Migration Strategies for Temporal Data based on Time-Segmented Storage Structure

Yun, Hongwon

School of Computer Engineering, Silla University, Sasang-gu, Pusan, 617-736 Korea
Tel: +82-51-999-5065, Fax: +82-51-999-5652
Email: hwyun@silla.ac.kr

Abstract – Research interests on temporal data have been almost focused on data models. There has been relatively less research in the area of temporal data management. In this paper, we propose two data migration strategies based on time-segmented storage structure: the migration strategy by Time Granularity, the migration strategy by LST-GET. We describe the criterion for data migration and moving process. We simulated the performance of the migration strategy by Time Granularity in order to compare it with non-segmentation method. We compared and analyzed two data migration strategies for temporal data.

1. INTRODUCTION

A great deal of work has been done in recent years in the field of Temporal Databases [1, 2, 3, 4]. Due to this effort, a large infrastructure, namely data model, query languages, index structures, etc., has been developed for the management of data evolving in time. However, there has been relatively less research in the area of temporal data management that is to say storage structure, data migration. Several storage structures, such as reverse chaining, accession lists, clustering, and cellular chaining are proposed in [5]. In these storage structures open and closed versions are maintained in two separate partitioned storage areas: the current store and the history store. Only the open version of an object is stored in the current store, whereas closed (past) versions are stored in the history store.

Sarda proposed that Historical DBMS divide a historical relation into three segments: history, current, and future. This segmentation is transparent to the user. Some tuples may move from current to past at this time. The concepts of a key for a historical relation can be helpful for relating tuples across the segments [6].

Traditional relational database does not take into account the special characteristic of temporal data, such as past data, current data, and future data. This conventional data management not required tuples moving among the segments. In this paper we propose two data migration strategies based on time-segmented storage structure related to temporal database.

The rest of this paper is organized as follows. Section 2 presents related work on temporal storage structure. In section 3 we describe two data migration strategies based on time-segmented storage structure. Section 4 presents the simulation model and the experimental results of our simulation. Finally, Section 5 will present conclusion and direction for future research.

2. RREVIOUS WORK

Many storage structures have been proposed, including reverse chaining, clustering, accession lists, stacking, and cellular chaining. The temporally partitioned storage structure has two storage areas, the current store and history store. The current store contains current versions which can satisfy all non-temporal queries, and possibly some of frequently accessed history versions. The history store holds the remaining history versions. Separating current data from the bulk of history data can minimize the overhead for conventional non-temporal queries, and at the same time provide a fast access path for temporal queries [5]. Ahn proposed several storage structures, however not discussed future segment in temporal data.

Sarda proposed HDBMS divide a historical relation into three segments: HISTORY, CURRENT and FUTURE, and DBMS that handle time must handle past, present, future data [6]. Some values may move from future segment to current segment when now reaches the earliest *from* time in the future segment. In this scheme, the concept of a key for a historical relation with real world time can be helpful for relating tuples across the segments. An advantage with this approach is simplicity. A problem with this approach is when now reaches the earliest from time in the future segment or the latest *to*

time in the current segment, relating tuples may move from future segment to current segment or from current segment to past segment. Data moving among segments occur at a granularity level (second, minute, and hour). This work is mainly concerned with granularity level as data is moved to other segment. Sarda introduced the concept of the temporally-partitioned store and briefly examined data moving.

Kouramajian proposed an archiving system that maintains temporal data on magnetic/optical disks. In this system, most of the historical data are maintained on optical media, whereas some recent history data and current data are maintained on magnetic media [7]. An archiving system have been described, however not discussed future segment in temporal data. In this paper we propose two data migration strategies based on time-segmented storage structure (past segment, current segment, and future segment).

3. DATA MIGRATION STRATEGIES FOR TEMPORAL DATA

In this section we propose two data migration strategies based on time-segmented storage structure. The symbols used throughout the section are given below:

- *PCB* (Past Current Bound) : The time point on time line divide a relation into two segments: past segment and current segment.
- *CFB* (Current Future Bound) : The time point on time line divide a relation into two segments: current segment and future segment.
- E_{ij} : j th version of entity version E_i.
- $E_{ij}.A$: all attribute of entity version E_{ij}.
- $E_{ij}.V_s$: valid start time of entity version E_{ij}.
- $E_{ij}.V_e$: valid end time of entity version E_{ij}.

Fig. 1. shows a entity versions to be moving or copy between the two segments.

Data moving and copy	Entity version
Future → Current (moving)	$(E_{ij}.V_s \geq PCB \wedge E_{ij}.V_e < CFB)$
Current → Past (moving)	$(E_{ij}.V_e < PCB)$
Future → Current (copy)	$(PCB \leq E_{ij}.V_s < CFB) \wedge (E_{ij}.V_e > CFB)$
Current → Past (copy)	$(E_{ij}.V_s < PCB) \wedge (PCB < E_{ij}.V_e < CFB)$

Fig. 1. Entity versions for moving or copy

Fig. 2. shows the dividing criterion based on now to separate temporal data.

	$E_{ij}.V_s \leq now$	$E_{ij}.V_e = now$	$E_{ij}.V_e > now$
$E_{ij}.V_s \leq now$	past data	past data	current data
$E_{ij}.V_e = now$	×	×	current data
$E_{ij}.V_e > now$	×	×	future data

Fig. 2. Dividing criterion of migration strategy by time granularity.

In the migration strategy by time granularity, entity versions are stored as follows:

- Entity versions are stored in the past segment: $= \{ E_{ij} \mid E_{ij}.V_e \leq now \}$
- Entity versions are stored in the current segment: $= \{ E_{ij} \mid E_{ij}.V_s \leq now < E_{ij}.V_e \}$
- Entity versions are stored in the future segment: $= \{ E_{ij} \mid E_{ij}.V_s > now \}$

In the migration strategy by LST-GET, time point LST is selected based on the least valid start time of current versions, to be the dividing criterion between past segment and current segment. Time point GET is the same as LST. The values of LST and GET are defined as:

$$LST = \min \{ E_{ij}.V_s \mid E_{ij}.V_s \leq now < E_{ij}.V_e \}$$
$$GET = \max \{ E_{ij}.V_e \mid E_{ij}.V_s \leq now < E_{ij}.V_e \}$$

Dividing criterion	Segment
$E_{ij}.V_e < LST$	Past segment
$(E_{ij}.V_s \geq LST) \wedge (LST < E_{ij}.V_e < GET)$	Current segment
$E_{ij}.V_s \geq GET$	Future segment
$(E_{ij}.V_s < LST) \wedge (LST < E_{ij}.V_e < GET)$	Past, Current segment
$(LST \leq E_{ij}.V_s < GET) \wedge (E_{ij}.V_e > GET)$	Current, Future segment

Fig. 3. Storage segments of entity versions by LST and GET

An entity version must be stored in a segment, given the dividing criterions in Fig. 3.

Migration	Entity versions
Future segment → Current segment	$(E_{ij}.V_s \geq LST) \wedge (E_{ij}.V_e < GET)$
Current segment → Past segment	$E_{ij}.V_e < LST$

Fig. 4. Entity versions of moving object

Migration	Entity versions
Future segment → Current segment	$(LST \leq E_{ij}.V_s < GET) \wedge (E_{ij}.V_e > GET)$
Current segment → Past segment	$(E_{ij}.V_s < LST) \wedge (E_{ij}.V_e < GET)$

Fig. 5. Entity versions of copy object

Fig. 4. and Fig. 5 show the entity versions must be moved and copied using the migration strategy by LST-GET.

4. PERFORMANCE EVALUATION

In this section, we simulated the performance of the time-segmented structure in order to compare it with non-segmentation structure for temporal data. We present a performance evaluation of the migration strategies discussed in section 3. The parameters we used for all experiments are shown in Fig. 6 and Fig. 7.

Parameter	Value
Tuple lifespan	30, 31, ... , 50
Tuple lifespan of LLT	300, 310, ..., 500
Relation lifespan	0 ~ 10,000
Rate of LLT	0, 1, 3, 5, 7, 9 %
Number of data	100,000 개

Fig. 6. Parameters and values for simulation

Parameter	Value
Average Seek, Read/Write (msec)	7.7 / 8.7
Track-To-Track Seek, Read/Write (msec)	0.98 / 1.24
Average Latency (msec)	2.99
Transfer Rate (Bytes/sec)	16,777,216
Track Size (Bytes)	115,078

Fig. 7. Parameters and values of hard disk used in simulation

The main queries put to study include temporal point query, temporal range query. A temporal point query was assumed to read only one segment in three segments. A temporal range query was assumed to read one, or two, or three segments. A temporal database was assumed to be implemented with a record-based storage system which supports tuple versioning.

Fig. 8. Shows the average response time of two data migration strategies at second granularity level with different data moving rate: 10, 20, 30, 40, and 50%. The main observation from the graph is that non-segmentation has the best average response time. This is because in data moving to other segments impose very large overhead, whereas non-segmentation not requires data moving. Fig. 9. Shows the average response time of two data migration strategies at minute granularity level with different data moving rate.

Fig. 10. shows the average response time of two data migration strategies without long lived tuples. Fig. 11. shows the average response time of two migration strategies for temporal queries with different query rate and case in exist long lived tuples.

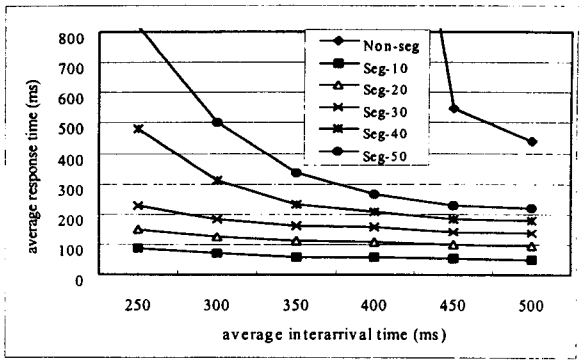

Fig. 8. Average response time at second granularity level for temporal queries.

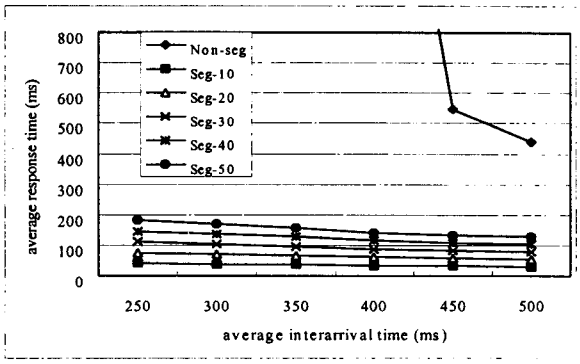

Fig. 9. Average response time at minute granularity level for temporal queries.

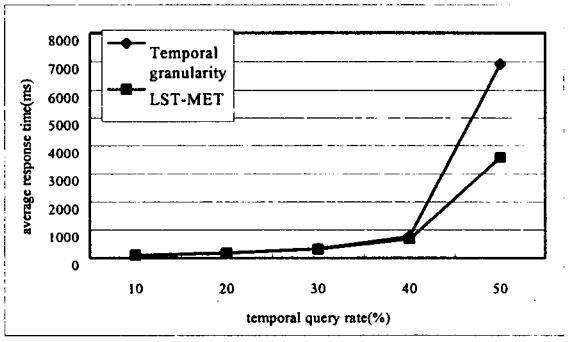

Fig. 10. Average response time for two migration strategies case in not exist LLT

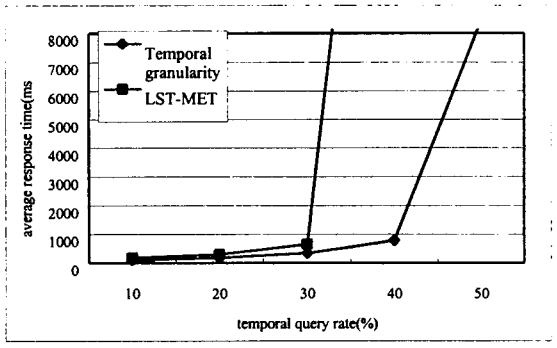

Fig. 11. Average response time for two migration strategies case in exist LLT

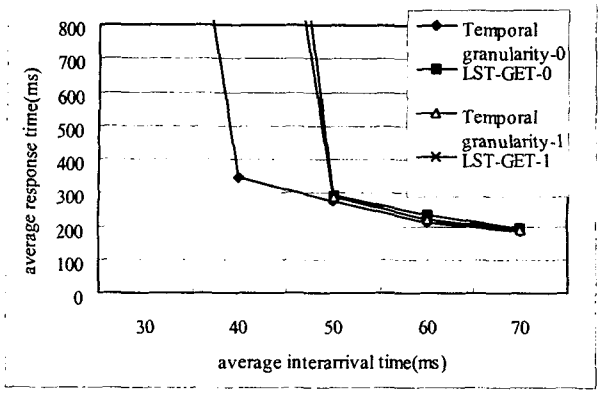

Fig. 12. Average response time for changing average interarrival time

Fig. 12. shows the response time of two migration strategies for temporal queries with different average interarrival times. Notice that the performance as interarrival time of temporal queries decrease, the performance gap between time granularity in case not exist LLT and LST-GET in case exist LLT increase.

5. SUMMARY

Numerous proposals for extending the relational data model as well as conceptual and object-oriented data models have been suggested. However, there has been relatively less research in the area of time-segmented storage structure and data migration strategies for temporal data. This paper presents the segmented storage structure in order to increment search performance and the two data migration strategies for segmented storage structure.

This paper presents the two data migration strategies: the migration strategy by Time Granularity, the migration strategy by LST-GET. In the migration strategy by Time Granularity, the dividing time point to assign the entity versions to the past segment, the current segment, and future segment is described. In the migration strategy by LST-GET, we describe the process how to get the value of dividing criterion. Searching and moving processes are described for migration on the future segment and the current segment and entity versions to assign on each segment are presented. We simulate the search performance of the segmented storage structure in order to compare it with conventional storage structure in relational database system. And extensive simulation studies are performed in order to compare the search performance of the migration strategies with the segmented storage structure.

REFERENCES

[1] M.D. Soo, "Bibliography on Temporal Databases," ACM SIGMOD Record, Vol. 20, N. 1, 1991.

[2] N. Kline, "An Update of the Temporal Database Bibliograhpy," ACM SIGMOD Record, Vol. 22, N. 4, 1993.

[3] V.J. Tsostras, A. Kumar, "Temporal Database Bibliograhpy," ACM SIGMOD Record, Vol. 25, N. 1, 1996.

[4] C.S. Jensen, et al., "A Consensus Glossary of Temporal Database Concepts-February 1998 Version," in O. Etzion, et al., Temporal Databases-Research and Practice, LNCS N. 1399, Springer-Verlag, 1998.

[5] I. Ahn, R. Snodgrass, " Partitioned storage for temporal databases, In Information Systems," Vol. 13, No. 4, 1988.

[6] N.L. Sarda., "Time-Grid: A file structure for historical databases," Technical report, Indian Institute of Technology, April 1992.

[7] V. Kouramajian, "Temporal Databases: Access Structures, Search Methods, Migration Strategies, and Declustering Techniques," Ph. D. Dissertation, The University of Texas at Arlington, 1994.

[8] Ahn, I. "Towards an implementation of database management systems with temporal support," In IEEE International Conference on Data Engineering, Feb., 1986.

[9] Elmasri, R., Jaseemuddin, M., and Kouramajian, V. "Partinioning of time index for optical disks", In IEEE International Conference on Data Engineering, Feb. 1992.

[10] Jensen, C. et al., "A consensus glossary of temporal database concepts", In ACM SIGMOD Record, 23(1), Mar. 1994

[11] A Tensel et al., Temporal Databases: Theory, Design, Implementation, Benjamin/Cummings, 1993.