DOI : 10.3745/JIPS.2008.4.1.009

Journal of Information Processing Systems, Vol.4, No.1, March 2008 **9**

Two-Tier Storage DBMS for High-Performance Query Processing

Sang-Hun Eo*, Yan Li*, Ho-Seok Kim, and Hae-Young Bae***

Abstract: This paper describes the design and implementation of a two-tier DBMS for handling massive data and providing faster response time. In the present day, the main requirements of DBMS are figured out using two aspects. The first is handling large amounts of data. And the second is providing fast response time. But in fact, Traditional DBMS cannot fulfill both the requirements. The disk-oriented DBMS can handle massive data but the response time is relatively slower than the memory-resident DBMS. On the other hand, the memory-resident DBMS can provide fast response time but they have original restrictions of database size. In this paper, to meet the requirements of handling large volumes of data and providing fast response time, a two-tier DBMS is proposed. The cold-data which does not require fast response times are managed by disk storage manager, and the hot-data which require fast response time among the large volumes of data are handled by memory storage manager as snapshots. As a result, the proposed system performs significantly better than disk-oriented DBMS with an added advantage to manage massive data at the same time.

Keywords: *DBMS, Storage Management, Query Processing*

1. Introduction

In order to provide fast response service in DBMS, many researchers have studied various techniques such as indexing techniques[1,2,3], buffer management techniques[4,5,6], materialized view techniques[7,8], etc. But these disk-oriented techniques couldn't support fast response time successfully. So those techniques are on the verge of being obsolete in today's hi-tech society. The arguments by some researchers stated that the whole database will soon fit in memory for certain applications because of the declining price of memory for their higher capacity counter-parts[9,15]. So there are various memory-oriented database techniques have been studied such as memory-oriented record format, page structures, indexing techniques and various memory-resident DBMSs have been developed[10,11,12,13]. They perform significantly better than disk-oriented DBMSs even when the disk-oriented DBMSs have all data in memory, because they have data structures and algorithms which are fitted only into the memory. However, memory-resident DBMSs have original restrictions of memory size. Such as video, image, and voice data, the amount of data, needed to be stored in databases, has grown at an even faster rate than memory size increment. Thus for the foreseeable future it is unlikely that the whole database will ever fit in memory, at least for large applications. In the present day, therefore, the main requirements of DBMS can be figured out using two aspects. The first is handling large amounts of data. And the second is providing fast response time. Both the disk-oriented DBMS and the memory-resident DBMS cannot fulfill these requirements. The disk-oriented DBMS can manage massive data but the response time is relatively slower than the memory-resident DBMS. On the other hand, the memory-resident DBMS can provide fast response time but it has original restrictions of database size.

These problems can be solved using a multi-level storage system[13,14]. This system stores data in several deferent levels. For fast response time, some parts of the data are stored in high-speed storage device like main memory, and for massive data secondary devices such as disk storage, tape storage, and etc can be used. And if it's necessary, network devices can also be put in the stack. There are no reasons that the entire data have to store in memory for fast response time. Because the data which require fast response time are only certain parts of the database, not the whole of database itself. Therefore, to provide fast response time to certain applications, which require fast response time for some parts of the database, moving the whole database from disk-oriented DBMS to memory-resident DBMS is not efficient. If the DBMS can stores the parts in high-speed storage devices, then the overall performance of the DBMS will improve. In this

Manuscript received January 2, 2008; revised February 18, 2008; accepted March 11, 2008.

Corresponding Author: Sang-Hun Eo

This research was supported by a grant(07KLSGC05) from Cutting-edge Urban Development - Korean Land Spatialization Research Project funded by Ministry of Construction & Transportation of Korean government.

* Dept. of Information Engineering, Inha University, Incheon, Korea (eosanghun@dblab.inha.ac.kr,leeyeon@dblab.inha.ac.kr, hybae@inha.ac.kr)

** LG Mobile Handset R&D Center of Mobile Communications Company, LG Electronics Inc, Seoul, Korea. (hskim94@lge.com)

paper, a two-tier DBMS is proposed. In the proposed system the hot-data which require fast response time are stored in memory storage as snapshots, and the cold-data which does not require fast response time or large volume of data are stored in disk storage.

The rest of the paper is organized as follows. Section 2 is related work. Section 3 presents the two-tier DBMS. Section 4 present preliminary experiment results. Section 5 concludes the paper and discusses future work.

2. Related Work

In this section, the reasons why memory-resident DBMS can provide more fast response time than disk-oriented DBMS even when the disk-oriented DBMS have all data cached in memory are described. In addition, the concept of a multi-level storage system is introduced.

2.1 Memory-resident DBMS vs. Disk-oriented DBMS

There are several reasons why memory-resident DBMS can provide more fast response time than disk-oriented DBMS even when the disk-oriented DBMS have all data cached in memory. Basically, in traditional disk-oriented DBMS, the whole database existed in disk. When the user requests happen, DBMS copy the request data into memory buffer first and then answers users after data manipulation. On the other hand, a memory-resident DBMS manages the whole database in memory. So, the data, which are needed for manipulating, are simply referenced by memory pointers.

Another reason is database indexing technique. Generally, traditional disk-oriented DBMS uses B-tree index, which focuses on the disk access efficiency. This technique is used for reducing disk I/O cost. While using B-tree index, the tree balance operation consumption which happened between each nodes should be considered. However, usually memory-resident DBMS uses T-tree index technique[13,15,16]. The purpose of the T-tree index is to reduce the calculation time and minimize the amount of memory usage. This technique is optimized for reducing the cost of rotation which is happened to balance the index tree. Thus, data search and structure modification operation is optimized in memory-resident DBMS. In addition, the cost of accessing data is relatively cheap because all data are resident in memory. This is because the cost of changing RID (Record ID) to memory pointer isn't necessary any more.

As mentioned above, the difference between data handling mechanism and memory oriented indexing technique shows that memory-resident DBMS can provide fast response time to users obviously even when the disk-oriented DBMS have all data cached in memory. But if the amount of database is extremely huge, it's not an advantage any more because the DBMS cannot load the whole database at all.

2.2 A Multi-Level Storage System

A multi-level storage system was designed for managing very large object. This kind of system reside time critical objects in main memory, other objects are stored disk resident, and the remainder managed in occupy tertiary memory[14].

Location	Main memory database	Disk database	Archive database
Main memory	Main memory objects In main memory format(M)	Cache of disk blocks	Cache of archive blocks
disk		Disk objects In disk format (D)	Cache of archive blocks
archive			Archive object In archive format(A)

Fig. 1. Logical model of a three level store

It is possible that the system has more than three levels, and some of these levels can be on remote hardware. The system can be divided by a collection of L logical devices that form a rooted tree. Hence there is a unique root, called main memory, with zero or more direct descendant devices, each of which can have zero or more descendants. Moreover, these L devices can be on various computer systems in a network. The system must be able to address the needs of the following tow kinds of applications. The first kind is real time application. These applications are needed sub-millisecond response times for requests to a main memory database along with conventional response times to disk based data. The second kind applications are need to manage massive databases, which are needed conventional response times to disk based data and reasonable response times to archival data.

3. Two-Tier Storage DBMS

In this section, we describe architectural design issues and propose the two-tier DBMS. In addition, how to interoperate the memory storage manager and disk storage manager using snapshots is introduced.

3.1 Architectural Design Issues

Until now, almost enterprises already constructed business database are complex and the amount is huge. In this situation, re-constructing the whole database to get the more fast response time is not efficient because it requires so much time and efforts. In fact, not the whole database is required for fast response time. That means there only some parts of database is needed fast response time.

While using traditional disk-resident DBMS, in order to get fast response time, the system automatically cache special data in buffer cache using LRU buffer change

strategy. This kind of methods can reduce the disk access time efficiently, but the cached by system's own strategy may not be the just required data for DBA and DB user. The DBA can not decide which kind of data should be cached in memory, although DBA has the information of hot accessed data.

On the other kinds of approach, as the current research, using two kinds of DBMS is the normal way to handle the whole massive database and to get the fast response time for certain data. One kind is a disk-oriented DBMS and the other is a memory-resident DBMS. In this environment, a disk-oriented DBMS manages the whole massive database and a memory-resident DBMS handles only some part of database, which is needed fast response time and duplicated from disk-oriented DBMS.

However, this kind of approach has some problems. First, it needs to prepare another DBMS (a disk-oriented DBMS or a memory-resident DBMS) to handle large volume data from disk-oriented DBMS and manage hot data from memory-resident DBMS to get the fast response time. Second, application programmers have to recognize where the disk-oriented DBMS and the memory-resident DBMS located. Third, application programmers have to keep replicate table on those two kinds of DBMSs. Fourth, they have to make synchronization module to preserve consistency between the disk table which is located in disk-oriented DBMS and memory table which is located in memory-resident DBMS. Last, Application programs, which are already made, have to be changed.

So in this paper, two-tier storage DBMS system is proposed which disk-oriented DBMS and memory-oriented DBMS are tightly combined. In this system, system manager can decide and design the importance of the data, and then decide the data archived style which includes disk table style based on disk-oriented DBMS and memory table style based on memory-oriented DBMS.

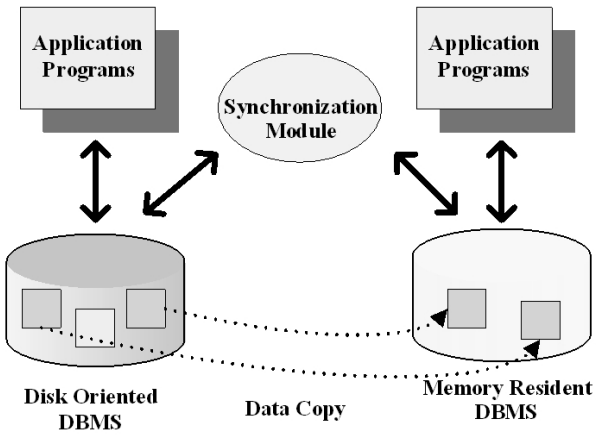


Fig. 2. The normal way to manage massive data with fast response time

3.2 The Proposed System Architecture

The proposed tow-tier DBMS consists of three major components. They are Disk Storage Manager, Memory Storage Manager, and United Query Processor. Basically, the whole database exists in disk storage. And some parts of the database, which is required to have fast response time, are duplicated in memory storage as snapshots for handling the data using memory-oriented techniques. A united query processor has been implemented by extending an existing query processor to control disk data as well as memory snapshots.

The united query processor takes user queries and analyzes the queries to decide whether the data is existed in snapshots, which are related to the queries, or whether the snapshots have all required records or not for answering the queries. If the required data exist in snapshots, which are related to the user queries, and the snapshots have full set of records, which are needed for answering the queries, then the united query processor gathers the required records with snapshots in memory, using memory-oriented techniques. But if there are no snapshots in memory related to the user queries or snapshots don't have full sets of records, which are needed for executing the queries, then the united query processor gathers records as snapshots using disk storage manager and memory storage manager.

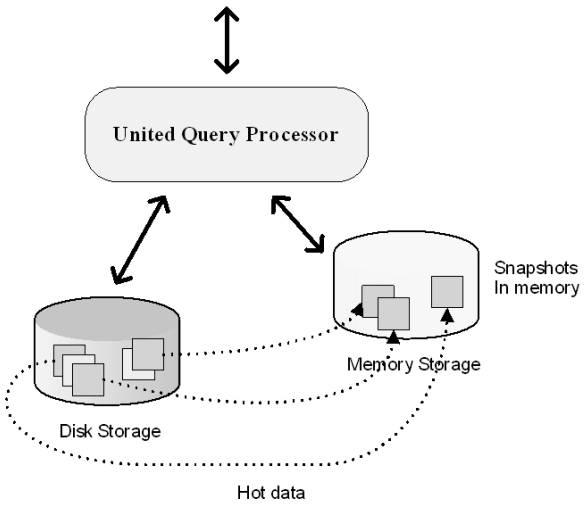


Fig. 3. The concept of a two-tier storage DBMS

3.3 Snapshots

The two-tier DBMS prepares data, which requires fast response time, from disk storage using access patterns, access frequency, a number of tables, period, etc and makes snapshots in memory storage by duplicating some parts of the disk data. When DBA find that a certain data table in disk-oriented DBMS is queried frequently, then DBA can cache the data table in memory as a snapshot in

order to provide quick answer in the future. User information table could be the example that should be cached on memory as a snapshot to make the user's login as fast as possible. On the other hand, when DBA find certain kind of data which have fixed access pattern, then DBA can decide to make snapshot for this kind of data periodically. For example, in the bank user account information system for bank web site, users usually want to get the information about he or she's recent transactions for one week, month or etc. In this case the bank DBMS DBA could make snapshot for the recent transaction information in memory-oriented DBMS.

Our prototype system stores historical query lists using background process. And it is periodically analyzed the query list to decide whether snapshots are needed. Snapshots can be made using projection, selection, and join conditions from disk tables.

For instance, in the web site users are required to login to access the web site, and there must be the table which includes login information such as user id and password. To decide whether the user has a right account to access the site, the web programmer uses SQL such as 'select id, password from users_table'. Considering that this kind of query is invoked frequently, the two-tier DBMS system creates snapshot with the related records dinamicly. After creating the snapshot, the two-tier DBMS system answers the query using snapshot with memory-oriented techniques. In this case, the snapshot is created by the projection of the disk table. In addition, snapshots can be made by selection of the disk table. If the query is 'select * from users_table where address = 'seoul'' then snapshot is created by selection of the users_table in disk storage. To avoid changing the exist application programs proposed system supports the creation of snapshots from views. Therefore, applications using the views don't need to be changed for using proposed system. Our prototype system already supports hybrid-queries (i.e. some of the required records exist in snapshot and rest of them are in disk storage). The united query processor classifies user queries into three types (i.e. memory-query, disk-query and hybrid query) and optimizes theses queries considering query types and the query classification overhead (i.e. actually, there is small overhead in united query processor but it can be ignored).

In the proposed system, the snapshot is made by DBA who can decide which kinds of data are accessed frequently should be cached as snapshot. So after the creation of the snapshot it will not be deleted very soon considering usability of the snapshot. But if the memory table is full, then the system uses LRU strategy to delete long run snapshot because of the limitation of memory space.

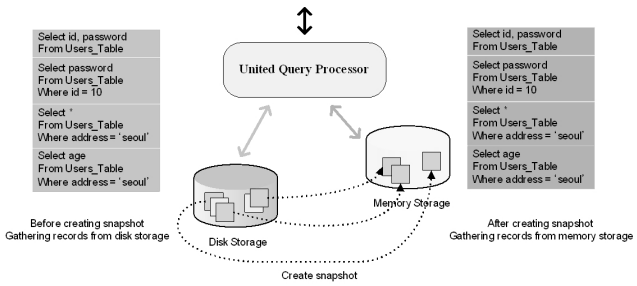


Fig. 4. Answering queries using memory snapshot. Existing application programs don't need to be changed for using the two-tier DBMS.

Through the previous performance test, we could figure out that if less than 10% of required record is in snapshot, then the proposed method could not perform well compare with disk only used DBMS. So search operation is faster while only using disk storage manager than using both memory storage manager and disk storage manager in this case. There are several ways to make snapshots in proposed system. For example, many kinds of snapshots can be made based on one single table, or one snapshot made by the two or more tables join process. While one table has 10 fields and 100 tuples, the snapshot may be made by only 5 fields or made by the user required 20 tuples. After the system got the user queries by using the proposed system, firstly the system analyzes queries and check the snapshots include the required data based on the table information, fields and tuples information. This is the addition cost of this proposed system. On the other hand, while using the disk-oriented DBMS to process the user queries, the system have to read the disk data into memory first. In this case, because of the system's loading strategy, the whole data could be loaded into memory while only 90% of the data is required. As a result, while using only 10% of the records cached in memory the query response time is little faster than the case while only using disk-oriented system. The experiment is showed in section 4.

4. Preliminary Experiment Results

To measure the performance benefits of the two-tier DBMS, a series of experiments are run using Wisconsin Benchmark database[17] and generated data sets. The experiments were performed on Solaris equipped with 8 CPUs of 1.2GHz and main memory of 4G bytes. The proposed system was implemented in C++ language. The Geomania Millenium Server[1][18] was extended as the two-tier DBMS.

[1] http://www.geomania.com

4.1 The Test Database and Query Set

As prescribed in the Wisconsin Benchmark Database, the test relation contains sixteen attributes – thirteen 4-byte integer attributes and three 52-byte string attributes. The relation has two candidate keys, unique1 and unique2, whose values range from 0 to 10,000.

Table 1. The Benchmark Queries

Query 1:	select * from tenktup1 where unique2D >= 8383 and unique2D <= 8482;
Query 2:	select * from tenktup1 where unique1D >= 8510 and unique1D <= 8609;

The two queries are chosen for obtaining results. Query 1 return 1% records of relation tenkup1 without using index. Query 2 selects 1% records of relation tenkup1 using non-clustered index.

And the generated data set contains sixteen attributes also – thirteen 4-byte integers attribute and three 52-byte strings attributes. The relation has one primary key, id whose values range from 0 to 1,000,000.

Table 2. The Test Queries

View 1:	create view view1pro as select * from large table where id >= 0 and id < 10,000;
Query 3:	select * from view1pro;
Query 4:	select * from view1pro where id = 5000;

View 1 was created as above definition. The view contains 10,000 records. For Query 3 and Query 4, performance evaluation has been conducted to compare the response time before creating the snapshot and after creating the snapshot.

4.2 Experiment Results

In the all experiment tests, the system extracts records using the disk storage manager first. Snapshots are created after the number of execution time is over than 300. After creation of snapshots, the system begins extracting records using the memory storage manager. The response times from the disk storage manager are almost identical with disk-oriented DBMS.

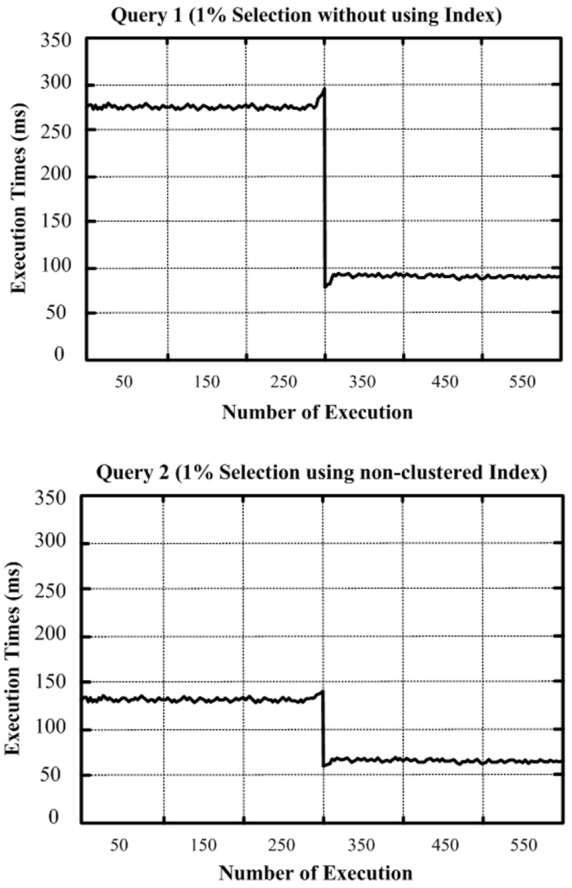


Fig. 5. Benefits of snapshots

Figure 5 depicts the benefits of snapshots when execute the Benchmark Queries. For each case of 1% of selection queries are executed without using index (i.e. Query 1) and 1% of selection queries are executed using index (i.e. Query 2). It can be studied that the second case's performance was significantly increased. The reason is that before creation of snapshots, the system must scan full record sets with disk-oriented techniques but after creation of snapshots, the system just needs to scan required records from the memory snapshots with memory-oriented techniques. As a result the queries execute time is reduced.

The results using generated large volume data sets are almost same (i.e. Query 3). Scanning 1,000,000 records spent so much time but after creation of snapshots the system only scanned 10,000 records from the snapshots. So the response time was extremely decreased. Query 4 selects one tuple. Before creation of snapshots, the system must scan full sets of 1,000,000 records but after creation of snapshots, the system just scanned one required record from the snapshots. So, the performance was increased remarkably. Another important experimental result is that constructing index on 1,000,000 records for query 4 spent about 2 minutes, but the spent time of creating snapshot for query 4 was just several seconds. That means proposed

system performs better than disk-oriented DBMS even when the disk oriented DBMS uses index.

Figure 7 shows the benefits of using caching methods. In this experiment, the number of whole records is increased from 10,000 to 100,000 while the records caching rates are changed from 0% to 90%. So the line TQ_HB(10%) means the 10% of the records that required by one application is cached in memory table. The first point of every test case shows the search time while 1,000 of 10,000 records are cached in memory table. While the caching rates are rose, the response time became short. But while the caching rates is 10% the search queries performance shows lower than the queries performance while only using disk table(TQ_DK). This is because the proposed method also needs cost to scheduling the queries, and the cost is more than the benefit of caching 10% records in memory table.

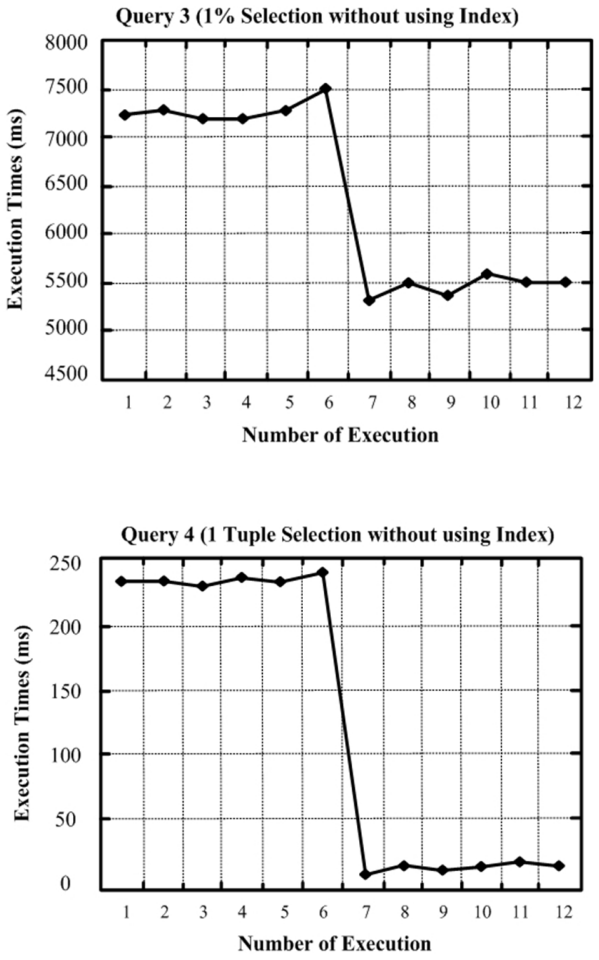


Fig. 6. Result of using generated large volume data sets

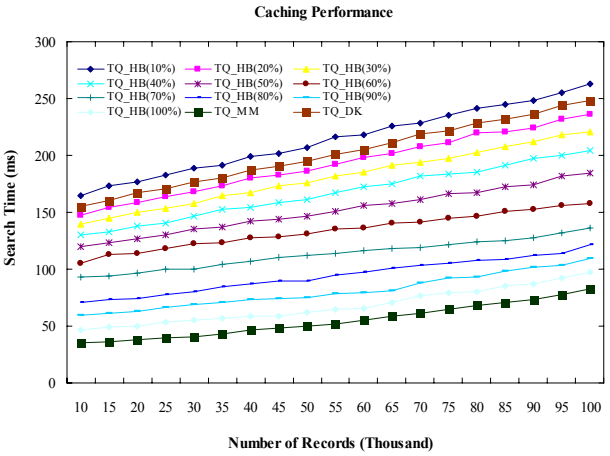


Fig. 7. Search operation with different caching rate

Figure 8 shows the benefit of using snapshot methods compare with using index methods in disk. In this experiment, the search queries are executed by using index methods and snapshot methods.

The number of records is increased from 10,000 to 100,000 during the whole experiment. And the search execution time is checked every 5000 records increased comparing the proposed system with the methods using B-tree index, etc. The searched record is 10% of the whole records stored in the system. From figure 8, it can be find that B-tree index method is performs well than searching only from disk without any index. But it is slower than the case while using the proposed system which caching data in memory and searching data from the snapshot. And while using T-tree index on the memory data, the search queries processing time is much decreased than the other methods, so the performance increased more.

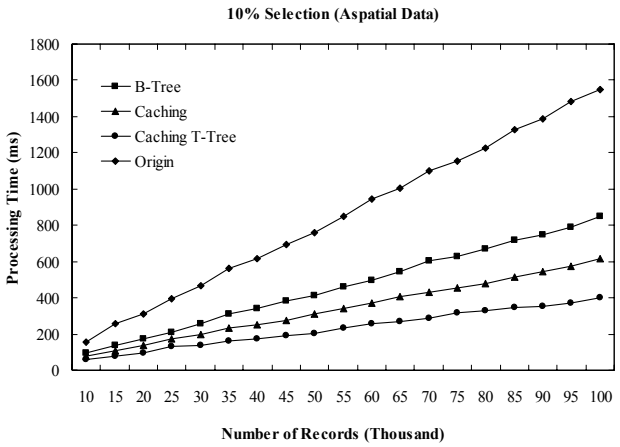


Fig. 8. Search operation using index and cache

5. Conclusion

In this paper, the design and implementation of a two-tier DBMS were described and its preliminary performance evaluation has been conducted. The results of the

performance test show that proposed system performs significantly better than disk-oriented DBMS with an added advantage to manage massive data at the same time. The proposed system can manage massive data and also can provide fast response time for certain parts of the data using snapshots. We are currently investigating query optimization and optimal synchronization technique between disk data and the memory snapshot data. For the synchronization, current prototype system uses lazy update method with assuming safe operation of the system (i.e. never die system). Therefore, efficient synchronization technique, intelligent snapshot making technique and recovery technique will be important challenges.

Reference

[1] Bijit Hore, Hakan Hacigumus, Bala Iyer, Sharad Mehrotrasss, "Indexing text data under space constraints," Proceedings of the thirteenth ACM international conference on Information and knowledge management CIKM '04, November 2004.

[2] Goetz Graefe, Michael Zwilling, "Transaction support for indexed summary views," Proceedings of the 2004 ACM SIGMOD international conference on Management of data, June 2004.

[3] Sven Helmer, Guido Moerkotte, "A performance study of four index structures for set-valued attributes of low cardinality," The International Journal on Very Large Data Bases, Volume 12, Issue 3, October 2003.

[4] Zhifeng Chen, Yan Zhang, Yuanyuan Zhou, Heidi Scott, Berni Schiefer, "Empirical evaluation of multi-level buffer cache collaboration for storage systems," Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems SIGMETRICS '05, Volume 33 Issue 1, June 2005.

[5] Wenhu Tian, Pat Martin, Wendy Powley, "Techniques for automatically sizing multiple buffer pools in DB2," Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research, October 2003.

[6] Stephane Bressan, Chong Leng Goh, Beng Chin Ooi, Kian-Lee Tan, "A framework for modeling buffer replacement strategies," Proceedings of the ninth international conference on Information and knowledge management, November 2000.

[7] Jonathan Goldstein, Per-Ake Larson, "Optimizing queries using materialized views: a practical, scalable solution," Proceedings of the 2001 ACM SIGMOD international conference on Management of data, 2001.

[8] James J. Lu, Guido Moerkotte, Joachim Schue, V. S. Subrahmanian, "Efficient maintenance of materialized mediated views," Proceedings of the 1995 ACM SIGMOD international conference on Management of data, 1995.

[9] Minwen Ji, "Affinity-based management of main memory database clusters," ACM Transactions on Internet Technology (TOIT), Volume 2 Issue 4, November 2002.

[10] Philip Bohannon, Peter Mcllroy, Rajeev Rastogi, "Main-memory index structures with fixed-size partial keys," Proceedings of the 2001 ACM SIGMOD International conference on Management of data SIGMOD '01, Volume 30 Issue 2, May 2001.

[11] Tobin J. Lehman and Michael J. Carey, "A Study of Index Structures for Main Memory Database Management Systems," Proceedings of the Twelfth International Conference on Very Large Data Bases, 1986.

[12] Ying Xia, Sung-Hee Kim, Sook-Kyoung Cho, Kee-Wook Rim, Hae-Young Bae, "Dynamic versioning concurrency control for index-based data access in main memory database systems," Proceedings of the tenth international conference on Information and knowledge management, October 2001.

[13] Tobin J.Lehman, J. Shekita and Luis-Felipe Cabrera, "An Evaluation of Starburst's Memory Resident Storage Component," IEEE Transactions on knowledge and data Engineering, Vol. 4, NO. 6, DECEMBER, 1992.

[14] Michael Stonebraker, "Managing Persistent Objects in a Multi-Level Store," SIGMOD Conference, pp2-11, 1991.

[15] Kong-Rim Choi, Kyung-Chang Kim, "T*-tree: a main memory database index structure for real time applications," Proceedings of the Third International Workshop on Real-Time Computing Systems Application (RTCSA '96), October 1996.

[16] Tobin Jon Lehman, "Design and performance evaluation of a main memory relational database system (t tree)," Doctoral Thesis, January 1986.

[17] D. Bitton, D. DeWitt, and C. Turbyfill, "Benchmarking simple database operations," in Proc. 9th Int. Conf. on Very Large Databases, Nov. 1983.

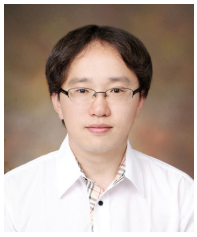
[18] S. Park, W. chung, and M. Kim GMS, "Spatial database management system," Proc. of the KISS Spring Conf, April, 2003.

Sang-Hun Eo

He received the BS in Computer Science & Engineering from Inha Univ. in 2003. And now he is undertaking a doctorate course as a member of the database lab at Inha Univ. His research interests include Spatial Database, Ubiquitous and Pervasive Computing, RFID middle ware, Context-Awareness System, Grid Database.

Yan Li

She received the MS in Computer Science & Engineering from Inha Univ. in 2008. And now she is undertaking a doctorate course as a member of the database lab at Inha Univ. Her research interests include Spatial Database, Spatial Data warehouse, Ubiquitous Computing, Grid Database.



Ho-Seok Kim

He received the Ph.D in Computer Science & Engineering from Inha Univ. in 2007. Now he works for LG Mobile Handset R&D Center of Mobile Communications Company, LG Electronics Inc as Senior Research Engineer. His research interests include Embedded Software System, Database, Ubiquitous and Pervasive Computing, Context-Awareness System.

Hae-Young Bae

He received a Ph.D. degree in Computer Science & Engineering from Soongsil Univ. in 1990. He has been a professor at Inha Univ. since 1982. His research interests are in the area of Data Management, RFID Systems, USN, Grid System. They include topics such as Spatial Database Management System, Spatial Database Cluster System, Grid Database Management System.