DATABASE PARTITIONING TECHNIQUES TO SUPPORT RELOAD IN A MAIN MEMORY DATABASE SYSTEM: MARS

Le Gruenwald Margaret H. Eich

Department of Computer Science and Engineering
Southern Methodist University
Dallas, Texas 75275

ABSTRACT

In a main memory database system, the primary copy of the database may be placed in *volatile memory*. When a crash occurs, a partial or complete reload of the database from archive memory (AM) into main memory (MM) is needed. To effectively perform the reload process without degrading the system performance, the most effective technique for structuring AM and MM should be determined. This paper reports on experiments performing a complete analysis of possible partitioning techniques in terms of the number of I/Os for reload and number of MM references incurred during transaction processing. Our analysis shows that horizontal and single vertical partitioning are actually the only possible candidates. Which one is best depends on the types of transactions executed.

Introduction

Recently, with memory costs decreasing, storage capability increasing, and the need for high performance database systems rising, many researchers have been examining the problem of designing *Main Memory Database (MMDB) systems* [1]. *MARS(MAin memory Recoverable database with Stable log)* is a main memory database system designed at Southern Methodist University. It assumes that the primary copy of the database is in main memory and that an *archive database* existing on secondary storage is used solely as a backup in the event of main memory media failure or system failure.

In a MMDB system, such as MARS, the primary copy of the database or a major portion of the database may be placed in a *volatile memory*. When a crash occurs, a partial or complete reload of the database from archive memory (AM) into main memory (MM) is needed. This reload process has a severe impact on the performance. Performance impediments are mainly caused by two problems: down time and page faults. When the system is down, no transactions can be processed. Page faults caused by referencing data not yet loaded are encountered if the system is brought online before the reload process completes. The goal of our ongoing research is to derive an effective way to perform the reload process without degrading transaction performance. One of the steps to achieving this goal is to examine how data can be stored on AM and MM.

* This material is based in part upon work supported by the Texas Advanced Research Program (Advanced Technology Program) under Grant No. 2265.

The objective of this paper is to examine the effect of different database partitioning techniques on the MMDB reload problem and, subsequently, derive the best technique to structure the AM and MM. The best technique is one that yields the minimum overall cost in terms of the number of I/O needed for reload and transaction processing time.

The paper is organized as follows. In section 2, possible partitioning techniques are introduced. Section 3 describes the overall approach we take to perform our analysis. Sections 4 and 5 contain the analysis. The paper is concluded in section 6.

Database Partitioning Techniques

Many database partitioning techniques have been developed to physically organize data on storage devices. Each technique divides the data into groups which are then assigned to physical pages. We classify these into the following six categories: horizontal partitioning, group horizontal partitioning, single vertical partitioning, physical vertical partitioning, group vertical partitioning, and mixed partitioning.

Horizontal partitioning subdivides a relation to form groups of tuples without taking tuple affinity into account. Each group contains a number of *complete* tuples. Due to its simplicity this technique has been used as a conventional storage structure.

The group horizontal partitioning technique decomposes a relation into groups of tuples based on their affinity. Tuples that are more frequently used together are placed in the same group. This technique has been implemented in several distributed database systems such as SDD-1 [2].

Single vertical partitioning focuses more on references to single attributes without considering attribute affinity. It vertically subdivides a relation into groups each of which contains only one type of attribute. This technique has been used in several database systems, such as ADABAS, to physically store their secondary keys [3].

In the physical vertical partitioning technique, each tuple is divided into physical groups of fixed sizes that are independent of attribute length. This technique has not been implemented in any database systems. The reason we consider it for MARS is due to its uniformity to all relations.

Group Vertical partitioning subdivides attributes of a relation into groups based on affinity. Attributes that are commonly required together are physically stored together. This technique has been extensively studied in a variety of computing environments [4], [5].

Mixed partitioning is introduced to take advantages of both horizontal and vertical partitioning [6]. Partitions of data are constructed by either applying group horizontal to physical/group vertical partitioning or vice versa. SDD-1 uses this technique to partition its database [2].

The group horizontal and mixed partitioning techniques are excluded from our analysis for MARS due to the high CPU cost needed for tuple affinity calculation. More discussion on this is given in [7].

Analysis Approach

The analysis of the database partitioning techniques is performed based on the eight properties that cover the database design and the query processing stages: 1) number of physical pages required to store an entire relation, 2) cost to store a relation into the main memory, 3) cost to delete an attribute from a schema, 4) cost to insert an attribute into a schema, 5) cost to project attributes in all tuples from a given relation, 6) cost to select a tuple from a given relation, 7) cost to insert a tuple into a relation, and 8) cost to modify some attributes within a specified tuple. The first property estimates the number of I/Os for reloading while the rest of the properties measure the number of MM references incurred during transaction processing.

These eight properties are complete in that they can be used to derive the cost (based on memory references and I/O requirements) for any type of transaction. In the following list we include in parenthesis following the type of database operation, a list of the properties used to estimate the cost of that operation: project attributes from a relation (1,5), select a tuple (6), modify a tuple (8), delete a tuple (6), join two relations (1,5,6), modify a schema (2,3,4), and insert a tuple (7).

We examine each property individually and then rank the six partitioning techniques based on their ability to satisfy the property, that is based on the cost incurred. A ranking of 6 indicates that this technique yields the lowest cost for that property, and ranking of 1 indicates that it yields the highest cost for that property. For a particular property, a technique with a ranking of 6 is the *best* technique, and the one with a ranking of 1 is the *worst* one.

After the rankings are done, a weight associated with each property which indicates how important the property is in transaction performance and in reloading is derived. "How important" here means the frequency of the property in contributing its cost to the transaction processing and reloading cost. The most important property has the highest weight. The total weighted value of each database partitioning technique is determined as follows:

$$\text{total weighted value} = \sum_{i=1}^{8} w_i r_i$$

where w_i is the weight associated with property i, and r_i is the ranking of the technique for the property i. The technique that yields the highest total weighted value is the choice for structuring the main memory and archive memory.

Analysis of The Eight Properties

Analyzing the eight properties based on the number of main memory references incurred by each partitioning technique, and using our ranking system with the highest rank for the best technique, we obtain the results in Table 1. Horizontal, Single, Physical, and Group stands for horizontal, single vertical, physical vertical, and group vertical respectively. Due to space limitations, the detailed analysis of each of the eight properties is not shown here. The interested reader is referred to [7].

Table 1. Ranking of Partitioning Techniques

Cost	Horizontal	Single	Physical	Group
Number of Pages	2	4	1	3
Implementation	3	4	2	1
Attribute Deletion	2	4	1	3
Attribute Insertion	3	4	2	1
Attribute Access	1	4	2	3
Tuple Selection	4	1	2.5	2.5
Tuple Insertion	1	4	2.5	2.5
Tuple Modification	4	3	1.5	1.5

Total Analysis

In this section, we give a total analysis of all six partitioning techniques to identify the technique which yields the lowest overall cost. The technique that has the highest total weighted value W incurs the lowest overall cost. Recall that W is defined as $W = \sum_{i=1}^{8} w_i r_i$, where w_i is the weight of the property i, and r_i is the rank of the property i. We use W_h, W_s, W_p, and W_g, to indicate the total weighted value for horizontal, single vertical, physical vertical, and group vertical partitioning respectively. Ranks are the results of the analyses of the eight properties (Table 1). Weights are determined based on the frequencies of use of typical relational transactions. The weight of each of the eight properties is computed as the sum of the frequencies of use of all transaction types which use the property to determine their processing costs. For example, if the property i is used to determined the cost of n transaction types, then the weight w_i of property i is the sum of the frequencies of use of all n transactions: $w_i = \sum_{j=1}^{n} f_j$ where f_j is the frequency of use of transaction type j.

The total weighted values for the database partitioning techniques are obtained as follows:

$$W_h = 3f_{proj} + 4f_{sel} + 4f_{tmod} + 4f_{tdel} + 7f_{join} + 10f_{schmod} + 3f_{tins}$$
$$W_s = 8f_{proj} + f_{sel} + 3f_{tmod} + f_{tdel} + 9f_{join} + 16f_{schmod} + 8f_{tins}$$
$$W_p = 3f_{proj} + 2.5f_{sel} + 1.5f_{tmod} + 2.5f_{tdel} + 5.5f_{join} + 6f_{schmod} + 3.5f_{tins}$$
$$W_g = 6f_{proj} + 2.5f_{sel} + 1.5f_{tmod} + 2.5f_{tdel} + 8.5f_{join} + 8f_{schmod} + 5.5f_{tins}$$

Since $W_p < W_g$, physical vertical partitioning is not the best technique in any case. We therefore exclude it from further analysis.

We divide the transaction types into three subsets: retrieval, update, and schema modification. The members of the retrieval subset are transaction types: select a tuple, project attributes, and join two relations. The update subset's members are modify a tuple, delete a tuple, and insert a tuple. The third subset has only one member: modify a schema. Let f_{ret}, f_{upd}, and f_{schmod} be the frequencies of the subsets which are defined as follows:

$$f_{ret} = f_{proj} + f_{sel} + f_{join}$$
$$f_{upd} = f_{tmod} + f_{tdel} + f_{tins}$$

f_{schmod} is the frequency of the schema modification transaction type itself. Since most real database applications require more retrievals than updates and more updates than schema modifications, it is reasonable for us to assume the following order: $f_{ret} > f_{upd} > f_{schmod}$.

We designed a computer program to examine the two cases:

1. Case a: Examination of the effects of the update frequency on the total weighted value given a fixed retrieval frequency between 50% to 100%. The update frequency varies from 30% to (100% - the given retrieval frequency) at 1% steps.
2. Case b: Examination of the effects of the retrieval frequency on the total weighted value given a fixed update frequency between 30% to 50%. The retrieval frequency varies from 50% to (100% - the given update frequency).

In both cases, the schema modification f_{schmod} is calculated as $(100\% - f_{ret} + f_{upd})$.
The relative orders among retrieval and update frequencies studied by the program are $f_{sel} > f_{proj} > f_{join}$ (S1), $f_{sel} > f_{join} > f_{proj}$ (S2), $f_{proj} > f_{sel} > f_{join}$ (P1), $f_{proj} > f_{join} > f_{sel}$ (P2), $f_{join} > f_{sel} > f_{proj}$ (J1), $f_{join} > f_{proj} > f_{sel}$ (J2), $f_{tmod} > f_{tins} > f_{tdel}$ (M1), $f_{tmod} > f_{tdel} > f_{tins}$ (M2), $f_{tdel} > f_{tmod} > f_{tins}$ (D1), $f_{tdel} > f_{tins} > f_{tmod}$ (D2) $f_{tins} > f_{tmod} > f_{tdel}$ (I1), and $f_{tins} > f_{tdel} > f_{tmod}$ (I2).

For each value of f_{ret} and of f_{upd}, the program determines two types of output: the percentage of time a partitioning technique gives the highest total weighted value for *each combination* of the member frequencies (type 1), and the percentage of time a partitioning technique gives the highest average total weighted value computed by averaging all total weighted values for *all combinations* of the member frequencies (type 2).

From all possible test runs for case a, we derive Table 2. The results of test runs for case b can be found in [7]. The technique that has the highest output of type (1) is the best technique under the conditions of the particular test run. If two techniques have the same or similar outputs of type 1, then the one that has the higher output of type (2) is chosen to be the best one.

Table 2. Results Based on Retrieval Frequency

Highest Retrieval Member	Highest Update Member	Relative order	Retrieval Frequency	Best Technique
Selection	Modification	S1, M1	50%-79%	Single Vertical
Selection	Modification	S1, M1	80%-99%	Horizontal
Selection	Modification	S2, M2	50%-99%	Horizontal
Selection	Deletion	S1, D1	50%-99%	Horizontal
Selection	Deletion	S2, D2	50%-99%	Horizontal
Selection	Insertion	S1, I1	50%-99%	Single Vertical
Selection	Insertion	S2, I2	50%-74%	Single Vertical
Selection	Insertion	S2, I2	75%-99%	Horizontal
Projection	Modification	P1, M1	50%-99%	Single Vertical
Projection	Modification	P2, M2	50%-99%	Single Vertical
Projection	Deletion	P1, D1	50%-99%	Single Vertical
Projection	Deletion	P2, D2	50%-99%	Single Vertical
Projection	Insertion	P1, I1	50%-99%	Single Vertical
Projection	Insertion	P2, I2	50%-99%	Single Vertical
Join	Modification	J1, M1	50%-99%	Single Vertical
Join	Modification	J2, M2	50%-99%	Single Vertical
Join	Deletion	J1, D1	50%-99%	Single Vertical
Join	Deletion	J2, D2	50%-99%	Single Vertical
Join	Insertion	J1, I1	50%-99%	Single Vertical
Join	Insertion	J2, I2	50%-99%	Single Vertical

From the results of all test runs, we see that either horizontal or single vertical is the best technique. When selection is the highest member of the retrieval subset, most of the time horizontal performs better than single vertical if either tuple modification or deletion is the highest update member. Otherwise, if tuple insertion has the highest update frequency, single vertical usually outperforms horizontal. When projection is the highest retrieval member, regardless of the update members, single vertical is always the best. When join has the highest retrieval frequency, most of the time single vertical performs better than horizontal. We conclude that if we have more selections than projections and joins, and more tuple modifications or tuple deletions than tuple insertions, then horizontal gives the best performance. Otherwise, single vertical does.

Conclusions

In this paper, we examine the effect of different partitioning techniques on the MMDB reload problem in terms of the number of I/Os for reload and number of MM references during transaction processing. The best technique is the one that yields the minimum overall cost consisting of both properties. Our analysis shows that horizontal and single vertical are actually the only possible candidates. Physical vertical never yields the best result. In some very rare cases, group vertical outperforms the other techniques. If the database system encountered performs more selections than projections and joins, and performs more tuple modifications or tuple deletions than tuple insertions then horizontal is the best technique. Otherwise, single vertical is the chosen technique. Our analysis also shows that if reload is the only concern, that is if we do not take into account the transaction performance, then single vertical is always the best choice.

References

[1] T. Lehman and M. Carey, "A Recovery Algorithm for a High-Performance Memory Resident Database System", *SIGMOD*, 1987.

[2] J. Rothnie *et al.*, "Introduction to a System for Distributed Databases (SDD-1)", *ACM Transactions on Database Systems*, Vol.5, No.1, p. 1-17, March 1980.

[3] P. Pratt and J. Adamski, *Database Systems Management and Design*, Boyd and Fraser Publishing Company, 1987.

[4] S. Navathe, *et al.*, "Vertical Partitioning Algorithms for Database Design", *ACM Transactions on Database Systems*, Vol.9, No.4, p. 680-710, Dec 1984.

[5] D. Cornell and P. Yu, "A Vertical Partitioning Algorithm for Relational Databases", *IEEE Database Engineering*, p. 30-35,1987.

[6] S. Ceri, *Distributed Database Design*, McGraw Hill, 1984.

[7] L. Gruenwald and M. Eich, "Database Partitioning Techniques to Support Reload in a Main Memory Database System: MARS", Technical Report 89-CSE-31, Southern Methodist University, June 1989.