记为[*Now*];在等价类[*Now*]中,所有节点描述的列表按照节点的有效时间的开始时间的偏序关系排列.

定理 5([***Now***]**等价类偏序结构的时态特性**). 对于任意的两个节点 u 和 v,两个节点都属于[*Now*]等价类,则有如下属性:

(1) 如果 $VT(u)\subseteq VT(v)$,则节点 v 在偏序结构中的位置先于节点 u;

(2) 对于任意的一个有效时间区间 *VI*,在等价类[*Now*]中,如果 $Vspan(VI)\neq Now$,则在等价类[*Now*]中不存在一个节点 u,使得如下的时态关系成立:$Equal(VI,VT(u))$=true,$Contains(VI,VT(u))$=true,$After(VI,VT(u))$=true,$Overlapped\text{-}by(VI,VT(u))$=true,$Met\text{-}by(VI,VT(u))$=true,$Started\_by(VI,VT(u))$=true,$Finishes(VI,VT(u))$=true,$Finished\_by\ (VI,VT(u))$=true.

基于定义 6 的结构,时态过滤算法中存在 3 种情况会涉及到 *Now* 变量:前两种情况是给定的查询条件的时间区间的结束时间为 *Now*,在非[*Now*]等价类和[*Now*]等价类中分别查找满足 $\theta(Qp,VT(v))$=true 的节点;第 3 种情况是给定的查询条件的时间区间的结束时间不为 *Now*,在[*Now*]等价类中查找分别满足 $\theta(Qp,VT(v))$=true 的节点.以下给出 3 种情况下基于 *Now* 变量的时态过滤算法.

算法 6. 基于 *Now* 的时态过滤算法 *NowTempNodeFilter*.

功能:对于给定的一个时间区间 *Qp*,在标签 *TagName* 对应的等价类划分{[*Flag*],*SPVList*}中找到所有节点 v,节点的有效区间 $VT(v)$应满足 $\theta(Qp,VT(v))$=true

输入:某个语义标签 *TagName* 对应的等价类划分{[*Flag*],*SPVList*},给定的时间区间 *Qp*,时态区间关系 θ,此算法 θ为 during;

输出:{*NodeList*}.

对于 *Flag*=*Now* 的时态等价类

{根据定理 5 属性(1),在 *SPVList* 列表中找到 $Vstart(v)$值最大的节点 v,v 满足 $Vstart(v)<=Vstart(Qp)$;

v 以及 v 之前的节点信息加到结果集{*NodeList*}中;}

对于 *Flag*<>*Now* 的时态等价类

根据定理 5 属性(2),If $Vend(Qp)$=*Now* then 找不到任何一个节点满足条件

算法 6 是第 2.1.1 节中算法 2 的特例补充,算法 2 的功能是进行时态关系过滤和匹配运算,但不涉及到 *now* 变量的任何处理,而算法 6 则针对所有涉及到 *now* 变量的情况进行时态关系运算的补充处理.

例 4(包含时态变量的时态关系运算实例):语义标签为 *A* 的基于偏序的时态等价类划分为{([1],2004^{75},2007^{99}),([2],2000^{15},2001^{87}),([5],2000^{115},2001^{78},2002^{178},2003^{194},2005^{233},2008^{245}),([*Now*],2000^{5},2001^{50},2005^{200},2007^{66})},请找出在[2002,2005]内有效的节点.

时间区间[2002,2005]的跨度为 3,目前共有 4 个等价类,找到大于等于 3 的最小的等价类[5](等价类[1]、等价类[2]中不可能有满足条件的节点),等价类[5]的节点跨度大于 3,首先可以用二分法/B 树快速找到小于或等于 2002 的最大 *start* 值的节点 2002^{178},然后找到大于 $Vend(Qp)-Flag$=2005−5−1=1999 的最小 *start* 值的节点 2000^{115},这两个节点之间的节点均满足条件.即{2000^{115},2001^{78},2002^{178}}.

在等价类[*Now*]中,找到小于或等于 2002 的最大 *start* 值的节点 2001^{50},该节点及其之前的节点均满足条件,即{2000^{5},2001^{50}}.

所以,满足时态过滤的节点为

$$\{2000^{115},2001^{78},2002^{178}\}\cup\{2000^{5},2001^{50}\}=\{2000^{115},2001^{78},2002^{178},2000^{5},2001^{50}\}.$$

在快照查询中有一种特殊的快照查询,即当前数据库快照,该查询会在整个 XML 文档记录的数据历史中获得所有对象当前的数据.即,对时态 XML 文档当前的“切片”.这是一类非常普遍的查询.由于所有结束时间为 *Now* 的节点都在同一个等价类中,所以当前快照的查询变得十分的简单,只需要将[*Now*]等价类中所有节点按照前缀编码排序,即可得到按深度遍历的当前快照的切片.

4 仿真与评测

为检验 TempPartialIndex 的基本性能,本文设计了相应的仿真实验.实验硬件环境为 Inter(R) Core(TM)i5-3230M CPU,主频为 2.60GHz,主存容量为 4GB;外存容量为 500GB.实验在 Windows 7 系统平台下进行.

因为索引 TempSumIndex[14]具有最好的时态 XML 查询效率,即,具有最低的时间复杂度,所以本文选择其作为仿真比较对象.

仿真使用的时态 XML 数据节点数从 10 万~45 万个,每次递增 5 万个节点,内部节点平均时间跨度为 500 个单位,叶节点平均时间跨度为 200 个单位,数据由自定义的程序随机生成,但数据均满足时态约束.

4.1 索引构建

由于 TempSumIndex 索引构建时态数据结构的效率的时间复杂度为 $O(n^2)$[14],而且 TempPartialIndex 索引构建时态数据结构的时间复杂度 $O(n\log n)$,而且在构建索引模型时,由于 TempSumIndex 索引是根据 1-index 路径摘要来进行分类的,所以比 TempPartialIndex 按照语义标签分类的搜索过程要复杂.

索引构建性能评测结果如图 4 所示.

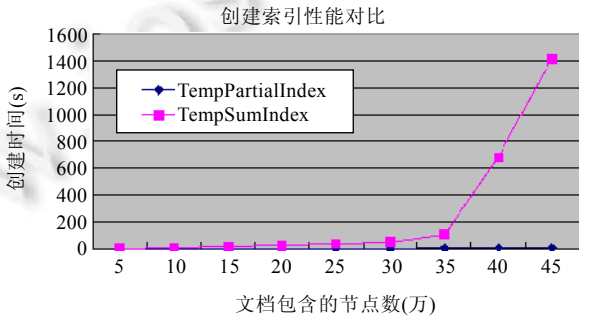

Fig.4 Performance evaluation of building index

图 4 构建索引性能评测

由图 4 也可以看出,语义协同时态 XML 索引 TempPartialIndex 的构建时间开销要远远小于 TempSumIndex 索引构建时间开销,这进一步验证了 TempPartialIndex 索引在进行索引构建时具有比较好的时间复杂度,体现了 TempPartialIndex 的优越性.

4.2 查询功能

4.2.1 TXPath 查询

实验采用的时态约束查询分为:

(1) 无时态约束类型:$Q_1(A//B)$;

(2) 时态约束类型:$Q_{2\text{-}1}(//A[VT(A)])$,$Q_{2\text{-}2}(A//B[VT(B)])$,$Q_{2\text{-}3}(A[VT(A)]//B)$,$Q_{2\text{-}4}(//A[VT(A)]//B[VT(B)])$.

$Q_1(A//B)$是不涉及时态的相对路径查询,其查询处理由语义查询器和结构查询器协同完成.在经过语义过滤后,如果子孙层有 Q 个节点,祖先层有 W 个节点,则祖先/子孙匹配效率会非常慢,本文采用了前缀编码结合 hash 映射的方法,取代了对“值”的缓慢搜索.通过时间复杂度为 $O(1)$,就可以将某个查询值对应的节点对象找出.由于子孙层有 Q 个节点,通过子孙节点的前缀编码得到祖先节点的前缀编码,再通过 hash 映射判断祖先层中是否存在着此前缀编码对应的节点,所以算法的效率为 $O(Qh)$;TempSumIndex 采用前序编码的方式判断祖先/子孙关系,匹配算法的效率为 $O(Q\log Q)$.因而,TempPartialIndex 在处理 $Q_1(A//B)$查询时具有优越性.性能对比如图 5 所示.

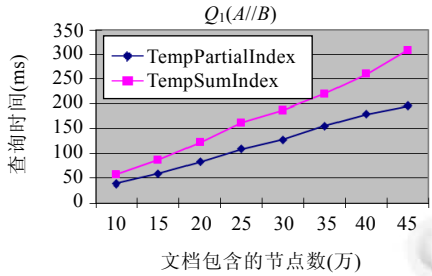

Fig.5 Performance evaluation of query $Q_1(A//B)$

图 5 $Q_1(A//B)$查询性能评测

$Q_{2\text{-}1}(//A[VT(A)])$,$Q_{2\text{-}2}(A//B[VT(B)])$,$Q_{2\text{-}3}(A[VT]//B)$,$Q_{2\text{-}4}(//A[VT(A)]//B[VT(B)])$均为带时态约束的查询类型,这类具有时态约束的查询整个查询过程分为 3 个部分:语义查询处理、时态查询处理、结构查询处理.通过第 1 类查询的比较,TempPartialIndex 的语义部分和结构部分的处理都具有一定的优势,在时态关系运算方面,TempSumIndex 时态运算的复杂度主要是 $O(mb\log n)$,m 为需要处理的时态摘要节点的个数,b 为每一个时态摘要节点的线序划分包含的线序分枝的个数,n 为其时态基本数据结构 LOB 中的元素个数;而 TempPartialIndex 时态运算的复杂度主要是 $O(kn\log m)$,k 是语义节点的个数,n 是需要监测的等价类数,m 是等价类中节点的个数.由于 TempSumIndex 的线序划分需要时间区间之间满足嵌套的关系,及其进行划分的条件要更严苛,所以线序划分的个数会更多.而且 TempPartialIndex 的时态运算对于跨度小于自己的等价类是不需要检查的,而 TempSumIndex 需要检查每一个线序分支.对于查询时间区间的结束时间是 *Now* 的时态运算,TempPartialIndex 只需要检查一个 *Flag=Now* 的时间区间,所以运算的效率为 $O(\log m)$,而基于 *Now* 的查询是一种非常普遍的时态查询.

第 2 类的 4 种时态运算的性能比较如图 6~图 9 所示.

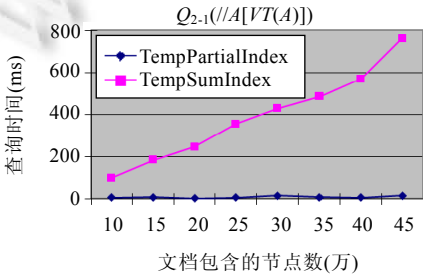

Fig.6 Performance evaluation of query $Q_{2\text{-}1}$

图 6 $Q_{2\text{-}1}$ 查询性能评测

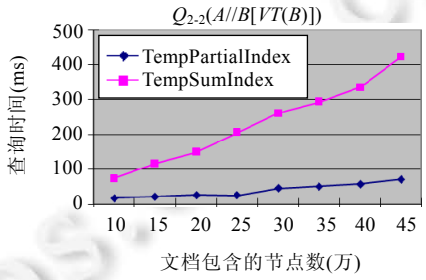

Fig.7 Performance evaluation of query $Q_{2\text{-}2}$

图 7 $Q_{2\text{-}2}$ 查询性能评测

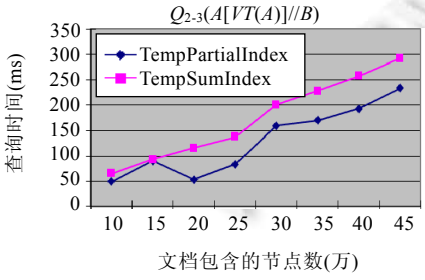

Fig.8 Performance evaluation of query $Q_{2\text{-}3}$

图 8 $Q_{2\text{-}3}$ 查询性能评测

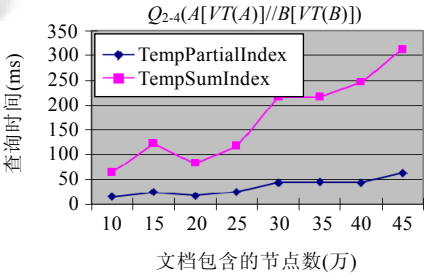

Fig.9 Performance evaluation of query $Q_{2\text{-}4}$

图 9 $Q_{2\text{-}4}$ 查询性能评测

4.2.2 快照查询

时态 XML 数据检索的另一种典型类型就是快照查询,也是时态数据所特有的查询方式.

对于时间点 VT_0=[VT_0,VT_0]的快照查询可以分为两种类型:$Q_{3\text{-}1}(A[VT_0(A)]//B[VT_0(B)])$和 $Q_{3\text{-}2}(VT_0)$两种类型.

对于 $Q_{3\text{-}1}(A[VT_0]//B[VT_0])$快照查询可以归结为是 TXPath 中 $Q_{2\text{-}4}(A[VT(A)]//B[VT(B)])$的一种特殊情形,一方面,$A$ 和 B 的时态约束相同;另一面,时态运算的关系不再是 during,而是 equal.由于 TempSumIndex 需要检查每一个线序分支,因而算法的复杂度为 $O(mb\log n)$;但索引 TempPartialIndex 只需要检查时间区间跨度等于 $VT(Qp)$跨度的时态等价类,满足条件的等价类最多只有 1 个,在此等价类中,只需要查询 VT_0 所在的位置,所以运算的效率为 $O(\log m)$,因而 TempPartialIndex 的效率较优.两种算法的性能对比如图 10 所示.$Q_{3\text{-}2}(VT_0)$最典型的查询就是当前快照查询,即,基于 *now* 的快照查询,该查询的性能分析详见第 4.2.4 节.

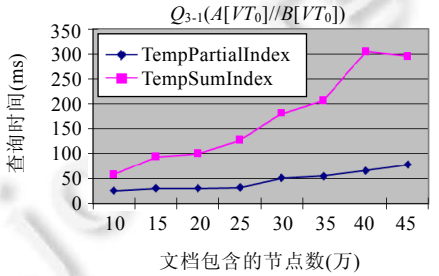

Fig.10 Performance evaluation of query $Q_{3\text{-}1}(A[VT_0]//B[VT_0])$

图 10 $Q_{3\text{-}1}(A[VT_0]//B[VT_0])$查询性能评测

4.2.3 时态 XML 更新

插入的对比实验有两组,第 1 组对比是在 50000~450000 节点的数据中插入 50 个节点(如图 11 所示),第 2 组对比实验实在 10 万节点的 XML 文档中分别插入不同的节点个数,节点的个数按照 10 递增(如图 12 所示).由图 11、图 12 可以看出,TempPartialIndex 要优于 TempSumIndex.由于 TempSumIndex 索引在插入和修改的过程都可能会引起其时态数据结构线序划分中相关线序分枝的分裂和新线序分枝的建立,而 TempPartialIndex 只需在其应该所在的等价类中插入即可,不会导致任何等价类的分裂和重建,所以其时间复杂度等效于时态关系运算的效率.而节点有效时间的修改首先要找到节点原来有效时间所在的等价类,进行删除,然后在新的等价类中插入.其时间复杂度为两倍时态关系运算的效率.

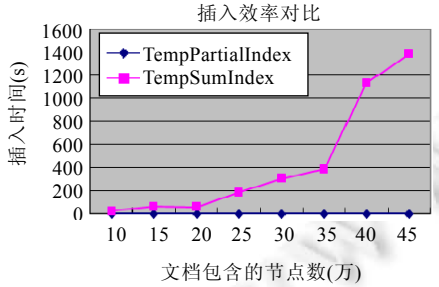

Fig.11 Performance evaluation of inserting (1)

图 11 插入性能评测(1)

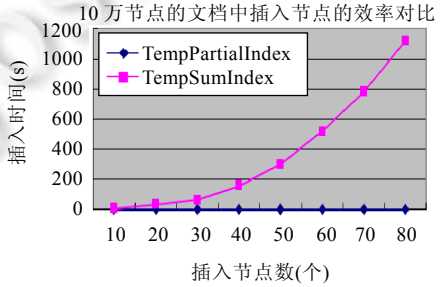

Fig.12 Performance evaluation of inserting (2)

图 12 插入性能评测(2)

4.2.4 基于 *Now* 变量的查询

由于索引 TempPartialIndex 中所有结束时间为 *Now* 变量的节点都在一个时态等价类中,这使得 *Now* 变量的时态过滤只涉及到一个等价类,等价类中的偏序结构使得只需要找到此等价类中查询范围的起点,所以运算的效率为 $O(\log m)$(m 为文档中结束时间为 *Now* 的节点总数).而 TempSumIndex 索引需要检查每一个线序分支

中符合时态约束的节点,因而算法的复杂度为 $O(mb\log n)$(m 为文档中结束时间为 *Now* 的节点总数,n 为 TempSumIndex 索引中线序分支的总数),所以 TempPartialIndex 的效率较优,性能对比如图 13 所示.而当前快照查询也是类似,对于索引 TempPartialIndex 是将时态等价类[*Now*]中所有节点取出,而 TempSumIndex 索引需要检查每一个线序分支.性能对比如图 14 所示.

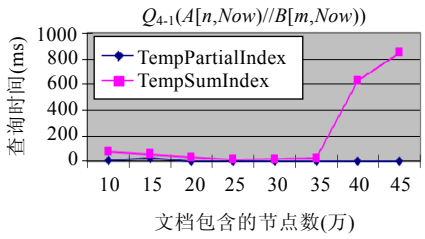

Fig.13 Performance comparison of query $Q_{4\text{-}1}$

图 13 $Q_{4\text{-}1}$ 查询性能对比

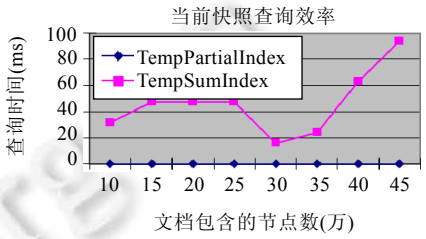

Fig.14 Query time comparison of current snapshot

图 14 当前快照的查询时间对比

5 结 语

时态 XML 的索引研究是提高时态 XML 数据查询效率的关键技术.时态 XML 索引通常将时态约束作为一种特殊的值约束处理,查询过程中,时态约束的处理往往放在最后.事实上,由于时态 XML 数据中每个节点都具有时间标签,数据时间信息的过滤和匹配在整个数据过滤过程中占有比较大的比重,先行进行时间信息过滤可以大大减少获取最终结果过程中的工作量,这类似于关系数据库中先选择投影再连接的查询优化思路.本文借鉴此思想,将基于偏序的时态等价类数据结构整合到非时态的 XML 索引的语义层之中,提出了一种新颖的时态 XML 索引 TempPartialIndex.数据处理过程采用了不同于现有相关工作的方式,即:先处理语义和时态约束,再进行结构的连接.仿真评估表明,TempPartialIndex 索引具有可行性与有效性.

References:

[1] Nørvåg K, Nybo AO. DyST: Dynamic and scalable temporal text indexing. In: Norvag K, ed. Proc. of the Temporal Representation and Reasoning. Budapest: IEEE, 2006. 204−211. [doi: 10.1109/TIME.2006.12]

[2] Bin-Thalab R, El-Tazi N, El-Sharkawi ME. TX-Kw: An effective temporal XML keyword search. Int'l Journal of Advanced Computer Science and Applications. 2013,4(6):217−226.

[3] Faisal S, Sarwar M. Temporal and multi-versioned XML documents: A survey. Information Processing and Management, 2014, 50(1):113−131. [doi: 10.1016/j.ipm.2013.08.003]

[4] Clifford J, Croker A, Grandi F, Tuzhilin A. On temporal grouping. In: Clifford J, ed. Proc. of the Recent Advances in Temporal Databases. Zurich: Springer London, 1995. 194−213. [doi: 10.1007/978-1-4471-3033-8_11]

[5] Kepser S. A simple proof for the turing-completeness of XSLT and XQuery. In: Proc. of the Extreme Markup Languages. 2004.

[6] Fernández M, Siméon J. Growing XQuery. In: Cardelli L, ed. Proc. of the ECOOP 2003—Object-Oriented Programming. Darmstadt: ECOOP, 2003. 405−430. [doi: 10.1007/978-3-540-45070-2_18]

[7] Nørvåg K. The design, implementation, and performance of the v2 temporal document database system. Information and Software Technology, 2004,46(9):557−574. [doi: 10.1016/j.infsof.2003.10.006]

[8] Mandreoli F, Martoglia R, Ronchetti E. Supporting temporal slicing in XML databases. In: Ioannidis Y, Scholl MH, eds. Proc. of the Advances in Database Technology (EDBT 2006). Berlin, Heidelberg: Springer-Verlag, 2006. 295−312. [doi: 10.1007/11687238_20]

[9] Rizzolo F, Vaisman A. Temporal XML: Modeling, indexing, and query processing. The VLDB Journal, 2008,17(5):1179−1212. [doi: 10.1007/s00778-007-0058-x]

[10] Amagasa T, Yoshikawa M, Uemura S. A data model for temporal XML documents. In: Ibrahim M, Küng J, Revell N, ed. Proc. of the Database and Expert Systems Applications. London: Springer-Verlag, 2000. 334−344. [doi: 10.1007/3-540-44469-6_31]

[11] Zheng TK, Wang XJ, Zhou YC. Indexing temporal XML using FIX. In: Liu WY, Luo XF, Wang FL, Lei JS, eds. Proc. of the Web Information Systems and Mining. Shanghai: Springer-Verlag, 2009. 224–231. [doi: 10.1007/978-3-642-05250-7_24]

[12] Zhao L, Wang XJ. Indexing temporal XML using UB-tree. Science of Computer, 2008,35(3):71–72 (in Chinese with English abstract).

[13] Mendelzon AO, Rizzolo F, Vaisman A. Indexing temporal XML documents. In: Nascimento MA, Özsu MT, Kossmann D, Miller RJ, Blakeley JA, Schiefer KB, ed. Proc. of the 30th Int'l Conf. on Very Large Data Bases—Vol.30. Toronto: VLDB Endowment, 2004. 216–227. [19] Dewey decimal classification. http://www.oclc.org/dewey/

[14] 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 Softwarer, 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]

[15] 叶小平,汤庸,郭欢,陈罗武,朱君,陈铠原.时态索引技术研究及其应用.中国科学(F 辑:信息科学),2009,19(12):1258–1270.

[16] Dyreson CE, Jensen CS, Snodgrass RT. Now in temporal databases. In: Liu L, Özsu MT, ed. Proc. of the Encyclopedia of Database Systems. Springer US, 2009. 1920–1924. [doi: 10.1007/978-0-387-39940-9_248]

[17] Ye XP, Tang Y. Semantics on "Now" and calculus on temporal relations. Ruan Jian Xue Bao/Journal of Softwarer, 2005,16(5): 838–845 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/16/838.htm

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

[19] Dewey decimal classification. http://www.oclc.org/dewey/

附中文参考文献:

[12] 赵林,王新军.使用 UB-tree 索引时态 XML.计算机科学,2008,35(3):71–72.

[14] 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]

[17] 叶小平,汤庸.时态变量"Now"语义及相应时态关系运算.软件学报,2005,16(5):838–845. http://www.jos.org.cn/1000-9825/16/838.htm

汤娜(1975－),女,浙江萧山人,博士,副教授,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,CCF 高级会员,主要研究领域为时空数据库.

汤庸(1964－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为协同工作与数据库.

彭鹏(1987－),男,软件设计师,主要研究领域为数据库技术,大数据管理.

杜梦圆(1990－),女,硕士生,主要研究领域为数据库技术,大数据管理.