Evaluating Partitioning Techniques for Main Memory Database: Horizontal and Single Vertical[1]

Yu Chin Cheng Le Gruenwald Greg Ingels M. T. Thakkar

School of Computer Science
University of Oklahoma
Norman, OK 73019

Abstract

Performance of the two partitioning techniques, horizontal partitioning and single vertical, is evaluated in the context of main memory database in terms of memory space and execution time. Issues in systems architecture related to cost assessments are addressed, quantitative results are obtained.

1 Introduction

Partitioning is the process of mapping logical entities of relations to its physical counterparts. In studying the reload problem for the MARS system [2], Gruenwald and Eich evaluated four partitioning techniques for *main memory database (MMDB)* [4, 5]: horizontal, single vertical, group vertical, and physical vertical. In their study, all procedures related to database processing are broken down into one or more of eight basic properties. (See Table 1.) In each property, the four partitioning techniques are ranked 1 (worst) to 4 (best) for their performance based on the estimated cost to execute the basic property. Performance of the partitioning techniques are analyzed in four different processing environments: *retrieval, update, schema modification, and mixed environment.* The performance of a technique in an environment is computed by summing the products of its ranks and the frequencies that the constituent basic properties of the environment are executed. It is concluded that horizontal and single vertical are the most viable choices for MMDB.

The purpose of this study is to provide a more quantitative evaluation for horizontal and single vertical partitioning for MMDB. In particular, we observe that in [4, 5], although numbers are used in the ranking system, the ranks do not reflect the difference quantitatively. For example, in evaluating the number of physical pages required to store a relation, horizontal and single vertical are ranked 3 and 4, respectively. While this indicates that single vertical is better than horizontal, it does not indicate *how much better.*

Table 1: Eight basic properties

no.	*property*
1	number of physical pages to store a relation
2	loading a relation into main memory
3	attribute deletion
4	attribute insertion
5	attribute projection
6	tuple selection
7	tuple insertion
8	tuple modification

In Section 2, we first examine the issues of system architecture that are directly related cost assessments in implementing a basic property. As a result, we will be able to accurately evaluate the costs related to, for example, address translation. In Section 3 memory space efficiency for the two partitioning techniques is evaluated. In Section 4, we derive the equations for discrepancies in execution time for the basic properties incurred by horizontal and single vertical, and compute the *performance boundaries* for horizontal and single vertical in terms of the frequencies the basic properties are executed. The paper is concluded in Section 5.

[1]This work was partially supported by National Science Foundation under grant No. IRI9201596.

2 System architecture

This section presents the foundation that is used to derive memory space and execution time affected by the two partitioning techniques. Specifically, we shall examines a few designs that are closely related to MMDB and the database parameters used in the performance evaluation.

Physical page layout A page stores entities of a relation. In horizontal partitioning, an ***entity*** is a tuple; in single vertical partitioning, it is an attribute. Each page is partitioned into a number of slots equal to the maximum number of entities it can store.

Addressing Data reside in physical pages are accessed through pointers in an index structure. A pointer can either be a ***logical address*** or a ***physical address. Logical address is used in this study,*** and every tuple has a unique logical address. (Note that one tuple corresponds to one physical entity for horizontal partitioning and multiple physical entities for single vertical partitioning.) A logical address is dissected into two parts: ***page number*** and ***offset.*** The page number is mapped into a physical starting address of a physical page, and the offset is used to calculate the actual physical location of the tuple inside a specific physical page.

In general, consider a page group of S pages, with each page partitioned into T slots, each of size W bytes. The logical addresses for the slots are $0, 1, \cdots, T-1, T, T+1, \cdots, 2T-1, \cdots, (S-1)T, (S-1)T+1, \cdots, ST-1$. Let $PT[0..S-1]$ be an array of size S, with each element storing the beginning physical address of a page in the page group. Let L be a logical address in the range $0..ST-1$, the physical address of L is computed by

$$PT[L \div T] + W \times (L \bmod T), \qquad (1)$$

where $\div$ is the integer division operator, and mod is the modulo operator.

For single vertical, in addition to the variables for address translation of horizontal partitioning, let m be the number of attributes, and let A_i and T_i denote the size of attribute i and the number of A_i's that can be stored in a page, then a tuple with logical address L is mapped into m entities with physical addresses at

$$PT_i[L \div T_i] + A_i \times (L \bmod T_i), \quad i = 0, \cdots, m-1, \qquad (2)$$

where PT_i is the page table for the i-th attribute.

Index structure Tuples in a relation are accessed through an index structure constructed from a set of unique search keys associated with the tuples. ***We assume that the same partitioning technique is used for the memory space used by the index structure.*** Therefore, costs related to index structure are the same for both partitioning techniques.

Table 2: Analysis Parameters

Parameter	*Meaning*	*Values*
REL_SZ	relation size (tuple)	$10^3 i, i = 1, 2, \cdots, 10$
TUP_SZ	tuple size (byte)	$50i, i = 1, 2, 3, 4, 6, 9$
PAGE_SZ	page size (byte)	512
ATRIB_SZ$_k$	size of attribute k	5,10,15 bytes
NUM_ATRIB	num. of attributes	10,20,30

Memory space management Memory space management must provide information for the free slots in each page so that when a new tuple is inserted into a relation, a free slot can be assigned and the logical address of the slot can be stored in the new node in the index structure. Also, when a tuple is deleted, the slot must be marked free to be used in the future. ***A free list of available slots is used in this study.***

Database information In order to calculate the amount of space and the execution time, specific database information must be available. We shall use the parameters listed in Table 2 as the sample database information. Table 2 is a part of the list (excluding those related to group vertical and physical vertical) compiled in [4] based on the studies in [1, 3, 7]. In addition, we shall assume that one main memory read takes the same amount of time t_r as one memory write, and all arithmetic operations ($+, -, \times, \div$, and mod) cost the same amount of time, t_a.

3 Memory space

Memory space performance is modeled by the number of physical pages required to store an entire relation. The amount of main memory space used by a relation consists of three parts: space to store data, space for the page table, and space for the index structure. In our design, the latter remains the same regardless of the partitioning techniques used. Furthermore, the number of entries in the page table is equal to the number of pages required by a technique; therefore, the amount of memory used to store data in a relation reflects the overall memory requirement. We shall consider only the space required to store the data.

Let P_H and P_V denote the number of pages required to store a relation. Based on the design in Section 2 and the parameters in Table 1, we have to following equations.

Horizontal:

$$P_H = \left\lceil \frac{REL\_SZ}{\left\lfloor \frac{PAGE\_SZ}{TUP\_SZ} \right\rfloor} \right\rceil \tag{3}$$

Single vertical:

$$P_V = \sum_{k=1}^{NUM\_ATRIB} \left\lceil \frac{REL\_SZ}{\left\lfloor \frac{PAGE\_SZ}{ATRIB\_SZ_k} \right\rfloor} \right\rceil \tag{4}$$

To obtain the average memory space usage, we shall assume that attributes in a tuple are of the same size. Notice that not all combinations of parameter sizes in Table 2 are legitimate. For example, a tuple of size 100 will not have 10 attributes of size 15. In general, a combination is legitimate if

$$TUP\_SZ \ = \ ATRIB\_SZ \times NUM\_ATRIB. \tag{5}$$

We shall assume that the legitimate combinations of the parameters in Table 2 are *equally likely* to be implemented. We define the *memory implementation efficiency* μ to be the actual amount of memory space occupied by the number of pages used P divided by the amount of data stored. Mathematically,

$$\mu = \frac{P \times PAGE\_SZ}{REL\_SZ \times TUP\_SZ}. \tag{6}$$

Since there are ten possible relation sizes and each with nine legitimate combinations, the *expected memory implementation overhead* of horizontal partitioning vs. vertical partitioning, denoted by E, is computed by the next equation:

$$\begin{aligned} E &= \sum_{R_i} \frac{1}{10} \sum_{A_j} \sum_{B_k} \frac{1}{9} (\mu_{H,A_j,B_k} - \mu_{V,A_j,B_k}) \\ &= \frac{1}{90} \sum_{R_i} \sum_{A_j \times B_k} (\mu_{H,A_j,B_k} - \mu_{V,A_j,B_k}), \end{aligned} \tag{7}$$

where R_i, A_j, B_k represents various values of REL_SZ, TUP_SZ, $ATRIB\_SZ$, respectively, and μ_{H,A_j,B_k} (μ_{V,A_j,B_k}) denote the implementation efficiency of the horizontal (single vertical) partitioning technique under the legitimate combination $A_j \times B_k$, with

$$\begin{aligned} R_i &\in \{1000, 2000, \cdots, 10000\} \\ A_j \times B_k &\in \{(50,5),(100,5),(100,10), \\ &\quad (150,10),(150,15),(200,10), \\ &\quad (300,10),(300,15),(450,15)\}. \end{aligned}$$

Using Equation (7), we have found that $E \approx 23\%$, indicating that *horizontal partitioning is expected to take 23% more space to implement a relation than single vertical partitioning.*

In addition to the cost of memory space (property 1 in Table 1,) a few events in database processing are directly related to the amount of memory space used: loading a relation from secondary memory to main memory (property 2 in Table 1) and reorganization of relation (usually caused by schema modification i.e., properties 3 and 4, in horizontal partitioning.) In all of these events, it is obvious that single vertical partitioning is the preferable technique.

4 Execution Time

Performance of an MMDB in a processing environment is measured in execution time. The comparison in execution time is significantly different from the memory space requirement. (1) We can only account for the discrepancies incurred by the use of horizontal or single vertical related to the architecture design in Section 2, in other words, there is a cost common to both techniques that is not accounted. (2) Properties 5 – 8 constitute the mixed environment. Similar to Gruenwald and Eich [4], we consider the *frequency weighted cost discrepancy*: the cost discrepancy is weighted by the frequencies that the constituent properties are executed.

4.1 Execution time discrepancy analysis

Tuple selection The basic steps of tuple selection are (i) search the index structure and return the logical address of the tuple, (ii) perform address translation to map the logical tuple address to its physical equivalent, and (iii) read every attributes in the tuple.

Step (i) is identical to both partitioning techniques. The cost in step (ii) is computed by counting the arithmetic operations and memory references in equations (1) and (2). Since the page table is in the main memory, 1 main memory read is required to get the beginning address for a physical page. From equation (1), we see that 5 arithmetic operations are needed: $L \div T$ plus base address of the table to get the page address accounts for two, and mod, $\times$, and addition account for the other three. Thus, total processing time for horizontal to implement step (ii) is $t_r + 5t_a$. Similarly, total processing time for single vertical to implement step (ii) is $3t_r + 9t_a$.

In step (iii), horizontal partitioning requires one main memory reference plus one arithmetic operation per attribute in order to obtain the attribute boundaries in the tuple. For single vertical, since the attributes in a tuple is known from step (iii), no additional calculation is necessary.

The execution time discrepancies incurred in tuple selection are summarized in the following.

Horizontal:

$$T_{SH} = (NUM\_ATRIB+1) \times t_r + (NUM\_ATRIB+5) \times t_a \quad (8)$$

Single vertical:

$$T_{SV} = 3 \times NUM\_ATRIB \times t_r + 9 \times NUM\_ATRIB \times t_a \quad (9)$$

Tuple insertion, modification, and deletion Tuple insertion involves the following steps: (i) get a logical address from the free list for the newly inserted tuple, (ii) perform address translation, and (iii) write the new tuple. Only step (ii) incurs different cost in both techniques. The cost discrepancies are the same as in tuple selection and is given by (8) and (9).

The basic steps for tuple modification are the same as in tuple selection except in the step (iii), where a number of attributes are modified with memory write operations. For simplicity, we will assume that all attributes are modified. Given this assumption, the cost discrepancy is the same as in tuple selection.

Tuple deletion involves the following steps: (i) delete a node from the index structure, (ii) return the logical address of the deleted tuple to the free list, The costs of tuple deletion are the same for both partitioning techniques. Note that in [4, 5] the cost of tuple deletion is considered equivalent to that of tuple selection.

Attribute projection Projecting an attribute requires reading the same attribute for all tuples in the relation. This involves, for each tuple, (i) getting its logical address from the index structure, (ii) performing address translation, and (iii) reading the attribute.

Steps (i) and (iii) cost the same for both techniques. In step (ii), the physical address of the attribute projected in horizontal partitioning requires the cost of performing address translation in (1) plus an offset:

$$T_{PH} = (REL\_SZ + 1) \times t_r + 6 \times REL\_SZ \times t_a. \quad (10)$$

For single vertical partitioning, the address of attribute i is computed using (2). However, the attribute size and the number of attributes per page information need to be looked up only once, and the offset address in looking up the page table and the number of entries in a page table can be computed externally, the time discrepancy is

$$T_{PV} = (REL\_SZ + 2) \times t_r + (5 \times REL\_SZ + 4) \times t_a. \quad (11)$$

Table 3: Frequencies

name	*frequency of*
f_{sel}	tuple selection
f_{proj}	attribute projection
f_{ins}	tuple insertion
f_{mod}	tuple modification
f_{del}	tuple deletion
f_{tup}	$f_{sel} + f_{ins} + f_{mod}$

4.2 Performance boundary

We are now ready to derive the performance boundary for a mixed processing environment which includes tuple selection, insertion, modification, and deletion, and attribute projection. The performance boundary is derived based on the frequency f_{tup} that the tuple oriented properties are executed, where $f_{tup} = f_{sel} + f_{ins} + f_{mod}$. Horizontal partitioning performs better than single vertical if f_{tup} is greater than performance boundary. For convenience of reference, Table 3 lists the variable names for the frequencies that are used in the analysis. Also, we shall let $t_r = kt_a$, where $k > 1$ and k depends on the semiconductor technology used to implement the arithmetic unit and the main memory.

Notice that $f_{tup} + f_{proj} + f_{del} = 100\%$. Also, since both partitioning techniques incur the same cost for tuple deletion, the performance boundary is obtained by equating the frequency weighted discrepancy for horizontal and single vertical:

$$f_{tup}T_{SH} + f_{proj}T_{PH} = f_{tup}T_{SV} + f_{proj}T_{PV}.$$

Using the relation $t_r = kt_a$ and substituting $f_{proj} = (1 - f_{del}) - f_{tup}$, we have

$$f_{tup} = \frac{(1 - f_{del})(REL\_SZ - 4 - k)}{REL\_SZ + (2k+8)NUM\_ATRIB - (2k+9)}. \quad (12)$$

The empirical study of Joyce and Warn [6] showed that retrieval accounts for 80% of database access. Notice also that in update, the frequency for modification is the highest among the three basic properties. This

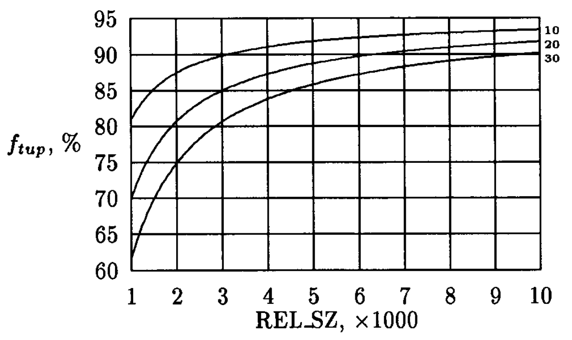

Figure 1: Curves of performance boundaries for mixed environment with $k = 5$

allows us to set f_{del} to no more than 5%. With this assumption, equation (12) becomes

$$f_{tup} \geq \frac{0.95(REL\_SZ - 4 - k)}{REL\_SZ + (2k + 8)NUM\_ATRIB - (2k + 9)}. \quad (13)$$

The performance boundary is plotted for $k = 5$ in Figure 1. The curves are labeled by the number of attributes (10,20,30). Horizontal axis is the relation size, and vertical axis is the combined percentage f_{tup} that Figure 1 indicates that

$$\begin{cases} \text{if } f_{tup} \leq 62\% & \text{single vertical is better} \\ \text{if } f_{tup} \geq 92\% & \text{horizontal is better} \\ \text{otherwise} & \text{depends on relation size} \\ & \text{and number of attributes} \end{cases}$$

Clearly, the performance boundary only says *when* one partitioning technique is better than the other, but not *how much*. We suspect that the latter question can only be answered when the common cost is completely accounted, or some other measure completely different from the one we are using in the paper must be used.

5 Conclusion

Performance of the horizontal partitioning technique and the single vertical partitioning techniques are evaluated for main memory database, using memory space and processing time. Cost models are based on the discrepancies incurred due to the different partitioning techniques used; a system architecture is used as a basis to obtain this cost model.

The conclusions are (1) single vertical is about 23% more space efficient. Since the amount of memory space used directly affects the loading cost and reorganization cost, single vertical is also preferable than horizontal in loading cost and schema modification. (2) Figure 1 indicates that if $f_{tup} \leq 62\%$ single vertical partitioning is always better; if $f_{tup} \geq 92\%$ horizontal partitioning is always better; otherwise, the performance of horizontal and single vertical depends on the relation size and the number of attributes.

References

[1] D. Bitton, D. DeWitt, and C. Turbyfill, "Benchmarking Database Systems: A Systematic Approach," *Proceedings of Very Large Scale Databases,* pp. 8-19, 1983.

[2] M. Eich. "Main Memory Database Research Directions," *SMU-CSE Technical Report* 88-CSE-35, November 1988.

[3] R. Epstein, "Creating and Maintaining a Database Using Ingres," *Electronics Research Laboratory, University of Calfornia, Berkeley, Memo ERLM77-71,* pp. 1-25, December 1977.

[4] L. Gruenwald and M. Eich, "Database Partitioning Techniques to Support Reload in a Main Memory Database System: MARS," *SMU-CSE Technical Report* 89-CSE-31, September 1989.

[5] L. Gruenwald and M. Eich, "Choosing the best Storage Techniques for a Main Memory Database System," *5th Jerusalem Conference on Information Technology,* pp. 1-10, October 1990.

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

[7] K. Salem and H. Garcia-Molina, "Crash Recovery Mechanisms for Main Storage Database Systems," *Princeton University, CS-TR-034-86,* April 1986.