CHOOSING THE BEST STORAGE TECHNIQUE FOR A MAIN MEMORY DATABASE SYSTEM *

Le Gruenwald Margaret H. Eich

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

Abstract. **Alternative partitioning strategies for database storage are examined within a Main Memory Database environment. An analytic performance analysis indicates that horizontal and single vertical are the only logical choices.**

Introduction

A *Main Memory Database (MMDB)* system assumes that the database is completely memory resident in a semiconductor memory ([5], [6], [9], [11]). Due to the many differences between RAM memory and secondary storage devices, many conventional database storage, access, and recovery approaches must be rethought in the MMDB context [7]. The purpose of this paper is to examine alternative techniques for storing database records (tuples) in an MMDB. Through an analytic performance analysis we compare the different choices and recommend the approaches to use based on overall processing requirements and types of transactions executed.

We assume an MMDB model where the primary copy of the database is in a semiconductor *Main Memory (MM)* and a backup database resides on a secondary storage device, *Archive Memory (AM)*. Periodically, data must be *checkpointed* from MM to AM. An alternative design would assume that MM is completely nonvolatile. This research is applicable under either design. Both MM and AM are divided into fixed size pages with logical records (tuples) contained within them.

In this paper we examine database partitioning techniques to determine which storage approach should be used to store data both in the AM and MM. We assume that the AM and MM structures are the same. This is needed due to the fact that data is checkpointed from MM to AM at a page level, and that efficient reload to MM from AM after a system failure is crucial. *Partitioning* is the process of assigning logical objects (relations) to several physical objects (pages in MM or blocks in AM) in a stored database [13]. Although we use the terms tuples and relations our results apply to any data model where conventional files (pages and records) are used for storage on AM. There are many different techniques one can use to partition a database into logical groups. The cost model used in this paper compares the different approaches based on the impact a partitioning approach has on all database processing functions including reload and transaction processing. This research represents a complete examination of partitioning techniques in an MMDB system. Its originality is that it is the first such endeavor.

The following sections describe the analysis in detail. In section 2, possible partitioning techniques are introduced. Section 3 describes the overall approach we take to perform our analysis. Section 4 examines and ranks each partitioning technique based on its performance for eight different properties which impact processing performance. Section 5 contains the complete analysis for all partitioning techniques. Section 6 summarizes results and makes recommendations for selecting the best approach.

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. In our discussions and analysis we assume that one technique is used for all pages in a database. Recommendations for best partitioning technique are made based on the processing characteristics of the

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

database. It is certainly conceivable that a different technique could be used for each relation. It is also quite possible for an index to have a different partitioning technique from its target relation. Since our results depend on the processing characteristics of the target system they are valid at the database, relation, or index level. The use of multiple partitioning techniques within the same MMDB system, however, requires much more research as its use would complicate the design of the MMDB DBMS code itself. Also, the address translation process required to access MM data would be more complex.

Horizontal partitioning subdivides a relation to form groups of tuples without taking tuple affinity into account. Horizontal partitioning is believed to be a natural way to partition data in many application environments [15]. Most commercial database systems use this partitioning strategy.

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 ([14], [16].

Single vertical partitioning subdivides a relation into groups each of which contains all tuple values for only one type of attribute. This technique is often used to implement inverted files.

In the *physical vertical partitioning* technique, each tuple is divided into physical groups of fixed sizes that are independent of attribute lengths. Some attributes may spread across several groups. In this technique, a group size is chosen independently of attribute sizes or tuple sizes. Therefore the same address translation mechanism can be applied to all relations.

Group vertical partitioning has been extensively studied in a variety of computing environments ([13], [4]). It subdivides attributes of a relation into groups based on affinity. Attributes that are used together are stored in the same group. Conversely, attributes which are not used together are stored in different groups.

The *mixed partitioning technique* has advantages of both horizontal and vertical partitioning ([1],[2],[3]). Partitions of data are constructed by either applying group horizontal to physical/group vertical partitioning or vice versa. SDD-1 distributed system is one of the database systems that uses mixed partitioning to partition its database [14].

Among the six possible techniques, we immediately exclude group horizontal partitioning from our analysis. This is due to the following reasons:

1) In order to group the tuples based on their affinity, we must examine the tuple affinity for all tuples in the relation. This cost can be tremendously high since the number of tuples in a relation is generally very large. The relative cost for such an operation in an MMDB system is much greater and the benefits much less than in a conventional database system.

2) Insertion of a tuple requires ***regrouping*** of tuples in the database. Since the regrouping process is very expensive as mentioned in (1), having to perform it often is intolerable in an MMDB system where processing times are fast.

Since mixed partitioning requires the existence of group partitioning, this leads to the exclusion of mixed partitioning from our analysis. In brief, from the set of six possible partitioning techniques, we will conduct our analysis on the single horizontal or horizontal for short, single vertical, physical vertical, and group vertical partitioning techniques.

Approach

To compare the four partitioning techniques, we introduce the following properties:

1) Number of physical pages required to store an entire relation.
2) Cost to initially load a relation in main memory.
3) Cost to delete an attribute from a schema.
4) Cost to insert an attribute into a schema.
5) Cost to project onto selected 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.
8) Cost to modify a tuple.

The first property is used to derive costs for reload as it indicates the number of pages (and thus I/Os) needed. The remaining properties will be used to determine the performance of a technique based on the number of MM references performed in processing database transactions. The eight properties are complete in that they can be used to derive the cost (based on memory references and I/O requirements) for each type of transaction. To delete a tuple, we must identify the tuple and free up the space occupied by the tuple. This is similar to selecting a tuple. Thus the tuple deletion cost can be calculated by using the property of the tuple selection cost (6). The join of two relations, say r and s, requires the product of $r \times s$ to be performed followed by the selection on common attributes, and finally followed by the projection of attributes. The cost to compute the product of two relations can be determined using the cost to select $n_r * n_s$ tuples from two relations r and s, where n_r and n_s are the number of tuples in r and in s respectively. Therefore, to compute the cost to join to relations, we can use the cost to select a tuple (6) and cost to access attributes (5) . Modification of a schema requires deletion/insertion of a single attribute or a group of attributes from/into the schema. Thus the cost to modify a schema is determined by the two properties: cost to delete an attribute from a schema (3) and cost to insert an attribute into a schema (4)

Our overall performance strategy is to first evaluate each partitioning technique based on each of these 8 properties. This will allow us to rank each partitioning technique on each property. Due to the many variables and parameters involved, it proved to be difficult to provide a more precise measure. However, the impreciseness and simplicity of this ranking approach is overcome in the second phase of the evaluation. In this phase we obtain a weight (based on frequency of usage) for each property which indicates how important the property is in transaction performance and in reloading. The most important property has the highest weight. We then determine the *total weighted value* for each database partitioning technique 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. A ranking of 4 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. The technique with the highest total weighted value is the best choice. However, since we can not predict the frequency of occurrence of a property, we examine all possible frequencies and recommend the best techniques for those frequencies. No one partitioning technique is best under all situations. Here is how we overcome any problems with the rankings. A change to the total weighted value can be due to either a change in a property weight or rank. By examining all possible frequencies we simulate changes to the ranks as well.

Analysis of the Eight Properties

In this section we summarize the results of ranking the eight properties. A more complete treatment of the evaluation can be found elsewhere [8].

We wrote a computer program to test the number of pages incurred in each technique under realistic combinations of parameter values (relation size, page size, tuple size, group size). In most of the cases (71.78%) single vertical performs either better or at least as well as the other three techniques. In 23.98% of the cases, group vertical yields the same or fewer number of pages than the other techniques. When group size is divisible by tuple size, physical vertical yields a good result. However, when this is not the case, it's behavior was usually much worse than the other approaches. Because of this unpredictability, we rank physical vertical as the worst for this technique. The final rankings obtained for this first property are thus as reported in Table 1: horizontal (2), single (4), physical (1), and group (3).

The load MM cost is the cost to store a relation into main memory. This cost may be incurred when a schema modification is needed. There are three major steps needed to store a relation into main memory: create address translation tables for MM access, perform address translations, and store values of the tuples into MM. All of these are very strongly related to the number of pages for the relation. Even though group vertical partitioning has lower translation table creation cost and address translation cost than physical vertical, the MM load cost incurred in it is higher than the one in physical vertical due to the cost of the attribute grouping process. When including this cost it becomes the worst choice. In brief, we derive the following ranking for the MM load property: horizontal (3), single (4), physical (2), and group (1).

Attribute deletion cost is the cost to delete an attribute from a schema. Except for the single vertical technique, the other techniques do require reorganization of either the entire database (horizontal and physical vertical) or a group of attributes (group vertical). Thus single vertical is best. Physical vertical performs the worst due to its behavior for property one. Group vertical is better than horizontal because of the smaller amount of data to be reorganized. The rankings for this property are derived as follows: horizontal (2), single (4), physical (1), and group (3).

Attribute insertion cost is the cost to insert an attribute into a schema and to store its values. As with the attribute deletion cost, a reorganization of the database is needed for all techniques except single vertical. Therefore the single vertical partitioning technique yields the lowest attribute insertion cost. Since the MM load cost dominates reorganization, using the results obtained from the MM load cost property, we find that the attribute insertion cost incurred by group vertical partitioning is the highest, by single vertical is the lowest, and by horizontal is lower than by physical vertical partitioning. We obtain the following rankings for this property: horizontal (3), single (4), physical (2), and group (1).

Attribute access cost is the cost to access (project) all values for a group of n attributes in a relation. There are three major steps needed to access a group of attributes in a relation: locate physical pages in which the attributes are stored, find the physical locations that are used to store the values of the attributes, and access the values of the attributes. The difference between the cost to access a group of n attributes incurred by the partitioning techniques is caused by the cost to determine the physical pages where the attributes are stored. This is dominated by the number of pages which are needed to store the desired attributes. As discussed earlier, horizontal needs more pages to store a relation than single vertical partitioning. Since physical vertical and group vertical would require that only a subset of the total pages be identified (those in which the attributes are stored), their behavior is better than horizontal with their relative behavior as shown for property one. The ranking for this property is derived as follows: horizontal (1), single (4), physical (2), and group (3).

Tuple selection cost is the cost to reconstruct (select) a tuple from a relation. There are two major steps needed to select a tuple given its identifier: determine the physical address of the tuple and access the tuple's values for all attributes. The difference between the cost of the four techniques comes from the number of address translations performed. The horizontal partitioning technique requires only one address translation for each tuple, therefore it yields the lowest cost. Single vertical requires address translation for each attribute in each tuple. Thus it is worse. The behavior for physical and group are similar, with the best determined by the number of groups involved. Generally we can conclude the following ranking: horizontal (4), single (1), physical (2.5), and group (2.5).

Tuple insertion cost is the cost to insert a tuple into an existing relation. There are three major steps needed: find free space to insert the tuple, allocate new pages and update translation tables if no room is found, and store the tuple's value. The major difference in approaches is due to the cost to examine for free space and to perform address translations. In the best case, each approach would require a constant overhead to check for free space. In the worst case each page must be examined. Since the number of pages, then, determines the worst case behavior, ranking in this category is largely determined by that for the first property. However, we assume that free space within each group is the same. That is, to find free space within each group, only one group need be examined. Free space within other groups is at the same relative position. Therefore, the performance of horizontal is worse than either physical or group vertical. In general we conclude the following rankings for this property: horizontal (1), single (4), physical (2.5), and group (2.5).

Tuple modification cost is the cost to modify n attributes in a specific tuple. The two major steps to modify n attributes in a tuple are: identify physical locations of n attributes, and modify the values of the attributes. The differences between the costs come from the number of address translations performed. The horizontal partitioning technique requires (n + 1) address translations (1 for the tuple, and n for n attributes). Single vertical requires (2 * n) address translations, while physical vertical as well as group vertical partitioning, each requires ((2 * number of groups) + n) address translations. In general, the number of attributes we want to modify is very small. Therefore, single vertical partitioning yields lower tuple modification cost than physical vertical and group vertical partitioning do. Whether physical vertical partitioning yields higher cost than group vertical partitioning depends on the number of groups each technique has. Generally we can conclude the following rankings: horizontal (4), single (3), physical (1.5), and group (1.5).

In Table 1 we provide a summary of the rankings found in the preceding paragraphs. Note that single vertical is the best in 6 of the 8 cases and second in another. However, for the property which we would expect to have the highest weight (tuple selection) it has the lowest rank. Thus the detailed analysis discussed in the next is warranted. As mentioned earlier, this additional phase will compensate for any errors in this subjective ranking.

Total Analysis

In this section, we give a total analysis of all four 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. The weight of each property 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. Table 2 shows for each transaction type the properties used to derive the cost for that transaction as well as the notation used in our analysis to indicate the frequency of occurrence for that transaction.

Cost	Horizontal	Single Vertical	Physical Vertical	Group Vertical
Number of Pages	2	4	1	3
Load MM	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

Table 1. Ranking of The Partitioning Techniques for the Eight Properties

Transaction Type	Frequency	Properties Needed
Project attributes	f_{proj}	1, 5
Select a tuple	f_{sel}	6
Modify a tuple	f_{tmod}	8
Delete a tuple	f_{tdel}	6
Join two relations	f_{join}	1, 5, 6
Modify a schema	f_{schmod}	1, 2, 3, 4
Insert a tuple	f_{tins}	1, 7

Table 2. Transaction Types and Properties Needed to Compute Their Costs

From the information given in the above table we derive the weights for the properties as follows: $w_1 = f_{proj} + f_{join} + f_{schmod} + f_{tins}$, $w_2 = f_{schmod}$, $w_3 = f_{schmod}$, $w_4 = f_{schmod}$, $w_5 = f_{proj} + f_{join}$, $w_6 = f_{sel} + f_{tdel} + f_{join}$, $w_7 = f_{tins}$, $w_8 = f_{tmod}$. The total weighted values W_h (horizontal), W_s (single vertical), W_p (physical vertical), and W_g (group vertical) are given by the formulae:

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

We observe that the total weighted value in physical vertical partitioning W_p is always less than the one in group vertical partitioning W_g. This indicates that physical vertical is not the best technique in any case. Therefore we exclude it from further analysis.

We compare the total weighted values of the partitioning techniques in four cases of transaction mixes:

1. Retrieval (Transactions allowed are selection, projection, and join.),
2. Update (Transactions allowed are tuple modification, tuple insertion, and tuple deletion.),
3. Schema modification (Only transaction allowed is schema modification.),
4. Mix of all three cases (All seven transaction types listed in Table 2 can be processed).

In each case, we examine the effects of the frequencies of use of related transaction types on the partitioning techniques. We then determine the situations in which a particular technique yields the highest weighted total value. In other words, we establish the conditions for the best techniques based on the frequencies of the related transaction types.

Retrieval Environment

In this environment, we assume that the only transactions that can be processed are project attributes (frequency f_{proj}), select a tuple (f_{sel}), and join of two relations (f_{join}). We have written a computer program to examine the impact of the selection, project, and join frequencies on the total weighted values of the partitioning techniques. The program, examines the impact of the six possible relative orders of the three frequencies. In each test run, one of the three frequencies is specified to be the highest frequency. Within each test run, the highest frequency varies from 50% to 100%, the second highest from 30% to (100% - the highest frequency) if the highest frequency does not exceed 70% and from 2/3 of (100% - highest frequency) to (100% - highest frequency) otherwise. Each frequency is varied at 1% increments. The lowest frequency is the remaining amount: (100% - the highest frequency - the second highest frequency). Since actual results vary, we report on the percentage of

Highest Frequency	Relative Order	Percentage of Time W is highest			Percentage of Time Average W is highest		
		Horizontal	Single	Group	Horizontal	Single	Group
Selection	$f_{sel}>f_{proj}>f_{join}$	45.20	18.43	37.37	68.63	7.84	25.49
Selection	$f_{sel}>f_{join}>f_{proj}$	80.56	0.00	21.21	92.16	0.00	9.8
Project	$f_{proj}>f_{sel}>f_{join}$	0.00	100.00	0.00	0.00	100.00	0.00
Project	$f_{proj}>f_{join}>f_{sel}$	0.00	100.00	0.00	0.00	100.00	0.00
Join	$f_{join}>f_{sel}>f_{proj}$	0.25	52.53	47.98	0.00	58.52	41.18
Join	$f_{join}>f_{proj}>f_{sel}$	0.00	100.00	0.00	0.00	100.00	0.00

Table 3. Test Run Results in Retrieval Environment

time a technique yields the highest total weighted value and the highest average total weighted value which is computed by averaging all total weighted values for all combinations of the second highest and third highest frequencies for each value of the highest frequency. In Table 3 we present the results of six test runs. Each row represents one test run. Regardless of the relative orders chosen, if selection is the highest frequency, horizontal is the clear cut winner. If projection or join has the highest frequency, then single vertical always yields the best results.

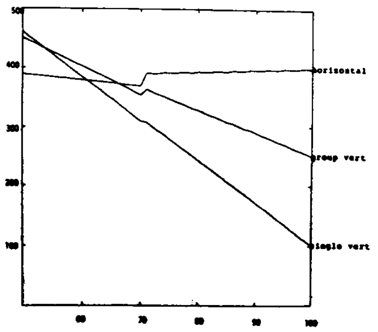
Figures 1a and 1b show the average total weighted value for the first two orderings. The weighted values for each value of the highest frequency are computed as the *average* weighted values over all combinations of the second and third highest frequencies. Figure 1a shows that when selection has the highest frequency, and $f_{sel} > f_{proj} > f_{join}$, single vertical performs the best if the selection frequency f_{sel} is below 53% of the total retrieval frequency, group vertical performs the best if f_{sel} is between 53% and 66%, and horizontal exceeds the other two techniques if f_{sel} is more than 66%. If the relative order, $f_{sel}>f_{join}>f_{proj}$ is chosen, in which join has higher frequency than projection, Figure 1b shows that single vertical is *always the worst*, and horizontal outperforms group vertical when the selection frequency exceeds 53%. Results for the two orderings in which project has the highest frequency, showed that single vertical is *always the best* regardless of the values of the project frequency and the orders between the join and select frequencies. When join has the highest frequency, horizontal is *always the worst.* If the select frequency is higher than the project frequency, and if the join frequency is smaller than 70%, then group vertical exceeds single vertical. Single vertical performs better than group vertical otherwise. When the project frequency is larger than the select frequency, single vertical is always the best.

Examining these figures we note that there are "jumps" in the average total weighted value that occur in each graph when the highest frequency changes from 70% to 71%. This takes place because in our experiment, when the highest frequency is less than or equal to 70%, the second highest frequency varies from 30% to (100% - the highest frequency) at 1% steps. When the highest frequency exceeds 70%, the second highest frequency varies from 2/3 of (100% - the highest frequency) to (100% - the highest frequency) at 1% steps. In brief, the "jumps" occur because the low limit of the second highest frequency is forced to change from 30% to 2/3 of (100% - the highest frequency) when the highest frequency exceeds 70%.

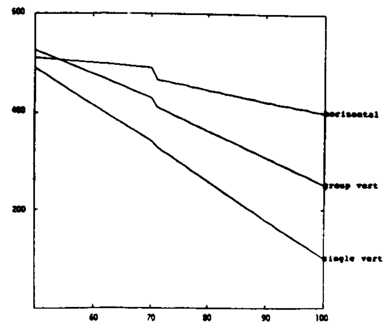
Based upon the results discussed above, if only retrieval transactions are used either horizontal or single vertical is the technique of choice. If it is known that selection occurs more often than other types of retrieval, horizontal is the best in most situations. However the results are not clear cut unless its frequency of occurrency is quite high (over 70%). For lower values group vertical or even single vertical may be better. However even in these cases the difference between the three techniques is small. The horizontal superiority is quite evident for higher selection percentages. In retrieval environments where project or join occur with the highes frequencies, single vertical partitioning is the clear cut winner.

Update Environment

In this environment, we assume that the only transactions that can be processed are modify a tuple (frequency f_{tmod}), delete a tuple (f_{tdel}), and insert a tuple (f_{tins}). The other transaction types never occur. Experiments similar to those for the retrieval environment were run to investigate the update transactions. In Table 4 we present the results of the six test runs.



a) $f_{sel}>f_{proj}>f_{join}$



b) $f_{sel}>f_{join}>f_{proj}$

Figure 1. Effect of f_{sel} on Average Total Weighted Value in Retrieval Environment

Highest Frequency	Relative Order	Percentage of Time W is highest			Percentage of Time Average W is highest		
		Horizontal	Single	Group	Horizontal	Single	Group
Tuple Modification	$f_{tmod}>f_{tins}>f_{tdel}$	25.76	74.49	0.00	43.14	58.86	0.00
Tuple Modification	$f_{tmod}>f_{tdel}>f_{tins}$	100.00	0.00	0.00	100.00	0.00	0.00
Tuple Deletion	$f_{tdel}>f_{tmod}>f_{tins}$	100.00	0.00	0.00	100.00	0.00	0.00
Tuple Deletion	$f_{tdel}>f_{tins}>f_{tmod}$	70.45	30.56	0.00	84.31	17.65	0.00
Tuple Insertion	$f_{tins}>f_{tmod}>f_{tdel}$	0.00	100.00	0.00	0.00	100.00	0.00
Tuple Insertion	$f_{tins}>f_{tdel}>f_{tmod}$	0.00	100.00	0.00	0.00	100.00	0.00

Table 4. Results of Test runs in Update Environment

Figures 2a, and 2b show the average total weighted value outputs for the first two test runs. In Figure 2a and Figure 2b, tuple modification is specified as the highest frequency. If tuple insertion takes place more often than tuple deletion, single vertical outperforms horizontal when the tuple modification frequency is less than 78%, and horizontal outperforms single vertical otherwise (Figure 2a). If tuple deletion occurs more often than tuple insertion, Figure 2b shows that horizontal is *always the best.* When tuple deletion frequency is the largest one among all update frequencies, if the tuple modification frequency is higher than the tuple insertion frequency then horizontal is *always* the best. Otherwise, single vertical performs better than horizontal when tuple deletion does not exceeds 68%. The opposite result is obtained when tuple deletion exceeds 68%. When tuple insertion is designated to be the highest frequency, single vertical *always* outperforms the other two techniques. These three experiments also confirm our conclusion of the group vertical: regardless of the values of the three frequencies, this technique never becomes the best.

Again we see that the technique of choice is either the horizontal or single vertical. When modification has the highest frequency the choice is not obvious, although for most cases horizontal is better. When tuple deletion occurs the most, horizontal is the winner in most cases. When tuple insertion is the highest frequency the single vertical technique is the only choice.

Schema Modification Environment

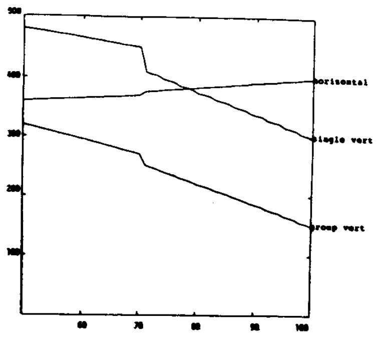
In this environment, the only transaction processed is schema modification (frequency f_{schmod}). All other transaction types never take place. The total weighted values of the partitioning techniques are given as follows:

$$W_h = 10f_{schmod},$$
$$W_s = 16f_{schmod},$$
$$W_g = 8f_{schmod}.$$

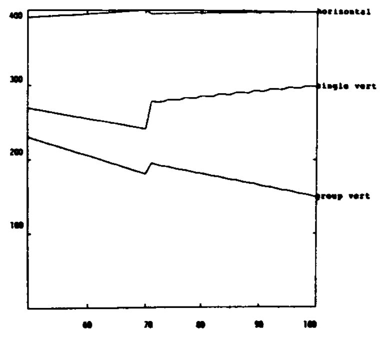
Obviously, in this environment, the single vertical is the best technique regardless of the value of the schema modification frequency.

Mixed Environment

In this environment, all transaction types (1-7) are processed. 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



a) $f_{tmod}>f_{tins}>f_{tdel}$



b) $f_{tmod}>f_{tdel}>f_{tins}$

Figure 2. Effect of f_{tmod} on Average Total Weighted Value in Update Environment

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 of real database applications require more retrievals than updates, ([10], [12]) and more updates than schema modifications, it is reasonable for us to assume the following order: $f_{ret} > f_{upd} > f_{schmod}$. In the following paragraphs, we examine the impact of these frequencies on the weighted values of the partitioning techniques.

We designed a computer program to examine the two cases:

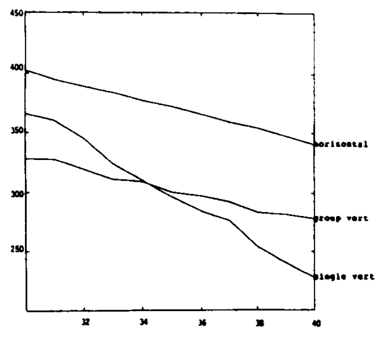
1. Case 1: Examination of the effects of the update frequency on the total weighted value, given a retrieval frequency between 50% to 100%. The update frequency varies from 30% to (100% - the given retrieval frequency) at 1% steps.
2. Case 2: Examination of the effects of the retrieval frequency on the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

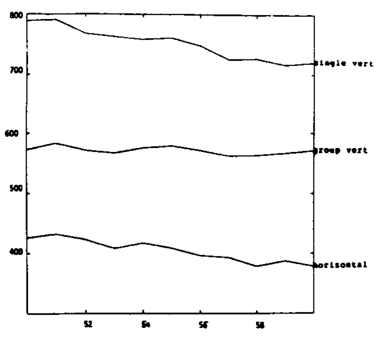
In each test run of case 1 and of case 2, one of the retrieval members is specified to have the highest retrieval frequency. This member is called the highest retrieval member. Similarly we specify the highest update member. Within the retrieval subset, we then have the highest, second highest, and third highest retrieval members. The way we vary the values of these members is the same as in the retrieval environment, except that the total frequency now is not 100% but f_{ret}. Similarly, within the update subset, we have the highest, second highest, and third highest members which are varied the same way as in the update environment where the total frequency is not 100% but f_{upd}. For each test run, the program produces the same measures as for the retrieval and update environments.

Figures 3a and 3b are two examples of test runs for case 1 and case 2. Figure 3a shows the average total weighted value of a test run for case 1 in which selection and tuple deletion are chosen to be the highest retrieval and update members. The relative orders used are $f_{sel}>f_{proj}>f_{join}$ and $f_{tdel}>f_{tmod}>f_{tins}$. The given retrieval frequency is 60%. The x-axis represents the update frequency which varies from 30% to 40%. In this test run, horizontal outperforms single vertical and group vertical regardless of the values of the update frequency. Figure 3b is obtained from a test run for case 2. The two relative orders $f_{proj}>f_{sel}>f_{join}$ and $f_{tins}>f_{tdel}>f_{tmod}$ are selected. The given update frequency is 40%. The x-axis shows the retrieval frequency varying from 50% to 60%. In this case, single vertical *always* performs the best.

Table 5 is constructed from test runs for case 1. Although not shown here, similar results were found for case 2. Examining this table, 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 *always* 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. From these results, we conclude that in general in a mixed environment, if we have more selections than projections and joins, and more tuple modifications or tuple deletions than tuple insertions,



a) Effect of f_{upd}



b) Effect of f_{ret}

Figure 3. Average Total Weighted Value in Mixed Environment

Highest Retrieval Member	Highest Update Member	Relative order	Retrieval Frequency	Best Technique
Selection	Modification	$f_{sel} > f_{proj} > f_{join}, f_{tmod} > f_{tins} > f_{tdel}$	50%-79%	Single Vertical
Selection	Modification	$f_{sel} > f_{proj} > f_{join}, f_{tmod} > f_{tins} > f_{tdel}$	80%-99%	Horizontal
Selection	Modification	$f_{sel} > f_{join} > f_{proj}, f_{tmod} > f_{tdel} > f_{tins}$	50%-99%	Horizontal
Selection	Deletion	$f_{sel} > f_{proj} > f_{join}, f_{tdel} > f_{tmod} > f_{tins}$	50%-99%	Horizontal
Selection	Deletion	$f_{sel} > f_{join} > f_{proj}, f_{tdel} > f_{tins} > f_{tmod}$	50%-99%	Horizontal
Selection	Insertion	$f_{sel} > f_{proj} > f_{join}, f_{tins} > f_{tmod} > f_{tdel}$	50%-99%	Single Vertical
Selection	Insertion	$f_{sel} > f_{join} > f_{proj}, f_{tins} > f_{tdel} > f_{tmod}$	50%-74%	Single Vertical
Selection	Insertion	$f_{sel} > f_{join} > f_{proj}, f_{tins} > f_{tdel} > f_{tmod}$	75%-99%	Horizontal
Projection	Modification	$f_{proj} > f_{sel} > f_{join}, f_{tmod} > f_{tins} > f_{tdel}$	50%-99%	Single Vertical
Projection	Modification	$f_{proj} > f_{join} > f_{sel}, f_{tmod} > f_{tdel} > f_{tins}$	50%-99%	Single Vertical
Projection	Deletion	$f_{proj} > f_{sel} > f_{join}, f_{tdel} > f_{tmod} > f_{tins}$	50%-99%	Single Vertical
Projection	Deletion	$f_{proj} > f_{join} > f_{sel}, f_{tdel} > f_{tins} > f_{tmod}$	50%-99%	Single Vertical
Projection	Insertion	$f_{proj} > f_{sel} > f_{join}, f_{tins} > f_{tmod} > f_{tdel}$	50%-99%	Single Vertical
Projection	Insertion	$f_{proj} > f_{join} > f_{sel}, f_{tins} > f_{tdel} > f_{tmod}$	50%-99%	Single Vertical
Join	Modification	$f_{join} > f_{sel} > f_{proj}, f_{tmod} > f_{tins} > f_{tdel}$	50%-99%	Single Vertical
Join	Modification	$f_{join} > f_{proj} > f_{sel}, f_{tmod} > f_{tdel} > f_{tins}$	50%-99%	Single Vertical
Join	Deletion	$f_{join} > f_{sel} > f_{proj}, f_{tdel} > f_{tmod} > f_{tins}$	50%-99%	Single Vertical
Join	Deletion	$f_{join} > f_{proj} > f_{sel}, f_{tdel} > f_{tins} > f_{tmod}$	50%-99%	Single Vertical
Join	Insertion	$f_{join} > f_{sel} > f_{proj}, f_{tins} > f_{tmod} > f_{tdel}$	50%-99%	Single Vertical
Join	Insertion	$f_{join} > f_{proj} > f_{sel}, f_{tins} > f_{tdel} > f_{tmod}$	50%-99%	Single Vertical

Table 5. Results Based on Retrieval Frequency in Mixed Environment

then horizontal gives the best performance. Otherwise, single vertical is best.

Summary and Future Research

In this paper, we discussed different techniques one can use to partition an MMDB: horizontal, group horizontal, single vertical, physical vertical, group vertical, and mixed partitioning. To determine which technique is the best, in terms of the overall cost which consists of both number of I/Os for reload and number of MM references in transaction processing, we introduced eight properties. We analyzed each of these properties to find out how the partitioning techniques compared to each other for the property. We then ranked the partitioning techniques for each property accordingly. In the final analysis, we derived the weights for the properties based on the frequencies of use of typical relational transaction types which use the properties to determine their costs. The total weighted value of each partitioning technique is computed based on the weights of the properties and the ranks of the the technique in these properties. The total weighted values thus depend upon the frequencies of use of transaction types in a database system. Given a database system with the information on these frequencies, we are always able to derive the total weighted values of the partitioning techniques. The technique that has the highest total weighted value is the desired one since it requires the minimum overall cost in terms of number of I/Os for reloading and number of MM references for transaction processing. It therefore satisfies the goals of structuring the main memory and archive memory.

Our analysis shows that horizontal and single vertical are actually the only possible candidates. Physical vertical never yields the highest total weighted value. 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 to use for partitioning the database. Otherwise, single vertical is the chosen technique. Our analysis also shows that if reload of the database from AM into MM is the only concern, that is we do not take into account the transaction performance, then single vertical is always the best choice because it requires the fewest number of pages to store a relation.

References

[1] Ceri, S., Navathe, S., "A Methodology for Distribution Design of Databases", *COMPCON 83*, San Francisco, March 1983.

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

[3] Chang, S., Cheng, W. "A Methodology for Structured Database Decomposition", *IEEE Transactions on Software Engineering*, Vol.SE-6, No.2, March 1980, pp. 205-218.

[4] Cornell, D., Yu, P. "A Vertical Partitioning Algorithm for Relational Databases", *IEEE Data Engineering Conference*, May 1987, pp. 30-35.

[5] DeWitt, D., Katz, R., Olken, F., Shapiro, L., Stonebraker, M., Wood, D., "Implementation Techniques for Main Memory Database Systems", *ACM SIGMOD*, June 1984, pp. 1-8.

[6] Margaret H. Eich, "A Classification and Comparison of Main Memory Database Recovery Techniques," *Proceedings of the Third International Conference on Data Engineering*, Los Angeles, California, February 1987, pp 332-339.

[7] Margaret H. Eich, "Main Memory Database Research Directions," *Proceedings of the 1989 International Workshop on Database Machines*, Deauville, France, June 1989, pp 251-268. (Earlier version available as SMU technical report 88-35.)

[8] Gruenwald, L., "Recovery in Main Memory Database Systems", PhD Dissertation, Southern Methodist University, Department of Computer Science and Engineering, 1990.

[9] Hagmann, R., "A Crash Recovery Scheme for a Memory Resident Database System", *IEEE Transactions on Computers*, Vol. C-35, No.9, September 1986, pp. 839-843.

[10] Joyce, J., Warn, D., "Command Use in a Relational Database System", *National Computer Conference*, 1983, pp. 247-253.

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

[12] Magalhaes, G., "Improving The Performance of Database Systems", University of Toronto, Department of Computer Science, Technical Report CSRG-138, December 1981.

[13] Navathe, S., Ceri, S., Wiederhold, G., Don, Y. "Vertical Partitioning Algorithms for Database Design", *ACM Transactions on Database Systems*, Vol.9, No.4, December 1984, pp. 680-710.

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

[15] Segev, A., "Optimization of Join Operations in Horizontally Partitioned Database Systems", *ACM Transactions on Database Systems*, Vol.11, No.1, March 1986, pp. 48-80.

[16] Stonebraker, M., Neuhold, E., "A Distributed Database Version of INGRES", *Proceedings of the 3rd Berkeley Workshop on Distributed Data Management and Computer Networks*, Lawrence Berkeley Lab., University of California, Berkeley, May 1977, pp. 19-36.