A Comparative Study of Main Memory Databases and Disk-Resident Databases

F. Raja, M.Rahgozar, N. Razavi, and M. Siadaty

***Abstract*—**Main Memory Database systems (MMDB) store their data in main physical memory and provide very high-speed access. Conventional database systems are optimized for the particular characteristics of disk storage mechanisms. Memory resident systems, on the other hand, use different optimizations to structure and organize data, as well as to make it reliable.

This paper provides a brief overview on MMDBs and one of the memory resident systems named FastDB and compares the processing time of this system with a typical disc resident database based on the results of the implementation of TPC benchmarks environment on both.

***Keywords*—**Disk-Resident Database, FastDB, Main Memory Database.

I. Introduction

THE idea of Main Memory Database (MMDB), using physical memory as primary storage and probably a disk subsystem for backup, has recently been an active research topic. MMDBs can achieve significant improvements in performance, processing time and throughput rates over conventional database systems by eliminating the need for I/O to perform database applications.

In contrast with the conventional DBMS systems, some problems are introduced in the MMDB environment.

The major problem deals with the volatility of main memory; so in this framework the issues concerned with efficient database recovery are more complex than in traditional DBMS systems [13].

In this paper we study the MMDB systems and their differences with Disc Resident Data Bases (DRDB) [10]; we use FastDB [1], a highly efficient main memory database system, to implement TPC benchmark [14, 15] on and compare its performance with a DRDB such as MS-SQL. FastDB assumes that the whole database is present in RAM and optimizes the search algorithms and structures according to this assumption. Our performance results indicate that elimination of the time overhead CAUSED BY transferring database files to the buffer pool and vice versa, makes FastDB work significantly faster than a traditional DATABASE; therefore it fits more efficiently to the requirements OF TODAY'S real time applications.

Authors are with the Database Research Group, Control and Intelligent Processing Center of Excellence, Faculty of ECE, School of Engineering, University of Tehran, Tehran, Iran (e-mail: f.raja@ece.ut.ac.ir, Rahgozar@ut.ac.ir, n.razavi@ece.ut.ac.ir, m.siadaty@ece.ut.ac.ir).

The remainder of this paper is organized as follows: In Section 3, MMDB and its corresponding issues are described; Section 4 and 5 will provide details of FastDB as an efficient main memory database and TPC as a performance benchmark; Section 6, details the system, environment, applications under which the experiments are conducted; Section 7, discusses the experimental results. At the end, Section 8 outlines the conclusions and Section 9 represents future directions of the work.

II. Main Memory Database (MMDB)

During the mid-1980s falling DRAM prices seemed to suggest that future computers would have such huge main memories that most databases could entirely be stored in them [4]. In such situations, it would be possible to eliminate all (expensive) I/O from DBMS processing. This would seriously change the architecture for a DBMS. An important question in a MMDBMS is how to do transactions and recovery in an efficient way. Some of the proposed algorithms assume that a (small) stable subset of the main memory exists. This stable memory for example can be used to place a redo log. Other algorithms do not assume stable memories, and still use I/O to write transaction information to a stable storage. These algorithms, hence, do not eliminate I/O, but minimize it, as the critical path in a MMDBMS transaction only needs to write the log; not data pages from the buffer manager [16].

A general MMDB architecture consists of a main memory implemented via standard RAM and an optional non-volatile (*stable*) memory. The main memory holds the primary copy of the database, while the stable memory holds the log and a backup copy of the database. A logger process flushes the logs asynchronously to the disk. Alternatively, an additional nonvolatile memory may be used as a shadow memory. This memory is intended to hold updates performed by active transactions and the tail end of the log that contains information about those updates [16].

A specific problem in MMDBMS is query optimization. The lack of I/O as dominant cost factor means that it is much more difficult in a MMDBMS to model query costs, as they depend on fuzzy factors like CPU execution cost of a routine. Therefore, DBMS query optimization tends to make use of simple cost models that contain "hard" constants obtained. One challenge in this area is to model the interaction between coding style, hardware factors like CPU and memory

architecture and query parameters into a reliable prediction of main memory execution cost [10][12].

MMDB systems are ideal for applications that require high throughput and a fast response time. With the increasing demand for high-performance systems and the steady decrease in memory cost, MMDBs have become an attractive alternative to DRDBs.

While meeting the high-performance demand of many real-time applications, MMDBs are naturally more vulnerable to failures than conventional DRDBs. Thus, the recovery component of an MMDB system must be properly designed, implemented and maintained. Three aspects of the recovery subsystem serve to ensure that the MMDB can recover from any failure: *logging*, *check pointing* and *reloading*.

III. FastDB

FastDB [1] is a highly efficient main memory database system with realtime capabilities and convenient C++ interface. FastDB is optimized for applications with dominated read access pattern. High speed of query execution is provided by the elimination of data transfer overhead and a very effective locking implementation. The Database file is mapped to the virtual memory space of each application working with the database. So the query is executed in the context of the application, requiring no context switching and data transfer. Synchronization of concurrent database access is implemented in FastDB by means of atomic instructions, adding almost no overhead to query processing. FastDB assumes that the whole database is present in RAM and optimizes the search algorithms and structures according to this assumption. Moreover, FastDB has no overhead caused by database buffer management and needs no data transfer between a database file and buffer pool. That is why FastDB will work significantly faster than a traditional database with all data cached in buffers pool [1].

FastDB supports transactions, online backup and automatic recovery after system crash. The transaction commit protocol is based on a shadow root pages algorithm, performing atomic update of the database. Recovery can be done very fast, providing high availability for critical applications. Moreover, the elimination of transaction logs improves the total system performance and leads to a more effective usage of system resources [1].

Although FastDB is optimized in the assumption that database as a whole fits into the physical memory of the computer, it is also possible to use it with databases, the size of which exceeds the size of the physical memory in the system. In the last case, standard operating system swapping mechanisms will work. But all FastDB search algorithms and structures are optimized under the assumption of residence of all data in memory, so the efficiency for swapped out data will not be very high.

FastDB is an application-oriented database. Database tables are constructed using information about application classes. FastDB supports automatic scheme evaluation, allowing users to do changes only in one place: in users' application classes. FastDB provides a flexible and convenient interface for retrieving data from the database. A SQL-like query language is used to specify queries. Such post-relational capabilities as non-atomic fields, nested arrays, user-defined types and methods, direct interobject references simplifies the design of database applications and makes them more efficient. FastDB uses a notation more popular for object-oriented programming than for a relational database. Table rows are considered as object instances, the table is the class of these objects. Unlike SQL, FastDB is oriented on work with objects, instead of SQL tuples. So the result of each query execution is a set of objects of one class [1].

IV. TPC Performance Benchmark

The Transaction Processing Performance Council (TPC) has defined a series of benchmark standards for database systems [14] [15].

In this study, we have tried to implement the same TPC environment (TPC A and TPC B) on two mentioned database systems so that we can use it for performance comparisons such as TPS measurements. In this work, we focus more particularly on comparing MMDB with DRDB for varying query loads, with varying size of DBs mentioned in the following.

V. Implementation of the Benchmark Environment

In order to compare performance time between MMDB and DRDB, we implemented a Visual C++.NET interface in conjunction with FastDB, and then using TPC-A benchmark and some extra testing benchmarks (to make homogeneous environments for both kinds of databases), we have used a set of intensive queries (including select, insert, update and delete) for varying number of table records and varying number of updates, implemented within the same benchmark environment, for both FastDB and MS-SQL databases. Notice that in the case of update, number of updates is equal to the value that we set at each run. Also, the recorded time is calculated from the beginning of an "existence checking" – which checks whether the produced random source/destination account/branch exists (or not) -to the end of the query execution. Random select query is designed so that it selects for half of the branches, all of the accounts which have a balance less than the original account balance. The experiment was performed on a Pentium IV (2.4 GHz) system with Windows XP as operating system and 256 MB of RAM.

VI. Experimental Results

In this section we describe the measured times in the different cases of intensive singleton queries (Insert, Update, and Select) for both MS-SQL and FastDB.

A. Insert Time Measurements

To observe the differences in performance of intensive insert queries between MS-SQL and FastDB, we ran the

benchmark in insert mode only with the number of target table records, varying from 1000 to 1000000 records; it should be mentioned that we cleared the database after each run before the next run. In this case, we inserted a specified number of records and measured the time of the whole insertion operation e.g. 19407 ms for 200000 records. The results of these measurements are shown in Fig. 1. As shown in this diagram, by increasing the numbers of records, the required time for performing update transactions increases. Also it is noticeable that FastDB insertion time is about 1000 times less than MS-SQL's. As a result, it is confirmed that in applications which require very short response times, (such as real-time applications), the conventional database systems will not satisfy the performance requirements as good as MMDBs.

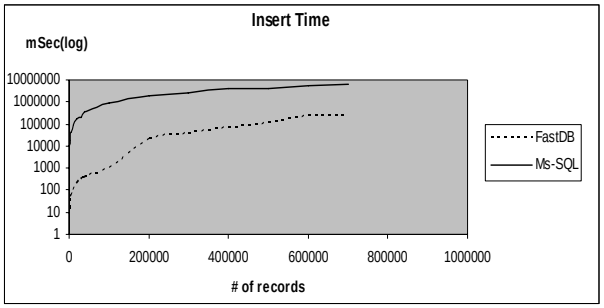

Fig. 1 Comparison of insert time between FastDB and MS-SQL for varying number of records

B. Update Time Measurements

For the case of intensive update queries, we performed the benchmark with constant number of updates and varying number of records in the target table. As the diagram below shows, for heavy loads (in contrast with low loads) FastDB does not perform much better than MS-SQL. Further analyses are needed to find out the reasons of such degradation in FastDB performance. We suggest to study the effect of swapping between main memory and hard disc to realize that why it could probably have such an important influence on the performance of FastDB.

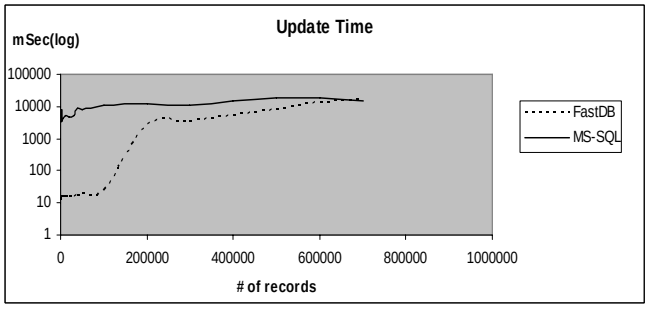

Fig. 1 Comparison of up date time between FastDB and MS-SQL for varying number of records in the table and with constant number of updates

C. Selection Time Measurement

We used a sample range query in this part of the benchmark that selects the accounts whose account balance is less than an original amount. Since random access to FastDB performs similar to the case of update we have chosen range selection to compare FastDB with MS-SQL in sequential access queries. In the following diagram (Fig. 3), results of this benchmark are shown. Here again like the case of update, MS-SQL has a more regular behavior than FastDB; for less than 150000 records, FastDB has a better response time than MS-SQL, but after this range, its response time grows increasingly. This shows the good performance of MS-SQL in such situations compared to FastDB.

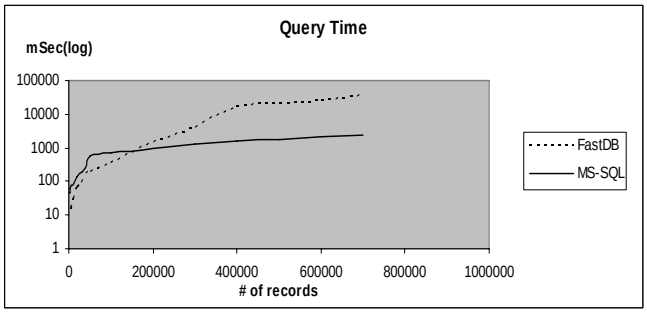

Fig. 2 Comparison of selection time between FastDB and MS-SQL for varying number of records

VII. Discussion

A wide range of application areas such as real-time/military applications require very short and predictable response time. Disks have long response time, and are not suitable as primary storage media for them. Main Memory has short response time, and its decreasing cost makes it affordable and suitable for these applications. Volatility of memory raises concerns about durability of in-memory data. Main Memory DBMS manages in-memory data and ensures ACID properties.

In this paper, we have evaluated one of the existing MMDBs, called FastDB[1], and implemented to compare it with MS-SQL. As the results show, FastDB is an application-orienant database, and is optimized for applications with dominated read access pattern. FastDB assumes that the whole database is present in RAM and eliminates the overhead time due to transferring database files to the buffer pool and vice versa; therefore since FastDB caches all its data in buffer pool, it works significantly faster than a traditional database until it does not need any swap between main memory and hard disc.

VIII. Conclusion

Like almost every research also the work presented here leaves some questions unanswered and even discovers new questions. Still many issues for future work remain; some of these open issues would be mentioned here:

- It needs further analysis to find out why FastDB shows unexpected behavior in heavy loads; it seems to be necessary to consider a mechanism to reduce the effect of swapping between main memory and disc in such DBs.
- FastDB uses T-tree for indexing the database and some

mechanism like "shadow paging" to guarantee quick and accurate recovery, therefore it is desirable to evaluate some other MMDB systems and compare them to FastDB.

ACKNOWLEDGMENT

We would like to thank Konstatin Knizhnik for providing the latest version of FastDB online and for his e-mail support and help with development of FastDB applications.

REFERENCES

[1] FastDB: a main-memory database object-relational database system, Available: http://www.garret.ru/~knizhnik/fastdb/FastDB.htm

[2] Inseon Lee, Heon Y.Yeon, Taesoon Park, "A New Approach for Distributed Main Memory Database Systems: A Causal Commit Protocol", *IEICE Trans. Inf. & Syst*., Vol.ES7, No.1 January 2004.

[3] Nicholas Carriero, Michael V. Osier, Kei-Hoi Cheung, Peter Masiar, Perry L. Miller, Kevin White, Martin Schultz, " *Exploring the Use of Main Memory Database (MMDB) Technology for the Analysis of Gene Expression Microarray Data",* Technical report, April 2004.

[4] Stefan Manegold, *"Understanding, Modeling, and Improving Main-Memory Database Performance"*, November 2002, Available: www.cwi.nl/htbin/ins1/publications?request=pdf&key=Ma:DISS:02.

[5] Philip Bohannon, Peter McIlroy, Rajeev Rastogi," Main Memory Index Structures with FixedSize Partial Keys", In Proceedings of SIGMOD Conference, 2001.

[6] S. Manegold, P. A. Boncz, and M. L. Kersten. "Optimizing Main-Memory Join on Modern Hardware", IEEE Transactions on Knowledge and Data Engineering (TKDE), Vol.14, No.4, pp.709–730, July 2002.

[7] J. Rao and K. A. Ross. "Making B+-Trees Cache Conscious in Main Memory", *Proc ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pp. 475–486, Dallas, TX, USA, May 2000.

[8] Tobin J. Lehman, Michael J. Carey, "A Study of Index Structures for Main Memory Database Management Systems", *Proc the Twelfth International Conference on Very Large Data Bases*, Kyoto, August, 1986.

[9] Rajeev Rastogi, S. Seshadri, Philip Bohannon, Dennis Leinbaugh, "Logical and Physical Versioning in Main Memory Databases", *Proc the 23rd VLDB Conference, Athens, Greece,* 1997.

[10] Hector Garcia-Molina, Kenneth Salem, *"Main Memory Database Systems: An overview"*, IEEE Transactions on Knowledge and Data Engineering, 4(6):509--516, Dec. 1992.

[11] Piyush Burte, Boanerges Aleman-Meza, D. Brent Weatherly, Rong Wu, *"Transaction Management for a Main-Memory Database",* Available: http://lsdis.cs.uga.edu/~aleman/mams/csci8370/paper/.

[12] Tobin J. Lehman, Michael J. Carey, "Query Processing in Main Memory Database Management Systems", Proceedings of the ACM SIGMOD International Conference on the Management of Data, 1986, pp.239-250.

[13] Margaret H. Eich, *"Main Memory Database Recovery"*, Proceedings of 1986 ACM Fall joint computer conference, Pages: 1226 – 1232, 1986.

[14] J. Baulier, P. Bohannon, S. Gogate, C. Gupta, S. Haldar, S. Joshi, A. Khivesera, H. F. Korth, P. McIlroy, J. Miller, P. P. S. Narayan, M. Nemeth, R. Rastogi, S. Seshadri, A. Silberschatz, S. Sudarshan, M. Wilder, and C. Wei. "*DataBlitz Storage Manager: Main-Memory Database Performance for Critical applications*", *Proc ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pp.519–520, 1999.

[15] M. Kauffman, "TPC BENCHMARK A Standard Specification", The Performance Handbook: for Database and Transaction Processing Systems, Transaction Processing Performance Council (TPC), 1991.

[16] Fremont, "TPC Benchmark B Standard Specification ", Waterside Associates, Transaction Processing Performance Council, CA., August 1990.