RouPar: Routinely and Mixed Query-Driven Approach for Data Partitioning

Ladjel Bellatreche[1], Amira Kerkad[1], Sebastian Breß[2], and Dominique Geniet[1]

[1] LIAS/ENSMA – Poitiers University
Futuroscope, France
{bellatreche,amira.kerkad,dgeniet}@ensma.fr
[2] University of Magdeburg
D-39016, Germany
bress@iti.cs.uni-magdeburg.de

Abstract. With the big data era and the cloud, several applications are designed around analytical aspects, where the data warehousing technology is in the heart of their construction chain. The interaction between queries in such environments represents a big challenge due to three dimensions: (i) the routinely aspects of queries, (ii) their large number, and (iii) the high operation sharing between queries. In the context of very large databases, these operations are expensive and need to be optimized. The horizontal data partitioning ($\mathcal{HDP}$) is a pre-condition for designing extremely large databases in several environments: centralized, distributed, parallel and cloud. It aims to reduce the cost of these operations. In $\mathcal{HDP}$, the optimization space of potential candidates for partitioning grows exponentially with the problem size making the problem NP-hard. In this paper, we propose a new approach based on query interactions to select a partitioning schema of a data warehouse in a divide and conquer manner to achieve an improved trade-off between the optimization algorithm's speed and the quality of the solution. The effectiveness of our approach is proven through a validation using the Star Schema Benchmark (100 GB) on Oracle11g.

1 Introduction

The big data era and the cloud bring two main aspects related to query optimization: (i) large amount of data [17] and (ii) the complexity of the repetitive nature of analytical queries that require multiple joins, aggregations, etc. One of the main characteristics of analytical queries in a data-warehouse environment is their interaction. *Multiple Query Optimization* (MQO) is a tool that captures this interaction by identifying common tasks among query plans (e.g., common sub-expressions, joins, etc.) through a single unified plan (sometimes called the *Multiple View Processing Plan* [19]). Unified plans represent a visualization tool of all queries and a data structure to capture the interaction among queries. Note that query interaction ($\mathcal{QI}$) has been largely used in the query optimization area in traditional databases [14] and recently in semantic databases (e.g., SPARQL

R. Meersman et al. (Eds.): OTM 2013, LNCS 8185, pp. 309–326, 2013.

queries) [8]. It contributed massively in selecting materialized views [19], caching and query scheduling [10,1].

On the other hand, the horizontal data partitioning ($\mathcal{HDP}$) is a pre-condition for designing databases on various environments: centralized [13], parallel [5], distributed [11], cloud [4,17], etc. Two versions of the $\mathcal{HDP}$ are available [3]: primary and derived $\mathcal{HDP}$. The primary $\mathcal{HDP}$ of a table $\mathcal{T}$ is performed using predicates defined on $\mathcal{T}$. The derived $\mathcal{HDP}$ is the partitioning of a table that results from predicates defined in other table(s). The derived $\mathcal{HDP}$ of a table $\mathcal{T}$ according to a partitioning schema of table $\mathcal{S}$ is feasible if and only if there is a join link between $\mathcal{T}$ and $\mathcal{S}$. This partitioning is well adapted in the context of relational data warehouses, where a fact table may be partitioned based on the partitioning schema of dimension tables. The derived $\mathcal{HDP}$ has two main advantages in relational data warehouses, in addition of classical benefits of data partitioning: **(1)** pre-computes join operation between fact table and dimension tables participating in the fragmentation process of the fact table. It uses the semi join operations to optimize the star join queries. The semi join has been largely used by commercial DBMS when optimizing star join queries [6]. **(2)** It optimizes selection operations defined on dimension tables. The number of horizontal fragments of the fact table generated by the partitioning procedure is given by the following equation: $\mathbf{N} = \prod_{\mathbf{i=1}}^{\mathbf{g}} \mathbf{m_i}$, where m_i is the number of fragments of the dimension table D_i. This number may be very large. Some studies propose to constrain this number [16].

The $\mathcal{HDP}$ has been implemented by academic and commercial DBMS and is nowadays an integral part of physical design in the most important DBMS such as *Oracle*, *DB2*, *SQL Server*, *PostgreSQL* and *MySQL*. A native Database Definition Language to support it is available by the means of various partitioning modes [15]: range, list, hash and composite.

The problem of selecting a partitioning schema is formalized as follows: Given a relational data warehouse with a set of d dimension tables $D = \{D_1, D_2, ..., D_d\}$ and one fact table F, a workload Q of queries $Q = \{Q_1, Q_2, ..., Q_n\}$, where each query Q_i $(1 \leq i \leq n)$ has an access frequency $AF(Q_i)$. The $\mathcal{HDP}$ selection problem consists in fragmenting the fact table F into N fragments minimizing the query cost subject to maintenance constraint $N \leq W$, where W is a threshold, configured by a database administrator (DBA), representing the maximal number of fact fragments that the partitioning algorithm may create. This problem has been shown as NP-hard [2,12]. A number of important algorithms were proposed in the context of databases and data warehouses. Most of these algorithms [2,9,11] do no consider the interaction between queries [18]. The main efforts focused on the algorithmic level, where several varieties of algorithms were proposed such as hill climbing, genetic, simulated annealing, and data mining driven algorithms. Each algorithm has its advantages and limitations. The algorithms that had a good performance use simple cost models to guide the exploration of solutions [12] and are time consuming. $\mathcal{HDP}$ has to integrate the routinely and mixed queries characterized applications build around the cloud. For example, in Alibaba's open data processing service distributed system

implemented based on the *MapReduce* framework for massive data analysis daily runs more than 20,000 queries due to the business needs and 40% of the statements share similar data operations and structural characteristics [7].

To incorporate the $\mathcal{QI}$ in the $\mathcal{HDP}$ selection process, three main alternatives may be distinguished: (1) the $\mathcal{QI}$ is delegated to the used cost model when estimating the overall query processing, (2) the development of $\mathcal{HDP}$ algorithms exploiting the data structure of the unified graph and (3) an hybrid alternative. The first alternative may increase the complexity of the algorithms, whereas the second and the third alternatives may reduce significantly this complexity and offering partitioning schema with high quality.

In this paper, we propose a new approach for partitioning relational data warehouses used by $\mathcal{SSDB}$ considering the interaction between queries at the data structure and cost model levels.

Our paper is structured as follows. In Section 2, a motivating example is described to show the basic ideas of our approach and to facilitate the description of our algorithms. Section 3 discusses the basic definitions and algebra for manipulating our data structure. Section 4 presents an algebra managing partitions. Section 5 presents our partitioning algorithm, called Elected Query Algorithm. Section 6 presents the results of our heuristics and compares the results with those obtained by state of art studies. The paper concludes in Section 7.

2 Motivating Example

Let us consider a star schema database with one fact table *Sales* and three dimension tables $\{Time$; $Product$; $Customer\}$. The data is accessed by a workload of 10 star join queries. The execution plans of all queries are merged in one graph called *Multi-View Processing Plan* (MVPP). MVPP is a graphical representation of a workload proposed in the context of Multi-Query Optimization [14]. Note that selection operations are pushed down after constructing MVPP. This graph has four main levels: **(a)** leaf nodes representing the base tables, **(b)** selection nodes that may be used to partition the databases (e.g., $\{s_1, s_2 \dots s_8\}$), **(c)** binary operation nodes (e.g., joins $\{j_1; j_2 \dots; j_9\}$) and **(d)** root nodes (e.g., grouping, ordering and projection $\{gop_1; gop_2; \dots gop_{10}\}$).

The join operation j_3 between table *Sales* and *Product* is shared by four queries Q_4, Q_7, Q_9 and Q_{10}. If we want to optimize the query Q_4 involving two selections $\{s_1, s_2\}$ and a join j_3, the $\mathcal{HDP}$ using the selection attributes may be a relevant optimization structure. The optimization of the query Q_4 impacts not only the query Q_4, but its benefit will propagate through all queries interacting with join node j_3: Q_7, Q_9 and Q_{10}. To spread the benefit along query plans, having the constraint of threshold[1] W limiting the number of fragments, the choice is very hard to be done. If we consider the query interaction in our example workload, we can group queries into four disjoint subsets depending on shared joins: $\{Q_1\}$; $\{Q_2; Q_3\}$; $\{Q_4; Q_7; Q_9; Q_{10}\}$; $\{Q_5; Q_6; Q_8\}$, that we call

[1] For the rest of the paper, N will refer to the number of generated fragments of the fact table in the partitioning schema.

groups. Note that the number of groups equals the number of direct joins with the fact table (four in the example). The reason is that queries in the same group are correlated by sharing at least the first join. This join is very expensive since it involves the largest table (fact table). If the first join is optimized using $\mathcal{HDP}$ based on its selections, all queries in the same group will benefit from the partitioning.

The threshold W imposes to start by most beneficial queries and predicates before the maximal number of fragments is reached. The problem at this level is : (1) which queries should we favor to steer partitioning, and (2) by which attributes should we start splitting and merging data?
The choice of important queries and attributes is crucial, because it allows lowering execution cost of the workload without exceeding the threshold W.

- If we choose to optimize the least expensive query of each group, we may ensure the gain propagation through all queries. This is caused by the interaction of each chosen query with the remaining ones in its group by sharing at least one common join. On the other hand, the cheapest query in a group may have fewer operations than others. These operations are concentrated in lower levels in the MVPP (e.g., join j_3 in query Q_4). The choice of the cheapest query allows to optimize lower nodes in the MVPP which are accessed by most (or even all) queries in the same group.
- From each elected query for partitioning, a set of attributes may participate in the $\mathcal{HDP}$ process. To start by the most important attributes, the usage rate may be a relevant criterion to spread a larger benefit through queries.

Observing this phenomenon, we deduced the impact of such properties to conduct $\mathcal{HDP}$. Therefore, we propose a new approach for $\mathcal{HDP}$ that exploits query interaction in a divide and conquer strategy to prune the large search space, and lower execution cost of the workload considering the constraint W.
We use a tailor-made data structure for representing partitioning schemas, which are potential solutions for $\mathcal{HDP}$. This structure contains participating attributes and sub-domain decomposition. If the data structure is based on our MVPP, it can be used to identify the candidate selection predicates for partitioning. The sub-domains decomposition considers existing predicates in the MVPP, and the least used attributes can be removed from the data structure in order to prune the search space.

3 Background

In this section, we present the formalization of our problem and the basic concepts that facilitate the understanding of our proposal.

We formalize the problem of $\mathcal{HDP}$ as follows: given (1) a data warehouse schema DW with one fact table F and a set of dimension tables $D_1, D_2, \ldots D_k$, (2) a workload $\mathcal{Q} = \{Q_1, Q_2, \ldots Q_n\}$, and (3) a constraint of a threshold W limiting the maximal number of fact fragments; the $\mathcal{HDP}$ aims at providing a new schema for the data warehouse that partitions dimension tables into dimension fragments using primary horizontal partitioning, and the fact table into N

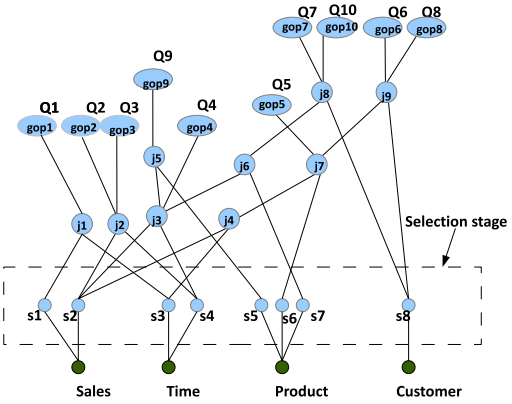

Fig. 1. Example of MVPP of 10 queries

Table 1. Existing Selections in the MVPP

Alias	Predicate	Table
s_1	$\sigma(quantity < 100)$	Sales
s_2	$\sigma(100 \leq quantity < 1000)$	Sales
s_3	$\sigma(season = "Automn")$	Time
s_4	$\sigma(season \in \{"Winter"; "Spring"\})$	Time
s_5	$\sigma(type = "T1")$	Product
s_6	$\sigma(color = "Color1")$	Product
s_7	$\sigma(type = "T2")$	Product
s_8	$\sigma(gender = "Female")$	Customer

fragments using derived horizontal partitioning from dimension fragments. The obtained schema DW' minimizes the execution cost of the workload $\mathcal{Q}$, and the number of fact fragments N must not exceed the threshold W: $N \leq W$.

3.1 Decomposition of an Attribute's Domain into Sub-domains

The $\mathcal{HDP}$ schema of a given $\mathcal{SSDB}$ is based on selection predicates defined in the queries of a given workload. Each selection predicate is defined as follows: $Att\ \theta\ Val$, where Att is the selection attribute, $\theta \in \{=, <, >, \leq, \geq, \in \ldots\}$ and Val is a value, a list or an interval in the values domain of the attribute Att. Table 1 contains the set of selection predicates in the MVPP shown in Figure 1.

The sub-domains are intervals, lists, or values in the domain of a selection attribute. The decomposition of a domain to many sub-domains is usually done either by the DBA based on experiments or by query selection predicates [16]. Some attributes are harder to decompose then others. In the traditional $\mathcal{HDP}$, the attribute domain decomposition is performed independently from the MVPP. As a consequence, this decomposition is static and fixed. In this paper, we propose an incremental encoding that does the decomposition systematically based on the MVPP. Figure 2-a illustrates an example of attribute domain decomposition obtained from the MVPP.

3.2 Encoding Schema

Usually, each $\mathcal{HDP}$ schema may be represented by a fixed coding [2], which is a juxtaposition of arrays representing selection attributes of the selection stage of the MVPP (Figure 1), where each array represents the domain decomposition of an attribute. The value of each cell of a given array representing an attribute A_i^k of dimension table D_k belongs to $[1, n_i]$, where n_i represents the number of sub-domains of the attribute A_i^k. An example of a $\mathcal{HDP}$ schema encoding is given in

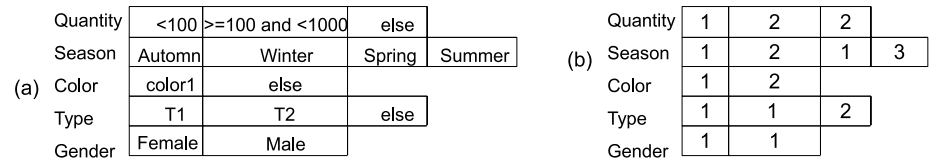

Fig. 2. Selection attributes and their sub-domains (a) and encoding schema (b)

Figure 2-b. Based on this representation, $\mathcal{HDP}$ schema of each dimension table D_j is generated as follows:

1. All cells of a partitioning attribute A_i^k of D_j have different values: this means that all sub-domains will be used to partition D_j. For instance, the cells of each partitioning attribute *Color* in Figure 2-(b) are different. So attribute *Season* will participate in partitioning process by splitting table *Time* into three disjoint subsets.
2. All cells of a partitioning attribute A_i^k have the same value: this means that it will not participate in the partitioning process. Attribute *Gender* for example will not participate in $\mathcal{HDP}$.
3. Some cells of an attribute sub-domains have the same value: their corresponding sub-domains will be merged into one fragment. For example, *Season* will split data into three subsets : {Automn,Spring}, {Summer} and {Winter}.

The primary horizontal partitioning on a dimension table D_i is applied using all participating attributes of this table. The set of participating attributes provide all different combinations of predicates that generate dimension fragments of D_i. For example, table *Product* is partitioned based on attributes *Type* and *Color* to generate four fragments as follows :

- $Product_1$: $Type \in \{T_1, T_2\}$ and $Color =$ "$Color1$"
- $Product_2$: $Type \notin \{T_1, T_2\}$ and $Color =$ "$Color1$"
- $Product_3$: $Type \in \{T_1, T_2\}$ and $Color \neq$ "$Color1$"
- $Product_4$: $Type \notin \{T_1, T_2\}$ and $Color \neq$ "$Color1$"

If attribute *Color* for example were not participating, table *Product* would provide two fragments : $Product_1$ where $Type \in \{T_1, T_2\}$, and $Product_2$ where $Type \notin \{T_1, T_2\}$. Most of partitioning approaches use a static coding to represent and handle their solutions. In this work, we propose an incremental way to generate the encoding in order to keep only relevant attributes and sub-domains for partitioning process. In the experimental study, we will also show that resolution algorithms are encoding-sensitive.

4 Algebra

The definition of an encoding and partitioning algebra are required to deal with $\mathcal{HDP}$. To do so, we propose two functions : *Horizontal Split* and *Vertical Split* to incrementally generate the encoding used to represent the partitioning schema.

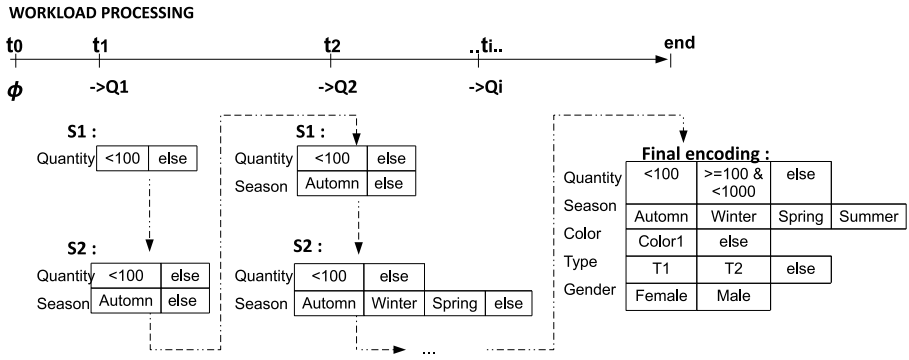

Fig. 3. Incremental encoding by Horizontal and Vertical Split on sub-domains of attributes

Then, two other functions : *Split* and *Merge* are used to manage partitions in the schema.

4.1 Generating Incremental Encoding

The encoding proposed in our work aims at providing a pruned set of attributes and sub-domains which are relevant to the MVPP. To generate new attributes in the encoding schema, a *Vertical Split* is applied to add a new array representing the new attribute. The set of sub-domains of each attribute A^k is updated by performing *Horizontal Split* on the existing array of A^k, which creates new ranges in sub-domains of A^k.

To obtain the set of selection attributes, we proceed by doing a first pruning of all attributes. The idea is to keep only attributes and their sub-domains that figure in the MVPP. The construction of the encoding is done incrementally by exploring query plans one by one, and updating the coding by new attributes and sub-domains, using the two functions : *Horizontal and Vertical Split*. The principle of this coding is to start from an empty set of attributes, and for each query, add its required selection attributes by creating new arrays. Each array contains one range. When a selection is found for the current query, three operations can be performed:

1. If the attribute does not exist in the schema, apply a vertical split to extend the schema vertically by adding a new array for the attribute. The range is split into many parts to cover the new sub-domains. An *else* range is added to ensure completeness. Figure 3 shows the added attributes to the empty set of selection attributes by query Q_1 (*Quentity, Season*).
2. If the attribute already exists, apply a horizontal split on the *else* range to add the new sub-domains. An example of added sub-domains by query Q_2 for attribute *season* is illustrated in Figure 3.
3. Finally, if the administrator knows the value remaining in the *else* range, it is replaced by this value. In Figure 3, the *else* is replaced by *"Male"* for attribute *Gender*, and by *"Summer"* for attribute *season*.

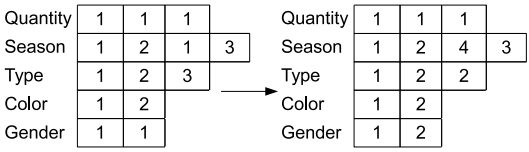

Fig. 4. Performing Split and Merge operations on the partitioning schema

The result of this phase is an encoding schema containing the set of selection attributes and their associated sub-domains. To run our encoding algorithm on our example, we start from an empty set of attributes. When the first query Q_1 is processed, two new attributes are added with their sub-domains in Q_1. The query Q_2 does not have new attributes, but it has new predicates that split horizontally the existing schema (Figure 3). Finally, when the last query Q_{10} is processed, the schema is adjusted with some known values to replace the *else* (Figure 3).

4.2 Partitioning Primitives

In most commercial DBMS, the $\mathcal{HDP}$ proposes primitives to manage partitions by the use of two basic functions: *Merge* and *Split*. The former consists in merging two partitions in one, and the later consists in splitting a partition into two partitions. These operations are performed on the physical level of a database. We propose to use these primitives to generate partitioning schemas.

More formally, the *Merge* and *Split* operations have the following signature: $Merge(sd_i^k, sd_j^k, A^k, PS) \rightarrow PS'$, where the *Merge* function is applied on two sub-domains sd_i^k and sd_j^k of the attribute A^k to get them in one partition. $Split(sd_i^k, A^k, PS) \rightarrow PS'$, where the *Split* function is applied on the sub-domain sd_i^k in order to be divided into two sub-domains if and only if it covers at least two sub-domains of attribute A^k. Figure 4 shows one *Merge* performed on attribute *Type*, and two *Splits* on attributes *Season* and *Gender*.

5 The Elected Query Algorithm

In this section, we show how query interaction is exploited in the $\mathcal{HDP}$ process by promoting overlapping join nodes. To exploit these nodes, we apply a divide and conquer algorithm (called *"Elected Query Algorithm"* (EQA)) to prune the space of relevant attributes and their sub-domains. We describe how EQA conducts the $\mathcal{HDP}$, and provide the detailed algorithm. Starting from the obtained set of candidate attributes kept in the encoding schema, a partitioning schema needs to be generated to decrease execution cost as possible. As already mentioned, existing solutions are either (1) *simplistic*, or (2) *heuristics*, based on cost models. To achieve an improved trade-off between the two approaches, we propose to add a new criterion to conduct the $\mathcal{HDP}$ process, to get a better efficiency and less complexity.

5.1 Electing Queries

As discussed in the motivating example, the join operation is very expensive, and at the same time, queries interacting with each other may share at least the first join. Considering the first join node is crucial, because it serves other queries sharing that node. To capture and exploit the interaction of queries and reduce the complexity of resolution algorithms, we propose an algorithm named *Elected Query Algorithm* (EQA) to get the set of most beneficial queries. Starting from the MVPP, the EQA generates groups of correlated queries, such that each couple of queries sharing at least one join node are in the same group. Figure 5 shows the obtained groups from the MVPP in the motivating example. From each group, the algorithm aims at electing one query to steer $\mathcal{HDP}$. We call this query : *Elected Query* (EQ) and consider it as the most important one in its group. The choice of the EQ_i for group i is done regarding the cost and the overlapping nodes of the query. We propose to choose the EQ as the one with minimal cost. The minimal cost allows having a minimal set of operations (less joins may sire less selections). The number of overlapping nodes is not considered, because it shares at least one join which may propagate the benefit through all queries in the same group. The EQA is given in the following algorithm. In addition, choosing the query with least cost allows optimizing lower nodes (operations) in the MVPP, which are the most used, so that it propagates the benefit through most queries.

The EQA aims at giving a subset of relevant selection attributes and sub-domains to guide the process of HDP. The returned set of elected queries may prune the search space and discard less efficient possibilities such as less frequent selections and joins. The obtained attributes from MVPP in the previous phase will be pruned again by considering, at first, only attributes participating in the elected queries. In the partitioning process, the algorithm satisfies the elected queries rather than randomly chosen predicates. This pruning phase may reduce considerably the complexity of the partitioning algorithm by reducing the size of the search space. The details of the algorithm of $\mathcal{HDP}$ using EQA are given in the next section.

Algorithm 1. ElectedQueryAlgorithm()

```
1: for all Q_i ∈ MVPP do
2:    Group(Q_i) := Q_i;
3: end for
4: for all (Q_i, Q_j) ∈ MVPP do
5:    if ( (i! = j)and(interaction(Q_i, Q_j) = true)) then
6:       merge(Group(Q_i), Group(Q_j));{grouping queries based on interaction}
7:    end if
8: end for
9: for all Group a do
10:   sort(a);{sort queries by minimal cost}
11:   elect(a, 1);{elect a query by minimal cost}
12: end for
```

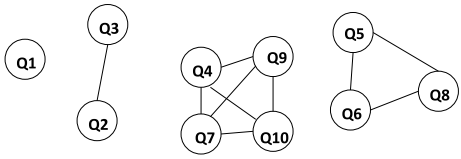

Fig. 5. Generating groups (subsets) of interacting queries from the MVPP

5.2 Exploiting EQA for HDP

Our algorithm for $\mathcal{HDP}$ called *Electing Queries for Horizontal Data Partitioning* (EQHDP) starts from the MVPP, and performs pruning of selection attributes by electing most "beneficial" queries using EQA. The EQA groups queries into disjoint subsets, then sorts queries inside each group by minimal cost. The EQA returns the set of elected queries EQ_i from each group i. The obtained set of elected queries from EQA is sorted by descending costs. That way, the most expensive queries are optimized first and the algorithm can ensure that the most expensive part of the workload is optimized before the threshold W is exceeded.

EQA computes first the elected queries and then prunes the schema of attributes depending on the elected queries requirements, i.e., only required attributes are taken, the others are ignored. The sub-domains are tagged with the total number of elected queries using each one. Let u_i be the number of EQ using a sub-domain of attribute a_i and let k_i be the maximal value of u_{ij} in a_i. The set of attributes is sorted by maximal use (value of k_i) in order to start by partitioning on most used attributes before the W is exhausted. After sorting the attributes, each attribute a_i is split/merged as follows:

1. The sub-domains, which are not used by any elected query ($u_i = 0$), are grouped in one partition P_0;
2. The most used sub-domains having k_i elected queries accessing them are all grouped in one partition P_k (where k_i is the maximal usage value of the attribute a_i);
3. Finally, if $N > W$ or $k_i \neq 0$ then, the remaining sub-domains are merged with the partition P_0; otherwise, the sub-domains accessed by $k_i - 1$ elected queries are grouped in a new partition. The same operation is repeated until $k_i = 0$ or $N > W$.

This allows creating partitions based on the requirements of most important queries. If partitioning is still possible ($N < W$), the obtained partitions are split depending on the correlation between queries accessing each sub-domain of the partition. If two subsets of elected queries require some sub-domains independently, a new partition is created to contain sub-domains used independently from the others.

If N is less than W after these merge and split operations regarding only the elected queries, then partitioning is still possible. For this reason, the optimization process moves to other queries to improve their performance as well. The next set of queries is the successors of current EQs that verify the same criterion as the already processed queries. If at least one group has still a successor,

Algorithm 2. AlgorithmeEQHDP()

```
1: generate_encoding();
2: EQA();{grouping by Elected Query Algorithm}
3: split_all();
4: E := elected()
5: while E not empty do
6:    prune_encoding(E);
7:    sort(E);
8:    S := required_attributes(E);
9:    usage(S);
10:   attributes_sort(S)
11:   while ((N < W)and(S not empty)) do
12:      for all sd ∈ SubDomains(a) do
13:         j := U(sd);
14:         merge(sd, P_j); {merge sub-domains in partition P_j}
15:         S := S − {a};
16:      end for
17:   end while
18:   split_disjoint();{split sub-domains used by disjoint sets of queries}
19:   E := elected_successors();
20: end while
```

a new set of elected queries is generated by the found successors of all groups. The same process is applied by extending the encoding schema with the new set of selection attributes incrementally. The partitioning is done until $N = W$ or no more queries are left in **any** group. We formalize our proposed EQHDP approach in the following algorithm:

6 Experimental Study

In this section, we contribute an extensive experimental study to validate our proposal. We start by presenting our data set and a simulator connected to Oracle11g for performing experiments. The obtained results are discussed and justified to reveal particularities of our proposal. Finally, we discuss another advantage of our proposal regarding query profiles and similarity that may support other optimization techniques.

6.1 Data Set

In our experimental study, we use the *Star Schema Benchmark* (SSB)[2] with a scale factor of 100, which generates 100 GB of data. The data warehouse contains a fact table *Lineorder*, and four dimension tables : *Customer, Supplier, Part* and *Dates* stored on Oracle11g located on a server of 32GB of RAM and an Intel® Xeon® CPU of 2x2.40GHz. The workload consists of 22 star join queries running under Oracle11g.

[2] `http://www.cs.umb.edu/~poneil/StarSchemaB.pdf`

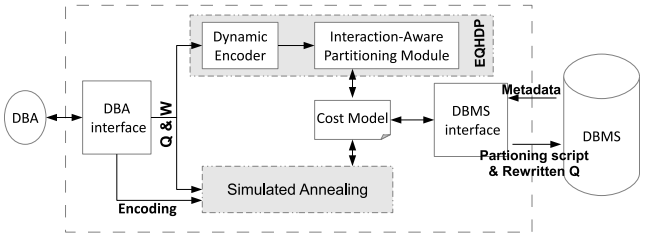

Fig. 6. The Components of our $\mathcal{HDP}$ Simulator

6.2 Infrastructure Components and Settings

To conduct our experiments, we use a simulator written in Java and connected to an Oracle11g instance. We illustrate the infrastructure of the experimental study in Figure 6. The main components are:

- *DBA interface* allows the DBA to provide the workload and the threshold $\mathcal{W}$ for both algorithms. The DBA provides also a pre-shaped encoding schema for Simulated Annealing.
- *DBMS interface* is responsible of extracting meta data from a Oracle11g DBMS, and at the same time, providing the partitioned schema with the rewritten queries to the DBMS.
- *Dynamic encoder* provides an encoding schema based on our algebra to be used by the partitioning module.
- *Interaction-Aware Partitioning Module* captures query interaction to steer the $\mathcal{HDP}$ using the generated encoding.
- *Cost Model* estimates the cost of executing the optimized queries on the obtained partitioning schema by both algorithms.

The entries are our database and its workload. The threshold W is varied to show its impact on performance. The metric used to quantify the quality of potential solutions is our cost model that provides the number of input/output operations required for executing queries. We compare EQHDP with *Simulated Annealing* (SA), because SA is widely used in such hard optimization problems. It computes results efficiently compared to a genetic algorithm [2]. To tune our SA, we set the iteration number to 500 and the temperature to 300. As the SA is a non-deterministic algorithm, it is run several times (from 5 to 10 times depending on the variance) to get the average performance. Contrary to our algorithm, which generates its encoding dynamically, the SA needs a pre-shaped encoding as additional input.

6.3 Simulation Results

Threshold Impact. In our first experiments, we compare the $\mathcal{HDP}$ schemas given by SA and our EQHDP algorithm. In Figure 7, the execution cost of the workload is given by the number of input/output operations. The results show that

EQHDP outperforms SA except for very large thresholds ($\mathcal{W} \geq 300$), with narrow differences in performance. The reason of that EQHDP efficiency is that it performs split/merge operations *on demand*, i.e., it aims at satisfying the largest number of queries, starting from the most efficient ones that contribute largely in improving the workload performance.

If $\mathcal{W} < 300$, SA performs a high number of iterations looking randomly for the best solution. In contrast, EQHDP detects and performs systematically the required moves to get the most beneficial partitioning schema. If $\mathcal{W}$ is very large (≥ 300), the EQHDP stops when all queries and their predicates are satisfied. Although further split/merge operations are still possible to decrease execution cost. These extra operations cannot be performed by the EQHDP, but can be reached randomly by SA. Note that $\mathcal{W}$ is the number of fact fragments, which is the number of resulting subschemas. The total number of fragments in the schema is much higher than $\mathcal{W}$.

We conclude from these experiments that the EQHDP can reach a high level of efficiency compared to a meta-heuristic. However, for very high values of threshold $\mathcal{W}$, the SA is more efficient with narrow differences.

Encoding Sensitivity. Another major advantage of our approach is that our algorithm is self-encoding. This means that the coding phase is a part of our algorithm and provides an encoding schema depending on the workload, contrary to the SA that needs a pre-computed encoding to run over. The given schema for SA is generally provided by the DBA, and may contain a better repartition of sub-domains for selection attributes.

To study encoding sensitivity of partitioning algorithms, we performed two experiments as follows: (1) The first one using incremental encoding process of the EQHDP, the obtained encoding is provided to the SA to run over it. (2) The second one is done using a DBA encoding schema for both algorithms. Figure 8 shows that using DBA encoding may make the SA more efficient, because more appropriate ranges and sub-domains are provided by the DBA, if he knows the workload and the data. But it may penalize the EQHDP because some query requirements do not fit in the given encoding.

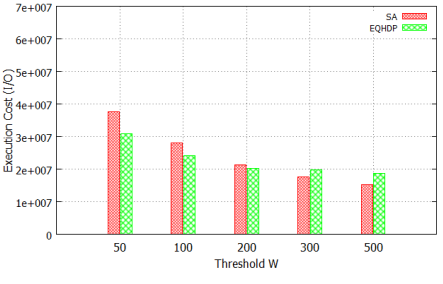

Fig. 7. Comparing algorithms performance by threshold variation

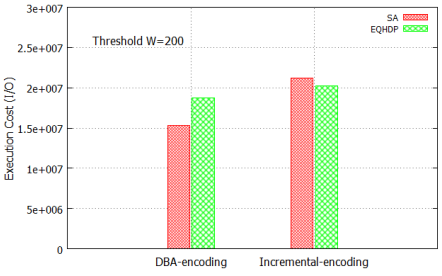

Fig. 8. Encoding sensitivity

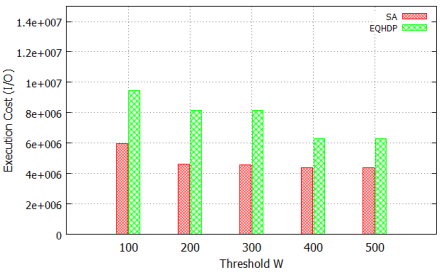

Fig. 9. Running SA and EQHDP on a workload with no interacting joins

Fig. 10. The impact of query interaction on the efficiency of both algorithms

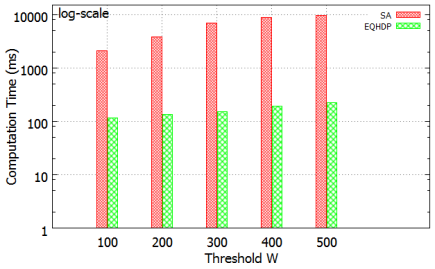

Fig. 11. Comparing reactivity of Simulated Annealing with the EQHDP algorithm

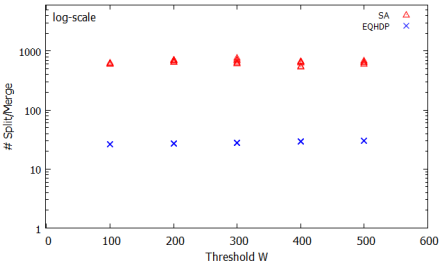

Fig. 12. Comparing #split/merge operations obtained by SA and EQHDP

From this experiments, we conclude that the resolution algorithms for the $\mathcal{HDP}$ are indeed encoding-sensitive. The SA cannot generate its own encoding. In the case where the DBA is not able to provide an efficient encoding schema, which is usually the case in real life, the EQHDP is more advantageous, because it is more autonomous and efficient.

Query Interaction Impact. In order to check the impact of query interaction on our approach, we use a new workload having no shared joins between queries (no common sub-expressions). Figure 9 shows the execution cost of the new workload optimized by both algorithms. The results show that the EQHDP is not efficient enough compared to the SA in case no interaction exists between queries. This is due to the fact that only a subset of queries is optimized, but the gain cannot propagate through other queries.

We compare now performance of EQHDP with and without query interaction. Figure 10 we can see that the EQHDP outperforms SA when queries are correlated, but execution cost of the workload is poor in case of no query interactions. From these experimentations, we argue that our approach is not efficient in case the workload does not have correlated queries, because the optimization process in our approach is guided mainly by query interaction.

Reactivity. We investigate the reactivity of both algorithms. Figure 11 shows the response time of the algorithms. By varying the threshold W, the runtime of SA increases dramatically, contrary to EQHDP's runtime, which remains very low compared to SA. This high computation cost is typical in such algorithms, contrary to our algorithm, which is much faster. This is due to the pruned search space phases and the interaction-based partitioning process.

Impact on Number of Split and Merge Operations. In the next experiment, we also check the speed of algorithms, but now in terms of moves and elementary operations of split and merge. Figure 12 shows the number of split/merge operations performed by the algorithms for different thresholds $\mathcal{W}$. As the SA is a non-deterministic algorithm, it is run several times, and different number of operations (split/merge) are given for each value of $\mathcal{W}$. On the other hand, the EQHDP is a deterministic algorithm, so that a unique solution is computed for each $\mathcal{W}$, and a fixed number of moves (split/merge) is performed. As shown in Figure 12, SA is very costly in terms of partitioning moves compared to the EQHDP for the same reasons as in previous experiments; i.e., our approach is interaction-based contrary to the SA, which explores a large search space randomly.

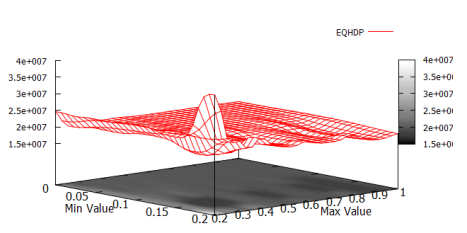

Fig. 13. Selectivity Factor intervals pour data partitioning

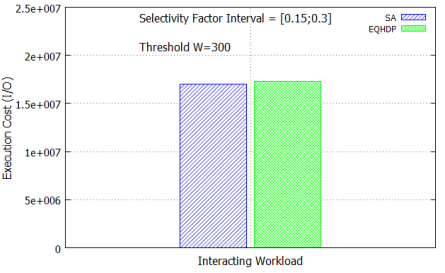

Fig. 14. Improving the EQHDP by defining selectivity factor interval

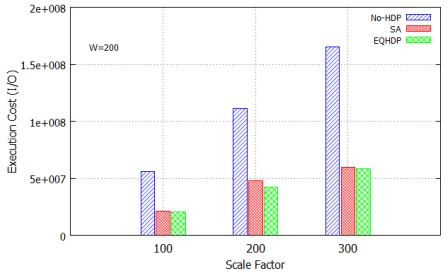

Fig. 15. Impact of Size Evolution on Query Performance

Fig. 16. Validating simulation results on Oracle11g

Selectivity Factor. Some relevant partitions for the EQHDP may not be suitable for partitioning process because of their selectivity factors which can be either too high or too low. Such selectivity factors generate unbalanced partitions and may penalize many queries. To avoid these partitions, we extend our algorithm by a new pruning criterion that consists of a selectivity factor interval.

We study the suitable selectivity factors for our workload as shown in Figure 13. In this experiment, we vary the minimal and maximal values of selectivity. Indeed, the highest (≈ 1) and lowest (≈ 0) selectivity penalize the workload. However, the best optimization values are found for the minimal value 0.15 and maximal value 0.30. This means that the sub-domains with selectivity factors in $[0.15; 0.30]$ are the most suitable ones for the optimization process. We compare the improved EQHDP algorithm with SA in the experiment illustrated by Figure 14, which shows that the yield of EQHDP is improved by this criterion even with higher Threshold ($\mathcal{W} = 300$) to be closer to SA performance.

Scale-up Analysis. In the final experiment, we study the impact of the database volume on the performance of optimization algorithms, as depicted in Figure 15. Varying the scale factor of the SSB allows to increase the database volume to 100 GB, 200 GB and 300 GB from a scale factor of 100, 200 and 300, respectively. When the database is in constant growth, the EQHDP algorithm remains more efficient than the SA for a Threshold of 200. This proves the effectiveness of our approach in the context of very large databases.

6.4 Validation

In order to validate our simulation results, we deploy both partitioning schemas obtained by SA and EQHDP with the same data set used in the simulations. Similarly to the theoretical results, Figure 16 shows the efficiency of our EQHDP approach compared with SA. From our experiments, we conclude that our approach remains considerably faster than SA especially when the threshold W is larger, which increases the computation cost of SA significantly. Additionally, our algorithm is deterministic, self-encoding, fast and capable to reach a high efficiency level of optimization based on query interaction. The algorithm is more efficient in highly correlated workloads, which is usually the case in a data warehouse environment.

7 Conclusion

The exploration of very extremely large databases is performed by the use of complex queries, involving joins and aggregations. These queries usually interact with each other, because the end users (scientists and decision makers) usually focus on identical parts of the databases. To optimize these queries, we propose the use of $\mathcal{HDP}$ because it is a pre-condition for designing several types of databases: centralized, parallel, distributed, cloud. In this paper, we motivate the consideration of query interaction in selecting $\mathcal{HDP}$ schemes for relational

data warehouses. This interaction is handled by the use of incremental encoding of any horizontal partitioning schema. We contribute an algorithm for $\mathcal{HDP}$ that considers our encoding data structure. Our approach promotes the more sharable nodes of a unified query graph. Contrary to other partitioning algorithms, where all queries have the same probability to partition the database, our algorithm exploits the property of the big data and cloud environments, where queries have a high interaction with other queries. Another dimension is that our algorithm discards queries involving selection predicates either with high or low selectivity factors. We compare our algorithm with the simulated annealing algorithm, which is known as an efficient solution for $\mathcal{HDP}$. The results of our algorithms in terms of query processing cost and execution time are encouraging.

Currently, we are working on deploying our proposal in a cloud platform in considering the data placement problem in private and public clouds.

References

1. Ahmad, M., Aboulnaga, A., Babu, S., Munagala, K.: Interaction-aware scheduling of report-generation workloads. VLDB Journal 20(4), 589–615 (2011)
2. Bellatreche, L., Boukhalfa, K., Richard, P.: Referential horizontal partitioning selection problem in data warehouses: Hardness study and selection algorithms. International Journal of Data Warehousing and Mining 5(4), 1–23 (2009)
3. Ceri, S., Negri, M., Pelagatti, G.: Horizontal data partitioning in database design. In: SIGMOD, pp. 128–136. ACM (1982)
4. Curino, C., Jones, E.P.C., Popa, R.A., Malviya, N., Wu, E., Madden, S., Balakrishnan, H., Zeldovich, N.: Relational cloud: a database service for the cloud. In: CIDR, pp. 235–240 (2011)
5. Curino, C., Zhang, Y., Jones, E.P.C., Madden, S.: Schism: a workload-driven approach to database replication and partitioning. PVLDB 3(1), 48–57 (2010)
6. Galindo-Legaria, C.A., Grabs, T., Gukal, S., Herbert, S., Surna, A., Wang, S., Yu, W., Zabback, P., Zhang, S.: Optimizing star join queries for data warehousing in microsoft sql server. In: ICDE, pp. 1190–1199. IEEE (2008)
7. Ge, X., Yao, B., Guo, M., Xu, C.: Lsshare: An efficient multiple query optimization system in the cloud. To appears in DEXA (2013)
8. Le, W., Kementsietsidis, A., Duan, S., Li, F.: Scalable multi-query optimization for sparql. In: ICDE, pp. 666–677. IEEE (2012)
9. Mahboubi, H., Darmont, J.: Enhancing xml data warehouse query performance by fragmentation. In: SAC, pp. 1555–1562. ACM (2009)
10. O'Gorman, K., Agrawal, D., El Abbadi, A.: Multiple query optimization by cache-aware middleware using query teamwork. In: Proceedings of the International Conference on Data Engineering (ICDE), p. 274 (2002)
11. Özsu, M.T., Valduriez, P.: Principles of Distributed Database Systems, 2nd edn. Prentice Hall (1999)
12. Papadomanolakis, S., Ailamaki, A.: Autopart: Automating schema design for large scientific databases using data partitioning. In: SSDBM, pp. 383–392. IEEE (2004)
13. Sanjay, A., Narasayya, V.R., Yang, B.: Integrating vertical and horizontal partitioning into automated physical database design. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 359–370 (2004)

14. Sellis, T.K.: Multiple-query optimization. ACM Transactions on Database Systems 13(1), 23–52 (1988)
15. Oracle Data Sheet: Oracle partitioning. White Paper (2007), http://www.oracle.com/technology/products/bi/db/11g/
16. Stöhr, T., Märtens, H., Rahm, E.: Multi-dimensional database allocation for parallel data warehouses. In: VLDB, pp. 273–284. Morgan Kaufmann Publishers Inc. (2000)
17. Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Zhang, N., Anthony, S., Liu, H., Murthy, R.: Hive - a petabyte scale data warehouse using hadoop. In: ICDE, pp. 996–1005. IEEE (2010)
18. Tzoumas, K., Deshpande, A., Jensen, C.S.: Sharing-aware horizontal partitioning for exploiting correlations during query processing. PVLDB 3(1), 542–553 (2010)
19. Yang, J., Karlapalem, K., Li, Q.: Algorithms for materialized view design in data warehousing environment. In: Proceedings of the International Conference on Very Large Databases, pp. 136–145 (August 1997)