

Query-Driven Horizontal Class Partitioning for Object-Oriented Databases*

Ladjel Bellatreche and Kamalakar Karlapalem and Gopal K. Basak

University of Science and Technology Clear Water Bay Kowloon, Hong Kong

E-mail : {ladjel, kamal}@cs.ust.hk E-mail: mabasak@uxmail.ust.hk

Abstract. Horizontal partitioning (HP) technique is a logical database design technique first developed for distributed relational (record structured) databases. Though the main objective of HP is to reduce irrelevant data access, there has been very little work done in quantifying this by means of a cost model, and developing algorithms which derive the horizontal partitioning schemes. In this paper, we develop a cost model for query processing in object oriented databases (OODBs) for both unpartitioned and horizontally partitioned object oriented databases. The problem of coming up optimal HP scheme is NP-complete. Therefore, we present and evaluate two algorithms for coming up with optimal (based on exhaustive search) and/or near optimal (based on heuristics) HP schemes.

1 Introduction

OODB technology has matured enough to support modern applications such as CAM/CAD, software engineering and multimedia databases. HP is often used as a means to achieve better performance out of database systems by reducing irrelevant disk accesses for executing a set of queries [6]. A partial list of benefits of horizontal class partitioning include: 1) different queries access only the needed fragments of a class, so HP will reduce the amount of irrelevant data accessed by queries, 2) HP reduces the amount of data transferred when processing queries across multiple sites distributed [4], 3) the decomposition of a class into horizontal class fragments (HCFs), each being treated as a unit, permits queries to be executed concurrently, and 4) HP can be treated as an object-oriented database schema design technique, which takes a class and generates a class hierarchy of HCFs so as to increase the efficiency of query execution, while providing additional semantics for the OODB scheme.

There has been little work done on the *HP* in OODBs [2, 4], in comparison to the relational model [3, 8, 7]. Ezeife et al. [4] presented a comprehensive set of algorithms for horizontally partitioning four class models: classes with simple attributes and methods, classes with complex attributes and simple methods, classes with simple attributes and complex methods, and classes with complex

* This research has been funded by RGC CERG HKUST747/96E

attributes and methods. They applied the concepts developed for relational models in [3, 8]. In [2] we presented primary and derived HP algorithms for generating HCFs of a given class. The primary algorithm is an adaptation of the horizontal partitioning algorithm proposed in the relational model [7]. The approach used is based on affinity between predicates, and we formalized “derived HP” concept and developed a scheme for generating derived HCFs. The work addressed in [2, 4] ignored the physical cost corresponding to the savings in the amount of irrelevant data accessed.

In this paper, we address the problem of HP, we show its utility for OODBs and finally, we present an algorithm for generating HCFs based on the cost model for executing a set queries. A major feature of this algorithm is that it minimizes the I/O disk accesses, and to the best of our knowledge, *it is the first algorithm* which uses the cost model for generating HCFs. The main contributions of this paper are:

1. Development of a mathematical formula to calculate the number of HP schemes for a given set of minterm predicates.
2. Development of a cost model for executing a set of queries on both horizontally partitioned classes and unpartitioned classes.
3. Development of two algorithms to achieve the horizontal class partitioning.

The rest of the paper is organized as follows : section 2 presents the cost model for unpartitioned classes and horizontally partitioned classes, section 3 presents two horizontal class partitioning algorithms and their evaluation, and section 4 presents a conclusion.

1.1 Basic Concepts

Definition 1. A *simple predicate* is a predicate defined on a simple attribute or a method and it is defined as [3]: *attribute/method operator value*, where operator is a comparison operator ($=, <, \leq, >, \geq, \neq$). The value is chosen from the domain of the attribute or the value returned by the method.

Let $\Pi = \{p_1, p_2, \dots, p_N\}$ be the set of all *simple predicates* defined on a class C . The set of *minterm predicates* $M = \{m_1, m_2, \dots, m_z\}$ is defined as [3] : $M = \{m_i/m_i = \bigwedge_{p_k \in \Pi} p_k^*\}$, where $p_k^* = p_k$ or $p_k^* = \neg p_k$.

Definition 2. A path P represents a branch in a class composition hierarchy and it is specified by: $C_1.A_1.A_2 \dots A_n$ ($n \geq 1$) where : C_1 is a class in the database schema, A_1 is an attribute of class C_1 , and A_i is an attribute of class C_i such that C_{i+1} is the domain of the attribute A_i ($1 < i \leq n - 1$). For the last class in the path C_n , you can either access an attribute A_n , or a method m_n that returns a value, or a set of OIDs. The length of the path P is defined by the number of attributes, n , in P . We call the first class C_1 the **starting class** and the last attribute (or method) A_n the **ending attribute or method** of the path. A predicate which is defined on path expression is called **component predicate** [2].

2 Cost Model for Horizontal Class Partitioning

Assume a database of object instances are stored on secondary memory which is divided into pages of fixed size. In this section, we present two analytical cost

models of executing a set of k queries $\{q_1, q_2, \dots, q_k\}$, where every query may contain a simple or component predicates. The first cost model is for executing these queries on unpartitioned classes (i.e., all classes in class composition hierarchy are unpartitioned), and second cost model is for executing the same queries on horizontally partitioned class(es) (i.e, we assume that there are some class(es) which are horizontally partitioned). The objective of our cost models is to calculate the cost of executing these queries, each of which accesses a set of objects. The cost of a query is directly proportional to the number of pages it accesses. The costs are calculated in terms of disk page accesses. The cost of executing a query is given by: **Total_Cost = IO_Cost + CPU_Cost + COM_Cost**, where *IO_Cost* is the input/output cost for reading and writing data between main memory and disk, *CPU_Cost* is the cost of executing CPU instructions, (for example, for evaluating predicate), and *COM_Cost* is the cost of network communication among different nodes. In this paper, we concentrate on the *IO_Cost* and disregard the *CPU_Cost* and *COM_Cost*. This is because for very large database applications with huge number of data accesses, the *CPU_Cost* contribution towards the *Total_Cost* will not be significant, and the database we consider is centralized, therefore the *COM_Cost* is not taken into account.

Cost Model Parameters

We can classify the parameters for the cost model into three categories: database, query and horizontal class fragments.

a- Database Parameters:

Cardinality of a class C_i ($|C_i|$) (i.e., number of objects), total number of pages occupied by the class C_i ($|C_i|$), object length or size (in bytes) in the class collection C_i (LC_i), and page size of the file system (in bytes) (PS).

b- Query Parameters :

Starting class of a component predicate $p_l(C_s^h)$, the number of object references for class C_i during the path evaluation process along the class composition hierarchy (REF_i), number of predicates used by all queries (N), selectivity of query q_h (SEL_h), the fan-out for the class composition hierarchy from class C_{i-1} to C_i ($Fan(C_{i-1}, C_i)$), selectivity of predicate p_l (sel_l), and the length of output result (L_{proj}).

c- Horizontal class fragments Parameters :

Number of HCFs of class C_i (m_i), cardinality of a HCF F_j ($|F_j|$) (i.e., number of objects), total number of pages occupied by the HCF F_j ($|F_j|$), the number of object references for HCF F_j during the path evaluation process along the class composition hierarchy (ref_j), selectivity of HCF F_j ($sel(F_j)$).

2.1 Cost of Executing a Single Query on Unpartitioned Classes

In order to make easier the formulation of the I/O cost model for both unpartitioned and horizontally partitioned classes, we develop it for a query q_h as shown Fig. 1, we assume that q_h contains one component predicate p_l . In the next subsection, we shall generalize this formulation for k queries. We assume that the starting class of component predicate as C_s^h and ending class as C_n^h . The length of path from C_s^h to C_n^h is n_h .

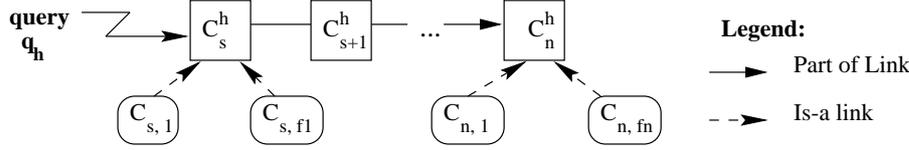


Fig. 1. Example of Unpartitioned Classes

Our cost model is based on [5]'s formulation. The cost for the query execution can be broken up into 2 components: the cost of evaluating a predicate (CEP), and the cost of building the output result (CBOR) which correspond to the number of pages to be loaded in order to evaluate a predicate, and the number of pages to be loaded to build the result, respectively.

Estimation of the Number of Pages in a Class Collection

Sequential scan and index scan are the two major strategies used for scanning a class collection. The objective of using an index is to attain faster instances access, while the objective of using horizontal class partitioning is to reduce irrelevant instances access. These two objectives are orthogonal and complementary. We concentrate on the sequential scan strategy; the use of index can also be incorporated into our model naturally as needed. We assume that the objects are smaller than the page size and do not to cross page boundaries. The total

number of pages occupied by a class collection C is given by: $|C| = \left\lceil \frac{\|C\|}{\lfloor \frac{PS}{LC} \rfloor} \right\rceil$

where $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$ are the ceiling and floor functions, respectively. The same formula can be used for a HCF F as follows: $|F| = \left\lceil \frac{\|F\|}{\lfloor \frac{PS}{LC} \rfloor} \right\rceil$

Estimation of the Number of Page Accesses for Predicate Evaluation

Let q_h be a query having a component predicate p_l with sel_l as its selectivity. We first define the number of distinct references involved in the path [5] $REF_i = (1 - Prob_i) \times \|C_i^h\|$, where $Prob_i$ is the probability of an object of collection C_i^h not involved in the path, we have:

$$Prob_i = \left(1 - \frac{1}{\|C_i^h\|}\right)^{(REF_{i-1} \times sel_l \times FAN(C_{i-1}^h, C_i^h))}, \text{ with } REF_1 = \|C_s^h\|.$$

In order to estimate the number of page accesses in class collection C_i^h (see Figure 2) during the evaluation of a predicate, we use the Yao function [9]: given n records uniformly distributed into m blocks ($1 \leq m \leq n$), each contains n/m records. If k records ($k \leq n$) are randomly selected from n records, the expected

number of page accesses is given by: $Yao(n, m, k) = m \times \left[1 - \prod_{i=1}^k \frac{n \times d - i + 1}{n - i + 1}\right]$

where $d = 1 - \frac{1}{m}$. We use the Yao's formula with $n = \|C_i^h\|$, $m = |C_i^h|$ and $k = REF_i$ as defined above.

Cost Formulate for Query Execution

The total cost of executing the query q_h having a component predicate p_l defined on unpartitioned classes is given by the following equation:

$Total_Cost = CEP + CBOR$, with: $CEP = \sum_{i=s}^n Yao(\|C_i^h\|, |C_i^h|, REF_i)$ and

$$CBOR = \left\lceil \frac{SEL_h \times \|C_s^h\|}{\lfloor \frac{PS}{Lprob} \rfloor} \right\rceil$$

Cost of Executing a Set of Queries on Unpartitioned Classes

We now, generalize the total cost for executing k queries $\{q_1, q_2, \dots, q_k\}$ which is given by the following equation:

$$Total_Cost = \sum_{h=1}^k \left[\sum_{i=s}^n Yao(|C_i^h|, |C_i^h|, REF_i) \right] + \sum_{h=1}^k \left[\left\lceil \frac{SEL_h \times |C_s^h|}{\lfloor \frac{PS}{Lprob} \rfloor} \right\rceil \right]$$

We note that all these equations are for queries having one component predicate. For a query having more than one predicate, we calculate the I/O cost for *each predicate* based on above equations, and finally we sum up all these I/O costs.

2.2 Cost of Executing a Single Query on Partitioned Classes

Let C_i^h be a class in the class composition hierarchy which is horizontally partitioned into m_i HCFs. Let us consider a query q_h having a component predicate invoking C_i^h as shown in Fig. 2. Before describing the cost on horizontally partitioned classes, we first estimate the number of page accesses in HCF F_j during the evaluation of a predicate. We can apply the same formula for unpartitioned class due to the representation of HCF as a class. Then the number of page accesses is given by: $Yao(|F_j|, |F_j|, ref_j)$ where: $ref_j = (1 - Prob_j) \times |F_j|$ where $Prob_j$ is the probability of an object of HCF F_j to be not involved in the path, we have: $Prob_j = (1 - \frac{1}{|F_j|})^{(REF_{i-1} \times sel_i \times FAN(F_j^h, C_{i+1}^h))}$

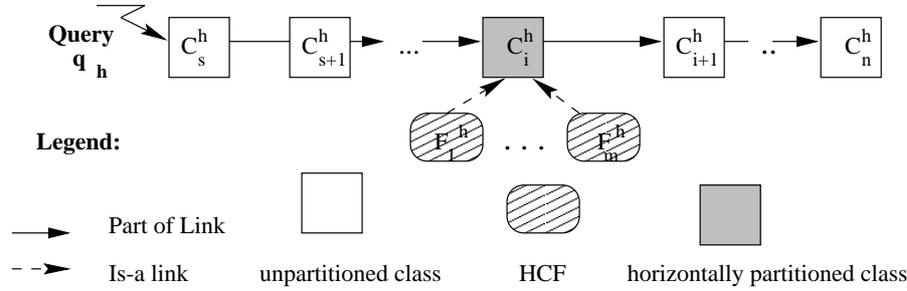


Fig. 2. Example Horizontal Class Partitioning

The predicate evaluation cost (CEP) is the sum of Yao's function over the classes involved in the query, but we need to consider the effect of HP of the class C_i^h which is represented by the binary variable $valid(p_l, F_j)$ that has the value 1 if the predicate p_l is valid in HCF F_j , 0 otherwise. That means we load only the HCFs which satisfy the predicates defined by the query:

$$CEP = \sum_{(i=s, i \neq h)}^n Yao(|C_i^h|, |C_i^h|, REF_i) + \sum_{i=1}^{m_i} valid(p_l, F_i) \times Yao(|F_i|, |F_i|, ref_i),$$

$$\text{and the cost of building the output result is: } CBOR = \left\lceil \frac{SEL \times |C_s^h|}{\lfloor \frac{PS}{Lprob} \rfloor} \right\rceil$$

We generalize the above formulas for i ($i > 1$) classes which are horizontally partitioned. We introduce a binary variable H_i which is of value 1, if the class C_i is horizontally partitioned, 0 otherwise.

The predicate evaluation cost will be defined by the following equation:

$$CEP = \sum_{i=s}^n \left[H_i \times \sum_{j=1}^{m_i} (valid(p_l, F_j) \times Yao(|F_j|, |F_j|, ref_j)) + (1 - H_i) \times Yao(|C_i^h|, |C_i^h|, REF_i) \right], \text{ the CBOR is given by:}$$

$$CBOR = (1 - H_s) \times \left\lceil \frac{SEL \times |C_i^h|}{\lfloor \frac{PS}{L_{proj}} \rfloor} \right\rceil + H_s \times \sum_{j=1}^{m_s} (valid(p_l, F_j) \times \left\lceil \frac{SEL \times |F_j|}{\lfloor \frac{PS}{L_{proj}} \rfloor} \right\rceil)$$

Based on the cost model for both unpartitioned and horizontally partitioned, we can make the following comparison in order to show the utility of HP. In unpartitioned classes, all instances of these classes are loaded into main memory, but in horizontally partitioned classes; the query needs to load only the HCFs which contribute to the result, that it is done by the valid function. We note that $|F_j|$ is always less than $|C_i^h|$ for horizontally partitioned classes, and then $Yao(|F_j|, |F_j|, ref_j) < Yao(|C_i^h|, |C_i^h|, REF_i)$. In this case, the HP will always improve the performance of evaluating the predicate, but if the selectivity of the query is very high, which means all HCFs of the class C_i^h are relevant to the query, there is an extra overhead which is due to cost of applying the *union operation* to reconstruct the result of a query. This overhead may deteriorate the performance of the HP.

3 Horizontal Class Partitioning Algorithms

3.1 Cost-Driven Horizontal Partitioning Algorithm

We define that the result of the HP process as a partitioning *scheme* which represents a set of HCFs. We consider set of queries accessing C_h using N simple predicates $\{p_1, p_2, \dots, p_N\}$. We assume that these simple predicates are *non-overlapping*, *complete* and *minimal* [8]. From these predicates, we generate the set $M = \{m_1, m_2, \dots, m_z\}$ of minterm predicates. We note that a minterm or a combination of several minterms using disjunction connector (\vee) forms one HCF denoted by $()$. From the 8 minterm predicates, we have several possibilities of HP scheme of a given class. For example: $[(m_1 \vee m_2), (m_3), (m_4 \vee m_5), (m_6 \vee m_7), (m_8)]$ forms a scheme with 5 HCFs. The number of all possible HP schemes for a given z minterm predicates is given by the theorem 1.

Theorem 1. Let z be the number of minterm predicates and S_z be the all possible schemes. Then S_z is given by: $S_z = \sum_{k=1}^z S_z^k$, where: S_z^k represents all schemes defined for z minterm predicates having k HCFs, with $S_z^k = k * S_{z-1}^k + S_{z-1}^{k-1}$.

Proof. See [1]. \square

We assume that the class C_h will be horizontally partitioned, and let $Q = \{q_1, q_2, \dots, q_k\}$ be set of queries accessing a class composition hierarchy including the class C_h . From the set of simple predicates $P = \{p_1, p_2, \dots, p_N\}$ defined on C_h , we generate all minterm predicates $\{m_1, m_2, \dots, m_z\}$, after that, we exhaustively enumerate all possible schemes S_z (see Theorem 1). For each scheme, we calculate the cost of executing all these queries. Finally, the scheme with minimal cost gives the best HP scheme of the class C_h .

We note that this algorithm needs an exhaustive enumeration strategy to generate all schemes. For small values of minterms, this procedure is not computationally expensive. However, for large values of number of minterms, the computation is a very expensive, for example, for 15 minterms, the number of schemes is 1382958545.

3.2 Approximate Horizontal Class Partitioning Algorithm

We note that the cost driven algorithm has exponential time complexity to generate all schemes and that the predicate affinity algorithm(PAA) [2] is very efficient algorithm ($O(k * N^2)$, where N and k represent the number of queries and their predicates, respectively), but in most cases it does not give the optimal HP scheme that minimizes the total number of disk accesses to execute all queries. Therefore, we have developed an approximate algorithm which takes the advantages of both algorithms PAA and CDA. It is based on hill-climbing technique in order to find a near optimal solution. Before the introduction of the approximate algorithm, we consider some basic concepts.

Representation of Horizontal Class Fragments

We note that each attribute or method has a domain(dom), then when a class is horizontally partitioned, the predicates define partitions of domain of each attribute and method. The cross product of partitions of an attribute or method by all the predicates determine a partitioning of the domains of all the attributes or methods into sub_domains.

Fragment	Duration		Cost		Location	
	d ₁₁	d ₁₂	d ₂₁	d ₂₂	d ₃₁	d ₃₂
F ₁	1	0	1	0	1	0
F ₂	1	0	0	1	1	0
F ₃	1	0	0	1	0	1
F ₄	1	0	1	0	0	1

Table 1. Representation of Class Fragments Using Sub-Domains

Example We assume that class *Project* (given in reference [2]) has been horizontally partitioned into four HCFs as follows:

F_1 given by clause $cl_1 : (\text{Duration} \leq 4) \wedge (\text{Cost}() \leq 7000) \wedge (\text{Location} = \text{"CA"})$, F_2 given by clause $cl_2 : (\text{Duration} \leq 4) \wedge (\text{Cost}() > 7000) \wedge (\text{Location} = \text{"CA"})$, F_3 given by clause $cl_3 : (\text{Duration} \leq 4) \wedge (\text{Cost}() \leq 7000) \wedge (\text{Location} = \text{"LA"})$, F_4 given by clause $cl_4 : (\text{Duration} \leq 4) \wedge (\text{Cost}() > 7000) \wedge (\text{Location} = \text{"LA"})$. The domains of the attributes and return values of the method are: $\text{dom}(\text{Duration}) =]0, 7]$, $\text{dom}(\text{Cost}()) =]0, 20000]$, $\text{dom}(\text{Location}) = \{\text{"CA"}, \text{"LA"}\}$. The partitioning of the attributes and methods defined in HCFs are as follows: $\text{dom}(\text{Duration}) = d_{11} \cup d_{12}$, where $d_{11} =]0, 4]$, $d_{12} =]4, 7]$, $\text{dom}(\text{Cost}()) = d_{21} \cup d_{22}$, where $d_{21} =]0, 7000]$, $d_{22} =]7000, 20000]$, $\text{dom}(\text{Location}) = d_{31} \cup d_{32}$, where $d_{31} = \text{"CA"}$ and $d_{32} = \text{"LA"}$. From sub-domains, we can represent all HCFs by a table whose rows and columns represent the HCFs and sub_domains respectively, as shown in Table 1. The value (F_i, d_{jk}) equals 1 if the HCF F_i is valid in sub-domain d_{jk} , otherwise, it is 0. For example, the value (F_1, d_{21}) is 1, because the HCF F_1 is defined by $\text{cost}() \leq 7000$. From Table 1, we define two operations: $\text{shrink}()$ and $\text{expand}()$ as follows:

- **Shrink()**: If the HCFs F_i and F_j are defined by two clauses which differ in one predicate p_k and if there are some queries with high frequencies which do not access predicate p_k , then we shrink these two HCFs into one HCF without

p_k . For example, the HCFs F_2 and F_3 have the same two predicates, except one predicate which is defined on Location attribute, then we apply the shrink operation, and we obtain: F_{23} : $(Duration \leq 4) \wedge (Cost > 7000)$.

• **Expand()**: If there is a HCF F_j of the HP scheme which is defined by a clause having f simple predicates, and if there is a query accessing the class using a clause with g , ($g > f$) simple predicates, then we expand the clause of F_j by augmenting additional simple predicates defined on the new attributes or methods. For example, we suppose that there is a query accessing class *Project* (given in reference [2]) with the following clause: $(Duration \leq 4) \wedge (Cost > 7000) \wedge (Location = \text{"CA"})$. Let F_j be a HCF defined by the clause cl : $(Duration \leq 4) \wedge (Cost() > 7000)$. In this case, F_j can be expanded into two HCFs F_{j1} and F_{j2} defined by the clauses cl_{j1} and cl_{j2} which cover the location attribute (we note that the location attribute has domain ("CA" and "LA")).

cl_{j1} : $(Duration \leq 4) \wedge (Cost() > 7000) \wedge (Location = \text{"CA"})$.

cl_{j2} : $(Duration \leq 4) \wedge (Cost() > 7000) \wedge (Location = \text{"LA"})$.

We conclude that the two operations(Expand and Shrink) defined above are realized by processing Table 1, for details refer to the technical report [1].

Approximate Algorithm

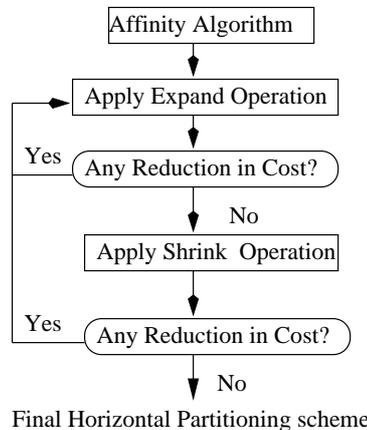


Fig. 3. The Steps of the Approximative Algorithm

We note that a hill-climbing technique needs four parameters: 1) the initial state, 2) how generate the next step, 3) determine the best state, and 4) stop test. The *initial state* is obtained by applying the affinity algorithm [2]. This is a good choice because the affinity algorithm is based on the best heuristic HP algorithm. We note that the result of this algorithm is a set of HCFs and each is HCF represented by one clause. In the *next step* a new scheme for HP is determined by applying the two operations (shrink and expand) described above. The *best state* is obtained by using the cost-driven approach's cost formulas to calculate the cost required for each of the new scheme to see if it is of lower cost than the cost of the current scheme. The *algorithm ends* if there is no new scheme with cost lower cost than the current scheme. The steps of this algorithm are shown in Fig. 3. In approximate algorithm, we apply expand() operation first,

and then shrink() (see Fig. 3), because the application of expand() operation generates many HCFs, and a query accesses only those HCFs having relevant data. The disadvantage of this operation is generation of many HCFs and then an additional cost is occurred due to the union operation.

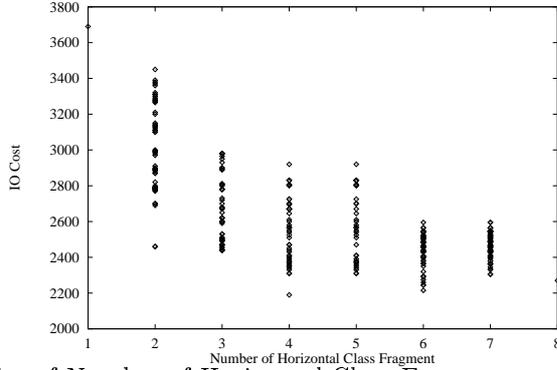


Fig. 4. Plot of Number of Horizontal Class Fragments vs. Total Cost

3.3 Evaluation of the Algorithms

We shall now take an example to compare the three algorithms: cost-driven, PAA, AA. We assume that the class *Project* will be horizontally partitioned, and we suppose there are queries accessing this class such as, $\{v.PId; v/ Project; v.Duration \leq 4 \wedge v.Location = \text{"CA"}\}$ (see the technical report [1] for details). From these queries, we enumerate all simple predicates, namely, $p_1: Duration \leq 4$, $p_2: Cost() \leq 7000$, $p_3: Cost() > 7000$, $p_4: Location = \text{"CA"}$, and $p_5: Location = \text{"LA"}$.

First, we apply the Cost-driven algorithm which generates 8 minterms from the above predicates. After that we generate **4140** HP schemes (see Theorem 1). For each scheme, we calculate the query processing cost. The plot of the total cost against number of HCFs is shown in Fig. 4. The best HP scheme is:

$CD_1: (Duration \leq 4) \vee [(Duration > 4) \wedge (Cost() > 7000) \wedge (Location = \text{"LA"})]$,
 $CD_2: (Duration > 4) \wedge (Cost() \leq 7000) \wedge (Location = \text{"CA"})$, $CD_3: (Duration > 4) \wedge (Cost() > 7000) \wedge (Location = \text{"CA"})$, $CD_4: (Duration > 4) \wedge (Cost() \leq 7000) \wedge (Location = \text{"LA"})$ with cost 2190.

After that, we apply the predicate affinity algorithm. Following HCFs are got:
 PA_1 with clause $cl_1: (Duration \leq 4) \wedge (Cost() \leq 7000) \wedge (Location = \text{"CA"})$,
 PA_2 with clause $cl_2: (Duration \leq 4) \wedge (Cost() > 7000) \wedge (Location = \text{"CA"})$,
 PA_3 with clause $cl_3: (Duration \leq 4) \wedge (Cost() \leq 7000) \wedge (Location = \text{"LA"})$,
 PA_4 with clause $cl_4: (Duration \leq 4) \wedge (Cost() > 7000) \wedge (Location = \text{"LA"})$,
 PA_5 with clause $cl_5: Duration > 4$.

The scheme obtained is one among those showed in Fig. 4 with cost 2830.

Finally, we apply the approximate algorithm which uses expand and shrink operations to HCFs generated by PAA so as to further reduce the total cost. The scheme obtained by AA has cost of 2520. We note that one shrink operation has been applied to the HCFs obtained by PAA. The HCFs produced by the

approximate algorithm are:

AA_1 with clause $cl'_1 : (\text{Duration} \leq 4) \wedge (\text{Location} = \text{"CA"})$, AA_2 with clause $cl'_2 : (\text{Duration} \leq 4) \wedge (\text{Cost}() \leq 7000) \wedge (\text{Location} = \text{"LA"})$, AA_3 with clause $cl'_3 : (\text{Duration} \leq 4) \wedge (\text{Cost}() > 7000) \wedge (\text{Location} = \text{"LA"})$, AA_4 with clause $cl'_4 : \text{Duration} > 4$.

4 Conclusion

In this paper, we addressed the problem of horizontal class partitioning in object oriented databases. We developed the cost model for executing a query in both unpartitioned and horizontally partitioned class. We have developed two algorithms namely, a cost-driven algorithm that uses the cost model for query execution to exhaustively search all schemes to find the optimal HP scheme, and an approximate algorithm(AA), wherein an initial solution got by PAA is refined by using a hill-climbing technique. The preliminary results indicate that the HP is a good technique to optimize the query processing. Further, the approximate algorithm provides better quality solution in comparison with PAA, and has lot less computational complexity in comparison with cost-driven algorithm. Our future work is based on executing queries on both horizontally partitioned classes and unpartitioned classes, allocating these HCFs to different sites in a distributed environment.

References

1. L. Bellatreche, K. Karlapalem, and G. K. Basak. Horizontal class partitioning for queries in object oriented databases. Technical Report HKUST-CS98-6, 1998.
2. L. Bellatreche, K. Karlapalem, and A. Simonet. Horizontal class partitioning in object-oriented databases. in *8th International Conference on Database and Expert Systems Applications (DEXA'97)*, Toulouse, *Lecture Notes in Computer Science 1308*, pages 58–67, September 1997.
3. S. Ceri, M. Negri, and G. Pelagatti. Horizontal data partitioning in database design. *Proceedings of the ACM SIGMOD International Conference on Management of Data. SIGPLAN Notices*, pages 128–136, 1982.
4. C. I. Ezeife and K. Barker. A comprehensive approach to horizontal class fragmentation in distributed object based system. *International Journal of Distributed and Parallel Databases*, 3(3):247–272, 1995.
5. G. Gardarin, J.-R. Gruser, and Z.-H. Tang. A cost model for clustered object-oriented databases. *VLDB*, pages 323–334, 1995.
6. K. Karlapalem, S.B. Navathe, and M. M. A. Morsi. Issues in distributed design of object-oriented databases. In *Distributed Object Management*, pages 148–165. Morgan Kaufman Publishers Inc., 1994.
7. S.B. Navathe, K. Karlapalem, and M. Ra. A mixed partitioning methodology for distributed database design. *Journal of Computer and Software Engineering*, 3(4):395–426, 1995.
8. M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, 1991.
9. S. B. Yao. Approximating the number of accesses in database organizations. *Communication of the ACM*, 20(4):260, April 1977.

This article was processed using the L^AT_EX macro package with LLNCS style