软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2020,31(11):3519−3539 [doi: 10.13328/j.cnki.jos.005826]

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

基于时序分区的时态索引与查询*

杨佐希, 汤 娜, 汤 庸, 潘明明, 李丁丁, 叶小平

(华南师范大学 计算机学院,广东 广州 510631)

通讯作者: 汤庸, E-mail: ytang@m.scnu.edu.cn, http://www.scholat.com/ytang

摘 要: 时态索引作为一种高效管理和检索时态数据的有效手段,一直是时态数据领域的研究热点.提出了一种基于时序分区的时态索引技术 TPindex.首先将海量时态数据的时态属性映射到二维平面上,对平面上的“有效时间”点进行采样处理,通过使用自上而下,自左而右的时序分区方法将平面划分成若干个均匀的区域.其次,使用基于拟序关系的线序划分算法对每个分区中的数据构建数据结构,并建立基于“有效时间戳”的全区索引,实现“一次一集合”的数据查询操作.再次,还提出了使用分文件存储线序索引的模式将分区线序索引磁盘化,同时可以结合多线程技术并行处理数据,充分利用现代化硬件资源以满足海量数据下的高性能需求,提高索引性能.另一方面,我们还研究了海量时态数据下 TPindex 的增量式更新操作.最后,设计相应的仿真实验,通过与现有的代表性工作进行对比评估,验证了所提出方法的有效性和实用价值.

关键词: 时态索引;时序分区;拟序关系;海量数据;并行化

中图法分类号: TP311

中文引用格式: 杨佐希,汤娜,汤庸,潘明明,李丁丁,叶小平.基于时序分区的时态索引与查询.软件学报,2020,31(11):3519−3539. http://www.jos.org.cn/1000-9825/5826.htm

英文引用格式: Yang ZX, Tang N, Tang Y, Pan MM, Li DD, Ye XP. Temporal index and query based on timing partition. Ruan Jian Xue Bao/Journal of Software, 2020,31(11):3519−3539 (in Chinese). http://www.jos.org.cn/1000-9825/5826.htm

Temporal Index and Query Based on Timing Partition

YANG Zuo-Xi, TANG Na, TANG Yong, PAN Ming-Ming, LI Ding-Ding, YE Xiao-Ping

(School of Computer Science, South China Normal University, Guangzhou 510631, China)

Abstract: Temporal index is one of key methods for temporal data managements and retrieval, which has been a hotspot in the field of temporal data. This paper presents a temporal index technique TPindex which is based on a temporal timing partition method. Firstly, the temporal attributes of massive amount of temporal data is mapped to a two-dimensional plane and the “Valid Time” points in this plane are sampled for timing partition. A “form up to down and form left to right” timing partition method is used to divide the plane into several balanced temporal areas and whole-partition index would be established at the same time. Once the steps above are completed, temporal data can be dynamically indexed by its querying schema of “one time, one set”. Secondly, the TPindex would build data structures through using “linear order partition” algorithm based on quasi-order relation for the data in each temporal area. Besides, a “Separated Files Model Index” based on disks and multi-threading parallel process technique that can be combined are proposed to make full use of modern hardware resources to meet the high performance needs under high-volume data, leading to better performance with index. On the other hand, the incremental updating algorithm was also studied. Finally, the corresponding simulation experiments are designed to compare with the current representative work to verify the feasibility and validity of the proposed algorithm.

* 基金项目: 国家自然科学基金(61772211, U181120009); 广州市产学研协同创新重大专项(201704020203); 广东省应用型科技研发专项(2016B010124008)

Foundation item: National Natural Science Foundation of China (61772211, U181120009); Major Project of Industry-university-research Collaborative Innovation of Guangzhou Municipality (201704020203); Special Fund for Applied Technology Research and Development of Guangdong Province (2016B010124008)

收稿时间: 2018-09-05; 修改时间: 2019-01-07; 采用时间: 2019-02-22

Key words: temporal index; timing partition; quasi-order relation; high-volume data; parallelize

时间和空间是客观事物存在和发展的基本形式,所有的信息都具有相应的时态属性[1].随着互联网信息技术的高速发展,全球的信息呈现出爆炸式增长,海量的历史数据被存储供企业进行查询和分析,这些数据与时间的联系十分密切.因此,如何对时态数据进行高效地存储与管理是时态数据库领域研究的热点.时态数据的存储与管理要解决的其中一项关键技术就是时态索引[2]的构建.

针对时态索引的构建问题,研究人员提出了多种解决方案.自 20 世纪 90 年代以来,传统的时态索引以时态数据模型[3]为理论基础(该模型主要包括“数据本体”和“时态信息”),一般通过结合 B+-tree(比如 MAP21[4])或 R-tree(比如 4R-tree[5]和 GR-tree[6])等技术对时态数据进行处理[7,8].根据不同的时态数据模型,时态索引的处理有 3 种处理方式:将“数据本体”和“时态信息”依次处理[4,5,9,10];将“时态信息”处理归结到非时态部分,对时态数据和非时态数据统一进行处理[11−13];将“数据本体”与“时态信息”协同处理[14−16].

传统的时态索引能够较好地适用于小规模数据的时态需求,然而在面对海量数据时,由于其数据类型复杂,传统的时态索引结构存在冗余、空间开销过大、更新维护成本过高、难以支持复杂的时态查询以及复杂查询效率低等问题因此,研究一种适用海量数据的时态索引显得很有必要.

另外一方面,支持并行、多核和大内存等的现代化硬件使用非常普遍,多核处理器已经是市场上的主流配置.传统基于单线程的时态索引显然不能很好地利用目前计算机体系结构的优势.因此,研究如何将基于多线程的并行化技术应用到时态索引,提出可并行化的时态索引方案,可以为时态数据的检索带来性能的提升.在这方面,文献[17]提出了一种基于内存的索引 Timeline,结合可并行化的现代化硬件,其优化的核心主要在于减少 CPU 的计算时间.但是其对机器性能需求过高,缺乏普遍适用性.而结合多线程的并行化技术,基于外存的时态索引仍然是一种有价值的解决方案,其优化的核心在于增加查询数据的命中率,减少 I/O.

因此,本文提出了一种基于时序分区的时态索引技术 TPindex.该索引通过时序分区和线序划分的方法将海量的时态数据分为分区层和索引层两个部分.在用户进行时态数据查询的时候,首先通过分区层快速地定位到目标数据对应的时序分区中,然后再通过相应时序分区中顺序结构的索引层进行目标数据的筛选,实现“一次一顺序序列”的高效据检索方式.本文的贡献如下:

(1) 提出了一种基于时序分区的时态索引方案,适用于海量数据存储与管理的时态索引 Tpindex.

(2) 在 TPindex 的基础上了,设计了一种基于多线程技术的并行化优化方案.同时设计分文件存储线序索引的模式将时序分区的线序索引磁盘化,减少内存中的存储压力,将时态索引外存化.

(3) 讨论了基于 PLOB 的时态数据增量式更新机制,实现了对大规模历史数据的有效动态管理.

(4) 通过多组仿真实验,实验结果表明,在海量数据的情况下,本文提出的 TPindex 索引具有一定的有效性和实用性.

本文第 1 节介绍相关的研究工作.第 2 节给出 TPindex 索引构建的详细步骤.第 3 节描述 TPindex 索引的多种查询模式.第 4 节讨论 TPindex 索引的动态增量式更新机制.第 5 节通过多个仿真实验进一步验证 TPindex 的有效性与实用性.第 6 节总结全文,并提出未来的工作展望.

1 相关研究

随着信息技术的高速发展,全球的信息量正在以指数级的速度增加,其增长速度已经超过摩尔定律[18].现实中的信息几乎都显示或隐式地包含时间特征,天然的具备时态属性.海量的历史数据被存储和管理,是企业分析和决策的重要资料.这些时态数据可以看成是由“数据本体”和“时间标签”两部分组成,所以在某种程度上时态数据存储和管理也可以看作是常规数据存储和管理的拓展.但是在传统的关系型数据库中,数据的时间属性通常不是显式的,只保留数据的当前状态,相当于一种“快照数据”.但是当数据发生变化的时候,常规数据库会通过覆盖原有的数据来实现数据的更新,这样就使得历史数据被“抹除”掉而导致出现历史数据丢失的问题.虽然常规的数据库也能够存储历史数据,但是这样会导致时态数据的完整性被打破,难以重建对象数据的历史状态、

跟踪数据变化和预测未来发展.另一方面,时间本身具有严格的单向性(时间的增长是严格的单调递增),多维性(包括有效时间、事务时间和自定义时间等)和相互关系的复杂度(ALLEN 时间关系[19]),常规的关系数据库无法根据这些特性来进行查询的优化,这就导致常规的关系数据处理框架难以高效地查询时态数据.近年来,数据库领域中还出现时空众包数据的管理技术[20−22],其核心思想是将具备时空属性的众包任务分配为众包参与者,并且要求参与者在完成任务的同时需要满足指定的时空约束,是一种新型的计算模式,但是该模式也仍然需要一种通用性强的时空索引结构[23].

随着 TimeDB[24]、TempDB[25]等时态中间件的出现,用户可以直接通过时态标准查询语言 TSQL[26]提出查询请求.时态查询操作变得更为自然友好和简便,时态数据库因而得以更广泛的应用,对时态数据库的进一步发展起到了推动的作用.而时态索引是实现时态数据高效存储和管理的关键部分.针对不同的应用场景,构建不同的时态索引模型,研究人员从不同角度提出了多种解决方案.文献[7]提出了一种基于 B+-tree 的 Time Index 索引方式,是最早提出的时态索引模型之一,它是单一的在时间属性的基础上建立索引.通过对时间属性和关键字进行索引创建,文献[8]提出了一种基于 B-tree 的 Multi-Version 索引方式,结合 I/O 优化,实现对时态数据的有效管理.总的来说,这一类的索引会将“数据本体”和“时态信息”依次处理,首先采用时态关系投影、选择和连接等[4,5,9,10]处理时态信息,再将处理得到的数据进行常规处理.该类方法主要通过“时态关系代数”等模型进行构建.而在时空数据[11−13]的应用场景下,时间常常被视为空间数据的一个“新”的维度,将“时态信息”处理归结到非时态部分,通过构建一维时间与二维空间的“三维”空间体进行处理,最终对时态数据和空间数据统一进行处理.该类处理方法能够结合 B-tree 和 R-tree 等技术实现高效地处理时态数据.但是,这种处理方式在某种程度上忽视了时间数据与空间数据的不同特性.时间数据和空间数据都可以转换为二维数组[X,Y]进行表示,但是两者的区别在于时间数据通常是具备顺序序列关系,即在二维数组中通常满足 $X \leqslant Y$.而在空间数据中,X 和 Y 一般不具备序列信息.还有一类处理手段是将“数据本体”与“时态信息”协同处理[14−16].这种方法不同上述的两种处理类型,这类方法充分考虑时间本身的特征,并把“数据本体”与“时态信息”的相互关联与相互约束结合在一起.通过建立具有时间特性的时态索引框架,并整合了非时态查询的需求,从而实现“时态查询”与“非时态查询”的协同.通过将有效时间与结构进行整合,文献[14,15]等提出了时态 XML 索引模型.索引的构建充分考虑时态数据、结构信息和语言信息各自的数据特性,将 3 类信息整合在一起,并优化“数据本体”与“时态信息”协同处理的顺序,最终实现对时态 XML 查询和非时态 XML 查询的优化,这些工作关注的都是“有效时间”的查询.文献[16]提出一个一般的时态索引框架 TDindex,该索引可以应用于多种场景中,并实现“数据本体”与“时态信息”的协同处理.而且其基于外存的存储模式可以有效地避免时态索引对于机器性能过高的需求,具有一定的普遍适用性.

这些索引模型是在小规模的历史数据上(部分还针对特定的数据类型)进行模型的构建.然而在数据规模增大、数据本体复杂的情况下,传统的时态索引在进行海量数据的存储和管理时遇到性能上的瓶颈.例如,存在结构存在冗余、空间开销过大的问题,这主要体现在大规模数据上创建时态索引的时间开销过大甚至当数据到达一定规模时难以创建完整时态索引,以及创建索引时需要较大的空间开销、存在不同程度上的空间浪费;在索引更新方面,部分索引没有考虑索引的更新问题或由于索引的结构过于复杂而导致索引的维护代价过高;另一方面,传统的时态索引结构对于时态数据不具有较优的过滤与筛选,在进行时态查询操作时需要遍历海量的索引树结构或进行大量的线性扫描,容易造成查询效率较低或查询效果不理想.

另外一方面,传统的时态索引技术大多是基于单线程技术设计的,相比之下对于并行化的时态索引模型研究较少.大量实验[17]表明,在支持多核、大内存和 NUMA 的现代化硬件(例如 SAP HANA[27])上,几乎所有传统的时态索引性能都不佳.经过深入研究发现,原因在于这些传统的索引不能很好地支持并行化处理,难以有效地利用目前计算机体系结构的优势.文献[17,28]提出一种新颖的时态索引结构 Timeline.它可以很好地结合并行化计算,是一种针对内存列存储方式的“事物时间”版本管理的索引技术.但是 Timeline 对于机器的性能要求极高,默认机器内存高达 192GB,限制了其索引模型的普遍应用,且在进行时态查询时需要执行大量的线性扫描.尽管如此,Timeline 提出的并行化优化方法、具有顺序结构的索引模型等方案,仍然具有很好的借鉴意义.

综合考虑上述问题,结合大数据时代的背景,通过对时态数据的“有效时间”进行深入分析,本文提出一种适

用于海量数据,同时可以结合并行化优化方案,具有外存存储模式的基于时序分区的时态索引技术 TPindex.通过特有的时序分区技术,可以使 TPindex 适用于海量数据的时态索引创建和查询.结合并行化的技术,可以在“拟序关系”的顺序结构中实现高效的时态检索.最后基于外存的存储模式,使得在减少时态索引对于运行机器的过高需求的同时又能充分的发挥现代化计算机体系结构的优势,具有一定的适用性.

2 TPindex 索引

时态数据(temporal data,简称 TD)可以定义为一个二元组 $TD=<RD,VT>$,其中 RD 表示常规数据,VT 表示一个有效时间期间标签.$VT=[VTs,VTe)$,其中 VTs,VTe 分别表示 VT 的开始时间与终止时间($VTs\leqslant VTe$).TD 有效时间记为 $VT(TD)$.如果将有效时间映射在二维平面上,那么不妨将 $VT=[VTs,VTe)$可以看作是二维平面(其中横坐标是 VTs,纵坐标是 VTe)的坐标点(VTs,VTe),则实现时间期间集 Γ 和平面 VTs-VTe 点集 $H(\Gamma)$一一对应.在本文中,对于 Γ 和 $H(P\Gamma)$,不做细致区分.

2.1 时序分区

定义 1(时序分区(timing partition)). 对于平面集 $H(\Gamma)$上满足拟序关系的数据集 E,按照某种特定方式进行平面分区的操作,称为时序分区.Γ 经过时序划分之后,得到若干个分区平面集 $H(P\Gamma)$,其中 $H_iP\Gamma$)表示第 i 个分区平面.同上,在本文中,对于 $H(P\Gamma)$与 $P\Gamma$ 不做详细区分.

事实上,在时态查询的操作过程中,满足查询条件的时态数据相较于整体的时态数据比例较小,存在着大量的“无关”数据,尤其是在海量数据的情况下,如果将数据全部读取再进行查询比较,其付出代价将会过于昂贵.因此,在进行时态查询操作之前,将大量的“无关”数据过滤,可以极大地减少中间查询比较过程的工作量,提高时态查询操作的效率.数据分区[29−31]作为海量数据预处理策略的一种,在缓解内存压力,提高索引效率等方面有着较好的性能提升.通过数据分区,使得数据能够有规则的“缩减”和尽可能的分布均匀.针对时态数据独特的属性组成,我们提出了一种基于时态降维的自上而下,自左而右的时序分区方法.在时序分区中,我们采用随机采样的方法.不妨设采样阈值为 α,数据总量为 S,综合考虑时态数据总量的稀疏程度与离散化程度,我们给出了公式(1)用以计算采样数量 D,公式(2)用以计算时序分区数量.

$$D=S\times\alpha \tag{1}$$

$$k=2+\log(1+\lambda e^{\sigma}) \tag{2}$$

其中,k 表示分区的数量(向下取整的正整数),λ 代表内存消耗系数($0\leqslant\lambda\leqslant1$),$\sigma$表示时态数据的膨胀系数($0\leqslant\sigma$),用以衡量时态数据的稀疏性与离散化程度.时态数据分布范围越广、越密集,说明膨胀系数越大.例如,给定时态数据集 $H_1(\Gamma)$={[1,10],[1,9],[1,8],[1,7],[1,6],[1,5],[1,4],[1,3],[1,2],[1,1]},$H_2(\Gamma)$={[1,10],[1,9],[1,6],[1,1]},可以看出,虽然 $H_1(\Gamma)$和 $H_2(\Gamma)$的数据分布范围都是由[1,10]到[1,1],但是 $H_1(\Gamma)$的数据相对于 $H_2(\Gamma)$而言更稠密,因此在进行时序分区时,可以认为需要划分更多的时序分区,即相应的膨胀系数也会更大.由公式(2)可以得到,索引默认的最小时序分区为 2(时序分区等于 1 时没有意义).通过上述公式计算的分区数量可以保证每个分区的数据加索引的体积不超过数据总量.

在确定分区数 k 之后,将随机抽取的样本按照自上而下、自左而右的时序排列并找出 k–1 个分段点.首先,根据采样的阈值大小得到相应的样本数据.针对时态数据特殊的时间属性 VT,本文通过时态降维的方法,通过映射函数 f 将二维平面上的任意 VT 区间转换成一维数据集 N.最后将 N 通过快速排序递归算法查找到 k–1 个分段点.本文给出了时态降维映射函数如公式(3)所示.

$$f=e^{\left(10^{\eta}\frac{VTs}{M}\right)}-\frac{VTe}{M} \tag{3}$$

其中,η 表示 VTs 的最大位数,M 表示所有时态数据中“有效时间”元组里最大的 VTe(即所有数据中处于最后一个元组中的 VTe).在进行时态降维过程中,主要考虑两个问题:当 VTs 不同时,VTs 越大,时态降维得到的结果越大;当 VTs 相同时,VTe 越大,映射的时态降维结果越小.通过上述公式,可以很好地区分不同时态点的映射位置,M 的平

滑化处理可以控制 *VTe* 对于整体映射结果的影响,即当 *VTs* 不同时,对于时态降维结果的大小判断几乎不产生影响(此时的大小由 *VTs* 的大小决定).当 *VTs* 相同时,又可以根据 *VTe* 的大小进行时态降维结果大小的判断.

例 1:下面给出有关于时态降维的例子.给定无序的 *VT* 点集 *I*={[1,8],[2,6],[3,6],[7,8],[2,5],[4,9],[5,8]},分别用 *a*,*b*,*c*,*d*,*e*,*f*,*g* 表示.根据给出的函数 *f*,得到的时序排序结果为 *a*<*b*<*e*<*c*<*f*<*g*<*d*.

综上所述,通过时态降维的处理方式,将时态数据特有的 *VT* 属性转换成一维相点,进行自上而下,自左而右的排序.可以看出,通过将二维时间区间转化为具有一定顺序排列的一维操作,形成了多个不同分分区层.在时态数据中具有大量与目标数据"无关"的时态点,因此在进行索引查找的时候,通过分区层过滤掉大量的"无关节点",再判断剩余时态点是否满足查询条件,可以很好地节省查询处理的时间开销.在整个时序分区过程中,算法的平均时间复杂度为 $O(n\log n)$,平均的空间复杂度也为 $O(n\log n)$.

2.2 时态拟序结构

定义 2(时态拟序关系(temporal quasi-ordering relation)). 若 *R* 是集合 *Γ* 中时态数据集 *E* 满足自反性和传递性的关系,则称 *R* 是 *E* 上的一个拟序.下面给出相关的拟序关系.其中,

- 自反性:若$\forall TD_1\in E$,满 $VT(TD_1)\subseteq VT(TD_1)$.
- 传递性:$\forall TD_1\in E,\forall TD_2\in E,\forall TD_3\in E$,若 $VT(TD_1)\subseteq VT(TD_2)$,且 $VT(TD_2)\subseteq VT(TD_3)$,则有 $VT(TD_1)\subseteq VT(TD_3)$.

定义 3(*col* 集合). 对于$\forall Y_0\in H(\Gamma)$,有 $col(Y_0)$表示 $H(\Gamma)$中 $VTs(P)=VTs(P_0)$的点 *P* 集合.

本文提出了时态索引技术是的线序划分基础上进行的,该方法引入了"拟序关系"概念,这是一种与传统的"代数"模式不同的关系模式.通过将 $H(PI)$上的时态数据按照拟序关系进行划分,进而将数据按照某种特定的结构进行组织构建,从而使得进行时态数据查询检索时,由该特定结构的良好过滤性而达到高效的检索需求.因此,我们需要对相应的线序构造算法进行探讨.其中最主要的思想是:为了在每个时序分区的时态数据中,TPindex 从 $H(PI)$"最左上方"的点开始,通过同列优先,而后从左到右的原则进行构建,直到所有的元素点进入相应的线序中.该算法伪代码如算法 1 所示.

算法 1. $H(PI)$"下右优先"线序划分构建算法.

输入:$H(PI)$.

输出:$L_1,L_2,...,L_n$.

1. $u(i,j)$=the "top left" point, $P=u(i,j)$, $L_k=\{P\}$, k=1;
2. **for** m=*i* to *j* **do**
3. traverse the $H(PI)$ the $col(i)$, while the $K(i,m)$=traverse point;
4. $L_k.add(K(i,m))$, $P=K(i,m)$;
5. **if** $\exists N(i+1,m)\in H(PI)$ **and** $VTe(N)\leqslant VTe(K)$ **then**
6. $L_k.add(N(i+1,m))$; goto Step 2;
7. **else** *i*++;
8. **end if**
9. **if** *i*==*m* **then**
10. goto Step 11;
11. **else** goto Step 2;
12. **end if**
13. **end for**
14. **return** L_k

通过上述算法,按照"下优先"和"右优先"的原则,经过有限次迭达之后,我们得到若干条完整的"路径"(在下文另有详细说明).我们在不妨设 $g=\max\{VTs(u)|u\in\Gamma\}$,$h=\max\{VTe(u)|u\in\Gamma\}$,则算法 1 的时间复杂度为$(g\times h)/2$.

例 2:假设在某一个时序分区内的数据为 $H(PI)$={[1,9],[1,7],[1,6],[1,5],[2,8],[2,7],[2,5],[2,3],[2,2],[3,8],[3,7],[3,4],[3,2],[4,8],[4,7],[4,4],[4,2],[5,10],[5,9],[5,7],[5,5],[6,10],[7,10],[7,9],[7,8],[7,7],[7,6]},则根据算法 1 可以得到

构建后的结构图如图 1 所示,其中若干条“路径”可以表示为<L_1,L_2,L_3,L_4,L_5>,L_1={[1,9],[1,7],[1,6],[1,5],[2,5],[2,3],[2,2],[3,2],[4,2]},L_2={[2,8],[2,7],[3,7],[3,4],[4,4]},L_3={[3,8],[4,8],[4,7],[5,7],[5,5]},L_4={[5,10],[5,9]},L_5={[6,10],[7,10],[7,9],[7,8],[7,7],[7,6]}.

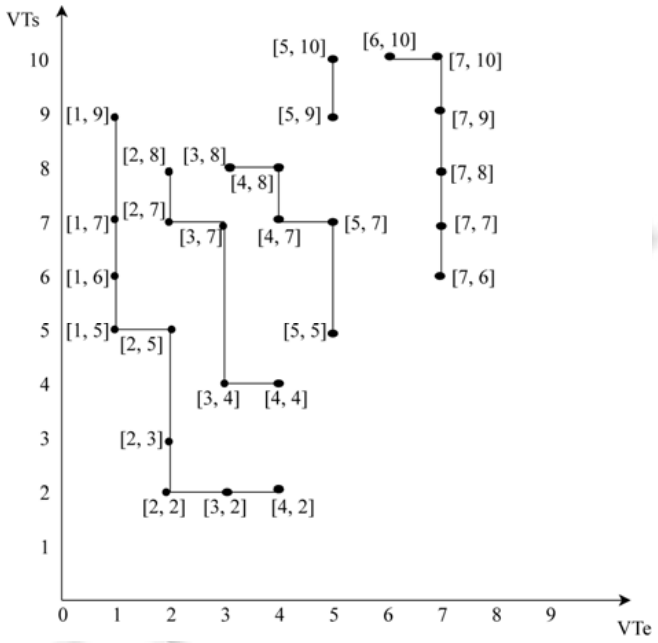


Fig.1 线序构造

图 1 Linear order building

2.3 时态索引TPindex构建

定义 4(分区线序划分(partition linear order partition,简称 PLOP)). 通过算法 1,我们得到的序列 $T(PI)$=<L_1,L_2,…,L_n>满足以下性质:

若$\forall L_i,L_j\in T\wedge i\neq j\Leftrightarrow L_i\cap L_j=\varnothing$,且$\cup L_i=I(1\leqslant i,j\leqslant n)$.这样的 $T(PI)$称为 $H_i(PI)$上的一个分区线序划分 PLOP,记为 $PLOP=T(PI)$,而 $PLOP$ 中每一个 L_i 称为该 $PLOP$ 的一个分区线序分枝,记为 PLOB(partition linear order branch).每个 $PLOP$、$PLOB$ 具有以下的基本性质:

1) 每个 $PLOP$ 是 $H_i(PI)$的划分.

2) 不妨设 $VTs(L_i)$和 $VTe(L_i)$是分别是 $L_i(1\leqslant i\leqslant n)$的起点和终点序列,则 $VTs(L_i)$单调增加,$VTe(L_i)$单调减少.

定义 5(分枝首元和尾元). 对于 $L\in PLOP(I)$,将每个分枝 L 中第 1 个节点称为分枝首元,记为 max(L),将 L 中最后一个节点称为尾元,记为 min(L).同时,将 $PLOP(I)$中的所有 max(L)集合记为 PImax,LOP(I)中的所有 min(L)集合记为 PImin.

定义 6(分区时态索引 TPindex). 关于 $H(PI)$上时态索引 TPindex 满足下列定义.

(1) 时态分区层(Partition level):Partition=<P1,P2,…,Pn>,用来保存若干个 $H(PI)$的分区点 $Pi(1\leqslant i\leqslant n-1)$.在时序分区操作结束后,对每个分区 $H(PI)$进行分区线序划分,由此组成 TPindex 的下层索引结构(2)~(5).

(2) 时态分区根节点层(Partition root Level):<r_1,r_2,…,r_m>,r_i=max($L_i(PI$max)),$1\leqslant i\leqslant m$(注:这里以某一具体的时序分区为例).

(3) PLOP MAX Level 层(PImax):在 $PLOP$ 上构建 $PLOP(PI\text{max})=\{L_i(PI\text{max})\}(1\leqslant i\leqslant|PLOP(PI\text{max})|)$,该层节点为 $L_i(PI$max).

(4) PLOP MIN(PImin)层:上层节点 L_i (PImax)中对应元素 $PLOB$ 集合构建 $PLOP(PI\text{min}(L_i(PI\text{max})))(1\leqslant$

$r \leqslant |PLOP(PI\text{-}\min(L_i(PI\text{-}\max)))|$),该层节点为 $PLOP(PI\text{-}\min(L_i(PI\text{-}\max)))$中 $L_i(PI\text{-}\min(L_i(PI\text{-}\max)))$.

(5) 叶节点层(PLOB):本层节点为 $L_i(PI\text{-}\min(L_i(PI\text{-}\max)))$对应元素 *PLOB*.

可以看出,通过线序划分算法,我们得到了一种具有拟序关系的“树形”结构 TPindex,在该“树形”结构(即TPindex)中,时态分区层属于上层分区结构,时态分区根节点层到 $PLOP(PI\text{-}\min)$层属于 TPindex 的“索引层”,叶节点层(*PLOB*)属于 TPindex 的元素层.下面我们给出 TPindex 一般的分区索引结构,如图 2 所示.

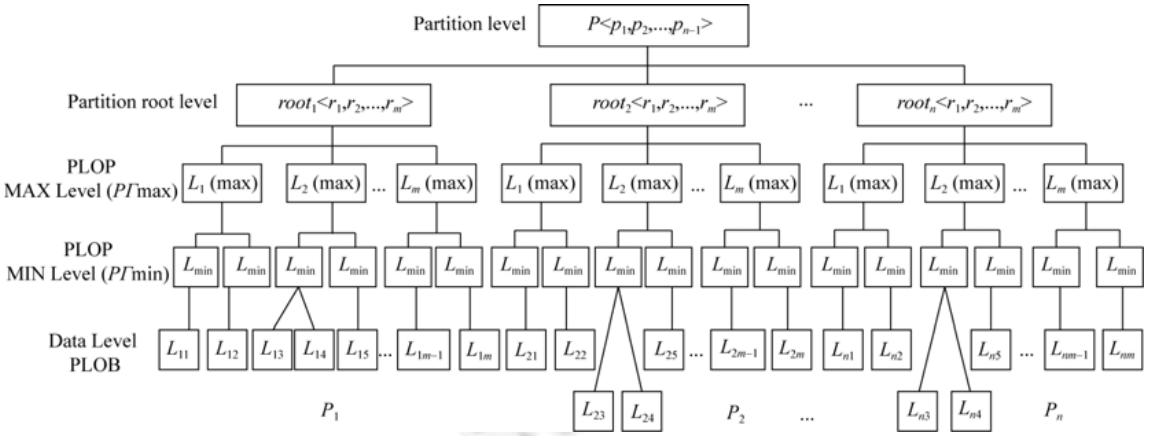


Fig.2 Partition index structure of TPindex

图 2 TPindex 的分区索引结构

3 TPindex 索引查询

在本节,我们将讨论时态索引 TPindex 的查询算法,包括基于 PLOB 的内部二分优化查询、基于 PLOB 的多线程查询以及基于外存的文件模式查询,以适用于支持多核、并行化等优势的现代化硬件,并给出相应的算法.

3.1 基于PLOB的内部二分优化查询

基于上述的 TPindex 索引结构,结合 PLOP 和 PLOB 对于时态数据的有效过滤,我们提出了基于 PLOB 的内部二分优化查询.在对 TPindex 进行时态数据二分优化查询的时候,我们首先要对查询窗口与 $H(PI)$进行时序分区点进行比较,为后面的数据查询缩小“范围”,减少不必要的时间开销.在本文的时态查询,是指在平面集 $H(I)$中找到包含查询窗口 Q 区间的所有时态数据.假设存在这样的数据{[1,10],[1,9],[2,8],[3,6],[4,6]},给定查询窗口 Q=[2,7],则满足查询要求的序列集合为{[1,0],[1,9],[2,8]}.

定义 7(分区点与查询窗口的关系比较). 不妨设查询窗口为 $Q<vts_0,vte_0>$,有 n 个待查询分区 $H(PI)$,即有 n–1 个时序分区点 $P_k<vts_k,vte_k>(1 \leqslant k \leqslant n-1)$,则对于 Q 与 P_k 的关系有以下几种情形.

(1) 若 $vts_0>vts_k$,则称 P_k 在 Q 的右边,符合查询条件的分区是 $P_1,P_2,\ldots,P_{k+1}$,记 $P_k>Q$.

(2) 若 $vts_0=vts_k$ && $vte_0>vte_k$,则称 P_k 在 Q 的上边,符合查询条件的分区是 $P_1,P_2,\ldots,P_{k+1}$,也记作 $P_k>Q$.

(3) 若 $vts_0=vts_k$ && $vte_0<vte_k$,则称 P_k 在 Q 的下边,符合查询条件的分区是 $P_1,P_2,\ldots,P_k$,记 $P_k<Q$.

(4) 若 $vts_0<vts_k$,则称 P_k 在 Q 的左边,符合查询条件的分区是 $P_1,P_2,\ldots,P_k$,也记作 $P_k<Q$.

综上所述,若 $P_k>Q$,则符合查询条件的分区是 $P_1,P_2,\ldots,P_{k+1}$;若 $P_k<Q$,则符合查询条件的分区是 $P_1,P_2,\ldots,P_k$.

在对相关的时序分区进行筛选之后,由定义 4,我们得到了 *VTs*(*PLOB*)的单增性和 *VTe*(*PLOB*)的单减性,不妨设 PLs_i 为当前 *PLOB* 的 *VTs*(*PLOB*)序列,$i(\max \leqslant i \leqslant \min)$,$PLe_j$ 为当前 *PLOB* 的 *VTe*(*PLOB*)序列,$j(\max \leqslant j \leqslant \min)$,则针对基于拟序的时态索引,本文给出了以下算法进行时态数据查询操作,如算法 2 所示.

算法 2. 基于 PLOB 的内部二分优化查询算法.

输入:查询窗口 Q.

输出:查询结果集 R.

1. **if** $VTs(Q)<PLs_{max}$ or $VTe(Q)>PLe_{max}$ **then**
2. $R=\varnothing$; goto Step 44;
3. **end if**
4. PLs_{mid}=the midpoint of PLs, PLe_{mid}=the midpoint of PLe; mid=Int((max+mid)/2);
5. **if** $VTs(Q)<PLs_{mid}$ and $VTe(Q)>PLe_{mid}$ **then**
6. **if** max<min **then**
7. min=mid–1; goto Step 4;
8. **else** goto Step 44;
9. **end if**
10. **else if** $VTs(Q)\geqslant PLs_{mid}$ and $VTe(Q)\leqslant PLe_{mid}$ **then**
11. **if** max<min **then**
12. max=mid+1; goto Step 4;
13. **else** goto Step 44;
14. **end if**
15. **else if** $VTs(Q)<PLs_{mid}$ and $VTe(Q)\leqslant PLe_{mid}$ **then**
16. min=mid–1; goto Step 20;
17. **else if** $VTs(Q)\geqslant PLs_{mid}$ and $VTe(Q)>PLe_{mid}$ **then**
18. min=mid–1; goto Step 32;
19. **end if**
20. PLs_{mid}=the midpoint of PLs;
21. **if** $VTs(Q)<PLs_{mid}$ **then**
22. **if** max<min **then**
23. min=mid–1; goto Step 4;
24. **else** goto Step 20;
25. **end if**
26. **else if** $VTs(Q)\geqslant PLs_{mid}$ **then**
27. **if** max<min **then**
28. max=mid+1; goto Step 4;
29. **else** goto Step 20;
30. **end if**
31. **end if**
32. PLe_{mid}=the midpoint of PLe;
33. **if** $VTe(Q)>PLe_{mid}$ **then**
34. **if** max<min **then**
35. min=mid–1; goto Step 15;
36. **else** goto Step 20;
37. **end if**
38. **else if** $VTe(Q)\leqslant PLe_{mid}$ **then**
39. **if** max<min **then**
40. max=mid+1; goto Step 15;
41. **else** goto Step 20;

42. **end if**

43. **end if**

44. Put all the points from the start point to the midpoint of PLOB into *R*

在上述二分优化查询算法中,由于*PLOB*具有独特的拟序关系,因此通过二分查找可以实现内部线序结构查找的高效检索效率.对于查询结果,若*PLOB*中包含查询窗口*Q*,则该*PLOB*上所有元素都视为查询结果.因此,通过二分优化查找找到*PLOB*上最后一个包含查询窗口*Q*的元素*M*,若存在这样的点,则从*PLOB*的第1个元素到元素*M*均为查询结果.设*n*为每个*PLOB*分枝节点的个数,则*n*=min–max+1,且算法2时间复杂度也为$O(\log n)$.

根据*PLOB*的拟序关系可知*VTs*的单增性与*VTe*的单减特性,因此在下面两组情况下,使用二分优化查找时,只在始点序列或只在终点序列中做比较即可,减少了结束有效时间序列的比较次数,实现了更加高效的检索时间.

(1) 当$VTs(Q)<PLs_{\text{mid}}$且$VTe(Q)\leqslant PLe_{\text{mid}}$,即查询窗口*Q*的开始有效时间小于中点的开始时间且*Q*的结束时间小于或等于中点的结束时间.

(2) 当$VTs(Q)\geqslant PLs_{\text{mid}}$且$VTe(Q)>PLe_{\text{mid}}$,即查询窗口*Q*的开始有效时间大于或等于中点的开始时间且*Q*的结束时间大于中点的结束时间.

例3:假设有查询窗口*Q*=[3,4],*PLOB*={[1,9],[1,7],[1,6],[1,5],[2,5],[2,3],[2,2],[3,2],[4,2]},则由算法2可得,算法初始的max=[1,9],*mid*=[2,5],min=[4,2],序列数据中的*PLs*集合为{1,1,1,1,2,2,2,3,4},*PLe*集合为{9,7,6,5,5,3,2,2,2}.观察查询窗口*Q*与*PLs*和*PLe*的关系,可以看出$\forall\tau\in PLs$,都满足$VTs(Q)\geqslant\tau$.因此,对于*VTs*来说,无论PLs_{mid}位于哪里,都满足查询窗口的检索要求.此时,只需要对*PLe*集合进行考虑即可.在*PLe*集合中,初始的PLe_{mid}=5,此时满足查询要求,即此时的查询结果集合*R*1={[1,9],[1,7],[1,6],[1,5],[2,5]}.在进行第2次二分查找时,此时max=[2,3],*mid*=[2,2],min=[4,2],*PLe*2集合为{3,2,2,2}.对于$\forall\omega\in PLe2$,都满足$VTe(Q)>\omega$,即在序列集合{[2,3],[2,2],[3,2],[4,2]}中,都不满足包含*Q*的查询要求,此时查询结果集合*R*2=∅.算法执行完毕,综合两次查询结果,可以得到最终的查询结果集合为$R=R1\cup R2$={[1,9],[1,7],[1,6],[1,5],[2,5]}.通过上述说明,可以看出,结合二分查询算法与线序分枝结构的性质,可以达到高效的时态数据检索效率.

3.2 基于PLOB的多线程查询

计算机硬件的不断发展也为时态索引的性能带来了新的性能提升空间.其中,并行化计算技术已经趋向于成熟,它可以通过多核处理器的强大计算能力和更大的存储资源来为海量数据进行高效的处理,很好地提高现代化硬件资源的利用率.因此,研究海量数据下如何结合现代化硬件的优势,带来索引性能的提升,是索引在进行构建时需要考虑的问题.为了充分地调用现代化计算机硬件的体系结构资源,利用多核处理器的强大计算能力及存储资源来解决大规模历史数据问题,在使用TPindex索引进行时态查询时,可以结合多线程技术并行化地处理用户的查询.同时,TPindex特有的上层“分区层”可以很好地结合并行处理机制,即每个分区可以有若干个线程,而这些线程可以分配到不同的“块”进行时态操作的处理.因此可以利用多线程技术并行处理多个分区的时态操作或者针对单个的分区分配若干个线程进行并行处理.

给定时态数据集$H(\Gamma)$,我们不妨设在该时态平面上建立*n*个TPindex分区时态索引结构,每个分区中包含若干个*PLOP*,而每个*PLOP*又有若干个*PLOB*.现有查询窗口*Q*,*t*个线程,每个分区将会根据线程个数,自动分配线程处理的线序分枝块,保证块内有序,直至调用完分区内的所有线程.进而,当运行到每个线序分枝块的具体查询时,使用上面提出来的基于*PLOB*内部二分优化查询算法,得到每一块的查询结果.最后把每个分区中的所有块的查询结果合并到一起,就是我们的所需要的查询结果集.基于*PLOB*的多线程查询模式如图3所示.

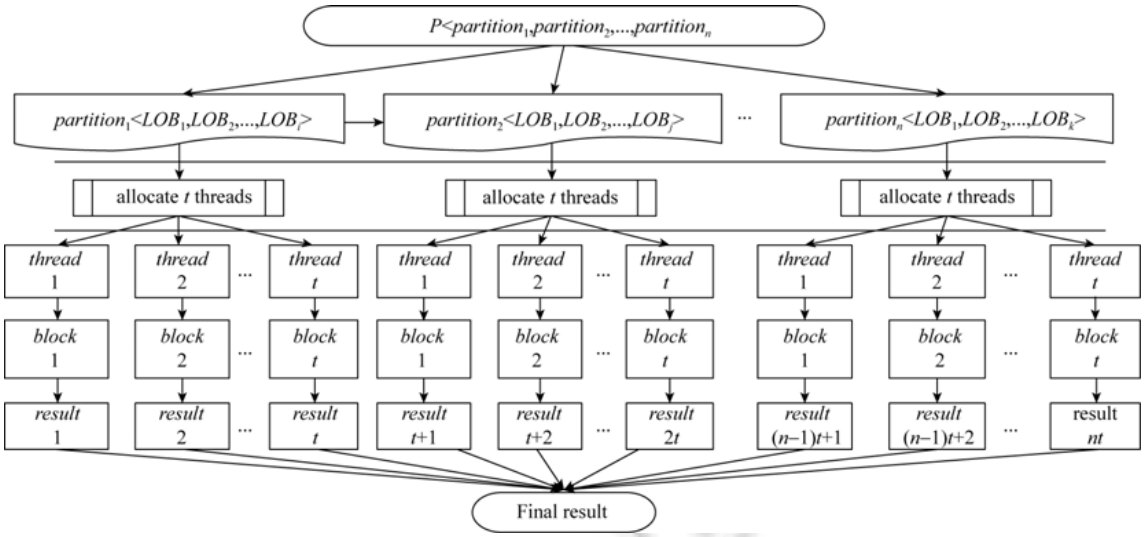


Fig.3 Multi-thread query model based on PLOB

图 3 基于 PLOB 的多线程查询模式

3.3 基于外存的分文件模式查询

考虑到在面对海量数据时,基于内存的索引结构存在因节点硬件条件限制而存在着内存溢出和计算速度缓慢等风险,从而影响索引查询的效率,结合上述数据结构,本文提出了一种基于外存的分文件模式查询.该模式通过在磁盘内以文件格式存放已经建立好的分区 TPindex 索引.每个分区都有一个分区根目录 $Plob\_root_i$,用来存储该分区内每个分枝的首元节点 $Plob\_td_{\max}$ 和尾元节点 $Plob\_td_{\min}$.其次再将每个分区的 TPindex 包含的每个 PLOB 存储为每个单独对应的文件,用于后续的磁盘数据索引的读取.线序分枝索引分文件存储模式如图 4 所示.

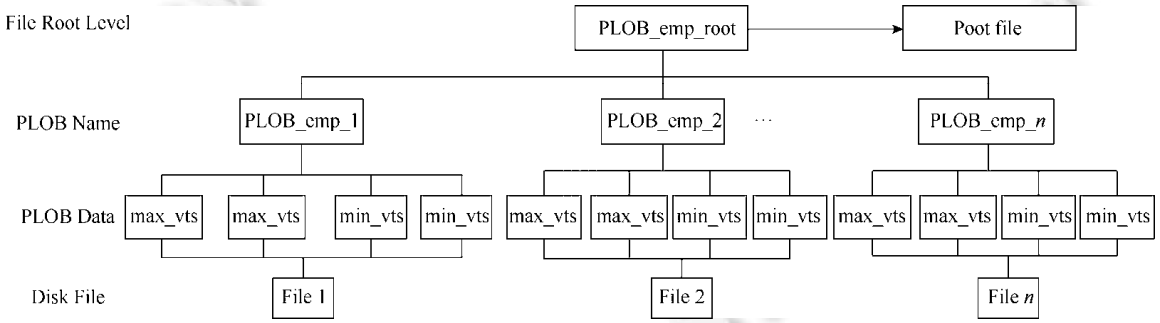


Fig.4 Query on Separated files mode

图 4 分文件模式查询

通过将索引磁盘序列化,将原有的 TPindex 分区层根节点与多个 PLOB 线序分枝分别一一映射到磁盘外存中,外存存储同样地具有两层结构(磁盘分区层与磁盘索引层).在进行时态查询时,TPindex 依次并行的读取磁盘的分区层根节点(即对多个 $Plob\_root_i$ 依次进行并行访问)元素,通过定义 7 确定目标数据所在的时序分区.由于外存中的磁盘索引层同样具有顺序结构,因此,只需将磁盘中线序分枝文件读取到内存中,重新组合成新的 PLOB 分枝.结合二分优化查找方案在索引层进行数据的筛选..由于磁盘外存存储的独立性和查询的部分性,使得在进行时态数据查询的时候,可以有效地减少访问磁盘的 I/O 次数和时间开销,进而达到了提高查询效率的目的.此外,通过外存存储的方式,可以将海量数据进一步“离散化”,而且磁盘分区层同样可以过滤掉大量与查询处理“无关”的时态点,再将顺序结构的磁盘索引读取进内存重新排列组合,进一步的判断剩余时态点是否满足查

询条件,高效地进行时态查询处理.

4 TPindex 索引更新

时态数据库的更新通常包括全量式更新和增量式更新,考虑到全量式更新在面对海量数据时存在难以重复利用原有索引和效率较低等问题,在 TPindex 中主要以增量式更新为主.接下来,我们讨论 TPindex 索引的时态数据插入和删除操作.

定义 8(局部域). 在时态平面上,对于$\forall y_0 \in H(PI)$,根据 y_0 所在的坐标位置,可以将对应的 $H(PI)$分成以下 4 个区域.

$L_{\text{Top}}(y_0)=\{y|y\in H(PI)\wedge VTs(y)\leqslant VTs(y_0)\wedge VTe(y_0)\leqslant VTe(y)\}$,

$L_{\text{Down}}(y_0)=\{y|y\in H(PI)\wedge VTs(y)\leqslant VTs(y_0)\wedge VTe(y)\leqslant VTe(y_0)\}$,

$R_{\text{Top}}(y_0)=\{y|y\in H(PI)\wedge VTs(y_0)\leqslant VTs(y)\wedge VTe(y_0)\leqslant VTe(y)\}$,

$R_{\text{Down}}(y_0)=\{y|y\in H(PI)\wedge VTs(y_0)\leqslant VTs(y)\wedge VTe(y)\leqslant VTe(y_0)\}$.

其中,$L_{\text{Top}}(y_0)$,$L_{\text{Down}}(y_0)$,$R_{\text{Top}}(y_0)$和 $R_{\text{Down}}(y_0)$分别代表 y_0 在 $H(PI)$中的“左上区域”“左下区域”“右上区域”“右下区域”.同时,为了方便相关阐述,将 $L_{\text{Top}}(y_0)$,$L_{\text{Down}}(y_0)$,$R_{\text{Top}}(y_0)$和 $R_{\text{Down}}(y_0)$的真子集分别定义如下.

$SL_{\text{Top}}(y_0)=\{y|y\in H(PI)\wedge VTs(y)<VTs(y_0)\wedge VTe(y_0)<VTe(y)\}$,

$SL_{\text{Down}}(y_0)=\{y|y\in H(PI)\wedge VTs(y)<VTs(y_0)\wedge VTe(y)<VTe(y_0)\}$,

$SR_{\text{Top}}(y_0)=\{y|y\in H(PI)\wedge VTs(y_0)<VTs(y)\wedge VTe(y_0)<VTe(y)\}$,

$SR_{\text{Down}}(y_0)=\{y|y\in H(PI)\wedge VTs(y_0)<VTs(y)\wedge VTe(y)<VTe(y_0)\}$.

在更新索引之前,通过 4 个局部域的划分,索引结构可以快速地定位的目标数据的所在位置,缩小索引结构更新的范围.同时.局部域可以保持更新前后时态数据的一致性,避免出现索引“缺失”、“断裂”等情况,为时态索引的增量式更新提供有效地基础支持.

定义 9(前邻居点和后邻居点). 设 $L_0\in PLOP$,如果$\exists x_0\in L_0$,且满足 $VTS(x_0)=VTS(y_0)\wedge VTe(x_0)=\min\{x|VTe(y_0)\leqslant VTe(x)\}$,则称 x_0 是 y_0 在 L_0 上的一个前邻居点,记为 $Front(y_0)$;如果$\exists x_0\in L_0$,且满足 $VTe(x_0)=VTe(y_0)\wedge VTe(x_0)=\min\{x|VTs(y_0)\leqslant VTs(x)\}$,则称 x_0 是 y_0 在 L_0 上的一个后邻居点,记为 $Back(y_0)$.

定义 10(最近下分枝)设 $Line(PLOB)$是 $H(PI)$上从 $\max(PLOB)$到 $\min(PLOB)$遍历 $PLOB$ 的路径,$L_0=<x_1,\ldots,x_{i-1},x_i,x_{i+1},\ldots,x_{j-1},x_j,\ldots,x_m>$,$L_{\text{next}}$ 表示在同一个 $PLOP$ 中,与 L_0 相邻的下一个 $PLOB$ 分枝,称 L_{next} 是 L_0 的最近下分枝.

通过上述的定义说明,我们将进一步讨论 TPindex 时态索引的动态更新机制,结合 PLOB 的结构特性,我们给出以下增量式的动态更新算法.

4.1 TPindex插入更新

时态节点 y_0 的插入主要分为 3 种情况:在当前的分枝上新增节点、新增线序分枝、引起当前分枝的变化或分裂.其主要的思想是:通过插入点 y_0 找到对应的 PLOB,若未找到相应的 PLOB,则将插入点单独新增为一个 PLOB.否则,结合定义 8 找到 $SL_{\text{Down}}(y_0)$,同时将 PLOB 中 $SL_{\text{Down}}(y_0)$的分枝部分切除,若该部分不为空,则递归重复以上操作,将切除的分枝部分插入到相应的 PLOB 中,直至该切除部分为空.新形成的线序分枝中的元素仍满足拟序关系,具体的算法插入过程如算法 3 所示.

算法 3. 基于 PLOB 的时态数据插入算法.

输入:插入节点 y_0,需要插入 $PLOB$.

输出:更新后的 $PLOB$.

1. select the corresponding L_0 according to the insert point y_0
2. **if** $PLOP\subseteq SR_{\text{Top}}(y_0)\vee PLOP\subseteq SL_{\text{Down}}(y_0)$ **then**
3. new $PLOB=\{y_0\}$;
4. **end if**

5. **if** $y_0 \in Line(L_0)$ **then**
6. $L_0.insert(y_0)$;
7. **end if**
8. **if** $L_0 \in PLOP$ and $(VT(y_0) \subseteq \min(L_0) \vee \max(L_0) \subseteq VT(y_0))$ **then**
9. $L_0.insert(y_0)$;
10. **end if**
11. **if** $\exists Front(y_0)$ and $\exists Back(y_0)$ **then**
12. new $PLOB=<x_1,\ldots,x_i,y_0,x_j,\ldots,x_m>$;
13. put the $<x_{i+1},\ldots, x_{j-1}>$ as a new insert set; goto Step 1;
14. **end if**
15. **if** $\exists Front(y_0)$ and $!(\exists Back(y_0))$ and $R_U(y_0) \cap L_0 = \varnothing$ **then**
16. new $PLOB=<x_1,\ldots,x_i,y_0,x_{i+1},\ldots,x_m>$;
17. **else if** $\exists Front(y_0)$ and $!(\exists Back(y_0))$ and $R_U(y_0) \cap L_0 \neq \varnothing$ **then**
18. new $PLOB=<x_1,\ldots,x_i,y_0>$;
19. put the $<x_{i+1},\ldots,x_m>$as a new insert set; goto Step 1;
20. **end if**
21. **if** $!(\exists Front(y_0))$ and $\exists Back(y_0)$ **then**
22. new $PLOB=<y_0,x_j,\ldots,x_m>$;
23. put the$<x_1,\ldots,x_{j-1}>$ as a new insert set; goto Step 1;
24. **end if**
25. **if** $!(Front(y_0))$ and $!(\exists Back(y_0))$ and $R_U(y_0) \cap L_0 = \varnothing$ **then**
26. new $PLOB=\{y_0\}$;
27. **else** new $PLOB<y_0,x_{i+1},\ldots,x_m>$;
28. put the $<x_1,\ldots,x_i>$ as a new insert set; goto Step 1;
29. **end if**

例 4:由算法 3 可得,TPindex 在进行插入更新时,会引起 3 种不同情况的变化.在第 1 种情况中,即插入新节点,而不引起现有线序分枝的变动,新插入的节点 y_0 独立为一个新的线序分枝,如图 5(a)所示.当插入节点 y_0 是[0,4]时,满足 $PLOP \subseteq SR_{Top}(y_0)$,此时 y_0 作为一个新的线序分枝;当插入节点是 y_0 是[9,10]时,满足 $PLOP \subseteq SL_{Down}(y_0)$,同样将 y_0 作为一个新的线序分枝.在第 2 种情况下,即插入新节点会引起某一个线序分枝的序列增加,如图 5(b)所示,插入节点 y_0 是[1,10],[7,5],[4,1]时,可以在原有的线序分枝的首元节点和尾元节点执行插入操作并更新原有的首元节点与尾元节点.在第 3 种情况中,插入的新节点会引起原有分枝的结构的变化或分裂,如图 5(c)、图 5(d)所示,当插入节点 y_0 是[5,6]时,则将原分枝中[5,7]与[5,5]之间的连接打断,再由 y_0 分别连接[5,7]与[5,5];当插入的节点 y_0 是[4,6]时,从图 5(d)中可以看到,此时原有的分枝会分裂(即[4,7]与[5,7]之间的连接断开),分裂后的[4,7]节点根据算法 1 连接到[4,7]节点,成为其所有分枝的新尾元节点;原有分枝中被“截断”的[5,7]和[5,5]则由算法 1 再重新连接到[5,9]节点之后,构成新的线序分枝.

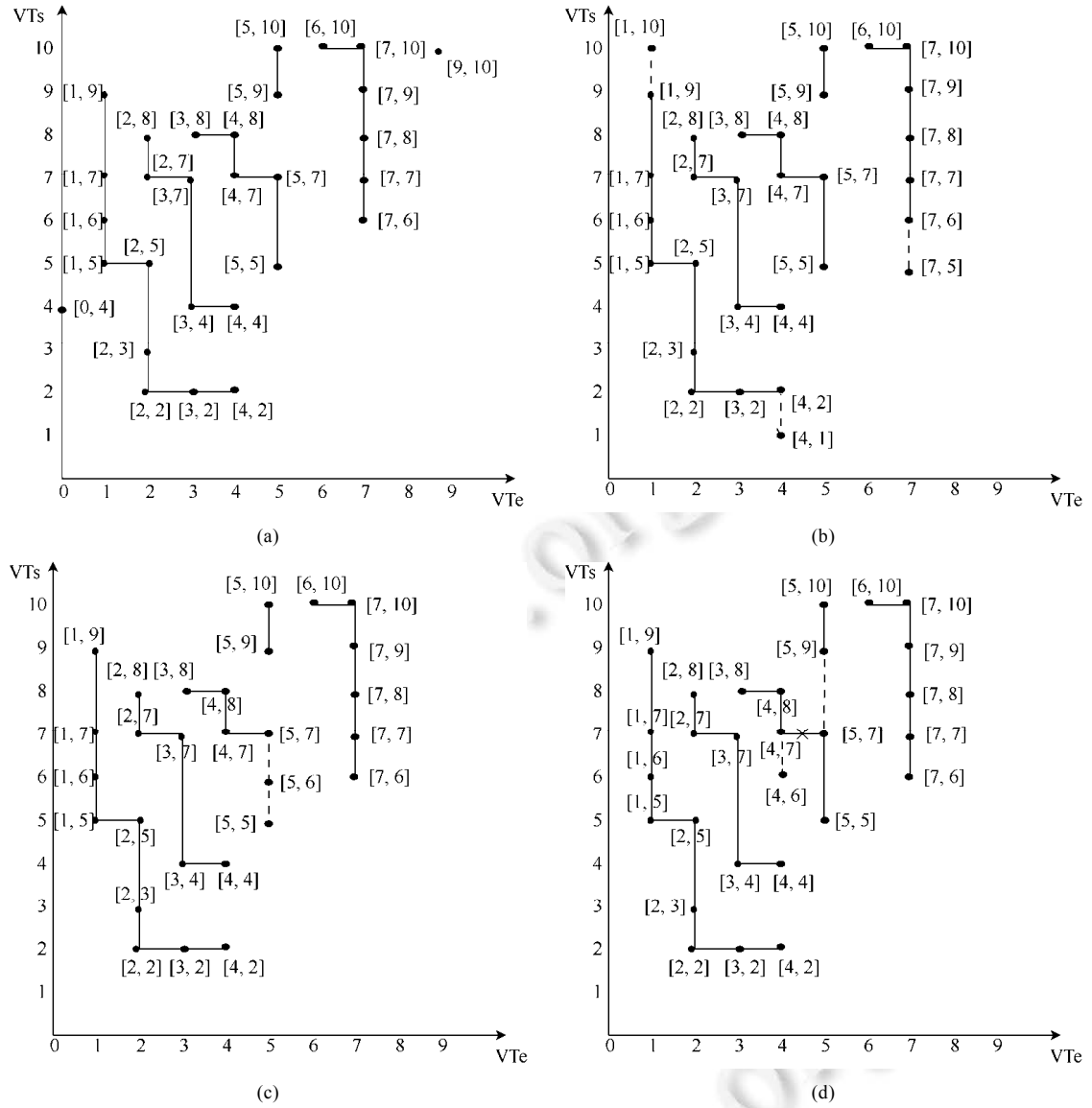


Fig.5 Insert update in TPindex

图 5 TPindex 插入更新

4.2 TPindex删除更新

为了保证时态数据在删除之后,线序分枝不会出现“断裂”情况,并且新形成的线序分枝元素仍满足拟序关系,因此有必要对时态索引的删除操作做出对于的结构更新,在这过程中,原有的线序的分枝会有以下情况;对应的线序分枝合并或删除.其主要是思想是讨论 $Front(y_0)$,$Back(y_0)$和 L_0 及 L_{next} 的位置关系,将节点 y_0 删除,将 L_0 和 L_{next} 的剩余元素再重新拼接.具体的删除过程如算法 4 所示.

算法 4. 基于 PLOB 的时态数据删除算法.

输入:删除节点 y_0 及其对应的 *PLOB*.

输出:更新后的 *PLOB*.

1. Select the corresponding L_0 according to the insert point y_0

2. **if** ($VTs(Front(y_0))=VTs(y_0)=VTs(Back(y_0))\vee(VTe(Front(y_0))=VTe(y_0)=VTe(Back(y_0)))$ **then**

3. $L_0.delete(y_0)$;
4. **end if**
5. **if** ($VTs(y_0)\leqslant VTs(Front(y_0))\wedge(VTe(Back(y_0))\leqslant VTe(y_0))$ **then**
6. d_0=[$VTs(Front(y_0))$,$VTe(Back(y_0))$];
7. **if** $d_0\notin L_{next}$ **then**
8. $L_0.delete(y_0)$;
9. a new *PLOB* is jointed and formed by the $Front(y_0)$ and $Back(y_0)$;
10. **else if** $d_0\in L_{next}$ **then**
11. a new *PLOB* is Linear Order joined and formed by the $Front(y_0)$ and $Back(y_0)$;
12. $L_{next}.delete(y_0)$;
13. a new L_{next} is Linear Order joined and formed by the $Front(d_0)$ and $Back(d_0)$
14. **end if**
15. **end if**
16. **if** ($VTs(Front(y_0))<VTs(y_0))\wedge(VTe(Back(y_0))\leqslant VTe(y_0))$ **then**
17. t_0=[$VTs(Front(y_0))$,$VTe(Back(y_0))$];
18. **if** $t_0\notin L_0$ **then**
19. $L_{next}.delete(y_0)$;
20. a new *PLOB* is Linear Order joined and formed by the t_0 joining $Front(y_0)$ and $Back(y_0)$;
21. **else if** $t_0\in L_0$ **then**
22. $L_{next}.delete(y_0)$;
23. a new *PLOB* is Linear Order joined and formed by the y_0 joining $Front(y_0)$ and $Back(y_0)$;
24. **end if**
25. **end if**

例 5:由算法 4 可得,TPindex 在进行删除更新时,同样会引起 3 种不同情况的变化.在第 1 种情况中,在某一线序分枝汇总删除首元节点或尾元节点而不引起其他分枝的变化.如图 6(a)所示,删除节点[6,10]或删除[7,6],则将删除节点从原有的线序分枝序列中移除,再由离删除节点最近的其他节点变成新的首元节点或尾元节点;在第 2 种情况中,删除线序分枝内部中的节点,但不引起其他分枝分结构变化.如图 6(b)所示,删除节点[7,9],则在原有的分枝中,分别断开[7,10]与[7,9]、[7,9]与[7,8]的连接,再将[7,10]与[7,8]进行连接;在第 3 种情况中,删除某一分枝内部中的节点,会引起其他分枝结构的变化.如图 6(c)所示,删除[3,7]节点,则原有的[2,7]与[3,7]、[3,7]与[3,4]之间的连接会断开,[2,8]与[2,7]形成新的线序分枝,而在首元节点[3,8]则会断开与[4,8]之间的连接,转向与[3,4]建立连接,这样[3,8],[3,4]与[4,4]形成新的线序分枝.而断开连接后的[4,8]成为新的首元节点,与[4,7],[5,7],[5,5]构成新的线序分枝.

综上所述,在 PLOB 的结构基础上,对 TPindex 进行时态数据的增加和删除操作时,只需要对相应的 PLOB 进行时态拼接即可得到更新后的索引结构,而不用对全部的时态数据进行更新,极大地提高了索引的更新效率和利用效率.设时态索引更新共有 k 个线序分枝,每个分枝中共有 m 个元素,则算法 3 的平均时间复杂度均为 $kO(\log m)$,最小的时间复杂度为 $O(1)$.算法 4 的最小时间复杂度也同样为 $O(1)$.可见其具有较好的时态数据维护效率和性能.

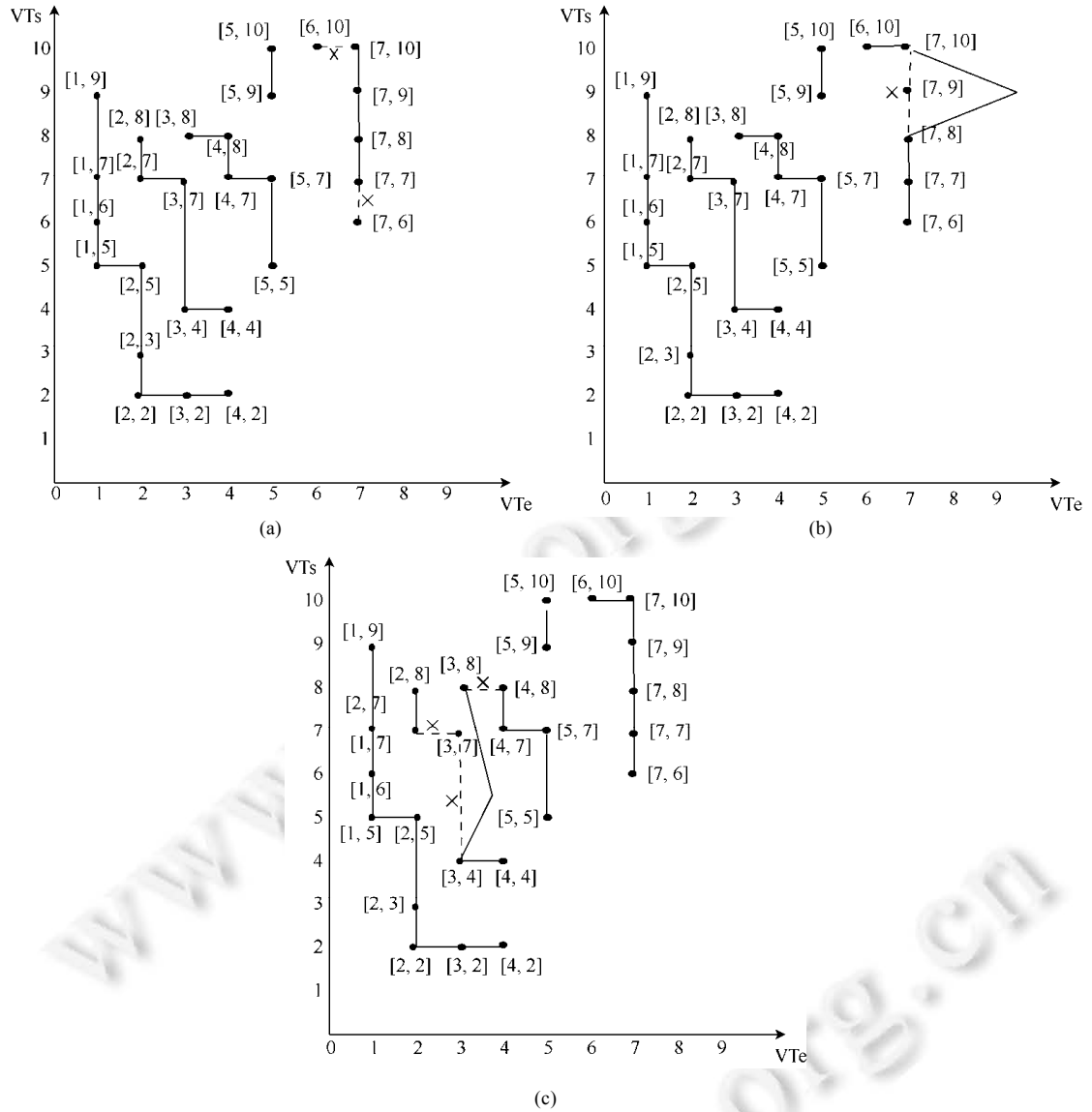


Fig.6 Delete update in TPindex

图 6 TPindex 删除更新

5 实验仿真

综上所述,TPindex 与 TDindex 都是结合时态数据的“有效时间”进行建模,是基于外存存储模式的索引结构.而 Timeline 更多地关注时态数据“事务时间”的版本管理,是一种内存列存储的索引结构.从本质上讲,TPindex 于 TDindex 更为相似,是一般的时态索引框架.TDindex 在外存存储的时态索引中具有最佳查询性能和较低的时间复杂度,所以本文选取其作为仿真比较对象.在本节通过设计多组 TPindex 与 TDindex 的对比实验,评估 TPindex 的索引构建和时态查询的性能.另一方面,因为 Mysql 与 TDindex、TPindex 同样具有磁盘外存存储模式(InnoDB 索引文件),是一种主流的数据库管理器,具有良好的索引性能.因此,本文还将 TPindex、TDindex 与 Mysql 进行实验对比,验证 TPindex 基于外存的分文件模式的有效性.

实验数据集利用时态数据生成器仿真而来,随机生成百万到千万量级的时态,其中时态属性为 1000-01-01~

3000-01-01.在进行时序分区操作时,采样阈值设定为 5%.实验的硬件环境为:Intel(R)Core(TM)i7-7700 CPU@3.60GHz,8 核 CPU,8GB 内存,硬盘 2T;软件环境为:64 位 Win10 操作系统,编程语言 Java,底层数据库 MySQL5.7.

5.1 TPindex磁盘索引构建

通过上述的理论说明,经过时序分区和线序划分的操作之后,时态数据均匀且具有一定顺序结构的分布在若干个时序分区上.实验中分别给定 1000w,2000w,3000w,4000w,5000w 的时态数据,分区个数均为 2,TPindex 磁盘索引构建的评测结果如图 7、图 8 所示.从图 7 中可以看出,每个时序分区的空间消耗基本相同,说明每个分区的时态数据基本划分均匀.在海量数据下,此时时态索引的空间开销应更着眼于索引与数据之间的空间比值.图 8 表明随着数据量的增大,无论索引大小与分区数据的空间比值和索引分枝个数与分区数据的数量比值都在逐渐减少,并且在最终趋向于稳定.这说明随着数据量的增大,TPindex 的空间消耗稳定而缓慢增长,可以预测到 TPindex 在海量数据的具有较小的空间开销,能够降低空间浪费,适合处理海量数据.

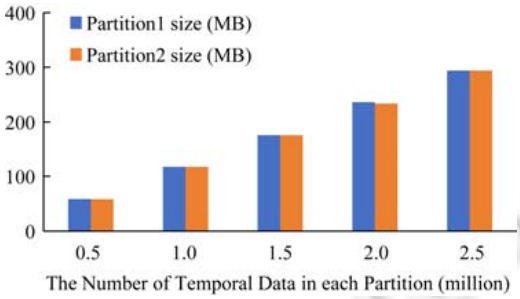


Fig.7 Number of PLOBs and space overhead in each partition

图 7 分区内的 PLOB 数量和空间开销情况

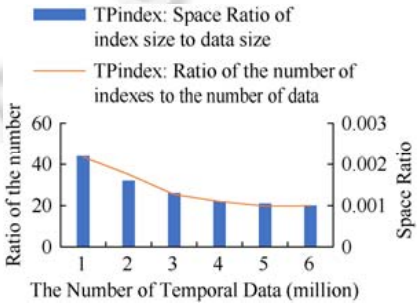


Fig.8 Space ratio of index size to data size and ratio of the number of indexes to the number of partition data

图 8 索引大小与分区数据的空间比值和索引个数与分区数据的数量比值

相对于 TDindex 将所有的时态数据进行统一划分的方式,时序分区的操作使得 TPindex 在创建每个分区的时间大大减少,从而减少了整体索引创建的时间开销.另外一方面,TDindex 是针对小规模数据设计的索引结构,不能很好地支持海量数据的索引构建.两者创建索引的时间开销与空间开销的实验结果如图 9、图 10 所示.由图 9 可以看出,当数据量超过 600W 时,TDindex 已经不能构建完整的索引,显然不适用海量历史数据的处理;另一方面,在数据量 100W~600W 时,随着数据量的增加,TDindex 索引构建的性能瓶颈越来越明显,耗费的时间开销急剧加大.而基于时序分区的 TPindex 可以很好地在海量数据下创建索引,并且 TPindex 索引的构建时间开销要远远小于 TDindex.另外一方面,图 10 中,TPindex 索引的空间消耗和分枝个数都略大于 TDindex,这是因为经过时序分区之后,TPindex 会得到至少 2 个的线序分枝,线序分枝的数量会略有增长,同时也带来空间开销上的部分增加.但是,针对时态数据而言,以“空间换时间”的数据处理模式也是可以接受的.

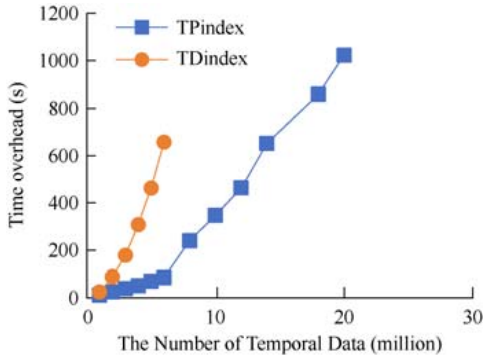


Fig.9 Comparison of time overhead of index building

图 9 索引构建的时间开销对比

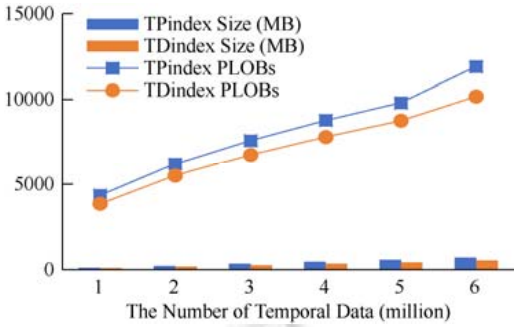


Fig.10 Space overhead and number of PLOBs of index building

图 10 构建索引的空间开销和 PLOB 数量

5.2 TPindex时态查询

综上所述,TPindex 支持时态查询操作包括快照查询(snapshot query)、跨度查询(interval query)、时态投影(projection query)和时态选择(selection query)等,现对 4 种查询分别进行实验评测:实验数据从 20w 到 240w 逐渐递增,随机数据量从 20w 到 400w 逐渐递增,依次随机生成 100 个时间版本的快照;随机生成 100 个时间跨度 T;随机做 100 次投影操作;随机生成 100 次查询窗口 Q,分别进行 4 次实验,实验结果如图 11 所示.可以看出,随着数据量的增加,快照查询和时态投影操作的时间开销曲线近似成直线,说明快照查询和时态投影运算的查询性能都较稳定;但跨度查询和时态选择操作的时间开销出现很大的波动,这与查询区间跨度的不确定性有关.因此,接下来的实验中将探究不同时间跨度对时态查询的性能影响.

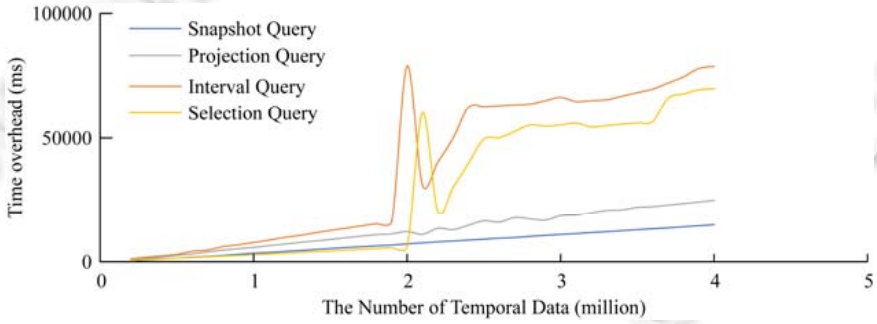


Fig.11 Comparison of the performances for TPindex on different temporal query operations

图 11 TPindex 时态查询的性能比较

不同的 Q 跨度对于时态查询性能的影响不同,本组实验将 TPindex 与 TDindex 索引通过对照进行说明.在 TDindex 索引的时态查询中,主要选取了 TDindex 的遍历查询(TDindex traveral)与内部二分查找(TDindex binary search)的查询算法.实验数据量为 600w,随机生成跨度分别为 10%T,20%T,30%T,40%T,50%T 和 60%T 的 100 组查询窗口,计算最终的平均时间开销.实验结果如图 12 所示.结果表明,随着 Q 跨度的增大,TDindex 和 TPindex 的时间开销都在逐渐减少,在 60%T 时达到最佳结果,并最后都趋向于稳定.但是无论在哪一个 Q 跨度下的查询,TPindex 的性能都优于 TDindex.

TPindex 特有的上层“分区层”可以快速地定位到目标数据所在的时序分区,再由二分优化查找算法在分区内部的顺序结构进行数据的筛选.通过“数据定位”与“过滤”,相对于 TDindex 一层的数据“过滤”,TPindex 可以实现更加高效的查询性能.本组实验中选取上述实验中最佳的时间跨度 60%T,数据量由 100w 到 600w 依次递增,

最后计算100次随机查询的平均时间开销,实验结果如图13所示.可以看出,TPindex的性能更加优越于TDindex.另一方面,查询区间的跨度越大查询效率越高;而对于查询跨度较小的查询窗口,则使用查询优化算法后查询效果会更好.通过上述实验图12和图13的实验结果,再次验证了图11中时态选择和时间时态跨度查询性能出现波动的原因.

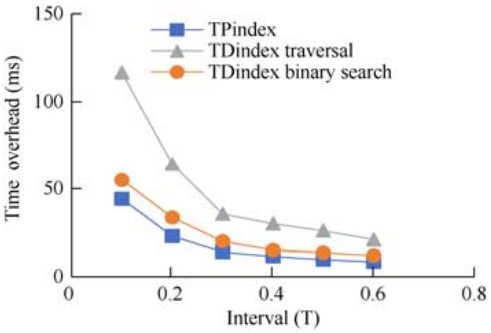


Fig.12 Comparison of time overhead under different query intervals

图12 不同查询跨度下的时间开销对比

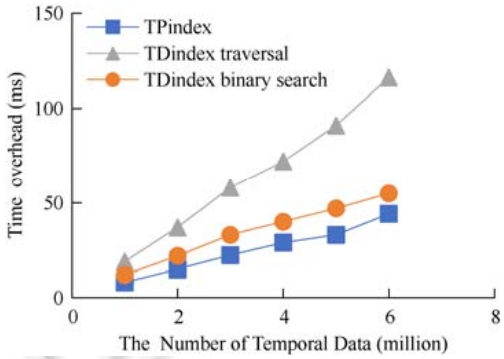


Fig.13 Comparison of time overhead under different query strategy

图13 不同查询策略下的时间开销对比

同时,TPindex的"分区层"还可以结合并行化技术,更好地利用目前计算机体系结构的优势.下面通过实验评估并行化优化方案对于TPindex的性能影响.第1组实验,数据量由100w到500w逐渐递增,且分别进行时序分区(分区个数统一为2),依次选择1、2、4、8、16线程进行100次随机时态查询(Q跨度统一为60%T),最后计算平均时间开销.第2组实验数据量固定为500w,Q跨度从10%T到60%T*依次*递增,随机生成100次Q,线程个数统一为8.实验结果如图14、图15所示.结果表明并行化的优化可以带来较好的性能提升;另外一方面,8线程查询性能和16查询性能几乎重合,因此在线程为8时,TPindex也能达到较好的效果.图15的实验结果表明,在线程是8的情况下,随着查询窗口跨度的不断增大,TPindex查询性能始终优于TDindex.

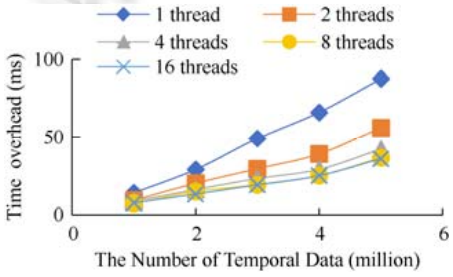


Fig.14 Comparison of time overhead under different threads

图14 不同线程下的时间开销对比

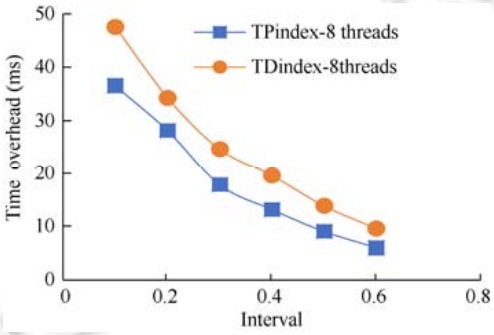


Fig.15 When the number of threads is 8, comparison of time overhead under different query intervals

图15 线程数为8时,不同查询跨度下的时间开销对比

为了评估TPindex基于外存的分文件模式的有效性,本文将TPindex、TDindex与Mysql的查询性能进行实验对比.第1组实验将Q窗口跨度设置为10%T,随机生成100组Q窗口,数据量由100w到600w逐渐递增;第2组实验将数据量设为600w,分别随机生成100组Q窗口(跨度分别为10%T,20%T,30%T,40%T,50%T,60%T),实验结果如图16、图17所示.结果表明,随着时态数据量的增加,TPindex的磁盘索引性能明显优于其他两种方法的磁盘索引,说明TPindex相对于Mysql和TDindex索引能够有效地降低I/O,验证了TPindex基于外存的分文件

模式的有效性.

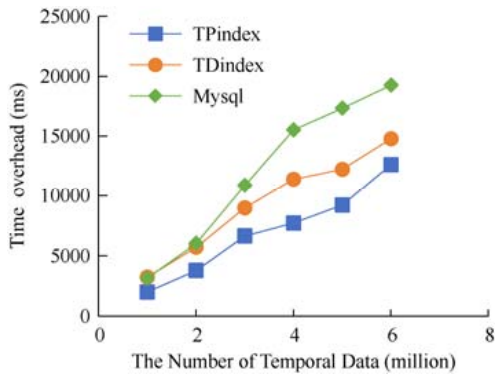


Fig.16 Comparison of the time overhead of 3 “disk-based” models

图 16 3 种基于“外存”模型的时间开销比较

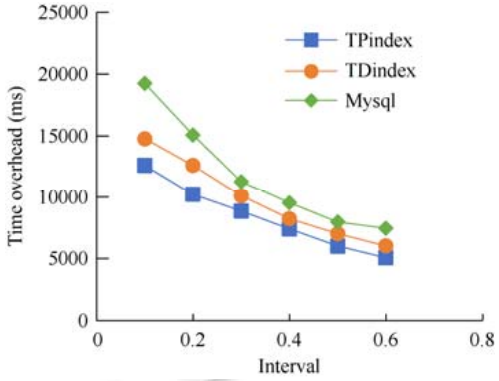


Fig.17 Comparison of the time overhead of 3 “disk-based” models under different query intervals

图 17 不同查询跨度下 3 种“外存”模型的时间开销比较

6 总 结

随着大数据时代的到来,时态数据管理技术受到越来越多的关注和重视,时态索引技术是高效存储和管理时态数据的一项关键技术.相对于传统的时态索引技术,针对海量数据、并行化优化的时态索引的研究工作较少.如何实现海量数据下对“数据本体”和“时态信息”的有机整合和高效管理,是一项具有挑战性的研究课题.首先,结合时态数据的“有效时间”,本文提出了一种基于时序分区,适用于海量数据的时态索引技术 TPindex. TPindex 特有的上层结构(分区层)可以很好地结合并行化技术,过滤掉大量的“无关”数据,在多个“块”中快速地找到目标数据所在时序分区.其次,在对应分区内部,具有顺序结构的索引层可以通过优化的查询算法高效地筛选数据.同时,本文还提出了一种基于外存的分文件模式,将时序分区内部的线序索引磁盘化.这种模式同样可以通过分区层快速的定位并将目标数据读入内存中,再由查询算法在内部具有顺序结构的索引层中进行高效的数据筛选,在海量数据的情形下,可以增加查询数据的命中率,减少 I/O.再次,本文从 TPindex 的索引结构出发,讨论了基于 PLOB 的增量式时态数据更新机制,提出了可以实现大规模历史数据动态管理的方法.最后,通过实验仿真说明了 TPindex 的性能更优越于 TDindex,表明其具有一定的有效性和可行性.由于目前的研究还局限于本地计算机系统上,在未来的工作中,将再继续拓展到分布式应用等领域中.

References:

[1] Ye XP. Model and algebra of object-relation bitemporal data based on temporal variables. Journal of Computer Research and Development, 2007,44(11):1971−1979 (in Chinese with English abstract).

[2] Kanhabua N, Anand A. Temporal information retrieval. In: Proc. of the Int’l ACM SIGIR Conf. on Research and Development in Information Retrieval. ACM, 2016. 1235−1238. [doi: 10.1145/2911451.2914805]

[3] Ye XP, Tang Y, Chen KY. Temporal data indexing technology based on valid time. Journal of Computer Research and Development, 2006,43(Suppl.):517−520 (in Chinese with English abstract).

[4] Nascimento MA, Dunham MH. Indexing valid time databases via B^+-trees. IEEE Trans. on Knowledge and Data Engineering, 1999, 11(6):929−947. [doi: 10.1109/69.824609]

[5] Bliujute R, Jensen CS, Sltenis S, Slivinskas G. Light-weight indexing of general bitemporal data. In: Proc. of the Int’l Conf. on Scientific and Statistical Database Management. 2000. 125−138. [doi: 10.1109/SSDM.2000.869783]

[6] Bliujute R, Jensen CS, Saltenis S, Slivinskas G. R-tree based indexing of now-relative bitemporal data. In: Proc. of the Int’l Conf. on Very Large Data Bases. 1998. 345−356.

[7] 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. 1990. 1–12.

[8] Becker B, Gschwind S, Ohler T, Seeger B, Widmayer P. An asymptotically optimal multiversion B-tree. The VLDB Journal—The Int'l Journal on Very Large Data Bases, 1996,5(4):264–275.[doi: 10.1007/s007780050028]

[9] Bozkaya T, Ozsoyoglu M. Indexing transaction time databases. Information Sciences, 1998,112(1):85−123. [doi:10.1016/S0020-0255(98)10024-5]

[10] Lomet D, Hong MS, Nehme R, Zhang R. Transaction time indexing with version, compression. VLDB Endowment, 2008,1(1): 870–881. [doi:10.14778/1453856.1453951]

[11] Tao Y, Papadias D. MV3R-Tree: A spatio-temporal access method for timestamp and interval queries. In: Proc. of the 27th Int'l Conf. on Very Large Data Bases. 2001. 431–440.

[12] Abdelguerfi M, Givaudan J, Shaw K, Ladner R. The 2-3TR-tree, a trajectory-oriented index structure for fully evolving valid-time spatio-temporal datasets. In: Proc. of the 10th ACM Int'l Symp. on Advances in Geographic Information Systems. 2002. 29–34. [doi: 10.1145/585147.585155]

[13] Procopiuc CM, Agarwal PK, Har-Peled S. Star-tree: An efficient self-adjusting index for moving objects. In: Proc. of the Workshop on Algorithm Engineering and Experimentation. 2002. 178–193.

[14] Rizzolo F, Vaisman AA. Temporal XML: Modeling, indexing, and query processing. The VLDB Journal—The Int'l Journal on Very Large Data Bases, 2008,17(5):1179–1212. [doi: 10.1007/s00778-007-0058-x]

[15] Guo H, Ye XP, Tang Y, Chen LW. Temporal XML index based on temporal encoding and linear order partition. Ruan Jian Xue Bao/Journal of Software, 2012,23(8):2042–2057 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4161.htm [doi: 10.3724/SP.J.1001.2012.04161]

[16] Ye XP, Tang Y, Lin YC, Chen ZY, Zhang ZB, Chen RX. Study and implementation of temporal index Tdindex. Scientia Sinica Information, 2015,45(8):1025–1045 (in Chinese with English abstract). [doi: 10.1360/N112013-00230]

[17] Kaufmann M, Manjili AA, Vagenas P, Fischer PM, Kossmann D, Farber F, May N. Timeline index: A unified data structure for processing queries on temporal data in SAP HANA. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. 2013. 1173–1184. [doi: 10.1145/2463676.2465293]

[18] Ma YZ, Meng XF. Research on indexing for cloud data management. Ruan Jian Xue Bao/Journal of Software, 2015,26(1):145–166 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4688.htm [doi: 10.13328/j.cnki.jos.004688]

[19] Allen JF. Maintaining knowledge about temporal intervals. Communications of the ACM, 1983,26(11):832–843. [doi: 10.1145/182.358434]

[20] Tong Y, She J, Ding B, Wang L, Chen L. Online mobile micro-task allocation in spatial crowdsourcing. In: Proc. of the IEEE International Conf. on Data Engineering. 2016. 49–60. [doi: 10.1109/ICDE.2016.7498228]

[21] Tong Y, She J, Ding B, Chen L, Wo T, Xu K. Online minimum matching in real-time spatial data: Experiments and analysis. VLDB Endowment, 2016,9(12):1053–1064. [doi: 10.14778/2994509.2994523]

[22] Song T, Tong Y, Wang L, She J, Yao B, Chen L, Xu K. Trichromatic online matching in real-time spatial crowdsourcing. In: Proc. of the IEEE Int'l Conf. on Data Engineering. 2017. 1009–1020. [doi: 10.1109/ICDE.2017.147]

[23] Tong YX, Yuan Y, Cheng YR, Chen L, Wang GR. Survey on spatiotemporal crowdsourced data management techniques. Ruan Jian Xue Bao/Journal of Software, 2017,28(1):35–58 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5140.htm [doi: 10.13328/j.cnki.jos.005140]

[24] TimeDB—A temporal relational DBMS. 2018. http://www.timeconsult.com/Software/Software.html

[25] Tang Y, Liu H, Guo H, Ye XP. TempDB: Temporal data manage system. Journal of Computer Research and Development, 2010, 47(Suppl.):442–445 (in Chinese with English abstract).

[26] Bohlen MH, Jensen CS, Snodgrass RT. Evaluating and enhancing the completeness of TSQL2. Technical Report, TR 95-5, Computer Science Department, University of Arizona, 1995.

[27] Färber F, May N, Lehner W, Grobe P, Muller I, Rauhe H, Dees J. The SAP HANA database—An architecture overview. IEEE Data Engineering Bulletin, 2012,35(1):28–33.

[28] Kaufmann M, Fischer PM, May N, Ge C, Goel AK, Kossmann D. Bi-temporal timeline index: A data structure for processing queries on bi-temporal data. In: Proc. of the 31st Int'l Conf. on Data Engineering. 2015. 471–482. [doi: 10.1109/ICDE.2015.7113307]

[29] Ghoshdastidar D, Dukkipati A. Uniform hypergraph partitioning: Provable tensor methods and sampling techniques. The Journal of Machine Learning Research, 2017,18(1):1638–1678.

[30] Cary A, Yesha Y, Adjouadi M, Rishe N. Leveraging cloud computing in geodatabase management. In: Proc. of the IEEE Int'l Conf. on Granular Computing. 2010. 73–78. [doi: 10.1109/GrC.2010.163]

[31] Zhong Y, Han J, Zhang T, Li Z, Fang J, Chen G. Towards parallel spatial query processing for big spatial data. In: Proc. of the IEEE 26th Parallel and Distributed Processing Symp. Workshops & PhD Forum (IPDPSW). 2012. 2085–2094. [doi: 10.1109/IPDPSW.2012.245]

附中文参考文献:

[1] 叶小平.基于时态变量对象关系模型及代数运算.计算机研究与发展,2007,44(11):1971–1979.

[3] 叶小平,汤庸,陈铠原.基于有效时间的时态索引技术.计算机研究与发展,2006,43(Suppl.):517–520.

[15] 郭欢,叶小平,汤庸,陈罗武.基于时态编码和线序划分的时态 XML 索引.软件学报,2012,23(8):2042–2057. http://www.jos.org.cn/1000-9825/4161.htm [doi: 10.3724/SP.J.1001.2012.04161]

[16] 叶小平,汤庸,林衍崇,陈钊滢,张智博,陈瑞鑫.时态数据索引 TDindex 研究与应用.中国科学:信息科学,2015,45(8):1025–1045.

[18] 马友忠,孟小峰.云数据管理索引技术研究.软件学报,2015,26(1):145−166. http://www.jos.org.cn/1000-9825/4688.htm [doi: 10.13328/j.cnki.jos.004688]

[23] 童咏昕,袁野,成雨蓉,陈雷,王国仁.时空众包数据管理技术研究综述.软件学报,2017,28(1):35–58. http://www.jos.org.cn/1000-9825/5140.htm [doi: 10.13328/j.cnki.jos.005140]

[25] 汤庸,刘海,郭欢,叶小平.TempDB:时态数据管理系统.计算机研究与发展,2010,47(Suppl.):442–445.

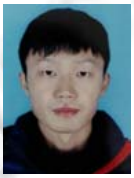
杨佐希(1995－),男,广东人,硕士,主要研究领域为时空数据库,社会与协同计算,数据挖掘.

潘明明(1994－),女,硕士,主要研究领域为时空数据库,XML 数据库.

汤娜(1975－),女,博士,副教授,CCF 专业会员,主要研究领域为时空数据库.

叶小平(1955－),男,博士,教授,博士生导师,主要研究领域为时空数据库,XML 数据库技术.

汤庸(1964－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为时空数据库,社会与协同计算,数据挖掘.



李丁丁(1982－),男,博士,副教授,CCF 专业会员,主要研究领域为计算机系统结构.