Data Compression Management Mechanism for Real-Time Main Memory Database Systems

Soon-Jo Lee and Hae-Young Bae
Department of Computer Science and Engineering
Inha University, In&on 402-751, KOREA
email: hybae@dragon.inha.ac.kr(165.246.10.3)

Abstract

In this paper we propose and study a data compression management mechanism for maximum available memory space and high performance in real-time main memory database systems (RT-MMDBSs). As a entire database is placed in main memory in a RT-MMDBS, the need for disk I/O operations to perform database transactions can be eliminated; therefore, the use of main memory database can achieve much better performance over real-time database systems. However, the system performance would be decreased when real-time data which is referenced outside its valid interval or non-real-time data which is infrequently referenced reside continuously in main memory. We define the characteristics of data and transactions in RT-MMDBSs, and we discuss major components that relate to the compression of such data.

1. Introduction

A real-time database system is one in which transactions have explicit timing constraints, such as deadlines. In this system, transaction processing must satisfy not only consistency constraints, but also timing constraints. Recently, a real-time database system is becoming important in a wide range of applications such as command and control, weapon systems, network management, traffic control systems, and automated manufacturing. To support those applications, it is essential to build a real-time database system that can process real-time transactions. To implement such a system, it has been proposed that the performance of conventional database systems should be enhanced. There are two approaches to achieve high performance. First, by putting more money; e.g., high speed devices can replace bottleneck devices like disk. Second, at the cost of some features of conventional database systems; e.g., some features are partly changed for real-time applications[9]. Main memory database systems, the entire database of which resides in main memory, are proposed as the best solution to process real-time transactions. Because main memory database systems can reduce disk I/O delay and the cost of memory purchase has been lowered[1,4].

Although RT-MMDBSs are capable of providing responses which are fast enough for real-time applications, they have some problems[12]. One of the problems is the size of main memory. As main memory database systems make entire database be resident constantly in main memory, and main memory size must be large enough to make entire database be resident in, to process transactions, and to store data which are produced continuously. In practice main memory size is limited, it is important to maximize the available space of main memory. It is not a perfect answer to design index and data structures which can maximize the available space of main memory. Furthermore, depending on the application, real-time systems may have to handle a large amount of multimedia information like audio, graphics, and images[10]. In this paper, to maximize the available space of main memory, a data compression mechanism is proposed, which retains the advantages of main memory database systems.

This mechanism compresses data separately in terms of timing constraints and reference probability. It takes some time to compress and decompress data. So in case that there is a real-time transaction to request data being compressed, it is impossible for the transaction to be processed within a deadline. It compresses only real-time data which is referenced outside valid interval and non-
real-time data which is requested infrequently so that it can maximize the available space of main memory without increasing deadline miss rate.

The motivation and method of the study are described in the first section. The characteristics of RT-MMDBSs, data types, and transaction types are described in the second section. The current methods for data compression are introduced and the proper criteria for real-time database are described in the third section. Major components for data compression management are presented in the fourth section. Conclusions and future research plans are described in the last section.

2. Related Works

In this section we introduce the characteristics of RT-MMDBS used in this paper, and describe the data types and the transaction types in that system.

2.1 Characteristic of RT-MMDBS

In a RT-MMDBS, data resides permanently in main memory; in conventional real-time database system(RT-DBS) it is disk resident. In a RT-DBS, disk-resident data may be cached into memory for further access; in a RT-MMDBS the memory resident data may have a backup copy on disk. So in both cases, a given object can have copies both in memory and on disk. The key difference is that in RT-MMDBS the primary copy lives permanently in memory. This has important implications as to how it is structured and accessed.

As RAM becomes cheaper and chip density increases, it is feasible to store larger databases in memory, making RT-MMDBS's a reality. Because data can be accessed directly in memory, RT-MMDBS's can provide much better response time and transaction throughputs, as compared to RT-DBS's.

A main memory has clearly different properties from that of magnetic disk, and these differences have profound implications on the design and performance of the database system. In designing RT-MMDBS the significant problem is the size of main memory. Main memory size must be large enough to deal with memory requests in processing transactions apart from residing database in main memory. It is important to maintain the high availability of memory space because so much data is stored in main memory of limited size. Therefore, data compression management must maximize the available space of main memory as in Figure 1, maintaining the merits of main memory database systems mentioned above.

The details of components of real-time main memory database systems in Figure 1 will be described in next section.

![Architecture of RT-MMDBS](image)

Figure 1. Architecture of RT-MMDBS

2.2 Transaction Types

There are two types of transactions in RT-MMDBS; real-time transactions and non-real-time transactions. The system should execute real-time transactions so that each transaction meets its deadline, and it should execute non-real-time transactions to minimize their average response time. Real-time transactions can be grouped into three categories: hard deadline, firm deadline, and soft deadline. The classification is based on how the system is affected badly by the violation of timing constraints[5,7].

Hard deadline transactions are those which may result in a catastrophe if the deadline is missed. One can say that a large negative value is imparted to the system if a hard deadline is missed. These are typically safety-critical activities, such as those that respond to life environment-threatening emergency situations.

Soft deadline transactions have some value even after their deadlines. Typically, the value drops to zero at a certain point past the deadline. If this point is the same as the deadline, we get firm deadline transactions, which impart no value to the system once their deadlines expire.
All transactions should be managed differently depending on transaction types to be processed properly. Transactions could be classified into hard, firm, soft, and non-real-time transactions by the criticalness factors and timing constraints of them at first. This paper presents methods with which to manage database according to data accessed by such classified transactions. This paper proposes the mechanism to manage RT-MMDBS according to the data type which is accessed by transactions classified like this.

2.3 Data Types

In RT-MMDBS, database consists of real-time data, non-real-time data, and system data. As shown in Figure 2, real-time transactions request only real-time data and system data, but non-real-time transactions request all types of data. System data is accessed very frequently and usually has low volume. Both real-time data and non-real-time data may be accessed infrequently. The former has low volume with stringent timing constraints, and the latter has high volume. If this is the case, it is possible to partition the data into one or more logical databases. Data should be exactly classified into real-time data type and non-real-time data type when the database administrator creates the database. Example of such real-time applications include telecommunications, radar tracking, and securities trading. In a telephone switching application, routing tables with which phone numbers are mapped to actual numbers, are real-time data; data for customers' monthly statements are non-real-time data. As above, data is classified by its type so that compression/decompression manager can easily select data to be compressed.

![Real-Time Database System](Image)

**Figure 2. Data types of RT-DBS**

**Real-Time Data**

Since real-time systems are used to monitor and to control physical devices, they need to store a large amount information about their environments. Such information includes input data from devices as well as system and machine states. In addition, many embedded systems must also store the system execution history for maintenance or error recovery purpose. Some systems may also keep track of system statistics such as average system load or average device temperature. Depending on the applications, real-time systems may have to handle multi-media information like audio, graphics, and images. Since systems are constantly recording information, data must have their temporal attributes recorded[5,10].

Often a significant portion of real-time database is highly perishable in the sense that it contributes to a mission only when used in time. In addition to deadline, therefore, other kinds of timing constraints could be associated with data in RT-DBS. For example, each sensor input could be indexed by the time at which it was taken. Data once stored in the database is valid in specific interval. To quantify this notion, data may be associated with a valid interval. Data outside its valid interval does not represent the current state, e.g. it would be transformed into non-real-time data type[10]. If there is a transaction to access data outside its valid interval, it is a non-real-time transaction. If we maintain data outside valid interval in main memory, system performance will be poor. It is natural that data outside valid interval be compressed and maintained until discard time. However, if a transaction requests real-time data and the data is compressed, it will take much time to decompress the data, so that the real-time transaction violates deadline condition. On the contrary, it enhances system performance not to compress real-time data.

**Non-Real-Time Data**

This type of data is requested by transaction with no timing constraints in that applications. It contains non-real-time data type declared in the beginning and real-time data type outside its valid interval. It is not a problem for non-real-time data to be delayed during compressing and decompressing. Because non-real-time data with no timing constraints like deadlines is requested in non-real-time transactions. Non-real-time data requested infrequently should be compressed so that it can maximize the efficiency of system. Non-real-time data requested frequently should not be compressed.

**System Data**

This type of data is requested during operating database and is produced autonomously by database systems. And it is always needed in processing all transactions. System data is associated with the information about database, table, user, index, access authority, and so on. As system data is hotspot data requested by all transactions, it is
natural that we keep system data from being compressed not to degrade the efficiency of system.

### 2.4 Data Compression Techniques

There has been no data compression techniques for real-time database systems. Therefore, this paper selects algorithm suitable for RT-MMDBS by comparison and evaluation of existing data compression algorithms. Criteria to select the algorithm which is suitable for data compression technique in RT-MMDBS are as follows.

1. fast compression/decompression
2. lossless decompression
3. using of small memory during the process
4. better compression ratio

<table>
<thead>
<tr>
<th>Computer</th>
<th>SUN Spark 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Speed</td>
<td>100 MIPS</td>
</tr>
<tr>
<td>Main Memory</td>
<td>32 MB</td>
</tr>
<tr>
<td>OS</td>
<td>SUN OS release 4.1.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Data Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size1</td>
<td>1KB</td>
</tr>
<tr>
<td>Size2</td>
<td>5KB</td>
</tr>
<tr>
<td>Size3</td>
<td>10KB</td>
</tr>
<tr>
<td>Size4</td>
<td>50KB</td>
</tr>
<tr>
<td>Size5</td>
<td>100KB</td>
</tr>
<tr>
<td>Size6</td>
<td>500KB</td>
</tr>
<tr>
<td>Size7</td>
<td>1MB</td>
</tr>
</tbody>
</table>

Figure 3. Evaluation environment

The algorithm based on above criteria should be evaluated on main memory, because the characteristics of RT-MMDBS are different from those of existing database. Also, only lossless algorithms such as Arithmetic, LZ77, LZW and AHC(Adaptive Huffman Coding) algorithm[9] is evaluated in terms of lossless decompression. Figure 4, 5, 6 show the evaluated data and Figure 3 shows the evaluation environment. Figure 3 describes the execution time of compression, Figure 4 describes the execution time of decompression and Figure 5 describes the ratio of compression. Data evaluated here is confined to text data. The result of evaluation shows that LZW algorithm has fast compression/decompression speed and high compression ratio, which is suitable for the compression algorithm of RT-MMDBS.
3. System Components for Data Compression Management

This section describes each component of real-time database system in Figure 1. Each component, especially the parts of each component related to compression management, is described.

3.1 Data Type Manager

Data type manager manages information on data to recognize and to compress real-time data, non-real-time data, time-out real-time data and system data of main memory resident database. Data type manager maintains not only some information to manage existing database, but also the following information to compress and manage data. This information is maintained and managed by table unit which we use as a compression unit.

1) real-time data/non-real-time data/system data

Data is classified into real-time, non-real-time and system data. When we classify data, the criterion depends on which transaction to access the data. If non-real-time transactions access the data, the data is classified into non-real-time data. And if both non-real-time transactions and real-time transactions access the data, or if only real-time transactions access the data, the data should be classified into real-time data. RT-MMDBS can manage data compression appropriately according to its data information.

2) valid interval of real-time data

Real-time data is associated with the valid interval during which it is used by system. If there is real-time data outside valid interval, it is no longer maintained as real-time data in system.

3) data compression flag

When a transaction refers to data, system must know whether or not the data is compressed. If the data to be referred is compressed, it must be decompressed.

4) size of decompressed data

When compressed data is decompressed, system must know original data size to allocate partition where compressed data is decompressed.

5) keeping time of compressed data

If there is compressed data which is referenced outside its keeping time and is no longer used by system, it is backed up into backup device like disk, tape and so on.

6) reference counter of data table

Only non-real-time data maintains this information. If reference counter of data is less than the number of counter determined by system, the data is ready to be compressed.

7) last reference time of data table

While data compression is proceeding, if there is a transaction to refer to the data, the compression must be canceled. For this facility, if the last reference to the data occurs after the data compression starts, the data compression is canceled.

Figure 7. Data Type Transition Diagram

Figure 7 shows functions of data type manager and compression/decompression manager to choose compression-ready data of entire database. Real-time data is ready to be compressed if it is referenced outside valid interval and non-real-time data is ready to be compressed if it is infrequently referred. But the reference to non-real-time data before its compression causes non-real-time data not to be compressed.

3.2 Memory Manager

Memory manager initially makes database resident in main memory and manages main memory by partition unit for continuously increasing data, to allocate and deallocate memory partitions required in compression and decompression, and maintain main memory for working area to be an appropriate size. Because system performance becomes poor in case that the size of working area is not appropriately maintained, the number of real-time transactions to be executed within their deadline would be decreased. Therefore, when the size of working area is smaller than an appropriate size, memory manager asks
3.3 Compression/Decompression Manager

Compression/decompression manager compresses time-out real-time data and non-real-time data infrequently referred, and discards compressed data past their keeping time as shown in figure 7. Such work is executed periodically, but if memory manager requests of the data compression to maintain working area appropriately, compression work may be executed aperiodically.

Figure 8. Data Reference Diagram

Figure 8 shows reference method of the data which is compressed like ① or isn't compressed like ②. When transactions request data, system retrieves the data with database name, table name, partition number and offset information. First, system acquires appropriate database information in DB_Catalog_Table and then retrieves information on the table of data from TB_Catalog_Table. System checks whether or not the data is compressed with retrieved information from data table. And if the data is compressed, it requests of data decompression or if the data isn't compressed, it refers to data through Partition_No_Tab.

\[
\text{Compress\_req() /* management process which invokes data compression */}
\]
\[
\text{input: TB\_Catalog\_Table}
\]
\[
\text{output: Updated TB\_Catalog\_Table, Compressed data}
\]
\[
\text{while (end of data != scan TB\_Catalog\_Table)}
\]

\[
\text{if (!Table\_Comp\_Flag and Table\_Comp\_State)}
\]
\[
/* check compressible state of data table and whether or not compressed */
\]
\[
\text{if (Table\_Size > p)}
\]
\[
/* target data table size larger than the specified size p */
\]
\[
\text{Allocate memory partition}
\]
\[
\text{Data compress}
\]
\[
\text{if (Table\_Ref\_Time > Comp\_Start\_Time)}
\]
\[
\text{Abort completely-compressed data}
\]
\[
\text{Release memory partitions}
\]
\[
\text{else}
\]
\[
\text{Update TB\_Catalog\_Table}
\]
\[
\text{Release memory partitions of compression target data Table}
\]
\[
\text{if (Table\_Comp\_Flag)}
\]
\[
\text{if (Discard\_Time < Current\_Time)}
\]
\[
\text{Discard Compressed Data /* copy to backup device */}
\]
\[
\text{Update TB\_Catalog\_Table}
\]
\[
\text{Release memory partitions of discarded compressed data}
\]
\[
\]
\[
\text{decompress\_req() /* management process which invokes data decompression */}
\]
\[
\text{input: TB\_Catalog\_Table information about request data table}
\]
\[
\text{output: updated TB\_Catalog\_Table, decompressed data}
\]
\[
\text{Allocate memory partition}
\]
\[
\text{Data decompress}
\]
\[
\text{Update TB\_Catalog\_Table}
\]
\[
\text{Release memory partition of compressed data table}
\]

Figure 9. Pseudo code for the Compression/Decompression Management Algorithm
In Figure 9, a pseudo code for compression/decompression management algorithm is presented. System executes compression work requested periodically or aperiodically as following.

First, if data compression is requested, system checks through the scan of TB_Catalog_Table whether or not the data is compressed and whether or not the data is ready to be compressed. If both conditions are satisfied, system checks whether the corresponding table size has a good effect on system in case of compression. Whether to execute this work is determined after system compares the table size with appropriately specified \( p \) value according to application. If the compression of data table would be beneficial, that is, Table_size > \( p \), system makes memory manager allocate memory partitions and start compressing data with the partitions.

After data compression is finished, system checks whether there is a transaction to refer to compressing data during compression work. If there is a transaction to refer to the table, system cancels compression work executed so far and proceeds the next step. If there is no transaction to refer to the table, system changes the partition number of TB_Catalog_Table, records compression status on Table_Comp_Flag and records the original table size. Then, system releases the partitions occupied by data table before compression.

If the checked data table is in a compression status, system checks the discard time and back up the data outside discard time into backup device and deletes the data from main memory database. Then, system updates TB_Catalog_Table and releases the partitions occupied by discarded data table.

If the requested data is in a compression status, system executes decompression work. The procedure of decompression is as following. First, system allocates memory partition for decompression, and then decompresses the data. After decompression work is finished, system updates TB_Catalog_Table and releases the partitions occupied by compressed data.

3.4 Performance Evaluation

We implemented a main memory file manager and a scheduler, and then evaluated the algorithm which is proposed in this paper. The focus of evaluation is how much effect the compression technique produce on transaction processing and how much the size of main memory is reduced. We used Classified Earliest Deadline First Algorithm[4] in which the priority and type of transaction is considered, because real-time transactions must be processed prior to non-real-time transactions. In systems which this algorithm is used in, compression/decompression does not effect on processing of real-time transactions. Therefore, only the effectness of main memory is evaluated in this paper. The environment of performance evaluation is as following.

- main memory size: 6MB
- used memory: 5MB
- available memory: 1MB
- data table size: 50KB (it takes less than one second to compress or decompress)
- non-real-time data table: 80%
- real-time data table: 20%
- database size: fixed (database increases or discards in a fixed size)

The result of performance evaluation is as following figure 10. In early times the size of used memory is decreasing, and then it is almost fixed. The reason is that the number of tables to be compressed is almost same to that of tables to be decompressed.

![Figure 10. Result of performance evaluation](image_url)

4. Conclusion and Future Research

This paper has proposed the technique of data compression and management to maximize the available space of main memory in real-time main memory database system, whose purpose is to handle real-time transactions with timing constraints. The proposed technique classifies data into real-time data and non-real-time data so that compression work may not have effects on processing real-time
transactions. For processing real-time data, the proposed technique compresses data outside valid interval, and for processing non-real-time data, it compresses infrequently referred data extremely so that system performance may not decrease. Considering working area size, it requests compression work aperiodically.

The application of the proposed technique keeps the size of working area appropriately, which is very important in real-time main memory database system, and maximizes the available space of main memory to store data which is increasing continuously. In addition, it manages transactions to minimize transaction deadline miss rate.

In future, research plans are to develop new compression technique because disk-based compression technique is not suitable for main memory database system, and to develop the index structure suitable for the new compression technique.

References