A MULTIKEY HASHING SCHEME USING PREDICATE TREES

Patrick VALDURIEZ, Yann VIEMONT

SABRE Project
INRIA, BP.105, 78153 Le Chesnay-Cédex, France

ABSTRACT

A new method for multikey access suitable for dynamic files is proposed that transforms multiple key values into a logical address. This method is based on a new structure, called predicate tree, that represents the function applied to several keys. A predicate tree permits to specify in a unified way various hashing schemes by allowing for different definitions of predicates. A logical address qualifies a space partition of a file according to its predicate tree. This address is seen as a single key by a digital hashing method which transforms it into a physical address. This method is used to address records in a file and to transform a retrieval qualification on a file into a set of partitions to access. Finally, a qualitative analysis of the behavior of the method is given which exhibits its value.

1. INTRODUCTION

The advent of relational database systems has greatly contributed to recent studies on multikey access methods. Non-procedural languages of these systems encourage the user to formulate complex queries, such as retrieval of a set of records that satisfy an expression involving more than one attribute. A retrieval expression is given in a conjunctive (or disjunctive) form of predicates (attribute op value), where op is one of the operators $=$, $<$, $\leq$, $\geq$, $\neq$.

Several access methods proposed for multikey searching have been reported in [BENT79]. Most of them manifest several deficiencies for handling dynamic files (files whose volume may vary rapidly). The problem of graceful adaptation to the content of changing files is well solved by two basically distinct classes of structures, balanced trees [BAYE72, COME79] and various dynamic hashings [FAGI79, LARS78, LITW78]. However, these structures are designed for the case where the access of records is based on a single key. B+-trees are an index organization which supports random accessing based on a key value as well as sequential accessing of records by key order. The key to address transform is computed by searching the index. Hashing provides random accessing only but generally with better performance than balanced trees, since the data structure it requires to represent the hashing function is smaller than an index. Intermediate structures appeared more recently for merging the advantages of both trees and dynamic hashing. They are essentially based on digital hashing [LITW81, LOME83]. Trie hashing [LITW81] represents the hashing function by a dynamic data structure, called trie, organized as a binary tree which stores values of key digits as separators to direct the search down to a correct leaf. DL-trees [LOME83] can be seen as variants of B+-trees where all records in a leaf share a common prefix of the key. Splitting a node which overflows is based on the bit of the key following the prefix P, so that records whose key is prefixed by P'0' are moved to a node while the others of prefix P'1' go to another node.

Extensions of dynamic hashing to multikey access have then been proposed by at least two methods. The grid file [NIEV81] is a multikey file structure where all the attributes are treated symetrically. Space partitionning is done by alternately using different attributes. Multidimensional linear hashing [OUKS83] is a linear hashing where successive expansions of the file are performed with functions on different attributes. This method treats the attributes in a hierarchical way.

An alternative to these schemes is to simply encode several keys into a single key which can be used by a classical method such as B+-trees [BLAS77]. The requirements for the encoding method are therefore that lexicographic order must be preserved. Such a method is heavily based on multikey order. The advantage is that no extension is needed to the unique key access method.

The variety of the proposed access methods is caused by the various applications for which they have been thought. This implies a serious problem for one who must choose an access method, since it can be restricted to a narrow range of applications. It has been shown [KNUT73] that a lot of hashing functions exist each having a privileged application domain.

This paper tries to bring a solution by a multikey hashing scheme. The function developped by the algorithm transforms multiple key values into a logical address, represented by a structure called predicate tree [GARD84]. A predicate tree (PT for short) permits to specify in a unified way various hashing methods, such as dynamic hashing, trie hashing, multi-attribute hashing and others. It can be seen as a generalization of these schemes. A PT defines space partitionnings of a file and is used for generating logical addresses based on the content of records. A logical address is represented by a bit string, seen as a single key by a digital hashing method [GARD84] employed for managing a directory which does the logical physical mapping for accessing records. PTs with the digital hashing method constitute the multikey access method of the SABRE relational database system [GARD83].

This paper is organized as follows. The PT structure and its powerful features are defined and illustrated in Section 2. The multikey hashing method using a PT is presented in Section 3. This method is used to find the logical addresses of records that must be inserted or deleted or a set of logical addresses of records specified by predicates. Section 4 gives a qualitative analysis of the proposed method in comparison with the multifield encoding method of [BLAS77]. Section 5 concludes and points out extensions of the method in a near future.

2. PREDICATE TREES

2.1. Definition

The primary requirement for a powerful and flexible multikey to address transform method is the ability of capturing and efficiently using information about the distribution of keys. The distribution of key values is seldom uniform, while the space partitionning of a file is wished as uniform. The method herein reported is based on the knowledge of the definition domain of some keys for partitionning a file. Another requirement is the ability to give some different weight to each key which can be used to define a hierarchy of these keys and hence a hierarchical partitionning. If these requirements are filled, it is then possible to favour certain requests more than others. A database administrator can thus precise the more frequent retrieval requests in order to minimize their processing time. PTs meet these needs in allowing for various ways to specify definition domains of keys of a file by predicates in a hierarchical fashion.

A more formal definition of PT is now presented. A predicate tree of height p associated with a file is defined by a balanced tree as follows. Each level i of the p levels describes a space partitionning of the file according to mi predicates {Pi1, Pi2, ..., Pimi}. A predicate Pij is a disjunctive normal form of atomic predicates. Each atomic predicate is of the form f (att) θ value, where att is an attribute of the file, θ an operator chosen among {=,≤,≥,<,>} and f a hashing function which can be omitted. Each node of level (i-1) has mi sons corresponding to the predicates Pi1, Pi2, ... Pimi. For a level i of a file F, there are mi partitions Fij such as :

1) for all j, for all k such that $j \neq k$:
   $Fij \cap Fik = \emptyset$
2) $F = \bigcup_{j=1}^{mi} Fij$

The number mi of predicates defining level i is called branching factor of level (i-1). The number of leaves of a PT is then $\prod_{i=1}^{p} mi$. A path from the root to a node is specified by the conjunction of predicates located in the nodes of this path. The access to a node in a PT starts from the root to the searched node.

Allowing a hierarchy of the p sets of predicates is an important feature of PTs. It permits the assignment of a decreasing priority to the sets of predicates. The predicates are more favoured as they are close to the root of the tree. For example, let us consider two sets {P11, P12, P13, P14} and {P21, P22}. The resulting PT is portrayed Figure 1.

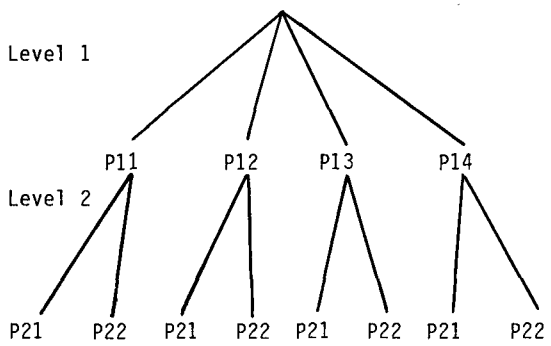

Figure 1. : First example of a predicate tree

The access to the partition defined by predicate P12 is done by locating only one node. The access to the partition of P22 needs to find four nodes (one for each predicate of level 1). However, the access to the partition of (P12 and P22) corresponds to only one node.

As a more meaningful example, let us consider the file containing the relation WINE (VINTAGE, YEAR, AREA, DEGREE, COLOR) and the following two sets of predicates expressed in a tuple relational calculus like language.

p1 = (YEAR≤1970 ; YEAR>1970 and ≤1980 ; YEAR>1980)

p2 = (AREA='Bordeaux" ; AREA='Bourgogne' ;
AREA='California' or 'New-York' ; OTHERS)

OTHERS is a key-word for generating the complement predicate (AREA≠'Bordeaux' and ≠'Bourgogne' and ≠'California and ≠'New-York'). The PT defined by {p1, p2} is depicted in Figure 2. The encircled predicate corresponds to the restriction predicate permitting to obtain the WINES for which their year is between 1970 and 1980 and their area is Bourgogne.

2.2. Properties of predicate trees

Let {A1, A2, ..., An} be a set of certain (possibly not all) attributes of a file F, each attribute Ai

takes values in a definition domain Di. The hashing function represented by the PT on the file F associates to each tuple of D1xD2x...xDn a space partition of F. The primary use of PTs is therefore the clustering of records of a file having common features, defined by predicates. In contrast to classical tree structures, a PT is not affected by variations of the content of the file since it defines a logical partitionning of the file. The logical-physical mapping can be ensured by a unique key access method based on digital hashing, which materializes the hierarchical partitionning defined by the PT according to a splitting policy. The digital access method is reported in [GARD84].

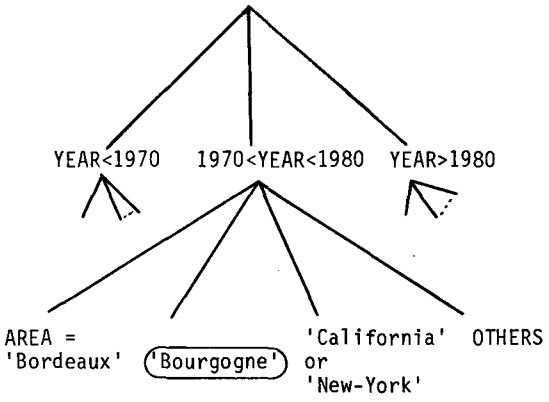

Figure 2. : A predicate tree for relation WINE

Parameters of a PT are the definitions of predicates at each level. Certain definitions of levels permit to implement as a particular case the same hashing functions of various known methods, such as classical hashing, variants of dynamic hashing, grid file and many others. The PT structure can hence be seen as a generalization of dynamic hashing schemes. This property is illustrated in the following.

A classical hashing scheme with rehashing for the management of overflows can be simply implemented by a PT with two levels, defined by h1(K)=K mod P and h2(K)=K mod Q, where h1 and h2 are remainder modulo P and Q respectively of the key K, P and Q being the number of primary and secondary buckets respectively. The first level generates predicates h1(K)=0, h1(K)=1,..., h1(K)=P-1 while the second one generates h2(K)=0, h2(K)=1, ..., H1(K)=Q-1. Overflows are then handled by the tree.

Dynamic hashing schemes are based on a hierarchy of hashing functions h0, h1, ..., hp applied to a key K. The idea is to dynamically change the hashing function according to variations of the content of the file. The functions are recursively defined as follows, where P is the number of primary buckets.

```
h0 (K) = K mod P
h1 (K) = h0 (K) or h0 (K) + P
.
.
hi (K) = hi-1 (K) or hi-1 (K) + 2**(i-1)*P
```

The PT of Figure 3 represents a dynamic hashing function. Level 0 is defined by h0(K). A level i is defined by the hashing function :

```
hi (K) = CHARi (K mod (2**i * P))
```

Let Pb be the number of bits to code the number P, the function CHARi (a) gives the ith leftmost digit of a, defined as a binary string of length Pb + i. The result '0' corresponds to hi-1 (K) while '1' corresponds to hi-1 (K) + 2**(i-1) *P. This general definition of a level i of the tree implies that new levels of the tree can be dynamically created according to increases in the size of the file. Thus, the PT grows as the content of the file grows.

A space partition is physically implemented by a bucket. Therefore a bucket of level i which overflows is split in two according to the predicates Pi1 and Pi2.

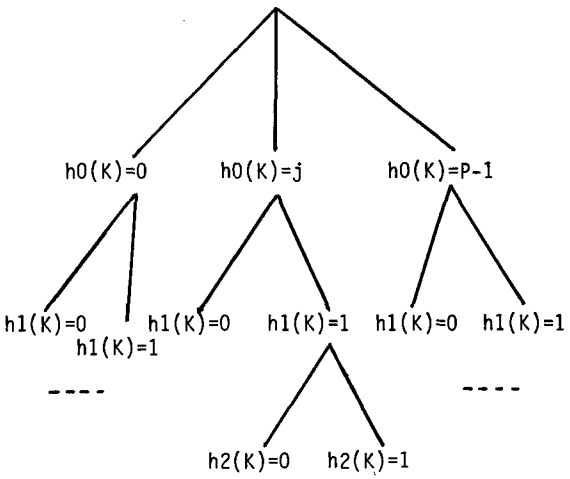

Figure 3. : A PT representing a dynamic hashing function

The grid file [NIEV81] is a multikey file structure where the space is recursively partitionned in two according to the median value of an attribute chosen in some sequence of the attributes of the file. All the attributes are treated successively. A PT representing the same function as this of the grid file is a binary tree where the attributes of the file appear in some order at each level. Let us consider the simple case with two attributes {A0, A1}. Each level is defined by two predicates as follows :

```
level 0 :    f0(A0)=0 ;    f0(A0)=1
level 1 :    f1(A1)=0 ;    f1(A1)=1
.
.
level 2i :     f2i(A0)=0 ;     f2i(A0)=1
level 2i+1 :   f2i+1(A1)=0 ;   f2i+1(A1)=1
```

The functions f2i and f2i+1 are used to divide a space partition in two. The splitting policy of the grid file is based on the median value of an attribute, calculated by knowing its minimum and maximum values.

The specification of f2i and f2i+1 is as follows, where min (A) and max (A) are the limits of the domain of attribute A and [a] means the smallest integer greater than or equal to a.

$$f2i\ (A0) = \left[\frac{2^{**}(i+1) * (A0-\min(A0))}{\max(A0) - \min(A0)} \right] \bmod 2$$

$$f2i+1(A1) = \left[\frac{2^{**}(i+1) * (A1-\min(A0))}{\max(A1) - \min(A1)} \right] \bmod 2$$

Probably a large number of possible level definitions is useful for certain applications, with various degrees of generality and complexity. The implementation of PTs in SABRE is actually restricted to a minimum set of predicate definitions, chosen for the following criteria. The specification of a level must be compact and simple because a PT must fit easily in memory. The consequence is a low processing cost of a PT. However, the minimum set of predicates should permit to represent combinations of various known methods. The first restriction is that a level consists of predicates defined on the same attribute. This facilitates the definition of a level by a database administrator and the algorithms. However, multikey predicates are specified with several levels. The following level definitions are provided :

1) classical hashing : hashing functions such as f(K) = k mod P are allowed.

2) digital hashing : the function CHARi(K) gives the rank of the ith character (if K is alphabetical or alpha-numerical) or the ith digit (if numerical key) or the ith bit (if bit string) in its alphabet ({A..Z}, {0..9}, {0, 1}, etc). As an example it permits to represent trie-hashing where the same key is involved at each level and respects key order.

3) interpolation hashing : this function is similar to that of [BURK83] and not far from that of the grid file. Giving the minimum and maximum values of a key and the branching factor m, the function is specified by :

$$f(K) = \left[\frac{m * (K - \min(K))}{\max(K) - \min(K)} \right]$$

It generates intervals by uniform partitionning of the space.

4) enumeration of a domain : the list of possible values of a domain of a key is enumerated. The key word OTHERS can be given by the designer of the PT for informing that a value not in the list can appear. If this word is not employed, the domain is completely defined which allows the PT to implicitely store integrity constraints on this domain. This feature will be seen to be useful in the next section.

5) enumeration of intervals : if interpolation hashing is not wished because of a totally non uniform distribution of values of data, it is then possible to enumerate the limits of the intervals in key order. Two key-words (SMALLEST and GREATEST) can be optionnally used for meaning the possibility of a value smaller than the minimum given or greater than the maximum given. As an example, the following list {smallest, 10, 30, 50} defines the predicates
K<10, 10≤K<30, 30≤K<50

This also implements in a simple way integrity constraints on a domain.

3. MULTIKEY HASHING SCHEME

The PT associated with a file is used in two basic operations : insertion of new records and retrieval of records qualified by predicates. Other operations on a file (e.g. deletion) are based on retrieval and are not reported here. The different key values of a record to insert are transformed in a logical address of a leaf of the PT. The different predicates qualifying a retrieval request are transformed in a set of logical addresses of leaves of the PT. A logical address is defined in the following.

3.1. Addressing in a predicate tree

A record to insert in a file is "signed" by the PT. A logical address called "signature" is affected to that record. A signature of a record is the path from the root to a leaf determined by its key values. It permits to locate the deepest possible partition for that record. A signature is therefore defined by the list of addresses of nodes on the path. A node of level i in a PT is logically addressed by the rank of the corresponding predicate. If this predicate is Pij, the node address is j. A signature of a record is therefore an abstract of its content. Examples of signatures are illustrated in Figure 4 by a PT of height 2.

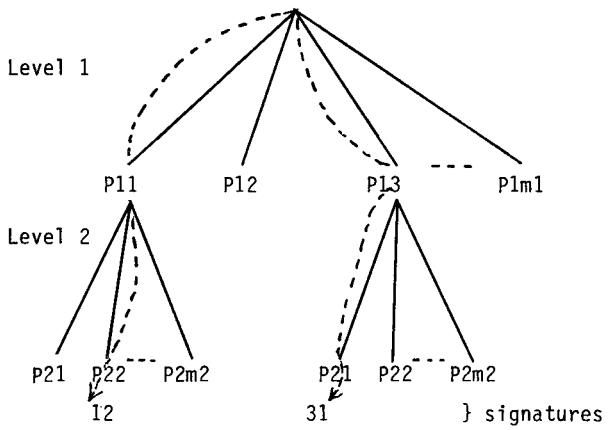

Figure 4. : A predicate tree and 2 leaf signatures

The definition of a predicate tree implies a signature of a record to be unique. The algorithme which signs a record is very simple. It consists of applying for each level partitionning an attribute Ai the hashing function to the Ai value of the record and then to generate the corresponding node address. A special case can arise when an integrity constraint on a domain is defined. For example, the level is an enumeration of intervals on SALARY and the lowest salary is 1000 $. Thus a record with a salary lower than 1000 $ violates the integrity constraint and is rejected. The algorithm is described in Figure 5 where Hi is the hashing function of level i defined on attribute Ai. We understand here the term hashing function in its general sense, as in [KNUT73].

```
procedure SIGNE-RECORD (r:record ; signature)
for i := 1 to p do
   begin
   S[i] := Hi (R.Ai) ;
   if S [i] < 0 or > mi - 1 then
      "reject r" ;
      exit ;
   end ;
```

Figure 5. : Signature generation for a record

The signature of the record is coded by a bit string to be used as a unique key by a digital hashing method. Thus, a signature must be compact since prefixes of signatures are stored in a directory to provide the physical mapping. By definition, each node address in a signature is bounded by mi, the branching factor of level i of the PT. Each number ni is then coded with bi bits in which mi can be represented. Then bi is such that :
$2^{**}(bi-1) < mi \leq 2^{**}bi$

For example, the signature of length 5 with the branching factors 16, 20, 20, 200, 10 is coded by the bit string

n1	n2	n3	n4	n5	
4	5	5	8	4	number of bits

This string needs 4 bytes and permits to generate 12800000 signatures. Each bit of the signature is successively employed by the unique key access method for applying the splitting policy [GARD84]. The advantage is that the signing process is completely independent of variations of the content of the file.

3.2. Selection algorithm

A PT serves to transform a multikey query qualification on a file (relational restriction operation) into a qualification on a smaller set of partitions. The selection algorithm generates partition addresses (signatures) which can contain records satisfying the query qualification. When the query is fully specified, that is, all the attributes in the PT are involved in the qualification with the operator '=', then a single signature is generated. A partially specified query, that is with some attribute(s) in the PT missing or an operator other than '=' is employed, may select a greater number of signatures. To decrease this number of signatures, a set of signatures corresponding to several subtrees of the PT is coded into a signature profile. A signature profile is an incomplete signature where missing node addresses are replaced by the unknown value, denoted by '.'. The unknown value appearing at the end of a profile can be omitted. Figure 6 portrays the signature profile with the associated PT for a query qualification specified by the predicate P21. This signature profile represents the six marked signatures.

Like a signature, a signature profile is coded by a bit string where some bits of a profile may be unspecified. Thus, a bit of a signature profile is coded with two bits in order to enable three values : '0', '1', '.'. It then can be possible to address only certain subtrees of a level corresponding to certain bits of a node address.

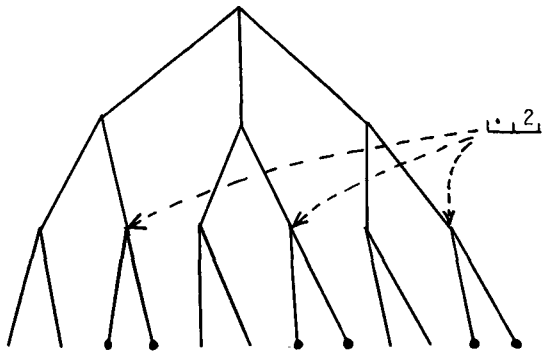

Figure 6. : A signature profile

This is useful for range qualifications where several predicates of a level are satisfied and the level is order preserving (e.g. intervals). An example of a signature profile coded in a bit string corresponding to the predicate (P13 v P14) is described by figure 7. Note that the node address of predicate P11 is coded as 00, that of P12 as 01 etc.

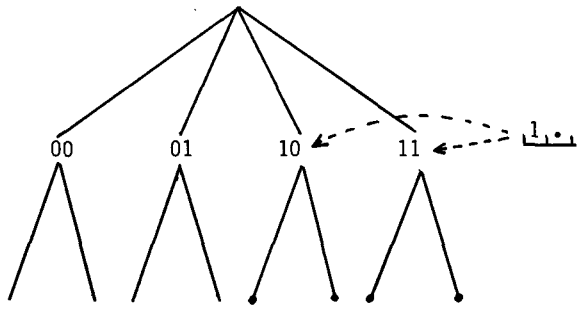

Figure 7. : A coded signature profile

With b bits, the number of different sets of nodes that can be addressed by a single profile part is $2^{**}b$. Two bits give the possible profile parts : .0, .1, 0., 1..

Let Q be the retrieval qualification, the selection algorithm compares Q with the PT associated with the file F in order to give one of the following responses :
- R = F, each record of F can satisfy Q
- R = $\emptyset$, no record of F can satisfy Q
- R = { Fij / Pij $\wedge$ (Pij $\Rightarrow$Q) }

In the latter case, the algorithm generates a set of signature profiles which determine the partitions containing candidate records that match Q. The principle of the algorithm is to select in the PT the predicates not contradictory with Q. Two predicates P and Q are said contradictory if, for any interpretation of P, P = not Q. For example, predicates DEGREE>11 and DEGREE<10 are contradictory while DEGREE>11 and DEGREE>12 are not. When comparing the PT with Q, two cases can occur. The first one is that a level i is defined on attribute Ai which is not involved in Q. Thus, each predicate of level i can be satisfied and each node address is selected. The unknown value is generated in the profile part. The other case is that level

i is defined on an attribute involved in Q. Then, for each predicate Pij not contradictory with Q, j must be kept as a candidate signature profile part. If no such j is found and no signature part already exists for level i, a special value, noted 'none' is generated in the profile part. None is the contrary of the value '.'. This occurs when Q violates an integrity constraint implemented on a domain by the PT. A set of profile parts (different node addresses for a same level) can be reduced by setting to the unknown value a bit which appears as '1' in a part and '0' in another one. This process is illustrated by Figure 7. If at least one signature part has been set to 'none' then R = ϕ and the file will not be accessed else the cartesian product of all signature profile parts determines the set of signature profiles to answer the query.

The algorithm which transforms a qualification Q into a set of profiles L proceeds in four main steps. Let Q be expressed in conjunctive normal form

$$Q = (Q11 \vee Q12 \ldots \vee Q1k) \wedge \ldots \wedge (Qn1 \vee Qn2 \ldots \vee Qnkn)$$

where Qij has the form (A θ value), A being an attribute of the file and θ a comparison operator. A different algorithm could be found for a qualification expressed in disjunctive normal form.

The algorithm is portrayed in Figure 8, where sets Li are supposed initialized to empty, noted []. The first step projects Q on the attributes involved in the PT. A boolean value is affected to each predicate Qij and set to true if the attribute of Qij is not in the PT and set to false otherwise. Then, simplification rules "P $\wedge$ true is replaced by P" and "P $\vee$ true is replaced by true" are applied. This avoids to test in the next step for disjunctions set to true. In particular, if Q becomes set to true then the whole file must be searched, the profile being all '.'. The second step selects the predicates in the PT which are not contradictory with a disjunction having not been set to true. For a disjunction Qk composed of predicates on various attributes, the attribute of deeper level is chosen. As an example, let us suppose that Qk=(Qk1 v Qk2), Qk1 being on attribute A1 and Qk2 on attribute A2. Level 2 is chosen (A2 is of level 2) since Qk2 must be tested for all sons of level 1. For the level i selected, if the set of profile parts Li is empty, it is set to none . Then, the hashing function of level i is applied to the value in predicates on Ai of Qk and each j such that Pij is not contradictory with Qk is kept as a possible signature part. The third step initializes all sets of profile parts Li, of which attribute Ai is not involved in Q (after simplification), to the element [.] meaning all possible values. If a set Li is still set to 'none' then L is set to 'none' and the algorithm terminates. The last step applies the cartesian product (noted X) to all sets Li giving the final set of signature profiles answering the qualification.

Like the signing process, the selection algorithm works at a logical level independently of the content of the file. Signature profiles are compared to signature prefixes stored in a directory and associated with addresses of pages in secondary memory. Then, records of the selected pages can be retrieved.

```
procedure PROFILE (Q : qualification ;
L : set of profiles) ;
L1, L2, ... Li, ... Lp : set of profile part ;

begin
simplify (Q) ;
for each remaining disjunction Qk do
   begin
   i := deeper-level (Qk) ;
   if Li = [] then Li := [none] ;
   for j := 0 to mi-1 do
      if Pij ∧ (Pij ⇒ Qk) then Li := Li V [j] ;
   end ;
for i := 1 to p do
   begin
   if Li = [none] then L := [none] ; exit ;
   if Li = [] then Li := [.] ;
   end ;
L := ⨉ (i=1 to P) Li
end ;
```

Figure 8. : Selection algorithm with a PT

4. COMPARISON WITH ANOTHER METHOD

A qualitative analysis of the proposed method is well expressed by a comparison with a multifield method [BLAS77]. The analysis is qualitative since performance of both methods strongly relates on these of the unique key access methods that perform the logical physical mapping. A quantitative analysis would require the complete definition of the unique key access method, which is beyond the scope of this paper.

The multifield encoding method codes multikey character strings into a single string and preserves lexicographic multikey order. This single key becomes the unique key of an indexed method based on key order : B+-trees. Simulation results of [BLAS77] indicate that the average overhead in the encoded data string for a string of L bytes is $\sqrt{2L}$ bytes. An average string length of 15 bytes gives approximately an overhead of 6 bytes per string. The proposed method herein reported codes multikey strings into a unique key, called signature, based on the content of multiple keys. This allows for a more general definition of the domain of keys than key order since various hashing functions can be represented by a predicate tree. As a particular case of hashing function, key order may be preserved. The representation of a signature by a bit string makes is more compact than an encoded data string. The size of a signature in independent of string lengths. Also, a query qualification can be encoded in the same fashion as a small set of signatures profiles. Profiles can be compared associatively with signature prefixes in a directory managed by an efficient digital hashing method [GARD84].

The multifield encoding method is based on an indexed method implemented by B+-trees to provide the unique-key to address transformation. The height of the index of the B+-tree is proportional to the size of the keys which must be fully con-

tained in the index nodes. Tree height is given by

$$\text{height} = \log_a (\text{file-size})$$

where a is the node fanout of the B-tree. The height gives the number of I/O operations for accessing a leaf containing records. To reduce this height, prefix B-trees [COME79] can be employed, which is not satisfactory for all kinds of keys. The proposed method is based on a digital hashing method which has common points with DL-trees. Any digital hashing method such as trie hashing [LITW81] could also be candidate. To some extent, extendible hashing [FAGI79], which uses the bits of the result of the hashing function, could also be adapted. With a digital hashing method, records having a same signature prefix are stored in the same page. Therefore, like with DL-trees, the height of the tree is less that of a B-tree. Hence, the number of I/O to read a page containing records is less. Also, B+-trees often lead to get a space partition of a file, which could fit in a single page, sharing two pages. This is because the splitting process is independent of any partitionning and is based on key order. This can result in doubling the number of I/O to access a partition. The proposed method appears more general and the unique key access method based on digital hashing more adapted to multikey partitionning.

5. CONCLUSION

A hashing method which transforms multiple keys into a logical address has been described. The functions applied to the keys of a file are represented by a predicate tree. A predicate tree can be seen as a generalization of various hashing schemes. The implementation of these methods suitable for specific applications suggests the development of predefined PTs representing various methods such as extendible hashing, grid file or unique key placement (by using certain digits of the key). All these methods would be based on the unified organization provided by PTs. A logical address, called signature, is an abstract of the content of the addressed partition. It is coded as a compact bit string and then is used as a unique key by a digital hashing method [GARD84] which maps signatures into page addresses. The advantage of such a method, like that of [BLAS77], is that no extension is required to the unique key access method. The transform method is useful for signing records to insert or delete in a file. It is also employed for translating a retrieval qualification on a file into a set of partitions identified by signature profiles. The advantages of such a method in comparison with a multifield encoding method are its generality, not restricted to key order, and its performance, due to the advantage of the unique key access method in comparison with B+-trees. By generalizing and combining various efficient methods, PTs permit to expect getting advantages of such methods.

This method has wide applicability. In order to be more adapted to certain applications, more general predicates can be allowed for defining space partitionning of a file. This would result in having predefined complex queries where the records to retrieve are in the same partitions. This certainly adds complexity and cost to the algorithms. The application here suggested for the method is the clustering of records according to the partitionning defined by the PT. PTs can also serve as a powerful tool for implementing secondary indexes in a more general way. For example, secondary access is useful when an attribute already used with a hashing function would need another function, that corresponds to another retrieval qualification. The PT implementing a secondary index would give secondary signatures associated by a directory with primary signatures, corresponding to partitions of the file.

Another fruitful way of investigation is the application of this method for speeding up join operations as well as retrieval operations. Relational join is generally considered as the most time consuming operation. It has been shown that parallel algorithms, without indices, are efficient only with a high number of processors [VALD84]. Using join attributes in the PTs defined on the files implementing the relations with the same function would permit to associate partitions of files where records can possibly match the join qualification. The method would then transform a join on two relations into several joins on partitions of relations having matching tuples. Such an algorithm could be based on parallel processing. Another possibility is to extend the definition of a PT to a whole database. Relation names would prefix attribute names for defining predicates as in non-procedural queries. Therefore, a join predicate could be specified for defining multirelation partitions having matching tuples.

6. REFERENCES

[BAYE72] R. BAYER, E. McCREIGHT : "Organization and maintenance of large ordered indexes", Acta Informatica, 1,3, 1972, pp 173-189.

[BENT79] J. BENTLEY, J. FRIEDMAN : "Data structures for range searching", Computing Surveys, Vol.11, n°4, Dec. 1979.

[BLAS77] M. BLASGEN, R. CASEY, K. ESWARAN : "An encoding method for multifield sorting and indexing", Com. of the ACM, Vol.20, n°11, Nov. 1977.

[BURK83] W. BURKHARD : "Interpolation-based index maintenance", 2nd ACM Symposium on PODS, Atlanta (Georgia), March 1983.

[COME79] D. COMER : "The ubiguitous B-tree", Computing Surveys, Vol.11, n°2, June 1979.

[FAGI79] R. FAGIN, J. NIEVERGELT, N. PIPPENGER, H.R. STRONG : "Extendible hashing : a fast access method for dynamic files", ACM-TODS, n°4, 1979.

[GARD83] G. GARDARIN, P. BERNADAT, N. TEMMERMAN, P. VALDURIEZ, Y. VIEMONT : "Design of a multiprocessor relational database system", IFIP, 9th World Computer Congress, Paris, Sept. 1983.

[GARD84] G. GARDARIN, P. VALDURIEZ, Y. VIEMONT : "Predicate trees", to appear in Proc. of the Computer Engineering Conference, IEEE Ed., Los Angeles, April 1984.

[KNUT73] D. KNUTH : "The art of computer programming : sorting and searching", Addison-Wesley, Vol.3, Reading, Mass., 1973.

[LARS78] P. LARSON : "Dynamic hashing", BIT 18, 1978.

[LITW78] W. LITWIN : "Virtual hashing : a dynamically changing hashing", Proc. of 4th Int. Conf. on VLDB, Berlin, Sept. 1978.

[LITW81] W. LITWIN : "Trie hashing", ACM-SIGMOD, Int. Conf. on Management of Data, Ann Harbour, Sept. 1981.

[LOME83] D. LOMET : "A high performance : universal key associative access method", ACM-SIGMOD, Int. Conf. on Management of Data, San Jose, May 1983/

[NIEV81] J. NIEVERGELT, H. HINTERBERGER, K.C. SEVCIK : "The GRID FILE : an adaptable symmetric multi-key file structure", Trends in Information Processing Systems, Proc. of 3r ECI Conf., 1981.

[OUKS83] M. OUKSEL, P. SCHEUERMAN : "Storage mappings for multidimensional linear dynamic hashing", 2nd ACM Symposium on PODS, Atlanta (Georgia), March 1983.

[VALD84] P. VALDURIEZ, G. GARDARIN : "Join and semi-join algorithms for a multiprocessor database machine", ACM-TODS, Vol.9, n°1, March 1984.