International Journal of Multimedia and Ubiquitous Engineering
Vol. 10, No. 1 (2015), pp. 71-88
http://dx.doi.org/10.14257/ijmue.2015.10.1.7

The Research and Implementation of Temporal Quasi-order Data Index

Xiuqin Deng[1], Hongyan Xing[2], Xiaoping Ye[3] and Chengyan Fang[4]
[1, 2corresponding author, 4]*School of Applied Mathematics, Guangdong University of Technology, Guangzhou, P.R. China*
[3]*School of Computer Science, South China Normal University, Guangzhou, P.R. China*
deng_xiuqin@126.com

Abstract

This study proposed a quasi-order-based temporal data structure (QOTDS) which differed from conventional, algebraic data management models. Based on this QOTDS, a temporal data index called the temporal quasi-order index (TQOindex) was established. Firstly, the study proposed the concepts of temporal quasi-order (TQO) and linear order partitioning (LOP) of time period sets and discussed the construction algorithm of LOP and the optimum (minimum) properties. On this basis, a temporal data structure was established based on LOP. This structure realized the set-at-a-time data operation-like relational data structure and improved the inquiry efficiency by using multiple threads. Subsequently, in the structural framework of TQO, we discussed the temporal data index (TQOindex) based on quasi-order extensions. This index was effectively applicable to various conventional database platforms depending on the disk (external memory)-based data management and also to big data dynamic index technology relying on the incremental updating mechanism. Finally, a corresponding experimental simulation and comparative evaluation were designed to verify the feasibility and effectiveness of TQOindex. Research and experiments showed that QOTDS were effective at temporal inquiry and management in cases involving the temporal processing and integration mechanisms in new data, such as semantic data, XML data, and moving object data.

Keywords: *QOTDS, TQOindex, linear order partitioning, incremental updating, simulation and evaluation*

1. Introduction

As a reflection of objective entities, computer data are often used to describe and deal with time domain problems, especially in the data management fields under networked environments such as the web, e-commerce, etc. Such time-variant data can be considered as a snapshot data. Driven by the need to govern the past state and predict future development and application of such data, the temporal attributes of data should be reflected in any explicit formulation and effectively processed. The data carrying time-based tags are temporal data. Data query is a basic function of data processing.

However, due to the special characteristics of time, *i.e.*, unidirectionality (monotonically increasing), multi-dimensionality (effective, transaction, and user-time dimension), and interrelationship complexity (ALLEN temporal relationship [1]) *etc*., temporal data can rarely be included in the processing framework of traditional data and are merely searchable based on a temporal data index. According to previous research, existing temporal indices are mainly researched using the following methods:

A. Processing the temporal and non-temporal parts in sequence: this method is mostly used for temporal relationship data queries [2-7] basically relying on the idea of establishing a set of time (time period) index systems. In this method, data are firstly processed temporally and

Copyright ⓒ 2015 SERSC

then conventionally treated after screening select times. The characteristics of this method are indicated as: temporal query technology is researched basing on the attributes of time. Supporting by the mature technologies, the processing in sequence is realized effectively.

B. Including the temporal processing in the non-temporal processing framework: this method is mainly used in temporal-spatial data queries ([8-15]) with the concept of treating time as a new one-dimensional space and the data concerning the one-dimensional temporal and two-dimensional spatial attributes as three-dimensional spatial data. This method focuses on the spatial attributes attached to time. Although it effectively employs spatial index technology for reference, it fails to reflect the basic characteristics in which time differs from space.

C. Integrating the temporal processing into the non-temporal processing: this method is mainly employed for queries in temporal extensive make-up language (XML) data and moving object data [16-22]. Its basic idea involves aiming at the characteristics of the data, such as, the structural characteristics of XML and the trajectory features of moving objects: a corresponding temporal index mechanism is developed and integrated into the non-temporal query modules. Being different in its processing sequence from method A and the "integral inclusion" of method B, it highlights the characteristics of time and the internal correlation and restriction between temporal and non-temporal data.

In fact, with the expansion of data models that have been applied (such as the temporal correlation model for temporal object data, the semi-structural model of temporal XML, and the trajectory model of moving object data), the temporal data index mode based on method C has drawn much attention. Here, two fundamental points are raised for discussion, namely, to effectively express the temporal data structure and index framework of the temporal characteristics and the integral mechanism of temporal and non-temporal queries [21-22].

The quasi-order-based temporal data structure (QOTDS) proposed here is basically characterized by forming an LOP-based framework structure through organising temporal data using time and time period sets. Since QOTDS is built on the mathematics of relationship, it shows the desired expand space and can be applied to new data fields such as: temporal XML, moving object data, semantic data *etc*. Meanwhile, QOTDS also realizes the set-at-a-time query and multi-thread operation-like relational data and thus are suitable for distributed data management modes and the data management mode under networked environments such as P2P; in addition, the QOTDS is also applicable to the dynamic management of various large temporal datasets under due to its ability to incrementally update the insertion and deletion of data. Within the framework of QOTDS, this study firstly established the external memory-based temporal data index mode (TQOindex) depending on the mapping relationship of linear order branches (LOB) with temporal numerical sequencing. This mode reflected structural correlations in the data based on the characteristics of temporal data (time period) (a quasi order relation) and provided a reference and ideas for the temporal and non-temporal integrations. Secondly incremental updating is a fundamental challenge in data management. TQOindex realised the dynamic index mechanism supported by the technology in the insertion and deletion of the QOTDS incremental data. Finally, relying on a time data disk storage management based on a B+ tree, TQOindex was usable in all kinds of conventional database platforms. Therefore, it is operable and extendable to real applications.

This paper is organized as follows: Section 2 introduces the concept of TQO and investigates the construction algorithm and basic characteristics of the minimum linear order. Moreover, it also establishes the QOTDS and discusses the query and updating of LOP. Section 3 discusses the TQOindex based on minimum linear order partition (MLOP). The data query in this section used the conventional B+ tree mechanism based on the mapping relationship between LOB and "time and time period" sequences. Section 4 presents the simulation and evaluation of TQOindex to verify its feasibility and validity.

2. Data Structure of Quasi-order

Temporal data refers to the two tuples $Td = M < D, T_{stamp} >$. Where, *D* is the non-temporal data, T_{stamp} is the time label. Without loss of generality, T_{stamp} was assumed to be a valid time period (VT), VT=[VTs, VTe), VTs and VTe are start and end points of VT respectively (VTs ≤ VTe); if VTs=VTe, we define VT=[VTs, VTe) as an instant. Let Td is a temporal data, the effective time period of Td is denoted by VT (Td).

A relation R on a set E is called a *quasi-order* if it is reflexive and transitive.

2.1. LOB and LOP

Definition 1 (TQO) Let E be temporal data set, If the relationship $\precsim$ on E is defined as: Td_1, $Td_2 \in$ E, $Td_1 \precsim Td_2 \Leftrightarrow VT(Td_1) \subseteq VT(Td_2)$, "$\precsim$" is called a *temporal quasi-order* (TQO) on E.

Let Γ be the time period set. If $\forall\ u \in \Gamma$, *u* = [VTs,VTe), *u* corresponded to point P(*u*) = (VTs, VTe) in the plane VTs-VTe. Such correspondence is a 1-1 correspondence and P(*u*) is called a 2-dimension time point corresponding to *u*. Here, *u* corresponded to point P(*u*) and Γ corresponded to a point set $P(\Gamma)$ in the plane VTs-VTe.

Let P_0 = (min{VTs (P)}, max{VTe (P)}), P $\in \Gamma$, starting from point P_0, the traversal sequence obtained by $P(\Gamma)$ from "top to bottom" and "left to right" is called the $P(\Gamma)$ *sequence*. In the following, Γ, $P(\Gamma)$ and $P(\Gamma)$ sequences are not distinguished in this study.

Example 1 Figure 1 shows an example of a $P(\Gamma)$ sequence.

Definition 2 (LOB and LOP) Let Γ be the time period set having quasi-order "$\precsim$". A whole order branch of Γ is called an *linear order branch* (LOB, or L) of Γ Ω represents all the LOB sets on Γ If $\forall LOB_i, LOB_j \in \Omega, i \neq j, LOB_i \cap LOB_j = \varnothing$, and $\bigcup_i LOB_i = \Gamma$, Ω is called an *linear order partition* (LOP) on Γand is denoted by LOP(Γ).

Algorithm l (The precedence algorithm under LOP)

It was assumed that there was a $P(\Gamma)$ sequence.

Step1. From the header element u_0 of $P(\Gamma)$ to $u_{i_{.0}} \in P(\Gamma)$:

$VT_s(u_{i_{.0}}) = VT_s(u_0) \wedge (VT_s(u_{i_{.0}+1}) \neq VT_s(u_0))$, $u_{i_{.0}+1}$ is the following element of $u_{i_{.0}}$ on $P(\Gamma)$.

Step2. From $u_{i_{.0}}$ to $u_{i_{.1}}$:

$VT_e(u_{i_{.1}}) = VT_e(u_{i_0}) \wedge (VT_s(u_{i_1}) = \min\{VT_s(u_j)\})$, where,

$u_j \in P(\Gamma) \wedge (\exists u_k \in P(\Gamma), VT_s(u_k) = VT_s(u_j) \wedge VT_e(u_k) < VT_e(u_j))$.

Step3. From $u_{i_{.1}}$: repeat Steps 1 and 2 until $u_m \in P(\Gamma)$, $\nexists\ u'_m \in P(\Gamma)$, such that $(VT_s(u_m) < VT_s(u'_m) \wedge VT_e(u'_m) < VT_e(u_m))$, the subsequence in $P(\Gamma)$ from u_0 to u_m is a LOB_1.

Step4. From the header element in $P(\Gamma) \setminus LOB_1$, repeat Steps 1 to 3. By calculation, $LOB_2, \cdots, LOB(\Gamma)$ were then obtainable.

If $VT_s(\Gamma) = \max\{VT_s(u) \mid u \in \Gamma\}$, $VT_e(\Gamma) = \max\{VT_e(u) \mid u \in \Gamma\}$, the maximum time complexity of Algorithm 1 is $VT_s(\Gamma) \times VT_e(\Gamma)/2$.

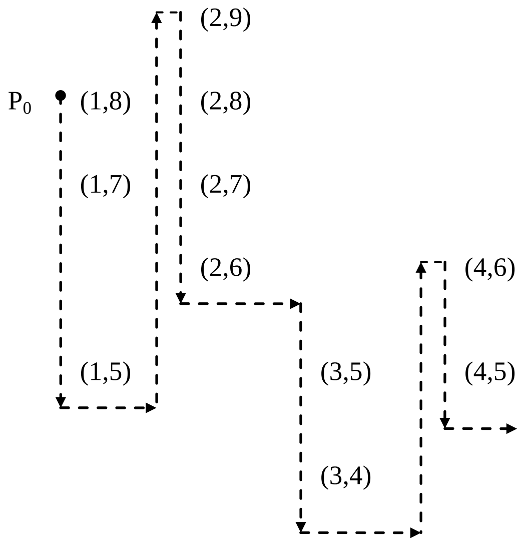

Figure 1. P(Γ) Sequence

Example 2 As for the P(Γ)=<[1,8), [1, 7), [1,5), [2, 9), [2, 8), [2, 7), [2, 6), [3, 5), [3,4), [4, 6), [4, 5)> in example 1, Algorithm 1 is realised according to Figure 2, we then obtain the following two LOBs:

LOB_1=<[1,8), [1, 7), [1,5), [3, 5), [3,4)>

LOB_2=<[2, 9), [2, 8), [2, 7),[2, 6),[4, 6), [4, 5)>

Definition 3 (MLOP) If LOP_0 is the LOP on Γ, if $|LOP_0| \leq |LOP|$ for every element LOP∈ Γ, LOP_0 is called *minimum linear order partition* (MLOP) on Γ.

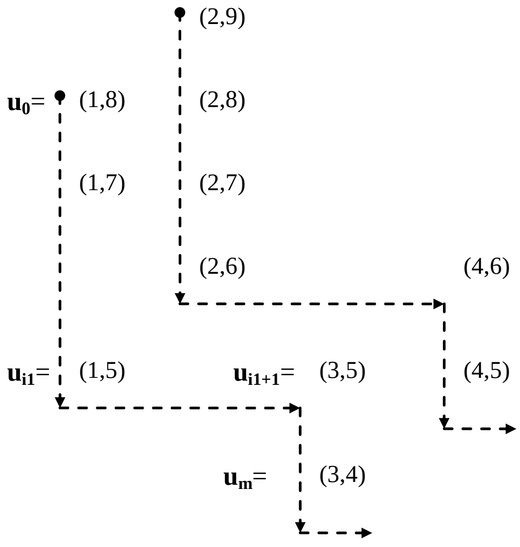

Figure 2. The Realization of Precedence Algorithm under LOP

$$\begin{array}{l}
i_1 j_m, \quad i_2 j_m, \quad \cdots\cdots, \quad i_{n-1} j_m, i_n j_m \\
i_1 j_{m-1}, \ i_2 j_{m-1}, \cdots\cdots, i_{n-1} j_{m-1} \\
\cdots\cdots, \quad \cdots\cdots, \quad \cdots\cdots, \\
i_1 j_2, \quad i_2 j_2 \\
i_1 j_1
\end{array}$$

Figure 3. Temporal Order Matrix TOM(Γ)

Definition 4 (TOM)

Let $i_1 = \min_\Gamma\{VT_s(u)\}$, $i_n = \max_\Gamma\{VT_s(u)\}$, $j_1 = \min_\Gamma\{VT_e(u)\}$, $j_m = \max_\Gamma\{VT_e(u)\}$. The time periods in Γ is $u = [VT_s, VT_e) = [i, j)$, and is denoted by $i, j\ (i_1 \le i \le i_m, j_1 \le j \le j_m)$. Let horizontal axis and vertical axis in the plane being the starting and ending points of the [*i*, *j*) respectively, the grid point set determined by $\{i, j \mid i_1 \le i \le i_n, j_1 \le j \le j_m\}$ (Figure 3) is called *Γ*-based *temporal order matrix* (TOM), and denoted by $TOM\ (\Gamma\}$.

For $u_{i_0 j_0} \in TOM\ (\Gamma)$, $TOM\ (\Gamma\}$ can be divided into four regions by $u_{i_0 j_0}$: $UL\ (u_{i_0 j_0}) = \{v_{ij} \mid i \le i_0, j_0 \le j\}$, $UR\ (u_{i_0 j_0}) = \{v_{ij} \mid i_0 \le i, j_0 \le j\}$, $DL\ (u_{i_0 j_0}) = \{v_{ij} \mid i \le i_0, j \le j_0\}$, and $DR\ (u_{i_0 j_0}) = \{v_{ij} \mid i_0 \le i, j \le j_0\}$. In the formula above, if only "<" is supported, corresponding regions are called *open regions*, and denoted by *OUL*, *OUR*, *ODL*, and *ODR* respectively. Figure 4 shows the "upper-left" *UL*(*23*) and "down-right" *DR*(*23*) regions of temporal node "23" in $TOM\ (\Gamma\}$.

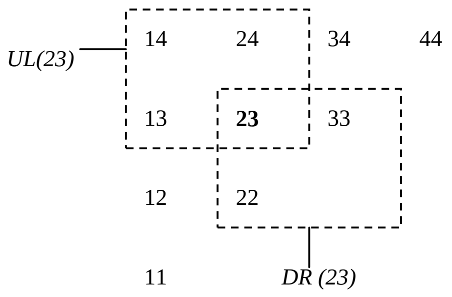

Figure 4. UL(23) and DR(23)

Theorem 1 (TOM and quasi-order) if $\forall u_0 \in TOM\ (\Gamma)$, we have

(1)$u_0 \subseteq v_0 \Leftrightarrow v_0 \in UL(u_0)$;

(2) $v_0 \subseteq u_0 \Leftrightarrow v_0 \in DR(u_0)$;

(3) $\neg(u_0 \subseteq v_0 \vee v_0 \subseteq u_0) \Leftrightarrow v_0 \in OUR(u_0) \vee v_0 \in ODL(u_0)$

Proof (1) Let $u_0=[i_0, j_0)$, $v_0=[k_0, l_0)$, clearly we know $u_0 \le v_0 \Leftrightarrow u_0 \subseteq v_0 \Leftrightarrow k_0 \le i_0, j_0 \le l_0 \Leftrightarrow v_0 \in UL(u_0)$

(2) and (3) can be proved in same way.

Theorem 2 (The properties of precedence algorithm under LOP)

The LOP obtained using the Algorithm 1 is MLOP.

Proof Supposing that $LOP_0 = < LOB_1, LOB_2, \cdots, LOB_n >$ is obtained by Algorithm 1, where, LOB_i is sequenced according to calculating order. $\forall u_{i_0} \in LOB_i\ (1 < i \le n)$, $\exists u_{k_0} \in LOB_{i-1}$,

$\neg(u_{i_0} \subseteq u_{k_0} \vee u_{k_0} \subseteq u_{i_0})$. Actually, it is only needed to explain that there are elements of LOB_{i-1} in $DL(u_{i_0})$. Otherwise, if such elements are absent, LOB_{i-1} locates in $UL(u_{i_0})$, and $ODR(\min LOB_{i-1})$ contains LOB_i. This result was contradicted with the *LOB* in Algorithm 1. If the nodes obtained by here were $u_1, u_2, \cdots, u_n$, we got $u_{i-1} \in ODL(u_i)$ $(1 < i \leq n)$, that is $\neg(u_i \subseteq u_{i-1} \vee u_{i-1} \subseteq u_i)$. Therefore, an arbitrary *LOP* contains *n LOBs* at least. The proof completed.

Definition 5 (Extended LOB and LOP)

Let *LOB*=< u_1…, u_i, u_{i+1},…, u_m>, u_i and u_{i+1}∈*LOB*, the folded line segments obtained using following methods are called the *extended LOB* on VTs-VTe, and is denoted by ELOB.

(1) For arbitrary adjacent u_i and u_{i+1} in LOB,

● if VTs(u_i)= VTs(u_{i+1})∨*VTe*(u_i)= *VTe*(u_{i+1}), u_i and u_{i+1} are connected using a linear segment.

●if ¬ (VTs(u_i) = VTs(u_{i+1})∨*VTe*(u_i)= *VTe*(u_{i+1})), a point *v* (*v* =[*VTs*(u_{i+1}),*VTe*(u_i))) is inserted between u_i and u_{i+1}. Then u_i and u_{i+1} are connected with *v* using line segments respectively.

(2)Connecting point *P*(u_m) of the minimum time period u_m on LOB and point P(v_0) on the diagonal on plane VTs-VTe was done using line segments, where, v_0=(VTs(u_m), VTe(u_m)).

If point u on ELOB belonged to Γ, point u is called *real instant*, and is denoted by u(r). Otherwise, it is called *fill instant*, and is denoted by u(f).

The set constituted by all the ELOBs corresponding to the LOB in MLOP (Γ) is called *extended MLOP*, and is denoted by EMLOP (Γ).

Example 3 For MLOP=< LOB_1, LOB_2> in Example 2, EMLOP=< $ELOB_1$, $ELOB_2$>, as shown in Figure 5. Where, "shallow" node corresponding to u(f).

$ELOB_1$=<[1,9), [1,8), [1, 7), [1,6), [1,5), [2,5) [3,5), [3,4), [4,4) >,

$ELOB_2$=< [2,9), [2, 8), [2, 7), [2, 6), [3,6), [4,6), [4,5), [5,5) >

EMLOP (Γ) and MLOP (Γ) can be constructed simultaneously. EMLOP(Γ) was stored and managed using a two-dimensional array EMLOP [VTs][VTe], with array elements N= (no, flag, [VTs, VTe)). Where, no is the ELOB number of time u=[VTs, VTe); flag is used to identify whether or not u belongs to MLOP. When u ∈ MLOP, flag=t (true); when u∈EMLOP\MLOP, flag=f (false). For example, in Figure 3, u=[1,5) ∈ ELOB1, thus no=1, u=[1,5)∈MLOP, flag=t; u=[3,6)∈$ELOB_2$, no=2, u=[3,6)∈EMLOP\MLOP, flag=f.

2.2. Dynamic Management of LOP

In terms of massive temporal data, incremental updating plays an essential role in realizing dynamic management. Hence, it was necessary to discuss the incremental updating of LOP.

2.2.1. Insertion Updating

Definition 6 (the nearest LOB of time period u)

If u denotes time and there is a LOB_i satisfying following conditions:

(∃v∈LOB_i (u⊆v))∧(∃w(u⊆w)→(w∈LOB_i∨w∈LOB_{i+k})),

Where k>0, then, LOB_i is called the *nearest LOB of u*.

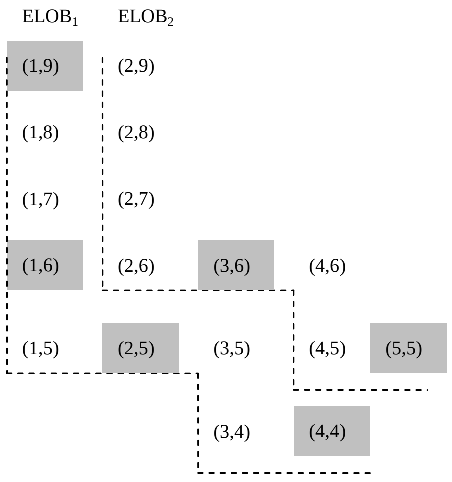

Figure 5. EMLOP (Γ)

Definition 7 (supremum and infimum of time)

It is assumed that u is a time period. For v∈LOB$_0$, if condition (u⊆v)∧(∃w∈LOB$_0$(u⊆w)→v⊂w) is satisfied, v is called the *supremum* of u in LOB$_0$, and is denoted by $v = \sup_{LOB_0}(u)$. If (v⊆u)∧(∃w∈LOB$_0$(w⊆u)→w⊂v), v is called the *infimum* of u in LOB$_0$, and is denoted by $v = \inf_{LOB_0}(u)$.

Algorithm 2 (EMLOP insertion algorithm)

Step1. Supposing that a new time period u was inserted into the given EMLOP(Γ) and LOB$_0$ serves as the nearest LOB of u. In LOB$_0$, sup(u)=u$_0$ and inf(u)=u$_1$. According to Algorithm 1, a new LOB$_0$ can be obtained by connecting u$_0$ with u$_1$.

Step2. Acting as the newly-inserted point in EMLP\ LOB$_0$, the point in LOB$_0$ segment <u$_0$,u$_1$>\{u$_0$,u$_1$} calls Step 1.

The maximum time complexity of Algorithm 2 is |EMLOP(Γ)|×max{|ELOB|}.

Example 4 As shown in Figure 5, u=[2, 4) was inserted into EMLOP(Γ) and the nearest LOB was LOB$_1$. In LOB$_1$, u$_0$=sup(u)=[1, 5), while u$_1$=inf(u)=[3, 4). Based on Algorithm 1, a new ELOB$_1$ was acquired, as shown in Figure 6. In this situation, LOB$_1$ segment <u$_0$, u$_1$>\{u$_0$, u$_1$}= {[3, 5)}. Then, the process was repeated as above using v=[3, 5) as the newly-inserted point, as shown in Figure 7. The final results are shown in Figure 8.

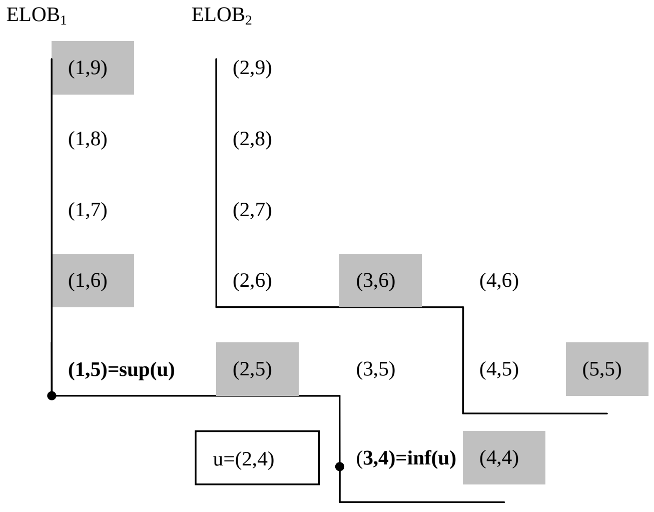

Figure 6. Insertion of u= [2, 4)

2.2.2. Deletion Updating

Algorithm 3 EMLOP deletion algorithm

Firstly, the following conditions were hypothesised: three points ($u_0 = [VT_s(u), VT_e(u))$, $v_0 = \sup(u)$, and $w_0 = \inf(u)$) in ELOB$_i$ need to be deleted; (Open) rectangular frame $R(u_0) = \{u \mid VT_s(u_0) < VT_s(u) < VT_s(w_0) \wedge VT_e(u_0) < VT_e(u) < VT_e(v_0)\}$ intersects with $ELOB_{i_0}, ELOB_{i_2}, \cdots, ELOB_{i_m}$.

Step1. In ELOP, all elements that were included in $R(u_0)$ and also belonged to Γ were deleted. Then according to Algorithm 1, a new $ELOB_{i_0}$ was produced by connecting v_0 with w_0 in EMLOP.

Step2. Let u_k denoted an element that was included in $R(u_0) \setminus \{u_0\}$ and also belonged to Γ. The construction process of $ELOB_{i_0}$ was to delete the corresponding u_k in succeed $ELOB_{i_1}, ELOB_{i_2}, \cdots, ELOB_{i_m}$, and then repeat Step 1.

The maximum time complexity of Algorithm 3 is |EMLOP (Γ)|×max{|ELOB|}.

Example 5 AS shown Figure 5, on the EMLOP (Γ), three points ($u_0 = [1,5)$, $v_0 = \sup([1,5)) = [1,7)$, and $w_0 = \inf([1,5) = [3,5))$) in $ELOB_1$ have been deleted. Figure 9 shows the two points ($u_1 = [2,6)$ and $u_2 = [2,7)$ belonged to ELOB$_2$) which needed to be deleted in $R(u_0)$. Based on Algorithm 1, $v_0 = [1,7)$ is connected with $w_0 = [3,5)$, and the new ELOB$_1$ was thereby obtained, as shown in Figure 9. To establish ELOB$_1$, we needed to delete $u_1 = [2,6)$ and $u_2 = [2,7)$ in ELOB$_2$, that is, $u_1 = [2,6)$ and $u_2 = [2,7)$ can be taken as the newly-deleted points in EMLOP (Γ)\ELOB$_1$. In ELOB$_2$, sup ([2, 7)) = [2, 8), while inf ([2, 6)) = [4, 6). According to Algorithm 3, by connecting [2, 8) with [4, 6), a new EMLOP was produced (Figure 10).

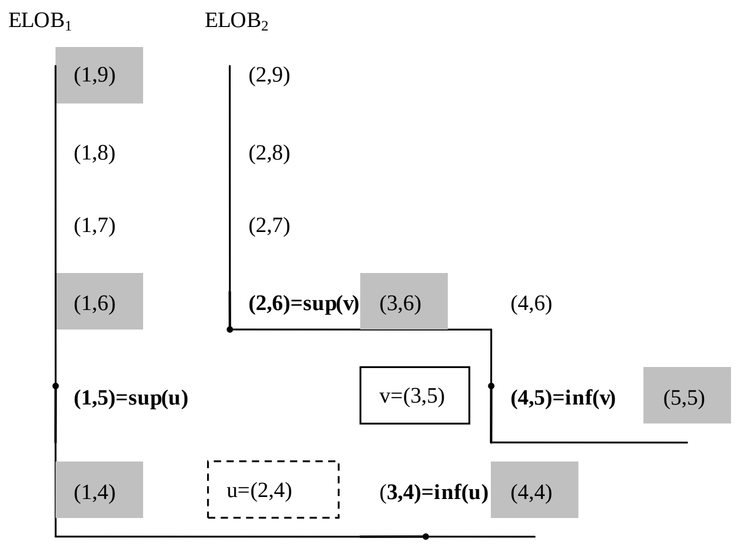

Figure 7. ELOB1 Resulting from the Insertion of u= [2, 4)

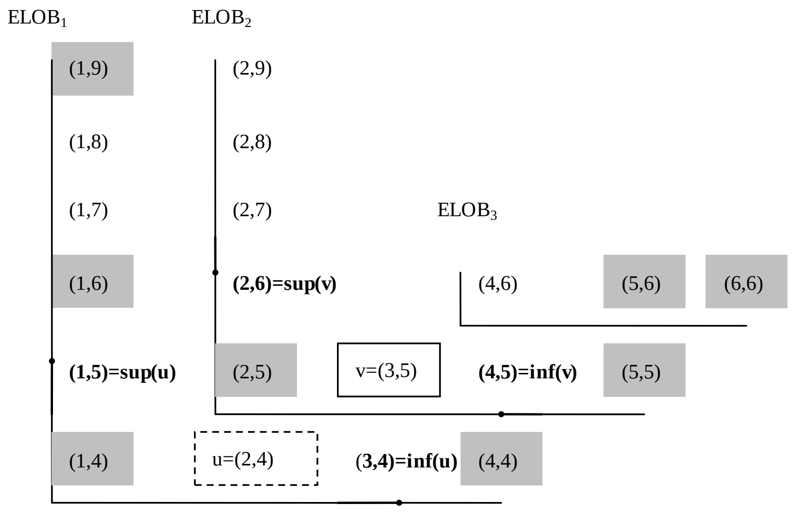

Figure 8. Final Results (ELOB1, ELOB2, and ELOB3)

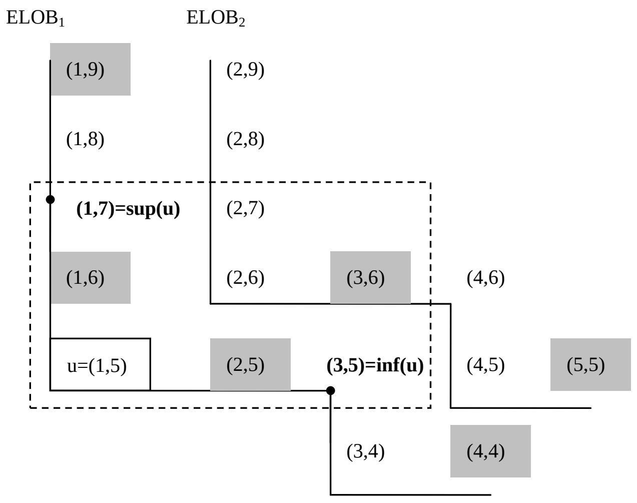

Figure 9. ELOB1 Resulting from Deletion

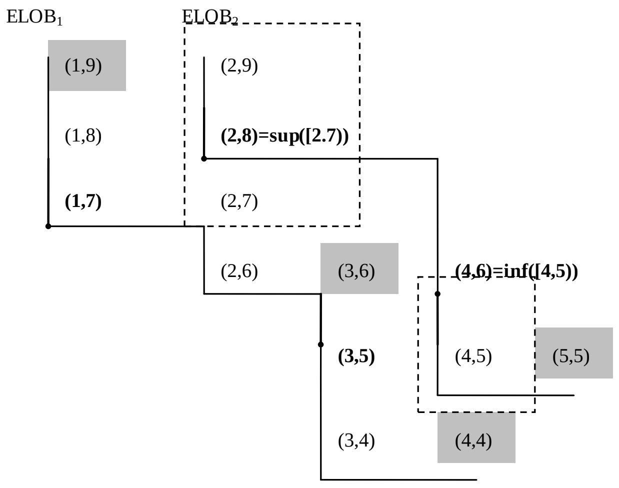

Figure 10. ELOB2 Arising from Cascade

3. The Data Index TQOindex

Data structure plays an essential role in index construction. Therefore, the index pattern of LOP-based temporal data structure (that is, the TQOindex) needs to be investigated.

3.1. Temporal Quasi-order Index Tree

Definition 8 (auxiliary query set (AQS) of MLOP)

For a given EMLOP(Γ), query period Q_0 = [VTs, VTe) and $\forall$ELOB$\in$EMLOP(Γ), according to quasi-order "$\precsim$", the minimum time point including Q_0 for every ELOB on ELOP is defined as an *auxiliary query point of Q_0*. The auxiliary query set (AQS) Q_0 on ELOP is denoted by AQS(Q_0).

Example 6 Let's consider the MLOP(Γ) in Example 3, the auxiliary point of the query point Q_0 = [2, 4) for $ELOB_1$ is [2,5), and that for $ELOB_2$ is [2,6). As seen in Figure 11, AQS([2, 4)) = {[2,5), [2,6)}, where [2,5) is a fill point, [2,6) is a real point.

Definition 9 (LOB-based time number) It was first supposed that u$\in$LOB; u=[VTs, VTe); and LOB$\in$MLOP, the serial number is no (LOB). The LOB-based time number of the time period u is defined as:

TN(u, LOB)= no(LOB)$\times 10^{2r}$ +Ve$\times 10^{r}$-Vs)

Where *r* refers to the bits of the maximum end-point for all time periods in the LOB.

Theorem 3 (basic properties of time number)

(1) For $u,v\in LOB$, $u \neq v$ iff $TN(u, LOB) \neq TN(v, LOB)$.

(2) For $u,v\in LOB$, $u \precsim v$ iff $TN(u, LOB) \leq TN(v,LOB)$.

Proof

(1) We presumes that u=[$VTs(u)$, $VTe(u)$) and v = [$VTs(v)$, $VTe(v)$). $U \neq v \Leftrightarrow VTs(u)-VTs(v) \neq 0 \vee VTe(u)-VTe(v) \neq 0$. But $TN(u,LOB)-TN(v,LOB)=(VTe(u)-VTe(v))\times 10^{r}-(VTs(u)-VTs(v))$. In this way, the conclusion was proved.

(2) If $u \precsim v$, then it is obvious that $u \subseteq v \Leftrightarrow VTs(v) \leq VTs(u) \wedge VTe(u) \leq VTe(v) \Leftrightarrow VTs(u)-VTs(v) \geq 0 \wedge VTe(u)-VTe(v) \leq 0$. However, $TN(u, LOB)-TN(v, LOB)=(VTs(u)-VTs(v))\times 10^{r}-(VTs(u)-VTs(v))$. As a result, required conclusion was verified.

It could be inferred from (1) in the aforementioned theorem that, for each element u in LOB, there was only one corresponding *TN* (*u*, *LOB*) determined. Meanwhile, from (2) it indicated that the quasi-order relationship between the elements of LOB could be interpreted by the corresponding time number.

Definition 10 (TQO-based index, TQOindex) Temporal quasi-order index of time period set Γ is denoted by TQOindex(Γ)=<EMLOP (Γ), MLOPB^{+}-tree (Γ)>. where

(1) EMLOP (Γ) was a two-dimensional array as provided in Definition 4.

(2) MLOPB^{+}-tree acted as a B^{+}-tree for storing the quasi-order set MLOP and the index object was the time number TM(u, LOB) corresponding to element u in the LOB.

Example 7 For MLOP in Example 3, the corresponding TQOindex (Γ) is shown in Figure 12.

Copyright ⓒ 2015 SERSC

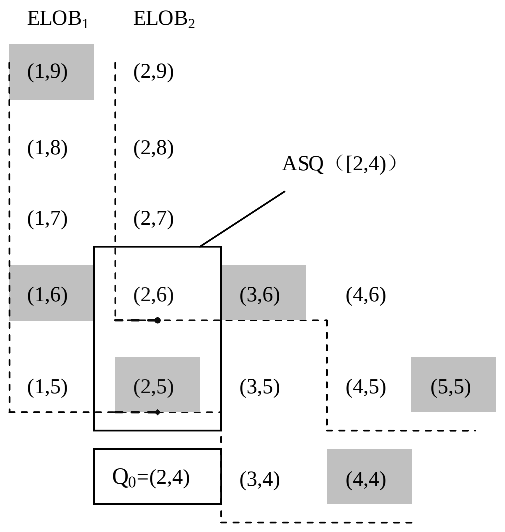

Figure 11. AQS(Q_0) of the Query Point $Q_0 = [2, 4)$

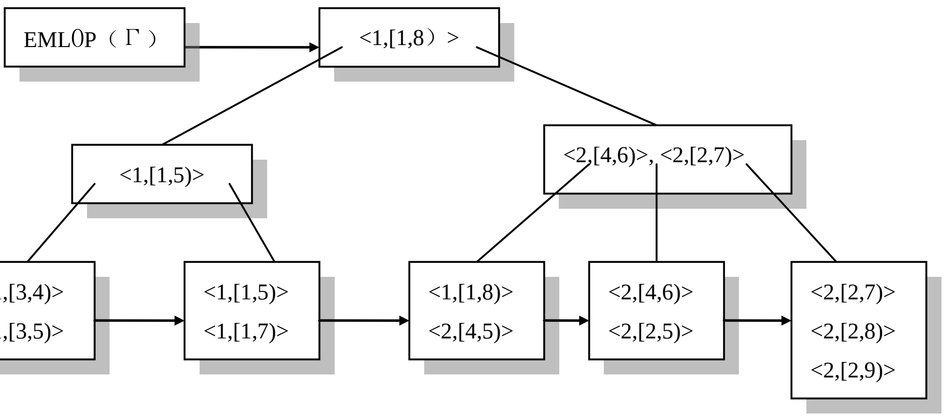

Figure 12. TQOindex (Γ)

3.2. Data manipulation

During data query, we put EMLOP (Γ) into the memory, and let the query time Q_0=[Vs(Q_0),Ve(Q_0)). Through sequencing storage of the two-dimensional array EMLOP (Γ), we may obtain AQS(Q_0)={(no, Vs, Ve)}. ∀ P_0∈ASQ(Q_0), the MLPB$^+$-tree (Γ) was queried by taking P_0 as the query target. We use B$^+$-tree to conduct a query to find out the minimum time number TN(v_0) that was larger than, or equal to, TN(P_0) in leaf nodes. Therefore, all TN(v) that were larger than or equal to TN(v_0) in the LOB including v_0 are the query result.

Algorithm 4 (TQOindex-based query)

Step1 Querying Q_0 = [Vs(Q_0), Ve(Q_0)) was transformed into TN (Q_0), and AQS(Q_0) was searched over EMLOP (Q_0). If AQS(Q_0) = ∅, no query result can be found in MLOP, and the query was ended. Otherwise, the query continued to Step 2.

Step2 Each u∈AQS (Q_0) had access to the MLOPB$^+$-tree (Γ) for simultaneous queries by multi-threads to yield the corresponding query results.

Step3 Converting results (time number) obtained in Step 2 into time periods and outputting them as the final results.

The maximum time complexity of Algorithm 4 in searching AQS (Q_0) in EMLOP (Γ) was |EMLOP(Γ)|×max{|ELOB|}. ∀ELOB∈EMLOP(Γ) and |E| represented the cardinal number of set E.

The basic process of TQOindex (Γ)-based temporal query is shown in Figure 13.

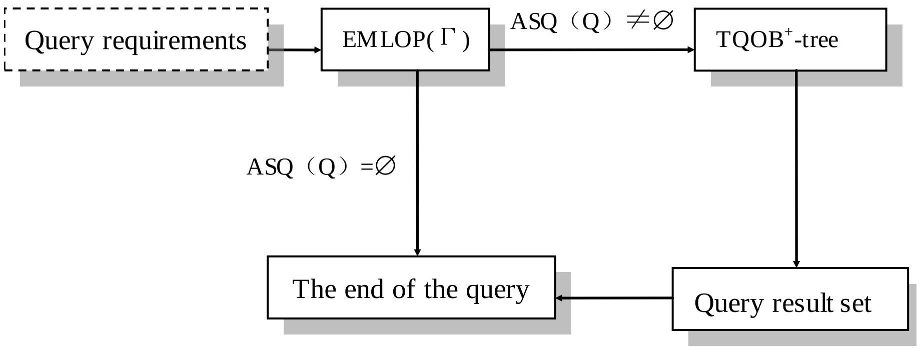

Figure 13. TQOindex (Γ)-basedQuery

The characteristics of TQOindex-based temporal query were as follows:

(1) If AQS (Q_0)=∅, then no result related to Q_0 in Γ and the query was thus ended. In this case, it was unnecessary to call MLOPB$^+$-tree (Γ). The query only proceeded in the memory and stayed off the disk. The query efficiency was thereby improved. If AQS (Q_0) ≠ ∅, MLOPB$^+$-tree (Γ) was proceeded. Elements of AQS (Q_0) were queried successively according to conventional B$^+$-tree. When LOB segments satisfying the conditions were distributed across many leaf nodes, subsequent leaf nodes can be discovered through the leaf node pointer. After that, with the aid of the LOB serial number, required results can be obtained.

(2) Based on LOB properties, if one result $v_0 \in LOB_0$ was found in LOB_0 in a certain leaf node of the MLOPB$^+$-tree (Γ), the segments after v_0 in LOB_0 were all the query results. Then, the query result set was obtained, namely set-at-a-time.

(3) The MLOP acts as the LOP of Γ. The LOBs therein were mutually unrelated. When |AQS(Q_0)|>1, diverse auxiliary query points can be simultaneously processed through multi-threads in the MLOPB$^+$-tree (Γ). The larger |AQS(Q_0)|, the higher the multi-thread efficiency.

Example 8. Assume that Q_0=(2, 4). From EMLOP (Γ) in Figure 5, it was inferred that AQS(Q_0)={(1, (2, 5)), (2, (2, 6)) }. According to TQOindex (Γ) in Figure 12 and auxiliary query point (1, (2, 5)), the query results <1,(1, 5)> and <1,(1, 8)> were acquired. Using the auxiliary query point (1, (2, 6)), the query results <2,(2, 7)>, <2,(2, 8)>, and <2,(2, 9)> were acquired. Therefore, the final query result set was {<1,(1, 5)>, <1,(1, 8)>, <2,(2, 7)>, <2,(2, 8)>, <2,(2, 9)>}.

4. Data Simulation and Evaluation

Map21-tree [23] was selected for comparative evaluation. The parameters involved in the experimental data were set as follows: the time periods including [0, maxTime) and the corresponding time period set Γ were generated randomly (maxTime denoted the maximum time end-point of the time periods generated); the disk block size was 1024 kB. Each test query was consisted of 50 operational sentences, and running expense corresponding to these I/O is the mean of the 50 operations.

4.1. Data Query

4.1.1. Based on Data Size

We set maxTime=2000 and the maximum time period span as 10% of maxTime. Then, data sizes of time periods randomly generated were: 1×10^5, 2×10^5, 3×10^5, 4×10^5, 5×10^5, 6×10^5, 7×10^5, 8×10^5, 9×10^5, and 1×10^6 respectively. In Figure 14, the horizontal axis denotes data size (the number of time periods), while the longitudinal axis represents the frequency of I/O access of the disk block. Figure 14 shows that, under a constant query span, the index node number increased with increasing dataset size. As a result, there were more nodes needing to be accessed. Besides, the I/O frequency presented a rising trend for both TQOindex and Map21-tree. However, compared with Map21-tree, TQOindex showed a slower increase of I/O frequency and better performance.

4.1.2. Based on Disk Block Size

Let maxTime=2,000 and query span as 10% of maxTime; 5×10^5 time periods were generated randomly; disk block sizes were: 2^9 B, 2^{10} B, 2^{11} B, and 2^{12} B respectively. In Figure 15, the horizontal axis denotes block size, while the longitudinal axis refers to I/O frequency required for the query. It can be inferred from Figure 15 that with the increase of block size, the I/O query frequency needed by both TQOindex and Map21-tree decreased. This was attributed to the idea that, under constant time period number, the larger the block size, the smaller the index node number and the node number that needed to be accessed in the query. In this situation, TQOindex outperformed Map21-tree.

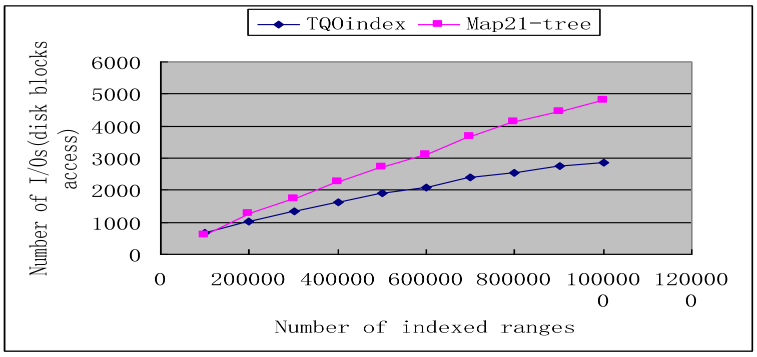

Figure 14. The Variation of I/O as the Data Size Increases

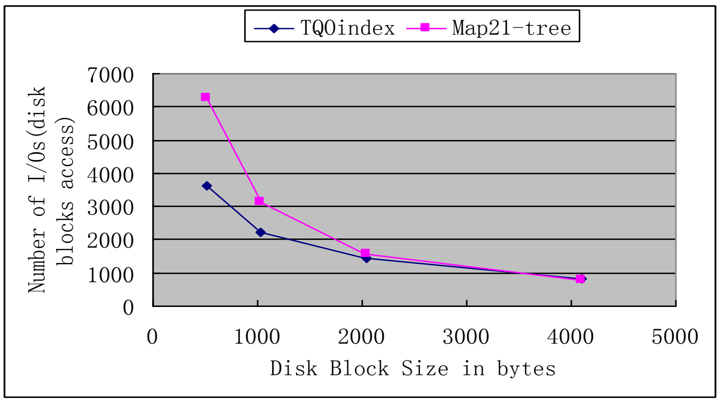

Figure 15. The Variation based on Different Disk Block Size

4.1.3. Based on Query Period Span

It was assumed that maxTime=2,000 and time period set Γ comprised 5×10^5 randomly generated time periods. The query period span accounted for 1%, 5%, 10%, 15%, 20%, 25%, 30%, 35%, 40%, 45% and 50% of maxTime respectively. In Figure 16, the horizontal axis denotes the query span, while the longitudinal axis represents I/O frequency. Figure 16 indicates that with the gradual increase of the query span, index data and the I/O frequency queried by both TQOindex and Map21-tree decreased. However, in this process, TQOindex outperformed Map21-tree.

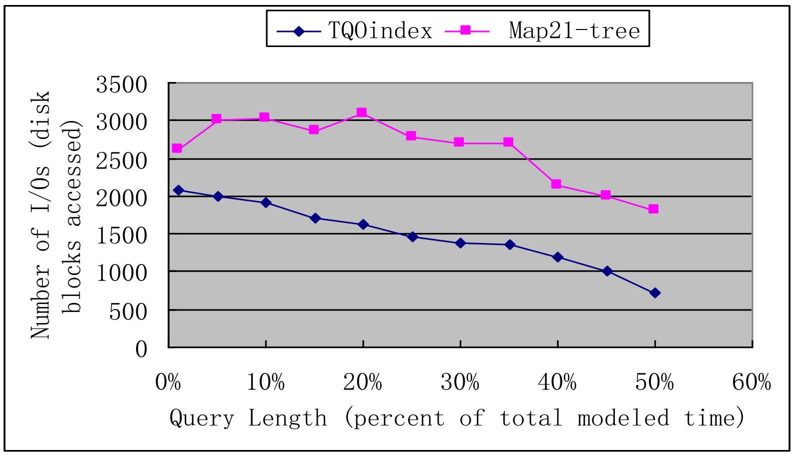

Figure 16. The Variation based on Different Query Spans

4.2. Data Updating

4.2.1. Data Insertion

We set maxTime=2,000, and the time period set Γ consisted of 5×10^5 randomly generated time periods, MLOP (Γ) comprised 1,277 LOBs produced in Γ using Algorithm 1. With 5×10^3 randomly generated time periods being inserted into Γ, some LOBs in MLOP (Γ) needed reorganising, *i.e.* incremental updating was produced. Meanwhile, 5,000 newly-inserted time periods were simulated and investigated to determine the number of time periods causing the reconstruction of 1 LOB, 2-5 LOBs, ……, respectively. The experimental results are shown in Table 1 and Figure 17. In figure 17, the horizontal axis shows the changes during the inserted time periods are located in the largest range, the vertical axis represents the corresponding number that need to rebuild the LOB.

4.2.2. Data Deletion

In the experimental data, the maximum time span is as maxtime=500 to 2000. Every time, 500000 time periods are randomly generated, there were 1,277 LOBs in total. From 500000 time periods, we select 5000 to do the deleting test. The results show that the total number of the affected time periods when 5000 time periods are deleted is about 5014, that is, the average branch number affected by deleting a time period is about 1.003. The experimental results are shown in figure 18. In figure 18, the horizontal axis shows the number of time periods that each time period is involved when 5000 time periods are deleted, the vertical axis represents the changes of the time span when 5000 time periods are deleted.

Table 1. Data Variation Induced by Insertion Updating

number of LOBs needed for reconstruction	1	2 to 5	6 to 10	11 to 25	More than 25
time periods causing LOB reconstruction (in 5,000 time periods)	3,623	698	220	220	239
percentage over 5,000 time periods	72.5 %	13.96 %	4.40 %	4.40 %	4.78 %

Copyright ⓒ 2015 SERSC

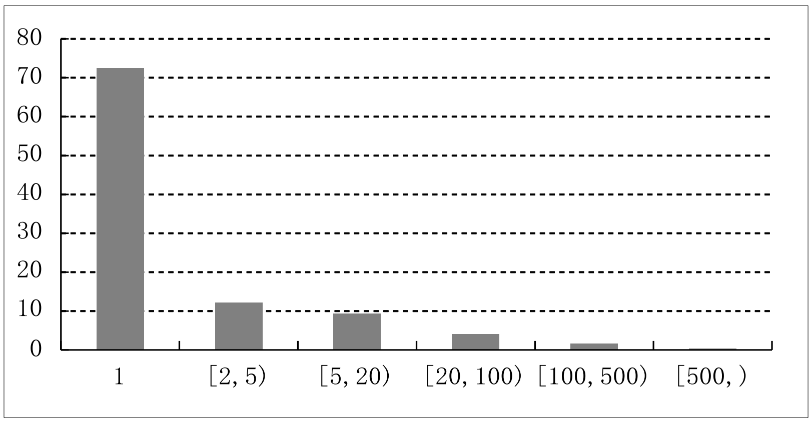

Figure 17. The Changes of the Affected LOB when Inserting New time Periods

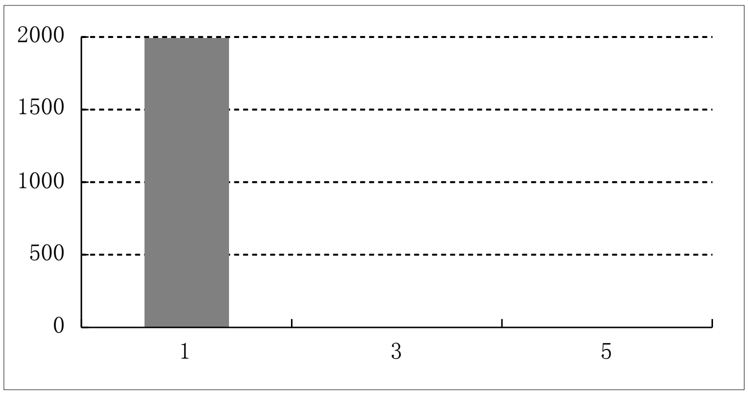

Figure 18. The Number of other Time Periods Effected by the Average Time Period after Deleting 5000 Time Periods when MaxTime Changes

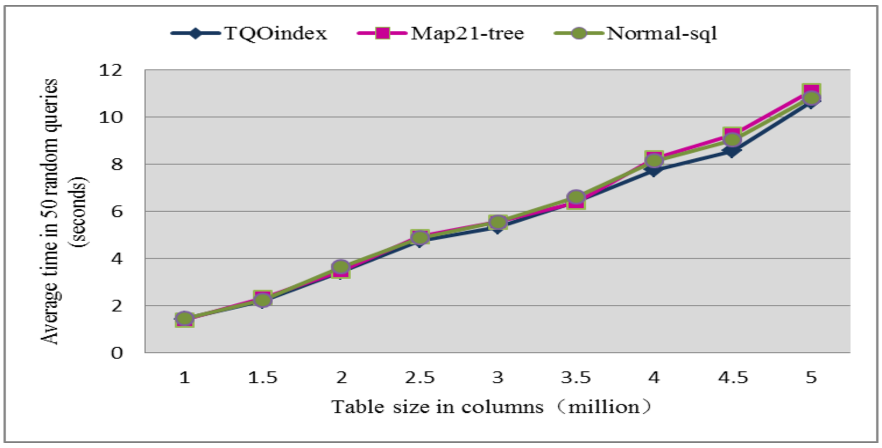

Figure 19. Temporal Query Test based on Temporal Relational Data

4.3. Temporal Relational Data Query

As a type of temporal data index structure, TQOindex was proved to be able to efficiently realize temporal information queries of the data in files of external memory according to the simulation. As most databases are disk-based, TQOindex is usable on practical database platforms. SQL Server 2008 was used in this research to randomly generate large-scale VT-based temporal relational data files. Meanwhile, VTs and VTe were processed as conventional attributes. By adopting SQL Server 2008, MAP21, TQOindex, and the tuple-meeting corresponding time conditions were filtered. The corresponding simulation results are shown in Figure 19.

Due to the presence of corresponding data structures, the TQOindex was applicable to non-classical data (such as: XML, object-oriented data, moving objects, *etc*.). By comparison, Map21 and conventional SQL queries were unsuitable for such extended application, and their temporal query efficiency for relational data was also much lower than that of TQOindex. This indicated that the results in this research can show adaptability and applicability in various situations.

5. Conclusions

Driven by applications in temporal and non-temporal data integration processes, this research focused on analyzing the applications of QOTDS to relational data management. Instead of conventional algebraic data patterns, this research investigated an order-based temporal data structure using following steps: firstly, according to the internal relationship beween non-temporal data and the time tags in temporal data, the concept of TQO was proposed. Subsequently, we discussed the precedence algorithm of LOB, established the temporal data structure (QOTDS) based on quasi-order relation was discussed and verified optimality (minimality) of QOTDS in certain. Moreover, based on QOTDS, the incremental algorithm for temporal data query and data updating was investigated. QOTDS was suitable for both internal and external memory since it realised set-at-a-time data queries and multi-thread optimisation mechanisms under mathematical support; besides, it met the management requirements of various new-type temporal data and showed the desired spatial expansion. Furthermore, a QOTDS-based data index (TQOindex) was also discussed. Our work indicated that TQOindex would fulfill index construction on the B+-tree platform supported by a conventional database system. Lastly, TQOindex has been simulated with large dataset sizes and the comparative evaluation of TQOindex and existing data index methods proved the efficacy. Above all, QOTDS was applicable to the data index of objects, XML, and moving objects.

Acknowledgements

This work was supported by the Industry-University Research Project funded by the Education and Guangdong Province Grant No. 2012B091100489 and the University Students’ Innovation project Grant No. 1184513258.

References

[1] J. F. Allen, Comm. of the ACM, vol. 26, no. 11, **(1983)**, pp. 832-843.
[2] T. Bozkaya and M. Ozsoyoglu, ACM Infor. Sci., vol. 112, **(1998)**, pp. 85-123.
[3] R. Bliujute, C. S. Jensen and S. Saltenis, “R-tree based indexing of now-relative bitemporal data”, Proceedings of the 24th VLDB Conference, **(1998)**.
[4] R. Bliujute, C. S. Jensen and S. Saltenis, Proceedings of the 12th International Conference on Scientific and Statistical Database Management, **(2000)**.
[5] B. Stantic, S. Khanna and J. Thornton, “An efficient method for indexing now-relative bitemporal data”, Proceedings of the fifteenth conference of Australasian database, **(2004)**.

[6] M. M. Moro and V. J. Nehme, “Transaction-time indexing”, In Temporal Database Entries for the Springer Encyclopedia of Database Systems, volume Time Center Technical Report TR-90, **(2008)**; New York: Springer US.
[7] D. Lomet, M. Hong and R.Nehme, “Transaction time indexing with version compression”, Proceedings of the VLDB Endowment, **(2008)**.
[8] Y. Tao and D. Papadias, “MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries”, Proceedings of the Intl. Conf. on Very Large Data Bases, VLDB, **(2001)**.
[9] D. Pfoser, C. S. Jensen and Y. Theodoridis, “Novel Approaches in Query Processing for Moving Object Trajectories”, Proceedings of the Intl. Conf. on Very Large Data Bases, VLDB, **(2000)**.
[10] V. P. Chakka, A. Everspaugh and J. M. Patel, “Indexing Large Trajectory Data Sets with SETI”, Proceedings of the Conf. on Innovative Data Systems Research, **(2003)**; Asilomar, CA, Janpan.
[11] M. Abdelguerfi, J. Givaudan, K. Shaw and R. Ladner, “The 2-3 TR-tree, A Trajectory-Oriented Index Structure for Fully Evolving Valid-time Spatio-temporal Datasets”, Proceedings of the ACM workshop on Adv. in Geographic Info. Sys., ACM GIS, **(2002)**.
[12] M. Lee, W. Hsu, C. Jensen, B. Cui and K. Teo, “Supporting Frequent Updates in R-Trees: A Bottom-Up Approach”, Proceedings of the Intl. Conf. on Very Large Data Bases, VLDB, **(2003)**.
[13] Y. Tao, D. Papadias and J. Sun, “The TPR*-Tree: An Optimized Spatio-temporal Access Method for Predictive Queries”, Proceedings of the Intl. Conf. on Very Large Data Bases, VLDB, **(2003)**.
[14] C. M. Procopiuc, P. K. Agarwal and S. Har-Peled, “STAR-Tree: An Efficient Self-Adjusting Index for Moving Objects”, Proceedings of the Workshop on Alg. Eng. and Experimentation, ALENEX, **(2001)**, pp. 178-193.
[15] S. Saltenis and C. S. Jensen, “Indexing of Moving Objects for Location-Based Services”, Proceedings of the Intl. Conf. on Data Engineering, **(2002)**.
[16] A. Mendelzon and A. Vaisman, “Indexing temporal XML documents”, Proceedings of the 30rd VLDB Conference, **(2004)**; Toronto, Canada.
[17] F. S. Wang and C. Zaniolo, “Publishing and querying the histories of archived relational databases in XML”, Proceedings of the Fourth International Conference on Web Information Systems Engineering, **(2003)**; California, U.S.A.
[18] F. Rizzolo and A. Vaisman, The VLDB Journal, vol. 17, no. 5, **(2008)**, pp. 1179-1212.
[19] X. P. Ye, K. Y. Chen and Y. Tang, Chin. J of Comp., vol. 30, no. 7, **(2007)**, pp. 1074-1085.
[20] D. Pfoser, C. S. Jensen and Y. Theodoridis, “Novel Approaches in Query Processing for Moving Object Trajectories, “Proceedings of the Intl. Conf. on Very Large Data Bases”, VLDB, **(2000)**.
[21] X. P. Ye, H. Guo, Y. Tang, L. W. Chen, C. Zhou and Q. Y. Liao, Chin. J of Comp., vol. 34, **(2011)**, no. 2, pp. 256-274.
[23] X. P. Ye, Y. Tang, L. W. Chen, H. Guo, J. Zhu and K. Y. Chen, Sci. in Chin. (E), vol. 52, no. 6, **(2009)**, pp. 899-913.
[24] M. Nascimento and M. Dunham, “Indexing Valid Time Database via B+-Tree”, The MAP21 Approch, Technical Report CSE-97-08, Dallas, USA: School of Engineering and Applied Sciences, Southern Methodist University, **(1997)**.

Copyright ⓒ 2015 SERSC