6 The number of access objects varies with the number of range query(DMOZ)

图 6 访问对象数随范围查询数变化(DMOZ)

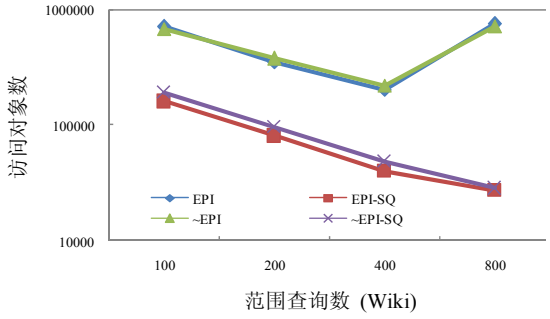


Fig.7 The number of access objects varies with the number of range query(Wiki)

图 7 访问对象数随范围查询数变化(Wiki)

- 响应时间随范围查询数的变化

图 8 和图 9 给出 DMOZ,Wiki 数据集响应时间随范围查询数的变化.为了清晰地描述响应时间和访问对象数的关系,图 8 和图 9 采用双纵坐标,左纵坐标为响应时间,右纵坐标为访问对象数.由图可知:EPI 和 EPI-SQ 的响应时间随查询数变化趋势和访问对象数随查询数变化趋势是相同的,可用公式(1)估算 EPI 和 EPI-SQ 的性能比.DMOZ 和 Wiki 数据集中,EPI 的响应时间分别是 EPI-SQ 的 20 倍~150 倍和 4 倍~20 倍.从图 6~图 9 可知:在查询数相同的情况下,SESIQ 方法在 DMOZ 数据集中访问了更少的对象,性能收益更加显著.主要原因是:Wiki 中的间隔以毫秒为单位,DMOZ 中的间隔以小时为单位,DMOZ 的数据密度(间隔总数/数据空间体积)相对 Wiki 更大,因而 DMOZ 数据集下,间隔查询间包含相同数据的几率增大,SESIQ 方法使得重复访问的数据减少地更多,其性能收益更大.

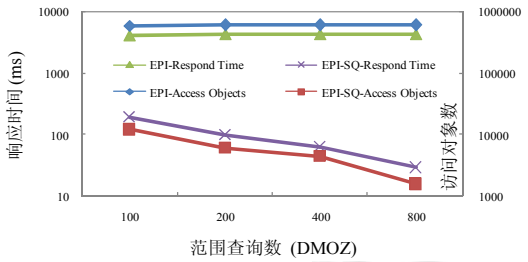


Fig.8 Respond time varies with the number of range query(DMOZ)

图 8 响应时间随范围查询数变化(DMOZ)

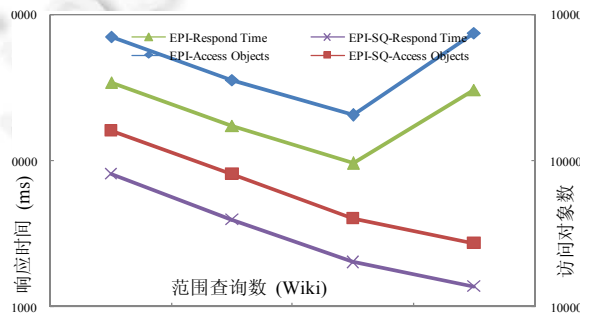


Fig.9 Respond time varies with the number of range query(Wiki)

图 9 响应时间随范围查询数变化(Wiki)

- 访问结点数随覆盖查询数的变化

EPI+MRST 中,MRST 索引响应覆盖查询.MRST 和 MRST-SQ 区分是否采用 SESIQ 方法,~MRST 和~MRST-SQ 表示 MRST 和 MRST-SQ 性能的估算值.DMOZ 数据集中,MRST 访问结点数随覆盖查询数增多而降低,MRST-SQ 访问结点数随覆盖查询数增加而波动,如图 10 所示.产生这种趋势的主要原因依然和选取的随机查询相关:MRST 是不平衡的二叉树,处理覆盖查询时执行点查询,不同的访问路径导致访问结点数不同. MRST 呈现的变化趋势是因为随机查询中访问短路径的查询比重更大,而 MRST-SQ 则因为两个查询具有覆盖关系的概率较小,增加覆盖查询数没有增多共享操作,因此,MRST-SQ 访问结点数随覆盖查询数增多而波动.但总体来说,MRST-SQ 访问结点数少于 MRST.图 10 所给出的访问结点数的估算值也存在波动,估算值和实际值二者有差异,但公式(9)获得的估算值依然可以作为评估性能的参考.

图 11 展示了 Wiki 数据集中覆盖查询数对访问结点数的影响,MRST 和 MRST-SQ 访问结点数均随查询数增加而增多,主要原因是访问长路径的随机覆盖查询所占比重更大.但 MRST-SQ 性能优于 MRST 是确定的,MRST 访问结点数是 MRST-SQ 的 1.5 倍~2 倍.图 11 中,MRST 和 MRST-SQ 访问结点数的估算值和实际值比较,也验证了公式(9)的可行性.

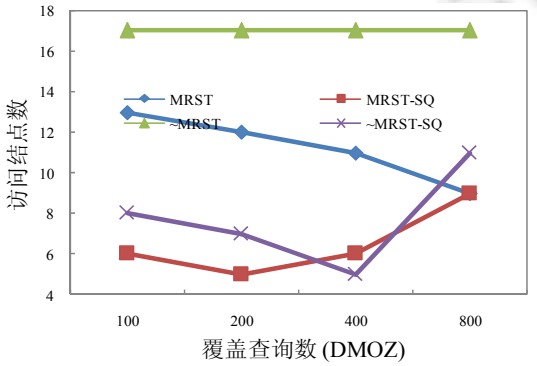


Fig.10 The number of access nodes varies with the number of cover query (DMOZ)

图 10 访问结点数随覆盖查询数变化(DMOZ)

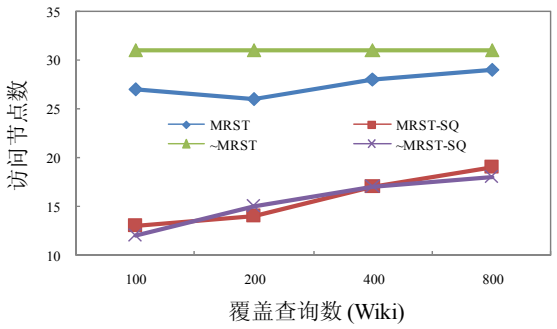


Fig.11 The number of access nodes varies with the number of cover query (Wiki)

图 11 访问结点数随覆盖查询数变化(Wiki)

- 响应时间随覆盖查询数的变化

本文也给出 DMOZ,Wiki 数据集响应时间随覆盖查询数的变化,类似于范围查询,图 12 和图 13 采用双纵坐标给出响应时间和访问结点数随查询数的变化趋势.从图 12 和图 13 可以看到:MRST 和 MRST-SQ 的响应时间和访问结点数二者随查询数变化趋势是相同的,可用公式(6)估算 MRST 和 MRST-SQ 的性能比.DMOZ 和 Wiki 数据集中,MRST 覆盖查询的响应时间分别是 MRST-SQ 的 1.2 倍~1.6 倍和 1.7 倍~2.3 倍.

对比图 10~图 13 可知:随着查询数的增加,SESIQ 在 Wiki 数据集中的性能收益更显著.主要原因在于:Wiki 中数据较稀疏,间隔查询间存在覆盖关系的几率更大;同时,Wiki 数据集索引树高更大导致 MRST 的访问路径较长,单个覆盖查询的响应时间更多;而 SESIQ 减少了向后台提交的查询,降低了查询的总体响应时间,因而性能收益较显著.

- 响应时间随间隔查询数变化

本节评估响应时间随间隔查询数的变化情况.从图 14 和图 15 可知:间隔查询的响应时间随查询数的变化趋势和范围查询接近,因为间隔查询的时间开销主要用于处理范围查询.EPIMRST-SQ 的检索性能明显优于 EPIMRST.DMOZ 数据集中,EPIMRST-SQ 的响应时间不足 1s,而 EPIMRST 的响应时间远大于 1s,是 EPIMRST-SQ 的 20 倍~50 倍.Wiki 数据量远远大于 DMOZ,EPIMRST 和 EPIMRST-SQ 的响应时间均在 1s 以上,EPIMRST 的响应时间是 EPIMRST-SQ 的 4 倍~25 倍.在间隔查询数相等的情况下,SESIQ 方法的性能收益在

DMOZ 数据集中更显著,由于 DMOZ 的数据密度相对 Wiki 更大导致.

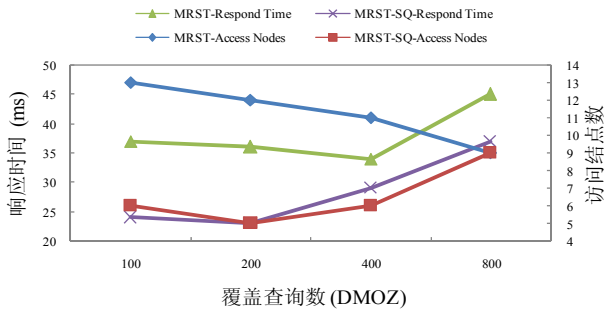


Fig.12 Respond time varies with the number of cover query (DMOZ)

图 12 响应时间随覆盖查询数变化(DMOZ)

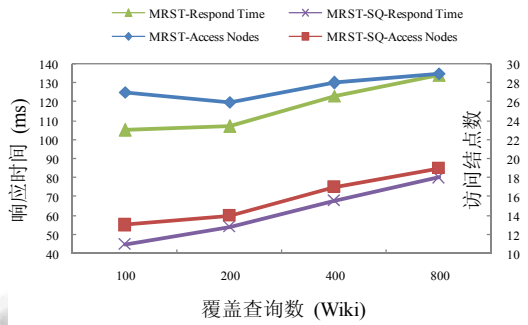


Fig.13 Respond time varies with the number of cover query (Wiki)

图 13 响应时间随覆盖查询数变化(Wiki)

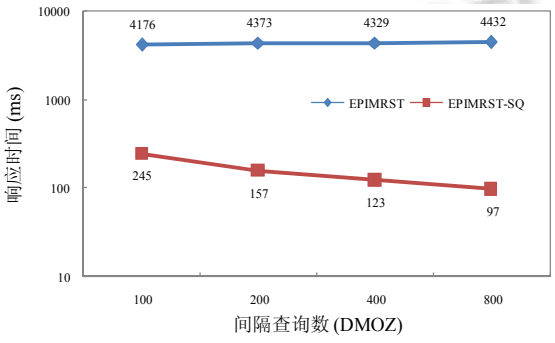


Fig.14 Respond time varies with the number of interval query (DMOZ)

图 14 响应时间随间隔查询数变化(DMOZ)

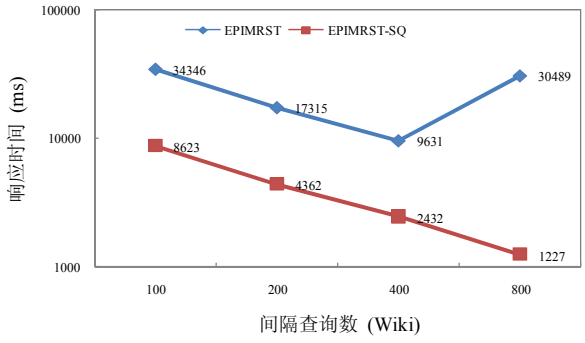


Fig.15 Respond time varies with the number of interval query (Wiki)

图 15 响应时间随间隔查询数变化(Wiki)

除了查询数这一影响因子,下面还评估了选择率、数据量对间隔查询检索性能的影响,主要关注间隔查询的平均响应时间随选择率和数据量的变化.

2) 选择率对响应时间的影响

图 16 展示了 DMOZ 数据集中,选择率对间隔查询的响应时间的影响.随着选择率增大,EPIMRST-SQ 和 EPIMRST 响应时间均增加.然而 EPIMRST-SQ 省略重复数据的访问和传输,增长趋势远低于 EPIMRST.同时可知:低选择率的情况下,EPIMRST-SQ 没有显露优势.由于分析查询间的可共享操作和重构检索结果增加了内存计算开销,检索性能甚至劣于 EPIMRST.当选择率升高到 0.1%后,EPIMRST-SQ 的性能显著优于 EPIMRST,且差异越来越大.当选择率为 10%时,EPIMRST-SQ 的性能比 EPIMRST 快了 20 倍.主要原因是高选择率下,间隔查询间相交的概率变大,可共享的数据更多,从而节省时间开销.

Wiki 数据集中选择率对间隔查询响应时间的影响如图 17 所示:随着选择率增长,EPIMRST-SQ 和 EPIMRST 的响应时间线性增加.而 EPIMRST 的响应时间远大于 EPIMRST-SQ,特别是 10%的选择率下,其响应时间是 EPIMRST-SQ 的 40 倍.

对比图 16 和图 17 可知,SESIQ 在 Wiki 数据集中的性能提升更显著.因为 Wiki 数据量较大,单个查询的响应时间更长,one-by-one 处理查询时,查询的等待时间相对较长,而 SESIQ 取消了查询间的等待,共享执行策略减少了大量重复数据的访问,因此其性能提升在 Wiki 中更显著.

3) 数据量对响应时间的影响

图 18 展示了间隔查询的响应时间随数据量变化趋势,实验中仅改变 Wiki 的数据量,不同数据量的查询用例是相同的.随着数据量的增加,EPIMRST-SQ 和 EPIMRST 的检索性能均呈线性下降,二者的下降趋势相同.但 EPIMRST-SQ 的检索性能依然优于 EPIMRST 约 4 倍.

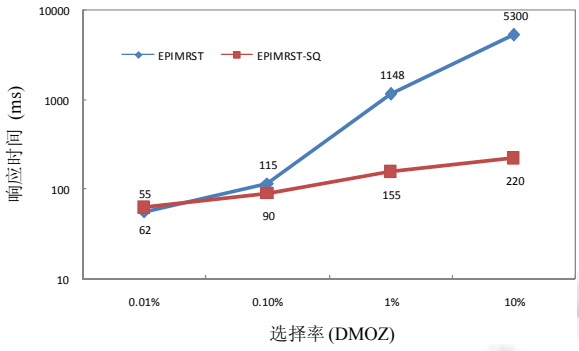


Fig.16 Respond time varies with query selectivity (DMOZ)

图 16 响应时间随选择率的变化(DMOZ)

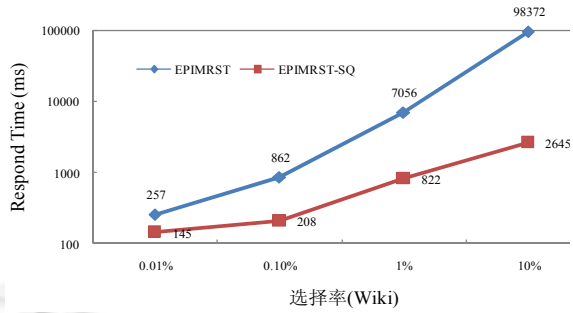


Fig.17 Respond time varies with query selectivity (Wiki)

图 17 响应时间随选择率的变化(Wiki)

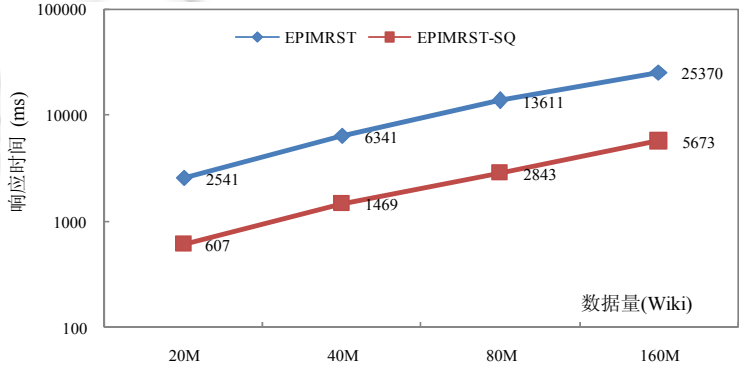


Fig.18 Respond time varies with the data size (Wiki)

图 18 响应时间随 Wiki 数据量的变化

以上实验从查询数、选择率、数据量这 3 个因子验证 SESIQ 的有效性.从结果可知:SESIQ 方法是有效的,特别是在数据量大、数据密度大的数据集中能够显著优化间隔查询,使其性能提升数十倍.

5 总结与展望

针对已有方法不能满足海量间隔查询性能要求的问题,本文首次将共享执行策略应用到间隔查询中,提出了采用共享执行策略优化间隔查询的方法 SESIQ.SESIQ 对接收到的间隔查询进行批处理,分析查询间可共享的操作,为多个查询生成一个全局的执行计划,从而减少重复数据的访问、提高检索性能.本文在已有分布式间隔索引基础上进一步优化间隔查询,基于理论分析和实验评估表明:SESIQ 是有效且高效的,其检索性能优于 NSESIQ 数十倍.

尽管本文提出的方案是高效可行的,然而本文还有一些问题待考虑:

1) 共享执行策略对内存需求相对较大.一般情况下,采用共享执行策略的应用都假设服务器的内存是足够大的,然而一些服务器的内存可能不够大,未来我们将研究内存受限的共享执行策略;

2) 本文在阐述 SESIQ 具体实施过程中仅对间隔查询进行了一维转换,然而一维转换使得 SESIQ 没有充分发掘间隔查询间的共享操作,因为范围查询的检索结果与覆盖查询的检索结果存在重复,如表 1 范围查询 q 的检索结果为{v,w},覆盖查询 q 的检索结果为{v,y,z},重复访问对象 v.

下一步我们将探索基于二维转换的 SESIQ 实现策略,从检索空间相交的角度深入挖掘间隔查询间重复访问的数据,进一步减少访问数据量,提升间隔查询的检索性能.

References:

[1] Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber R. Bigtable: A distributed storage system for structured data. In: Proc. of the Operating Systems Design and Implementation (OSDI). Seattle: USENIX Association, 2006. 205−218.

[2] Apache Hbase. http://hbase.apache.org/

[3] Cooper BF, Ramakrishnan R, Srivastava U, Silberstein A, Bohannon P, Jacobsen HA, Puz N, Weaver D, Yerneni R. PNUTS: Yahoo!'s hosted data serving platform. In: Proc. of the VLDB Endowment. Auckland: ACM Press, 2008. 1277−1288. [doi: 10.14778/1454159.1454167]

[4] Lakshman A, Malik P. Cassandra: A decentralized structured storage system. ACM Operating Systems Review (SIGOPS), 2010, 44(2):35−40. [doi: 10.1145/1773912.1773922]

[5] DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W. Dynamo: Amazon's highly available key-value store. In: Proc. of the 21st ACM Symp. on Operating Systems Principles (SOSP). Stevenson: ACM Press, 2007. 205−220. [doi: 10.1145/1294261.1294281]

[6] Chen Y, Chuang K, Chen M. Spatial-Temporal query homogeneity for KNN object search on road networks. In: Proc. of the 22nd ACM Int'l Conf. on Information and Knowledge Management (CIKM). San Francisco: ACM Press, 2013. 1019−1028. [doi: 10.1145/2505515.2505524]

[7] Salzberg B, Tsotras VJ. Comparison of access methods for time evolving data. ACM Computing Surveys (CSUR), 1999,31(2): 158−221. [doi: 10.1145/319806.319816]

[8] Elmasri R, Wuu GTJ, Kim YJ. The time index: An access structure for temporal data. In: Proc. of the 16th Int'l Conf. on Very Large Data Bases (VLDB). Brisbane: Morgan Kaufmann Publishers, 1990. 1−12.

[9] Kouramajian V, Kamel I, Elmasri R, Waheed R. The time index+: An incremental access structure for temporal databases. In: Proc. of the 3rd Int'l Conf. on Information and Knowledge Management (CIKM). Gaithersburg: ACM Press, 1994. 296−303.

[10] Ang C, Tan K. The interval B-tree. Information Processing Letters, 1994,53(2):85−89. [doi: 10.1145/191246.191298]

[11] Stantic B, Topor R, Terry J, Sattar A. Advanced indexing technique for temporal data. Computer Science and Information Systems (COMSIS), 2010,7(4):679−703. [doi: 10.2298/CSIS101020035S]

[12] Kolovson C, Stonebraker M. Segment indexes: Dynamic indexing techniques for multi-dimensional interval data. SIGMOD Record, 1991,20(2):138−147. [doi: 10.1145/115790.115807]

[13] Bliujute R, Jensen CS, Saltenis S, Slivinskas G. Light-Weight indexing of general bitemporal data. In: Proc. of the 12th Int'l Conf. on Scientific and Statistical Database Management (SSDBM). Berlin: IEEE Computer Society, 2000. 125−138. [doi: 10.1109/SSDM.2000.869783]

[14] Kumar A, Tsotras VJ, Faloutsos C. Designing access methods for bitemporal databases. IEEE Trans. on Knowledge and Data Engineering (TKDE), 1998,10(1):1−20. [doi: 10.1109/69.667079]

[15] Zheng C, Shen G, Li S, Shenker S. Distributed segment tree: Support of range query and cover query over DHT. In: Proc. of the 5th Int'l workshop on Peer-To-Peer Systems (IPTPS). Santa Barbara, 2006.

[16] Wang J, Wu S, Gao H, Li J, Ooi B.C. Indexing multi-dimensional data in a cloud system. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD). Indianapolis: ACM Press, 2010. 591−602. [doi: 10.1145/1807167.1807232]

[17] Wu S, Wu KL. An indexing framework for efficient retrieval on the cloud. IEEE Data Engineering Bulletin, 2009,32(1):75−82.

[18] Sfakianakis G, Patlakas I, Ntarmos N, Triantafillou P. Interval indexing and querying on key-value cloud stores. In: Proc. of the 29th IEEE Int'l Conf. on Data Engineering (ICDE). Brisban: ACM Press, 2013. 805−816. [doi: 10.1109/ICDE.2013.6544876]

[19] Nishimura S, Das S, Agrawal D, Abbadi AE. MD-HBase: A scalable multi-dimensional data infrastructure for location aware services. In: Proc. of the 12th IEEE Int'l Conf. on Mobile Data Management (MDM). Luleå: IEEE Computer Society, 2011. 7−16. [doi: 10.1109/MDM.2011.41]

[20] Hsu YT, Pan YC, Wei LY, Peng WC, Lee WC. Key formulation schemes for spatial index in cloud data managements. In: Proc. of the 13th IEEE Int'l Conf. on Mobile Data Management (MDM). Bengrluru: IEEE Computer Society, 2012. 21−26. [doi: 10.1109/MDM.2012.67]

[21] Chen K, Jin P, Yue L. Efficient buffer management for PCM-enhanced hybrid memory architecture. In: Proc. of the 17th Asia-Pacific Web Conf. (APWeb). Guangzhou: Springer-Verlag, 2015. 29−40. [doi: 10.1007/978-3-319-25255-1_3]

[22] Jin P, Ou Y, Härder T, Li Z. AD-LRU: An efficient buffer replacement algorithm for flash-based databases. Data & Knowledge Engineering (DKE), 2012,72:83−102. [doi: 10.1016/j.datak.2011.09.007]

[23] Ou Y, Jin P, Härder T. Flash-Aware buffer management for database systems. Int'l Journal of Knowledge-Based Organizations (IJKBO), 2013,3(4):22−39. [doi: 10.4018/ijkbo.2013100102]

[24] Nguyen T, He H, Chen Y. S*e*TPR*-tree: Efficient buffering for spatiotemporal indexes via shared execution. The Computer Journal (CJ), 2013,56(1):115−137. [doi: 10.1093/comjnl/bxs020]

[25] Giannikis G, Alonso G, Kossmann D. SharedDB: Killing one thousand queries with one stone. VLDB Endowment, 2012,5(6): 526−537. [doi: 10.14778/2168651.2168654]

[26] Xue ZB, Zhou X, Wang S. TOF: A throughput oriented framework for spatial queries processing in multi-core environment. In: Proc. of the Int'l Conf. on Database Systems for Advanced Applications (DASFAA). Hanoi: Springer-Verlag, 2015. 241−256. [doi: 10.1007/978-3-319-18123-3_15]

周新(1987－),女,安徽宿州人,博士生,CCF 学生会员,主要研究领域为时空数据管理.

薛忠斌(1985－),男,博士,主要研究领域为内存数据库,移动数据管理.

张孝(1972－),男,博士,副教授,CCF 高级会员,主要研究领域为高性能数据库,非结构化数据管理,数据的智能获取与应用.

王珊(1944－),女,教授,博士生导师,CCF 会士,主要研究领域为高性能数据库,内存数据库,非结构化数据库,数据仓库与数据分析.