스냅샷 데이터를 갖는 다중 레벨 저장 시스템에서의 효율적인 리프레시 기법

주붕*, 어상훈*, 김명근*, 조숙경**, 배해영*
*인하대학교 컴퓨터 · 정보공학과
**인천대학교 컴퓨터공학과
e-mail : cqzhoupeng@hotmail.com

An Efficient Refresh Method in Multi-Level Storage System with Snapshot Data

Peng-Zhou*, Sang-Hun Eo*, Myoung-Keum Kim*, Sook-Kyoung Cho**, Hae-Young Bae*
*Dept. of Computer Science & Engineering, Inha University
** Dept. of Computer Science & Engineering, Incheon University

Abstract

In multi-level storage system with snapshot data, some snapshots which are from selection portions of the base tables are kept in main memory. So how to efficiently refresh snapshots in response to changes on their base tables for preserving consistency which requires snapshots reflect the current state of the base tables referenced by the snapshot query is a very import research issue. In this paper, a method for efficiently refreshing snapshots is proposed. In this method, it uses a data structure to store metadata which contains some necessary information of every snapshot and an updating log that records the history of changes on its base tables. Synchronization process scans the metadata and refreshing process is executed using appropriate logs after it finds anyone of the snapshot need to be refreshed.

1. Introduction

Recently, spatial data has emerged as central to many applications, such as graphic information system, computer-aided design, image processing, etc. all of which have massive core spatial data that must be stored and queried, and displayed. In these applications, some especial parts of massive data need to be efficiently handled with faster response time. In order to fulfill that demand, multi-level storage system with snapshot data is required [3].

In multi–level storage system with snapshot data, some snapshots which are from frequently used parts of the base tables are kept in main memory [1]. However, snapshots need to be refreshed periodically in response to changes on their base tables to preserve consistency which requires snapshots reflect the current state of the base tables referenced by the snapshot query [6, 8]. So how to efficiently refresh snapshots has become a vital research issue in field of the multi-level storage system with snapshot data [2, 5, 8, 10].

In this paper[1)], a method for efficiently refreshing snapshots is proposed. First, a data structure to store metadata which contains some necessary information of every snapshot for refreshing is designed. Second, the structure of updating log which records the history of the changes on base tables of every snapshot is also proposed. Using this log, system can refresh corresponding snapshots to a consistent state according to their base table's changes. Synchronization process scans the metadata and refreshing process is executed using appropriate logs after it finds anyone of the snapshots need to be refreshed.

In the remainder of this paper is organized as follow. We begin with a brief introduction of the snapshot and an overview of the architecture of the multi-level storage system with snapshot in section 2. In section 3, first the structure of the metadata table and updating log is presented, second synchronization process how to efficiently refresh snapshots using metadata table and updating log is explained. Our conclusion and future work are given at last.

2. Related Work

In this section, the concept of snapshot in database and the architecture of the multi-level storage with snapshot data are

1) This research was supported by the MIC (Ministry of Information), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Assessment).

introduced.

2.1 Snapshot in Database

As we known, in a relational database system, a database may be composed of both base and derived relations. A derived relation or view is defined by a relational expression i.e., a query evaluated over the base tables. A derived relation may be virtual, which corresponds to the traditional concept of view, or materialized, which means that the resulting relation is actually stored and corresponds to the concept of materialized view or snapshot [1]. In the remainder of this paper, that kind of view is called snapshot instead of materialized view. In the following Figure 1, it gives out an example of snapshot. As Figure 1 shown, snapshot can provides fast access to some selection portions of the database with faster response time.

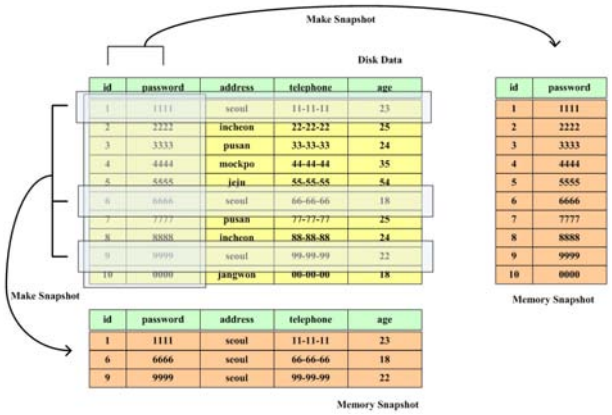

Figure 1: An Example of Snapshot

2.2 Architecture of Multi-Level Storage System with Snapshot Data

A multi-level storage system with snapshot data consists of main three components, namely Disk Storage Manager, Memory Storage Manager and United Query Processor. In order to avoid re-construct the whole of the system, memory storage manager is added to existing disk storage manager and existing query processor is also extended to control both disk data and memory data [3,12].

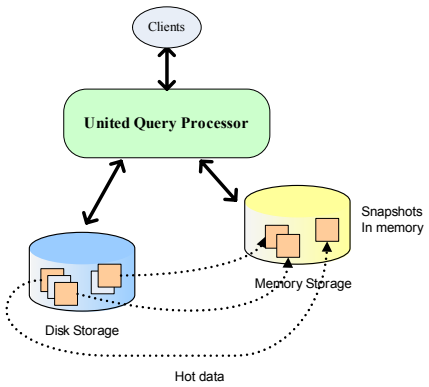

Figure 2: Architecture of Multi-level Storage System With Snapshot Data

As shown in Figure 2, to speed up the query process in multi-level storage system, some snapshots which are from frequently used parts of the base tables are kept in main memory. In that system, a snapshot is a read-only table whose value is defined by over one or more base tables, so it must be periodically refreshed to preserve consistency which requires its value to reflect the current state of the base tables referenced by the snapshot query.

In the remainder of this paper, to simplify description, we assume that every snapshot is derived from a single base table and it is created only by selection and projection operation on base table.

3. A Refresh Method of Snapshot

In this section, the structure to store metadata and the updating log using in refreshing process are presented, also synchronization process how to use metadata and updating log to efficiently refresh snapshots in main memory is showed.

3.1 Data Structure of Metadata

To efficiently refresh snapshots in multi-level storage system with snapshot data, system must store some necessary information of every snapshot, which is useful during the refreshing process, at the same time when the snapshot is created in main memory. That information which contains “data about the snapshot” is called metadata in the remainder of this paper [11].

A table structure to store metadata of every snapshot is proposed in the following Figure 3.

SN	TN	Refresh Time	TimeStamp	SQL Definition

Figure 3: Structure of Metadata Table

In the metadata table, every tuple corresponds to one existing snapshot in main memory and it includes five entries. These are:

SN: The name of a snapshot. It is unique for every snapshot.

TN: The names of base table referenced by a snapshot query. In this paper, only the case that snapshot is derived from a single base table is considered.

Refresh Time: The refresh time of a snapshot, such as every month, every day and so on. It is defined by user when it is created in main memory.

TimeStamp: The last refresh time of a snapshot.

SQL Definition: The SQL definition of a snapshot, it gives out the restriction of base table which defines the contents of a snapshot.

The relevant information of a snapshot is written into the metadata table tuple at the same time when it is created in main memory. In the following of this section about synchronization process how to use the information stored in metadata table to efficiently refresh every existing snapshot will be showed

3.2 The Updating Log

As the database changes because of updates applied to the base tables, snapshots need to be refreshed to preserve

consistency. A snapshot can always be brought up to data by re-evaluating the relational expression that defines it. That is the simplest way to refresh snapshots to a consistent state according to their base relation's changes. However, complete re-evaluation is often wasteful, and the cost involved may be unacceptable [7, 8, 10].

To efficiently refresh snapshots, a log which records the history of changes on the base tables, named updating log, is used. The updating log records the changes of every base table referenced by the snapshot query during the refresh interval. So refreshing process can only compute the changes in the base tables to update snapshots to a consistent state, namely refresh only a parts of the snapshot changes in response to changes to the base tables in the refreshing process. [2, 4, 9]

The structure of the updating log is proposed in the following Figure 4:

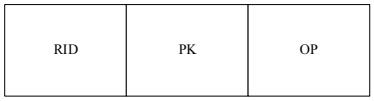

Figure 4: Structure of Updating Log

Every snapshot has its updating log to recode changes happened on base tables referenced by it. Every record of updating log contains three entries. These are:

RID: The record ID of tuple which is updated on base table.

PK: The primary key of that updated tuple on base table.

OP: Operation Type happened on updated tuple, namely Insert, Delete, or Update.

To use that updating log, every snapshot must be extended to include an extra attribute containing the primary key of its base table if its definition does not include corresponding primary key of base table. For example, as the following Figure 5 shown, because snapshot 2 is created by projecting some base table's attribute, but its attributes do not include primary key of its base table, so an extra attribute, namely PK, is added to snapshot 2.

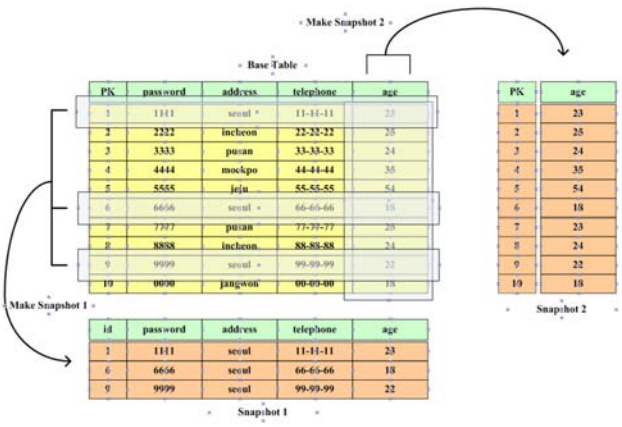

Figure 5: Snapshot Must Include PK

Because every snapshot's tuple exists with its relevant primary key in main memory, so a tuple, which needs to be refreshed, can be easily found according to its PK recorded in updating log during the refreshing process.

3.3 The Refreshing Process

In this part, how to use a synchronization process combined with the metadata table and updating log to efficiently refresh snapshot is showed.

Because the refresh time of every snapshot is different. To synchronize the refreshing process of every snapshot, a synchronization process which controls the whole of refreshing procedure is used. The main function of the synchronization process is to scans the metadata table and executes the refreshing process using appropriate updating logs after it finds anyone of the snapshots need to be refreshed.

The refreshing process is illustrated in the following Figure 6.

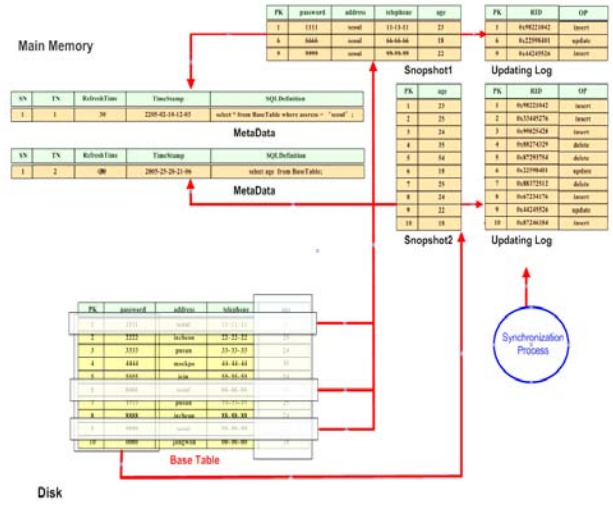

Figure 6: The Refreshing Process

As Figure 6 shown, the refreshing process includes following main point:

1) The information about every snapshot should be written into metadata table at the same time when it is created in main memory by synchronization process. For example, in Figure 6, the metadata of snapshot 1 and snapshot 2 is recorded into metadata table by synchronization process when it is created in main memory by user.

2) The synchronization process scans the metadata table to get every snapshot's refresh time. As we known, the refresh time of every snapshot and its last refresh time are stored as Refresh Time and TimeStamp field in metadata table, respectively. So if TimeStamp subtracting from clock is greater than Refresh Time, the corresponding refreshing process will be executed by synchronization process

3) The refreshing process of every snapshot will scan the relevant updating log. According to the value of OP field in the every record of updating log, there are three cases which need to be processed differently:

i) If the value of OP field is Delete, that means a tuple is

deleted in base table, so according to PK of that record, refreshing process finds corresponding tuple in snapshot and deletes it.

ii) If the value of OP filed is Update, that means the corresponding tuple in base table is updated. According to the RID and PK of that record, refreshing process get the new value of that tuple in base table and the corresponding tuple in snapshot, respectively. Then refreshing process refreshes corresponding tuple in snapshot.

iii) If the value of OP field is Insert, that means a new tuple is inserted in base table. According RID of that record, refreshing process gets the value of that new tuple, Then it insert corresponding tuple in snapshot.

4) The history of changes on base table is recorded in appropriate updating log of every snapshot during its refresh interval by synchronization process.

5): The synchronization process will update the value of TimeStamp field in metadata table after last refresh time of corresponding snapshot is changed and relevant metadata also will be deleted by synchronization process after corresponding snapshot's life is over in main memory.

6) After one refreshing process of a snapshot is over, there is no need to store record which written time is prior to TimeStamp in corresponding updating log, and in fact the record of log before that time can be deleted or overwritten safely by synchronization process.

As the above shown, synchronization process plays an important role during the refreshing process. Using that way, it is expected not only preserving the consistency between snapshot and base table, but also getting a better performance of multi-level storage system with snapshot data.

We stress that the above is only a heuristic. For instance, if an entire base table is deleted, it may be cheaper to recompute a snapshot that depends on the deleted table (if the new snapshot will quickly evaluate to an empty table) than to compute the changes to the view. So, some special cases that need to be processed differently, it must be considered in multi-level storage system with snapshot data.

4. Conclusion and Future Work

A method for preserving consistency between snapshots in main memory and base tables in disk which requires snapshots reflect the current state of the base tables referenced by the snapshot query in multi-level storage system with snapshot data is presented. The method consists of three parts. First, necessary information of every snapshot for refreshing is stored in metadata table. A data structure to store that information is proposed. Second, the updating log is used during the refreshing process. At last, a synchronization process is designed to synchronize refreshing process.

A simulation system based on a multi-level system storage system with snapshot data will be implemented and some relevant experiments will be done in the future.

References

[1] Adiba, Michel and B.G. Lindsay, "Database Snapshots", In Proceedings of the 6th International Conference on Very Large Database, pages 86-91, June 1980.

[2] J.A. Blakeley, P.A. Larson, and E.W. Tompa, "Efficiently Updating Materialized Views", In Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 61-71, Washington, D.C., June 1986

[3] M. Stonebraker, "Managing Persistent Objects in a Multi-Level Store", Electronics Research Laboratory Memorandum, M/91/72, University of California, Berkeley, March 1992.

[4] S. Ceri and J. Widom, "Deriving Production Rules for Incremental View Maintenance", In Proceedings of the Seventeenth International Conference on Very Large Data Bases, pages 577-589, Barcelona, Spain, September 1991.

[5] B. Lindsay, L. Haas, C. Mohan, H. Pirahesh, and P. Wilms, "A Snapshot Differential Refresh Algorithm", In Proceedings of the ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 1986

[6] A. Gupta, I, S. Mumick, and K.A. Ross, "Adapting Materialized Views after Redefinitions", In Columbia University TR CUCS-010-95, pages 211-222, March 1995.

[7] J.A. Blakeley and F.W. Tompa, "Maintaining Materialized Views without Accessing Base Table", In Information Systems, 13(4): pages 393-406, 1988.

[8] A. Gupta, I.S. Mumick, S. Jose, and M. Hill, "Maintenance of Materialized Views: Problems, Techniques, and Applications", Data Eng. Bulletin, Vol 18,No 2, June 1995.

[9] A. Segev and J. Park, "Updating Distributed Materialized Views", IEEE Transaction on Knowledge and Data Engineering, 1(2): pages 173-184, June 1989.

[10] O. Shmueli and A. Itaj, "Maintenance of Views", In Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 240-255, Boston, Massachusetts, May 1984.

[11] J-P. Kent, M. Schuerhoff, and S. Netherlands, "Some Thoughts about a Metadata Management System", In Proceedings of the 9th International Conference on Science and Statistical Database Management, pages 174-185, Olympia Washington.

[12] E. Morenoff and J. B. McLean, "Application of Level Changing to a Multilevel Storage Organization", Communication of ACM 10, pages 149-154, March 1967.