An Evolutionary Approach to Schema Partitioning Selection in a Data Warehouse

Ladjel Bellatreche[1] and Kamel Boukhalfa[2]

[1] LISI/ENSMA - Poitiers University - France - E-mail : bellatre@ensma.fr
[2] Université de Laghouat - Algeria - E-mail: k.boukhalfa@mail.lagh-univ.dz

Abstract. The problem of selecting an optimal fragmentation schema of a data warehouse is more challenging compared to that in relational and object databases. This challenge is due to the several choices of partitioning star or snowflake schemas. Data partitioning is beneficial if and only if the fact table is fragmented based on the partitioning schemas of dimension tables. This may increase the number of fragments of the fact tables dramatically and makes their maintenance very costly. Therefore, the right selection of fragmenting schemas is important for better performance of OLAP queries. In this paper, we present a genetic algorithm for schema partitioning selection problem. The proposed algorithm gives better solutions since the search space is constrained by the schema partitioning. We conduct several experimental studies using the APB-1 release II benchmark for validating the proposed algorithm.

1 Introduction

The main characteristics of data warehouses are (1) their data complexity due to the presence of hierarchies between attributes, (2) the huge amount of data, and (3) the complexity of their queries due to the presence of join and aggregate operations. Several queries optimization techniques were proposed in the literature and supported by commercial systems. These techniques can be classified into two categories: (1) *redundant-structures* and (2) *non redundant-structures*. Techniques in the first category compete for the same resource representing the storage cost and incur maintenance overhead in the presence of updates [12]. We can cite: materialized views and indexes. Techniques in the second category do not require an extra space compare to those in the first category. We can cite vertical and horizontal partitioning [11]. Horizontal partitioning (HP) allows access methods such as tables, indexes and materialized views to be partitioned into disjoint sets of rows that are stored and accessed separately. On the other hand, vertical partitioning allows a table to be partitioned into disjoint sets of columns. Like indexes and materialized views, both kinds of partitioning can significantly impact the performance of the workload i.e., queries and updates that execute against the database system, by reducing cost of accessing and processing data. In this paper, we are interesting to a non redundant structure, which is the HP. Several work and commercial systems show its utility and impact in optimizing OLAP queries [11, 3, 2, 8, 13]. But none study has formalized the problem of selecting a HP schema to speed up a set of queries and proposed selection algorithms. In this paper, we use fragmentation and partitioning interchangeably. HP in relational data warehouses is more challenging compared to that in relational and object databases. This challenge is due to the several choices of partitioning schemas [3] that can be found:

1. *partition only the dimension tables* using simple predicates defined on these tables [4]. This scenario is not suitable for OLAP queries, because the sizes of dimension tables are generally small compare to the fact table. Therefore, any partitioning that does not take into account the fact table is discarded.
2. *partition only the fact table* using simples predicates defined on this table. Note that a fact relation stores foreign keys and raw data which is usually never contain descriptive (textual) attributes because it is designed to perform arithmetic operations. On the other hand, in a relational data warehouse, most of OLAP queries access dimension tables first and then the fact table. This choice is also discarded.
3. *partition some/all dimension tables using their predicates, and then partition the fact table* based on the fragmentation schemas of dimension tables. This approach is best in applying partitioning in data warehouses. Because it takes into consideration star join queries requirements (these queries impose restrictions on the dimension values that are used for selecting specific facts; these facts are further grouped and aggregated according to the user demands. The major bottleneck in evaluating such queries has been the join of a large fact table with the surrounding dimension tables [13]). In our study, we opt for last solution.

To show the procedure to fragment a fact table using this scenario, suppose that a dimension table D_i is fragmented into m_i fragments: $\{D_{i1}, D_{i2}, ..., D_{im_i}\}$, where each fragment D_{ij} is defined as: $D_{ij} = \sigma_{cl^i_j}(D_i)$, where cl^i_j $(1 \leq i \leq g, 1 \leq j \leq m_i)$ represents a conjunction of simple predicates. Thus, the fragmentation schema of the fact table F is defined as follows: $F_i = F \ltimes D_{1i} \ltimes D_{2i} \ltimes ... \ltimes D_{gi}$, with $\ltimes$ represents the semi join operation. In order to illustrate this procedure, let consider a star schema with three dimension table (Customer, Time and Product) and one fact table Sales. Suppose that the dimension table Customer is fragmented into two fragments Cust_1 and Cust_2 defined by the following clauses: $Cust\_1 = \sigma_{Sex=`M'}(Customer)$ and $Cust\_2 = \sigma_{Sex=`F'}(Customer)$. Therefore the fact table Sales can be fragmented using the fragmentation schema of the dimension table Customer into two fragments Sales_1 and Sales_2 such as: $Sales\_1 = Sales \ltimes Cust\_1$ and $Sales\_2 = Sales \ltimes Cust\_2$.

The initial star schema (Sales, Customer, Product, Time) is represented as the juxtaposition of two sub star schemas S_1 et S_2 such as: S_1 : (Sales_1, Cust_1, Product, Time) (sales activities for only male customers) et S_2 : (Sales_2, Cust_2, Product, Time) (sales activities for only female customers).

To the best of our knowledge, the proposed work is the first article that addresses horizontal fragmentation schema selection problem in relational data warehouses and uses a genetic algorithm to select a right solution that minimizes the per-

[3] The fragmentation schema is the result of the data partitioning process.

[4] A simple predicate p is defined by: $p : \quad A_i \ \theta \ Value$, where A_i is an attribute, $\theta \in \{=, <, >, \leq, \geq\}$, and value $\in Dom(A_i)$

formance of OLAP queries, and reduces the maintenance cost.

This paper is divided in five sections: The section 2 formalizes the fragmentation selection problem in data warehouses modeled using star schemas. Section 3 presents a genetic algorithm with its four steps (selection, coding, mutation, and fitness function). Section 4 gives the experimental results using benchmark APB-1 release II benchmark. The Section 5 concludes the paper by summarizing the mains results and suggesting future work.

2 Complexity of generated fragments of the fact table

Let a star schema with d dimension tables and a fact table. Let g ($g \leq d$) be the number of fragmented dimension tables. The number of horizontal fragments of the fact table (denoted by N) is given by: $\mathbf{N} = \prod_{\mathbf{i=1}}^{\mathbf{g}} \mathbf{m_i}$, where m_i is the number of fragments of dimension table D_i. This fragmentation technique generates a large number of fragments of the fact table. For example, suppose we have: *Customer* dimension table partitioned into 50 fragments using the State attribute [5], *Time* into 36 fragments using the Month attribute, and *Product* into 80 fragments using Package_type attribute, therefore the fact table will be fragmented into 144 000 fragments ($\mathbf{50} \times \mathbf{36} \times \mathbf{80}$). Consequently, instead of managing one star schema, we will manage 144 000 sub star schemas. It will be very hard for the data warehouse administrator (DWA) to maintain all these sub-star schemas.

Therefore it is necessary to reduce the number of fragments of the fact table in order to guarantee two main objectives: (1) avoid an explosion of the number of the fact fragments and (2) ensure a good performance of OLAP queries. To satisfy the first objective, we give to DWA the possibility to choose the number of fragment maximal that he/she can maintain (threshold W). For the second one, we can increase the number of fragment so that the global performance will be satisfied. The problem of selecting an optimal fragmentation schema consists in finding a compromise between the maintenance cost and the performance cost.

In order to satisfy this compromise, we use genetic algorithms [1, 4] since they explore a large search space. Our problem is similar to the problems multiprocessor document allocation [6], and data replication [9], where genetic algorithms gave good results.

3 Genetic Algorithms

Genetic algorithms (GAs) [7], are search methods based on the evolutionary concept of natural mutation and the survival of the fittest individuals. Given a well-defined search space they apply three different genetic search operations, namely, *selection*, *crossover*, and *mutation*, to transform an initial population of chromosomes, with the objective to improve their quality. Fundamental to the GA structure is the notion of chromosome, which is an *encoded* representation of a feasible solution, most commonly a bit string. Before the search process starts, a set of chromosomes is initialized to form the first generation. Then the three

[5] case of 50 states in the U.S.A.

genetic search operations are repeatedly applied, in order to obtain a population with better characteristics. An outline of a generic GA is as follows:

Generate initial population ;
Perform selection step;
while stopping criterion not met **do**
    Perform crossover step;
    Perform mutation step;
    Perform selection step ;
end while. Report the best chromosome as the final solution. We demonstrate the design of our algorithm in details by presenting our encoding mechanism and then the selection, crossover and mutation operators.

3.1 Representation of Solutions

Representation of solution or chromosome is one of the key issues in problem solving. In our study, a solution represents a fragmentation schema. Note that any fragmentation algorithm needs application information defined on the tables that have to be partitioned. The information is divided into two categories [10]: quantitative and qualitative. Quantitative information gives the selectivity factors of selection predicates and the frequencies of queries accessing these tables ($Q = \{Q_1, , Q_n\}$). Qualitative information gives the selection predicates defined on dimension tables. Before representing each solution, the following tasks should be done:
1. extraction of all simple predicates used by the n queries,
2. assignment to each dimension table $D_i (1 \leq i \leq d)$, its set of simple predicates ($SSPD_i$),
3. each dimension table D_i having $SSPD_i = \phi$ cannot be fragmented. Let $D_{candidate}$ be the set of all dimension tables having a non-empty $SSPD_i$. Let g be the cardinality of $D_{candidate}$ ($g \leq d$),
4. use the COM_MIN algorithm [10] to each dimension table D_i of $D_{candidate}$. This algorithm takes a set of simple predicates and then generates a set of complete and minimal.

Representation of Horizontal Fragments Note each fragmentation predicate has a domain values. The clauses of simple predicates representing horizontal fragments defines partitions of each attribute domain into sub domains. The cross product of partitions of an attribute by all predicates determines a partitioning of the domains of all the attributes into sub domains.

Example 1. Consider three fragmentation attributes [6] Age, Gender and City of dimension table Customer and one attribute Season of dimension table Time. The domains of these attributes are defined as follows: $Dom(Age) =]0, 120]$, $Dom(Season) = \{``Summer'', ``Spring'', ``Autumn'', ``Winter''\}$, and $Dom(Gender) = \{`M', `F'\}$. Suppose that on attribute Age, three simple predicates are defined as follows: p_1 : Age ≤ 18, $p_2 : Age \geq 60$, and $p_3 : 18 < Age < 60$. The domain of this attribute ($]0, 120]$) is then partitioned into three sub domains

[6] A fragmentation attribute is an attribute participating in the fragmentation process

(p_1, p_2, and p_3). Dom(Age) $= d_{11} \cup d_{12} \cup d_{13}$, with $d_{11} =]0, 18]$, $d_{12} =]18, 60[$, $d_{13} = [60, 120]$. Similarly, the domain of Gender attribute is decomposed into two sub domains: Dom(Gender) $= d_{21} \cup d_{22}$, with $d_{21} = \{`M'\}$, $d_{22} = \{`F'\}$. Finally, domain of Season is partitioned into four sub domains : Dom(Season) $= d_{31} \cup d_{32} \cup d_{33} \cup d_{34}$, where $d_{31} = \{$"Summer"$\}$, $d_{32} = \{$"Spring"$\}$, $d_{33} = \{$"Autumn"$\}$, and $d_{34} = \{$"Winter"$\}$.

Each fragmentation attribute can be represented by an array with n cells, where n corresponds to number of its sub domains. The values of these cells are between 1 and n. If two cells have the same values, then they will be merged to form only one. Each fragmentation schema is represented by a multi-dimensional arrays. Suppose we have the following representation Gender: (1, 1, -, -), Season(2, 1, 3, 3) and Age (2, 1, 2, -). We can deduce that the fragmentation of the data warehouse is not performed using the attribute Gender, because all its sub domains have the same value. Consequently, the warehouse will be fragmented using only Season and Age. For Season attribute, three simple predicates are possible: P_1 : Season = "Spring", P_2 : Season = "Summer", and P_3 : Season = "autumn" $\vee$ Season = "Winter". For Age attribute, two predicates are possible: $P_4 : Age \leq 18 \vee Age \geq 60$ et $P_5 : 18 < Age < 60$ With these simple predicates, the data warehouse can be fragmented into six fragments defined by the following clauses: $Cl_1 : P_1 \wedge P_4$; $Cl_2 : P_1 \wedge P_5$; $Cl_3 : P_2 \wedge P_4$; $Cl_4 : P_2 \wedge P_5$; $Cl_5 : P_3 \wedge P_4$; and $Cl_6 : P_3 \wedge P_5$. The coding that we proposed satisfies the correctness rules (completeness, reconstruction and disjointness [10]) and the new chromosomes generated by cross over operations belong to the relevant sub domains (it does not generate invalid solutions). This coding can be used to represent fragments of dimension tables and fact table.

3.2 Selection Mechanism

Selection in genetic algorithms determines the probability of individuals being selected for reproduction. The principle here is to assign higher probabilities to filter individuals. The roulette wheel method is used in our algorithm (it allocate a sector of the wheel equaling to the i^{th} chromosome and creating an offspring if a generated number in the range of 0 to falls inside the assigned sector of the string). In this method, each chromosome is associated with its fitness value calculated using the cost model defined in section 3.4. The chromosomes with high fitness values have chances to be selected.

3.3 Types of crossover

We selected a two-point crossover mechanism to include in our GA for the following reason: note that fragments are represented by arrays. The chromosomes are crossed over once for each predicate. If the crossover is done over one chromosome, the predicates with high number (example of city) will have a probability greater than predicate with small predicate like gender. This operation is applied till none reduction of the number of suitable fragments of fact table (W). The rationale behind crossover operation, is that after the exchange of genetic materials, it is very likely that the two newly generated chromosomes will possess the good characteristics of both their parents (building-block hypothesis [7]).

3.4 Fitness value

The quality of each chromosome is measured by computing its fitness value. This function gives a percentage for each performance parameters (respect of threshold and performance of queries). A number of points is assigned to these two parameters: (1) *threshold*: 55 points over 100 are assigned (by default). If the number of obtained fragments is equal plus or minus 5 per cent of the threshold, then all points will be assigned. Otherwise, less points will be assigned to this parameter, and (2) *performance of queries*: a number of points (45) is assigned to all queries in an uniform manner (we have used 15 queries). To compute the cost of each query, we developed a cost model calculating the number of inputs and outputs. As in the previous case, we assign all points (3 per query) if the cost of a query is less than a given number. If the number of IOs increases, less we assign points, following a linearly decreasing function. When the number of IOs of a given query is very high, none point is assigned.
To estimate the cost of queries, we assume that all dimension tables are in the main memory. Let $D^{sel} = \{D_1^{sel}, ..., D_k^{sel}\}$ be the set of dimension tables having selection predicates, where each selection predicate p_j (defined on a dimension table D_i) has a selectivity factor denoted by $Sel_{D_i}^{p_j}$ ($Sel_{D_i}^{p_j} \in [0, 1]$). For each predicate p_j, we define its selectivity factor on the fact table, denoted by $Sel_F^{p_j}$ ($Sel_{D_i}^{p_j} \neq Sel_F^{p_j}$). For example, if we consider the selection predicate Gender="Female" defined on the dimension table. Suppose that its selectivity factor is 0.4. This is means that 40% of salespersons are female and 60% are male. But, female sales activities may represent 70% of the whole sales. To execute a query Q over a partitioned star schema $\{S_1, S_2, ..., S_N\}$, we shall identify the relevant sub star schema(s). To do so, we introduce a boolean variable denoted by $valid(Q, S_i)$ and defined as follows: $valid(Q, S_i) = 1$ if the sub star schema S_i is relevant for Q, 0 otherwise. The number of IOs for executing a query Q over a partitioned star schema is given by: $Cost(Q) = \sum_{j=1}^{N} valid(Q, S_j) \prod_{i=1}^{M_j} (\frac{(Sel_F^{p_i} \times ||F|| \times L)}{PS})$, where, M_j, F, L and PS represent the number of selection predicates defining the fact fragment of the sub star schema SDE_j, the number of tuples present in a fact table F, the width, in bytes, of a tuple of a table F and the page size of the file system (in bytes), respectively. In this study, the selectivity factors are chosen using an uniform distribution (UD) and a non uniform distribution (NUD).

3.5 The mutation

Although crossover can put good genes together to generate better offspring. It cannot generated new genes. Mutation is needed to create new genes that may not be present in any member of a population and enables the algorithm to reach all possible solutions (in theory) in the search space. Mutation is an operation aiming at restoring lost genetic material and is performed in our algorithm by simply flipping every bit with a certain probability, called the mutation rate. We have chosen a mutation rate between 30 and 6 percent (rate often used). Mutations are done for fragmentation attributes. Initialization of the first generation is performed by randomly generating half of the population while the rest is obtained from the solutions previously found by algorithm. In practice, there could be more intervals distinct or a merged intervals. In the same way, mutations could occur on several attributes of the individual.

4 Experimental Studies

In our experiments, we use the dataset from the APB1 benchmark [5]. The star schema of this benchmark has one fact table Actvars ($||Actvars||$ = 24786000 tuples, with a width = 74) and four dimension tables: Prodlevel ($||Prodlevel||$ = 9 000 tuples, with a width = 72), Custlevel ($||Custlevel||$ = 900 tuples, with a width = 24), Timelevel ($||Timelevel||$ = 24 tuples, with a width = 36), and Chanlevel ($||Chanlevel||$ = 9 tuples, with a width = 24). This warehouse has been populated using the generation module of APB1. Our simulation software was built using Visual C performed under a Pentium IV 2,8 Ghz microcomputer (with a memory of 256 Mo). We have considered 15 queries. Each query has selection predicates, where each one has its selectivity factor. The crossover and

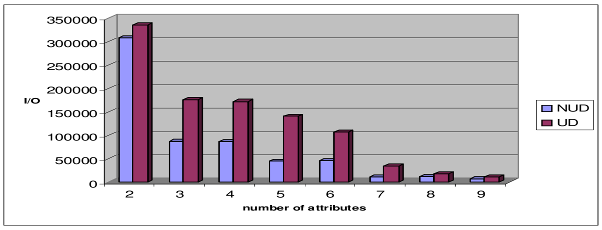

Fig. 1. Impact of fragmentation attributes on the query performance

mutations rates used in our experiments are 70% and 30% in the beginning. After several generation, the mutation rate of 6% was used to avoid a redundant search. We have used 1 500 generations (40 chromosomes per generation). 9 fragmentation attributes were considered.

If the DWA chooses the threshold as 2000, the dimension tables will be fragmented as follows: table Prodlevel in 48 fragments, table Timelevel in 7 fragments, table Custlevel in 2 fragments, table Chanlevel in 3 fragments,and the fact table in 2016 fragments. Figure 1 shows the evolution of IOs over the number of fragmentation attributes. The results show the impact of this number on the performance of queries. We note also that the non uniform distribution gives better performance than the uniform distribution.

In Figure 2, we have studied the effect of the number of fragmentation attributes over the number of fact table. We realize that the type of distribution does not have an effect on the total number of fragments.
Figure 3 shows the effect of the horizontal fragmentation and its role in reducing the global cost of executing a set of queries. These results confirmed the existing theoretical studies.

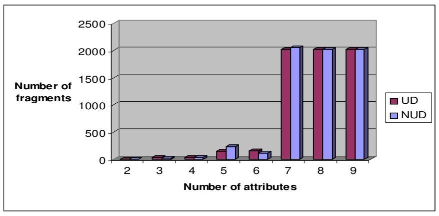

Fig. 2. The impact of the number of fragmentation attributes

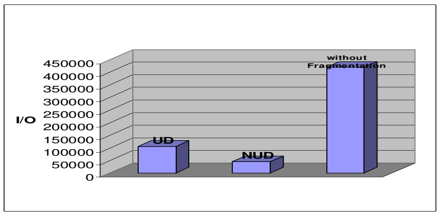

Fig. 3. Savings in query processing cost versus type of distribution

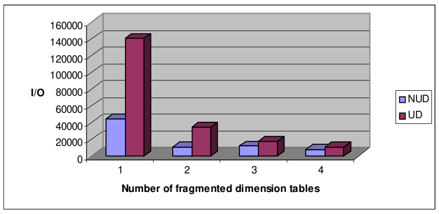

Fig. 4. The effect of number of fragmented dimension tables on query processing cost

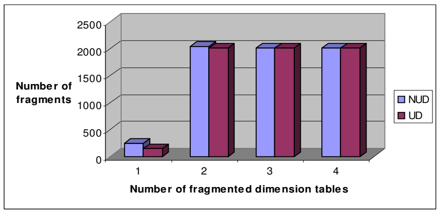

Fig. 5. Number of fragments versus number of fragmented dimension tables

In Figure 4, we have studied the effect of the dimension tables participating on the fragmentation process. The performance of OLAP queries is proportional with the number of these tables. Figure 5 shows that the number of fragments of the fact table increases when the the number of dimension tables participating on the fragmentation process increases. But our algorithm controls this augmentation. When we used less than six fragmentation attributes, the rate between

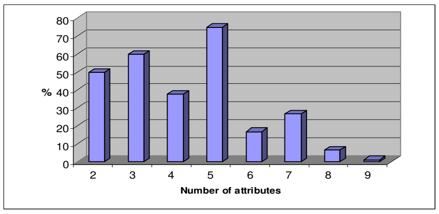

Fig. 6. Number of attributes versus number of possible fragments

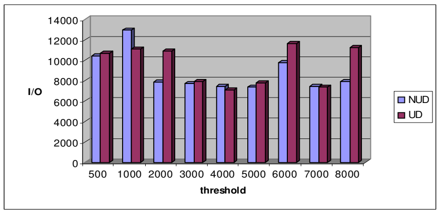

Fig. 7. Evolution of IO cost over the threshold

the number of fragments return by the algorithm and the possible number of fragment is high (more than 35%) because the possible number of fragments is small. From six attributes, this rate is small (less than de 2%) when we used 9 attributes (Figure 6). To get a better performance of queries, the threshold is varied, and experiments show that this performance is obtained when threshold is equals 4000.

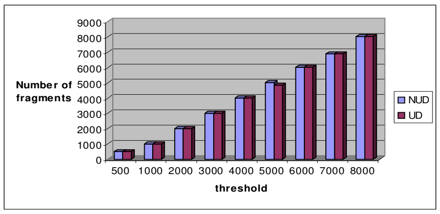

Fig. 8. Number of fragments versus the threshold

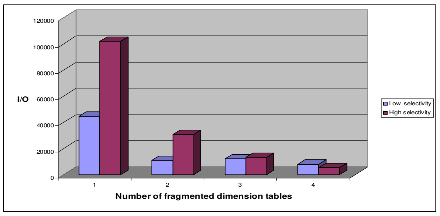

Fig. 9. Number of IOs versus selectivity type and number of fragmented dimension tables

Note that the number of fragments increases when the threshold increases, but it remains closer to the threshold (Figure 8).

In Figures 9, 10 and 11 we changed the selectivity factors of predicates in order to see their effect on the number of fragments and performance of queries. We realized that when we increase these factors, the number of IOs increases. This is due to the fact that an high selectivity implies a large number of tuples satisfying predicates. But the selectivity factors do not have a strong effect on the final number of fragments.

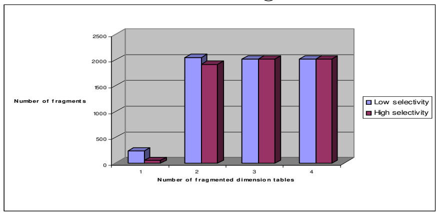

Fig. 10. Number of fragments versus the selectivity type and the number of fragmented dimension

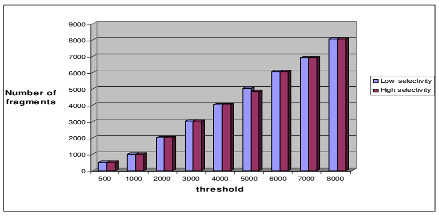

Fig. 11. Number of fragments versus the selectivity type and the threshold

5 Conclusion

In this paper, we have formalized the problem of selecting an horizontal fragmentation schema in relational data warehousing environments. First we have developed a methodology for fragmenting a star schema using the fragmentation schemas of the dimension tables. We have also shown the complexity of the generated fragments of the fact table. This number can be very huge and then it will be difficult for the data warehouse administrator to maintain all fragments. To reduce this number and guarantee a good performance, we proposed a genetic algorithm. Before applying this algorithm, we presented a coding mechanism for all possible solutions. A cost model for evaluating the cost of a set of frequently queries performed on a fragmented star schema is developed. This model is also used to measure the quality of the final solution. Finally, we conducted experiments to show the utility of the horizontal fragmentation and capture different points that can have effect on the performance of OLAP queries and respecting the threshold fixed by the administrator.

It will be interested to develop or adapt our algorithm to take into account the dynamic aspect of a warehouse due to the evolution of the schema and queries.

References

1. T. Bäck. *Evolutionnary algorithms in theory and practice*. Oxford University Press, New York, 1995.
2. L. Bellatreche, K. Karlapalem, and M. Mohania. What can partitioning do for your data warehouses and data marts. *Proceedings of the International Database Engineering and Application Symposium (IDEAS'2000)*, pages 437–445, September 2000.
3. L. Bellatreche, M. Schneider, H. Lorinquer, and M. Mohania. Bringing together partitioning, materialized views and indexes to optimize performance of relational data warehouses. *Proceeding of the International Conference on Data Warehousing and Knowledge Discovery (DAWAK'2004)*, pages 15–25, September 2004.
4. K. P. Bennett, M. C. Ferris, and Y. E. Ioannidis. A genetic algorithm for database query optimization. *in Proceedings of the 4th International Conference on Genetic Algorithms*, pages 400–407, July 1991.
5. OLAP Council. Apb-1 olap benchmark, release ii. *http://www.olapcouncil.org/research/bmarkly.htm*, 1998.
6. O. Frieder and H.T. Siegelmann. Multiprocessor document allocation : A genetic algorithm approach. *IEEE Transactions on Knowledge and Data Engineering*, 9(4):640–642, July 1997.
7. J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan, 1975.
8. P. Kalnis and D. Papadias. Proxy-server architecture for olap. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2001.
9. T. Loukopoulos and I. Ahmad. Static and adaptive distributed data replication using genetic algorithms. *in Journal of Parallel and Distributed Computing*, 64(11):1270–1285, November 2004.
10. M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems : Second Edition*. Prentice Hall, 1999.
11. A. Sanjay, V. R. Narasayya, and B. Yang. Integrating vertical and horizontal partitioning into automated physical database design. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 359–370, June 2004.
12. A. Sanjay, C. Surajit, and V. R. Narasayya. Automated selection of materialized views and indexes in microsoft sql server. *Proceedings of the International Conference on Very Large Databases*, pages 496–505, September 2000.
13. T. Stöhr, H. Märtens, and E. Rahm. Multi-dimensional database allocation for parallel data warehouses. *Proceedings of the International Conference on Very Large Databases*, pages 273–284, 2000.

This article was processed using the LaTeX macro package with LLNCS style