

New Objective Function for Vertical Partitioning in Database System.

© Thanh Hung Ngo

Don State Technical University
nthungla@yahoo.com

PhD supervisor: Michael V. Grankov

Abstract.

In this paper we introduce the objective function for vertical partitioning in database systems. It has been built with the new evaluative criterion: cache hit probability. Testing the validity of the derived evaluative formula via program simulation shows its high accuracy.

Index terms: vertical partitioning, objective function, evaluation criterion, cache hit probability.

1 Introduction

Vertical partitioning of a table splits it into two or more tables (which we refer to *sub-tables*), each of which contains a disjoint subset of the *attributes* of the original table, except for the *key attributes*. Since a subset of attributes is more frequently accessed by transactions than the subset of the rest attributes, so vertical partitioning can reduce the amount of data that needs to be scanned to answer the transactions.

As indicated in [1], many data partitioning algorithms have been developed basing on statistical and pattern classification and analysis. They cluster data using various criteria. The most common used criterion is the square-error. The lack of these algorithms is that they don't estimate the "goodness" of partition schemes in the relation with an index of system performance. In [2], authors develop the complex method, in which they first create a set of "candidate" partition schemes, and then analyze each of them via optimization unit of database system to find the best scheme. The work is of great interested, but it is too complicated. On the other hand the performance of optimization unit strongly depends on the current tuning of database system.

In this paper we will build the objective function for vertical partitioning with the evaluative criterion: the cache hit probability. As many algorithms in this area

we use an attribute usage matrix (AUM) as the input data. We will consider that number of sub-relations equaling to two, and both of them being located in the same site.

2 Cache system and database system specification

Let's consider a table T in a database system. Tuples of T have M attributes, and cache memory for its caching is limited to L bytes.

Assume that each transaction queries a subset of attributes of a tuple by its record number in the table. It means that a transaction is a sequence of three operations: select table, select a subset of attributes and select a tuple by record number in table. Then a transaction can be defined as a triplet (T, Δ_j, ID_j) , where, T – table name, Δ_j – the sub-set of query attributes, ID_j – the query record number. Assume that these selects are independent between one and the others, and the select of record number obeys uniform law. Let us have Q different transactions and there attribute usage is described in table 1:

Trans\Attrs	A ₁	A ₂	...	A _M
Tr ₁	u ₁₁	u ₁₂	...	u _{1M}
Tr ₂	u ₂₁	u ₂₂	...	u _{2M}
...
Tr _Q	u _{Q1}	u _{Q2}	...	u _{QM}

Table 1: attribute usage matrix.

where u_{ij} equals either p_i or 0, and p_i is the execution probability of transaction Tr_i . Thus the following expressions are the case:

$$p_1 + p_2 + \dots + p_Q = 1,$$

$$0 \leq p_i \leq 1, \quad 1 \leq i \leq Q.$$

Cache is divided into parts, each of which contains tuples of one table. Write to cache and read from it are executed per tuple. A tuple is loaded into cache if at least one of its attribute is claimed. At first time cache is empty. In operation it is filled with tuples. For the current transaction if copy of the query tuple is in

cache, then it is fetched from cache to answer the query. Otherwise it is fetched from disk and its copy is inserted into cache. In the first case we say that it was a cache hit, in the last case – a cache miss. After cache having been filled, a new tuple will substitute one among the present tuples. We will consider that the cache replacement strategy being randomized. Since the period, while cache hasn't been filled yet, is a transitional period, we can derive the objective function, having skipped this period.

The assumptions, are described in this section, has been made as a result of deductive analysis the literature [1], [2], [3], [4]. Note that these assumptions, except for the attribute usage matrix, are only for the purpose of mathematical deriving and validity testing of the estimated formula, but not for implemented areas of the derived objective function.

3 Method implementing vertical partitioning

In this and in the following sections we will consider the vertical partitioning of table T into two sub-tables T_1, T_2 . Attribute sets of tables T, T_1, T_2 are S, S_1, S_2 correspondingly. These following expressions are the case:

$$\begin{aligned} S_1 \cup S_2 &= S, \\ S_1 \cap S_2 &= SK_T, \\ |S_1| &> |SK_T|, \\ |S_2| &\geq |SK_T|, \end{aligned}$$

where, SK_T – the sub-set, containing all key attribute of table T .

Then the table T is replaced with two sub-tables T_1, T_2 . Using method simulating the vertical partitioning [2], we create regular tables, one for each sub-table. The queries referencing the original table T , is rewritten to execute against the sub-tables T_1, T_2 . If assume that any query contains at least one of non-key attributes, then a query (T, A_j, ID_j) is replaced only in one of three variants:

- 1-st variant: by one query, referencing sub-table $T_1, (T_1, A_j, ID_j)$, if all the query attributes are in S_1 , i.e. $(A_j \subset S_1)$;
- 2-nd variant: by one query, referencing sub-table $T_2, (T_2, A_j, ID_j)$, if all the query attributes are in S_2 , i.e. $(A_j \subset S_2)$;
- 3-rd variant: by two queries $(T_1, A_j \cap S_1, ID_j)$ and $(T_2, A_j \cap (S_2 \setminus SK_T), ID_j)$ simultaneously in the rest cases, i.e. $((A_j \cap S_1) \neq \emptyset) \wedge ((A_j \cap S_2) \not\subset SK_T)$.

The cache allocated for caching the table T is now divided into two parts for caching its sub-tables. Sizes of these parts are the parameters for partitioning algorithm.

4 Deriving of the objective function

In this section we will derive the objective function, which estimate the cache hit probability for caching the original table and its sub-tables. The cache system and database system have been specified in section 2. The method implementing the vertical partitioning was described in section 3.

4.1 Preliminaries

The following are used in derivation of the objective function.

N – the tuple count of table T as well as T_1 and T_2 .

M – the attribute count of tuples of table T .

l_i – length of attribute i of table T for $i = 1, 2, \dots, M$.

l_T, l_{T_1}, l_{T_2} – tuple length of table T, T_1, T_2 .

L, L_1, L_2 – size of part of cache memory, allocated for caching table T, T_1, T_2 . L_1, L_2 – the parameters, which satisfy the restriction $(L_1 + L_2 \leq L)$.

K, K_1, K_2 – number of tuples of table T, T_1, T_2 , which can exist in the corresponding part of cache memory at the same time.

Operation $\lceil \rceil$ – operation achieving the integer part of a fraction.

4.2 Cache hit probability for caching the original table

In this case any transaction queries a tuple of table T , which may be one among K tuples, copy of which has already existed in cache, or may be one among the rest $(N - K)$ tuples, which have not any copies in cache. If the first occurrence is the case, finding of query tuple in cache will successfully finish, i.e. it will be a cache hit. Since the select of tuples obeys the uniform law, the cache hit probability is estimated by expression:

$$P_T = \frac{K}{N} \quad (1)$$

$$\text{where, } K = \min \left(N, \left\lceil \frac{L}{l_T} \right\rceil \right) = \min \left(N, \left\lceil \frac{L}{\sum_{\forall r: A_r \in S} l_r} \right\rceil \right).$$

4.3 Cache hit probability for caching the sub-tables

Let's mark the occurrence, when the original query is replaced in the first, the second, the third variant, the event C_1, C_2, C_3 ; the occurrence, when the finding of query tuple in cache finishes successfully, the event B . According to formula of complete probability, cache hit probability is estimated by expression:

$$P_{T_1 T_2} = \sum_{i=1}^3 (p(B | C_i) \cdot p(C_i))$$

For $i = 1$:

$$p(C_1) = \sum_{\forall j: A_j \subset S_1} p_j;$$

since the query (T, A_j, ID_j) is replaced by the query (T_1, A_j, ID_j) , then the cache accessing for the original query is successful, if and only

if the cache accessing for the substituting query is successful. Similarly to (1):

$$p(B/C_1) = \frac{K_1}{N}$$

with

$$K_1 = \min\left(N, \left\lceil \frac{L_1}{l_{T1}} \right\rceil\right) = \min\left(N, \left\lceil \frac{L_1}{\sum_{\forall r: A_r \in S_1} l_r} \right\rceil\right).$$

For $i = 2$: in the same way we can achieve

$$p(C_2) = \sum_{\forall j: \Delta_j \in S_2} p_j;$$

$$p(B/C_2) = \frac{K_2}{N}$$

with

$$K_2 = \min\left(N, \left\lceil \frac{L_2}{l_{T2}} \right\rceil\right) = \min\left(N, \left\lceil \frac{L_2}{\sum_{\forall r: A_r \in S_2} l_r} \right\rceil\right).$$

For $i = 3$:

$$p(C_3) = 1 - p(C_1) - p(C_2);$$

since the query (T, Δ_j, ID_j) is replaced by query $(T_1, \Delta_j \cap S_1, ID_j)$ and query $(T_2, \Delta_j \cap (S_2 \setminus SK_T), ID_j)$, then cache accessing for the original query is successful, if and only if the cache accessing for both of the substituting queries are successful.

$$p(B/C_3) = p(B_{T1}) \cdot p(B_{T2}) = \frac{K_1}{N} \cdot \frac{K_2}{N}$$

$$p(B/C_3) = \frac{K_1 \cdot K_2}{N^2}.$$

Thus the evaluative formula is

$$P_{T_1 T_2} = \frac{K_1}{N} \cdot p(C_1) + \frac{K_2}{N} \cdot p(C_2) + \frac{K_1 \cdot K_2}{N^2} \cdot p(C_3) \quad (2)$$

Let's mark

$$a = \frac{p(C_1)}{N}; \quad b = \frac{p(C_2)}{N}; \quad c = \frac{p(C_3)}{N^2}$$

Formula (2) is rewritten as

$$P_{T_1 T_2} = a \cdot K_1 + b \cdot K_2 + c \cdot K_1 \cdot K_2 \quad (3)$$

Notice: if $S_2 = SK_R$, then $p(C_2) = p(C_3) = 0$. In this case the partition is not the case. So the cache memory allocated for table T is completely assigned to table T_1 . As a result of (2) we achieve formula (1). So the case, when the original table is not partitioned, is only a special case of its partitioning.

4.4 The objective function

For a table T , a usage matrix of its attributes, lengths of its attributes and limited to L (bytes) cache memory,

allocated for its caching, the best scheme of partitions is defined as the result of the programming problem (*):

$$\mathbf{P}(S_1, K_1, K_2) = a \cdot K_1 + b \cdot K_2 + c \cdot K_1 \cdot K_2 \rightarrow \max \quad (4)$$

With these restrictions:

$$SK_T \subset S_1, S_1 \subseteq S, \quad (5)$$

$$K_1 \cdot l_{T1} + K_2 \cdot l_{T2} \leq L, \quad (6)$$

$$K_1 \geq 0, K_2 \geq 0, \quad (7)$$

$$K_1, K_2 \text{ are the integers} \quad (8)$$

Since the estimated formula is easy for computation, the programming problem (*) can be solved via full search algorithm for input data with small dimensionality. For large dimensionality it's necessary that the more effective algorithm have been developed.

5 Simulation program and validity testing

Validity testing of the derived estimated formula has been carried out via program simulation. The program has been written in Delphi. It uses the exhausted search algorithm to solve the programming problem (*). It allows simulating the performance of cache system and database system, which have been defined in section 2. It also allows registering the cache hit rate. Using the program, we have carried out a number of experiments with randomized data input. For all that we randomized the value of M and Q in interval $[5..10]$. The relative error in all these experiments is less than 2.5%. This confirms the high accuracy of derived formula.

For example let's consider the table T with 6 attributes, where A_1 is key attribute. Lengths of attributes are 30, 37, 51, 21, 46, 11 (bytes). The attribute usage matrix is defined in table 2.

Trans \ Attrs	A ₁	A ₂	A ₃	A ₄	A ₅	A ₆
Tr ₁	.194	0	.194	.194	0	.194
Tr ₂	.089	.089	0	0	0	.089
Tr ₃	.281	.281	0	0	.281	0
Tr ₄	.157	.157	0	0	.157	.157
Tr ₅	.279	0	0	0	.279	.279

Table 2: example of attribute usage matrix.

The cache size is 2000 (bytes), and the tuple count is 200 (tuples).

For this example the best scheme of partitions is

$$S_1 = \{A_1, A_2, A_5, A_6\}, \quad S_2 = \{A_1, A_3, A_4\},$$

and cache lengths for T_1, T_2 are

$$K_1 = 16, \quad K_2 = 0.$$

With this scheme the cache hit probability is 0.0649, and it has increased to 1.27 (times) according to non-partitioning case.

Simulation has been carried out with 10000 run-throughs, in each of which has been executed 5000 queries. Result gives the average value of cache hit rate equaling to 0.065.

6 Conclusions

In this paper we have derived the objective function for vertical partitioning with a new estimated criterion: cache hit probability. We also have carried out validity testing of the achieved formula via program simulation. The simplicity of the achieved formula and its high accuracy confirm the availability of its using in database design and database reconstruction to achieve the significant enhancement of system performance.

We are currently developing a heuristic algorithm for solving the programming problem (*). We will discuss it in future work.

References

- [1] Sharma Chakravarthy, Jaykumar Muthuraj, Ravi Varadarajan, Shamkant B. Navathe. An Objective Function for Vertically Partitioning Relations in Distributed Databases and its Analysis. In *Distributed and Parallel Databases* 2(2): 183-207(1994).
- [2] Sanjay Agrawal, Vivek Narasayya, Beverly Yang. Integrating Vertical and Horizontal Partitioning into Automated Physical Database Design. In *SIGMOD* 2004, June, 2004.
- [3] C. J. Date. An Introduction To Database Systems – Seventh edition. *Addision-Wesley Longman*, Inc, 2000.
- [4] William Page, David Austin, Willard Baird II, Nicholas Chase and others. Special Edition: Using Oracle8/8I. *QUE Corporation*, 1999.