HAL open science

Enhancing XML Data Warehouse Query Performance by Fragmentation

Hadj Mahboubi, Jérôme Darmont

To cite this version:

Hadj Mahboubi, Jérôme Darmont. Enhancing XML Data Warehouse Query Performance by Fragmentation. 24th Annual ACM Symposium on Applied Computing (SAC 2009), 2009, Hawaii, United States. pp.1555-1562. hal-00411232

HAL Id: hal-00411232

https://hal.science/hal-00411232v1

Submitted on 26 Aug 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Enhancing XML Data Warehouse Query Performance by Fragmentation

Hadj Mahboubi and Jérôme Darmont
University of Lyon (ERIC)
5 avenue Pierre Mendès-France
69676 Bron Cedex
France
{hadj.mahboubi,jerome.darmont}@eric.univ-lyon2.fr

ABSTRACT

XML data warehouses form an interesting basis for decision-support applications that exploit heterogeneous data from multiple sources. However, XML-native database systems currently suffer from limited performances in terms of manageable data volume and response time for complex analytical queries. Fragmenting and distributing XML data warehouses (e.g., on data grids) allow to address both these issues. In this paper, we work on XML warehouse fragmentation. In relational data warehouses, several studies recommend the use of derived horizontal fragmentation. Hence, we propose to adapt it to the XML context. We particularly focus on the initial horizontal fragmentation of dimensions' XML documents and exploit two alternative algorithms. We experimentally validate our proposal and compare these alternatives with respect to a unified XML warehouse model we advocate for.

Keywords

XML data warehouses, Multidimensional model, XML-native databases, performance, fragmentation.

1. INTRODUCTION

Decision-support applications currently exploit more and more heterogeneous data from various sources. In this context, the eXtensible Markup Language (XML) is becoming a standard for representing complex business data [3] and can greatly help in their integration, warehousing and analysis. Many efforts toward XML data warehousing have indeed been achieved in the past few years [7, 21], as well as efforts for extending the XQuery language with near On-Line Analytical Processing (OLAP) capabilities such as advanced grouping and aggregation features [3]. This research notably aims at taking into account specificities of XML data (e.g., heterogeneous number and order of dimensions or complex measures in facts, ragged dimension hierarchies, etc.) that would be intricate to handle in a relational environment.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SAC'09 March 8-12, 2009, Honolulu, Hawaii, U.S.A.
Copyright 2009 ACM 978-1-60558-166-8/09/03 ...$5.00.

XML-native database management systems (DBMSs) supporting XQuery should form the basic storage component of XML warehouses. However, they currently present poor performances when dealing with the large data volumes and complex analytical queries that are typical in data warehousing. Distributing a warehouse on a grid-like network can contribute to improve storage and query performance. Such a framework indeed provides both computing power and distributed storage resources. Thus, it can be used to handle large data warehouses efficiently.

Traditionally, the distribution process starts with data fragmentation. Fragmentation consists in splitting a data set into two or more parts (fragments) such that the combination of the fragments yields the original warehouse without any loss nor addition of information. In the relational context, derived horizontal fragmentation is acknowledged as best-suited to data warehouses, because it takes decision-support query requirements into consideration and avoids computing unnecessary join operations [2]. Several approaches have also been proposed for XML data fragmentation, but they do not take data warehouse multidimensional architectures (i.e., star-like schemas) into account.

In this paper, we thus propose to adapt derived horizontal fragmentation techniques developed for relational data warehouses to the XML context. We particularly focus on the initial horizontal fragmentation of dimensions and adapt and compare the two major algorithms that address this issue: the predicate construction [17] and the affinity-based [16] strategies.

Adapting these relational techniques onto XML warehouses requires a well-identified XML warehouse model. Unfortunately, although XML warehouse architectures from the literature share a lot of concepts (mostly originating from classical data warehousing), they are nonetheless all different. Hence, as a secondary contribution of this paper, we propose a unified, reference XML data warehouse model that synthesizes and enhances existing models, and on which we can base our fragmentation work.

The remainder of this paper is organized as follows. First, we introduce the state of the art regarding XML data warehouses, as well as our own reference XML warehouse model (Section 2). Then, we present general definitions about fragmentation and discuss existing research related to relational data warehouse and XML data fragmentation (Section 3). We detail the specifics of our adaptation to XML data warehouse fragmentation (Section 4) and experimentally demonstrate that proper fragmentation significantly reduces the

execution time of analytical XQueries (Section 5). We finally conclude this paper and hint at future research directions (Section 6).

2. XML DATA WAREHOUSING

2.1 Related work

Several studies address the issue of designing and building XML data warehouses. They propose to use XML documents to manage or represent facts and dimensions. The main objective of these approaches is to enable a native storage of the warehouse and its easy interrogation with XML query languages.

Pokorný models a XML-star schema in XML by defining dimension hierarchies as sets of logically connected collections of XML data, and facts as XML data elements [21]. Hümmer *et al.* propose a family of templates, named XCube, enabling the description of a multidimensional structure (dimension and fact data) for integrating several data warehouses into a virtual or federated warehouse [11]. Rusu *et al.* propose a methodology, based on the XQuery technology, for building XML data warehouses. This methodology covers processes such as data cleaning, summarization, intermediating XML documents, updating/linking existing documents and creating fact tables [22]. Facts and dimensions are represented by XML documents built with XQueries. Park *et al.* introduce a framework for the multidimensional analysis of XML documents, named XML-OLAP [20]. XML-OLAP is based on an XML warehouse where every fact and dimension is stored as an XML document. The proposed model features a single repository of XML documents for facts and multiple repositories of XML documents for dimensions (one repository per dimension). Eventually, Boussaïd *et al.* propose an XML-based methodology, named X-Warehousing, for warehousing complex data [7]. They use XML Schema as a modeling language to represent user analysis needs.

2.2 XML data warehouse reference model

The studies enumerated in Section 2.1, though all different, more or less converge toward a unified XML warehouse model. They mostly differ in the way dimensions are handled and the number of XML documents that are used to store facts and dimensions. A performance evaluation study of these different representations showed that representing facts in one single XML document and each dimension in one XML document allowed the best performance [6]. Moreover, this representation allows to model constellation schemas without duplicating dimension information. Several fact documents can indeed share the same dimensions. Furthermore, since each dimension and its hierarchical levels are stored in one XML document, dimension updates are more easily and efficiently performed than if dimensions were either embedded with the facts or all stored in one single document.

Hence, we adopt this architecture model. More precisely, our reference data warehouse is composed of the following XML documents (Definition 1):

1. *dw-model.xml* that represents warehouse metadata, the XML graph representing warehouse metadata is denoted $G_{dw-model}$;
2. a set of $facts_f.xml$ documents that each store information related to set of facts f;
3. a set of $dimension_d.xml$ documents that each store a given dimension d's member values.

Definition 1. *An XML document is defined as a graph (XML graph) whose nodes represent document elements or attributes, and edges represent the element / sub-element (or parent-child) relationship. Edges are labeled with element or attribute names.*

A $facts_f.xml$ document stores facts (Figure 1(a)). The document root node, *FactDoc*, is composed of *fact* subelements that each instantiate a fact, i.e., measure values and dimension references. These identifier-based references support the fact-to-dimension relationship. The XML graph representing fact set f is denoted G_{facts_f}.

A $dimension_d.xml$ document helps instantiate one dimension, including any hierarchical level (Figure 1(b)). Its root node, *dimension*, is composed of *Level* nodes. Each one defines a hierarchy level composed of *instance* nodes that each define the level's member attribute values. In addition, an *instance* element contains *Roll-up* and *Drill-Down* attributes that define the hierarchical relationship within dimension d. The XML graph representing dimension d is denoted $G_{dimension_d}$.

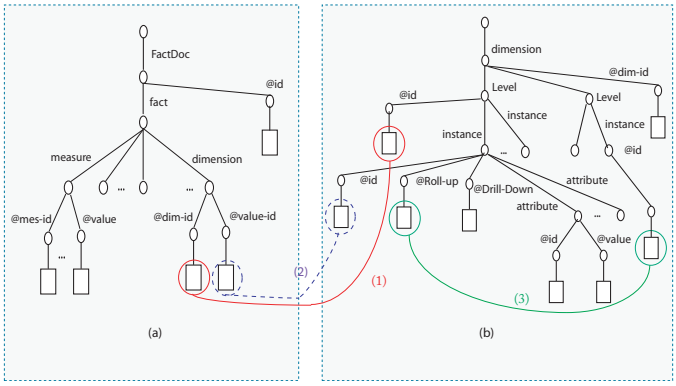

Figure 1: $facts_f.xml$ (a) and $dimension_d.xml$ (b) graph structure

3. DATABASE FRAGMENTATION

3.1 Definition

There are three fragmentation types in the relational context [2]: vertical fragmentation, horizontal fragmentation and hybrid fragmentation.

Vertical fragmentation splits a relation R into sub-relations that are projections of R with respect to a subset of attributes. It consists in grouping together attributes that are frequently accessed by queries. Vertical fragments are built by projection. The original relation is reconstructed by joining the fragments.

Horizontal fragmentation divides a relation into subsets of tuples using query predicates. It reduces query processing costs by minimizing the number of irrelevant accessed instances. Horizontal fragments are built by selection. The

original relation is reconstructed by fragment union. A variant, derived horizontal fragmentation, consists in partitioning a relation with respect to predicates defined on another relation.

Finally, hybrid fragmentation consists of either horizontal fragments that are subsequently vertically fragmented, or vertical fragments that are subsequently horizontally fragmented.

3.2 Data warehouse fragmentation

Many research studies address the issue of fragmenting relational data warehouses either to efficiently process analytical queries or to distribute the warehouse.

To improve ad-hoc query performance, Datta *et al.* exploit a vertical fragmentation of facts to build the Cuio index [8], while Golfarelli *et al.* apply the same fragmentation on warehouse views [10]. Bellatreche and Boukhalfa apply horizontal fragmentation to a star-schema [2]. Their fragmentation strategy is based on a query workload and exploits a genetic algorithm to select a optimal partitioning schema that minimizes query cost. Finally, Wu and Buchmaan recommend to combine horizontal and vertical fragmentation for query optimization [24]. A fact table can be horizontally partitioned according to one or more dimensions, it can also be vertically partitioned according to its dimension foreign keys.

To distribute a data warehouse, Noaman *et al.* exploit a top-down strategy that uses horizontal fragmentation [17]. The authors propose an algorithm for deriving horizontal fragments from the fact table based on queries that are defined on all dimension tables. Finally, Wehrle *et al.* propose to distribute and query a warehouse on a computing grid [23]. They use derived horizontal fragmentation to split the data warehouse and build a so-called *block of chunks*, a data set defining a fragment.

In summary, these proposals generally exploit derived horizontal fragmentation to reduce irrelevant data access rate and efficiently process join operations across multiple relations [2, 17, 23]. In the literature, the prevalent methods used for derived horizontal fragmentation are the following [12].

- **Predicate construction**. This method fragments a relation by using a complete and minimal set of predicates [17]. Completeness means that two relation instances belonging to the same fragment have the same probability of being accessed by any query. Minimality garantees that there is no redundancy in predicates.

- **Affinity-based fragmentation**. This method is an adaptation of vertical fragmentation methods to horizontal fragmentation [16]. It is based on the predicate affinity concept [25], where affinity defines query frequency. Specific matrices (predicate usage and affinity matrices) are exploited to cluster selection predicates. A cluster is defined as a selection predicate cycle and forms a dimension graph fragment.

3.3 XML database fragmentation

Recently, several fragmentation techniques for XML data have been proposed. They split an XML document into a new set of XML documents. Their main objective is either to improve XML query performance [9] or to distribute or exchange XML data over a network [4, 5].

To fragment XML documents, Ma *et al.* define a new fragmentation type: *split* [13], which is inspired from the oriented-object domain. This fragmentation splits XML document elements and assigns a reference to each subelement. The references are then added to the Document Type Definition (DTD) defining the XML document. Andrade *et al.* propose to apply fragmentation to an homogeneous XML collection [1]. They adapt traditional fragmentation techniques to an XML document collection and base their proposal on the Tree Logical Class algebra (TLC) [19].

Bose and Fegaras use XML fragments for data exchange in a peer-to-peer network (P2P), called XP2P [5]. XML fragments are interrelated and each is uniquely identified by an *ID*. The authors propose a fragmentation schema, called *Tag Structure*, to define the structure of data and fragmentation information. Bonifati *et al.* also define XML fragments for a P2P framework [4]. An XML fragment is obtained and identified by a single path expression, a root-to-node path expression *XP*, and managed on a specific peer.

In summary, these proposals adapt classical fragmentation methods to split XML data. An XML fragment is defined and identified by a path expression [4] or an XML algebra operator [1]. Fragmentation is performed on a single XML document [13] or on an homogeneous XML collection [1].

4. FRAGMENTING XML DATA WAREHOUSES

4.1 Motivation

Approaches dealing with fragmentation in XML databases adopt only primary horizontal fragmentation applied onto one XML document (Section 3.3). They use fragmentation to minimize XML query expression execution cost. However, in XML data warehouses, decision-support queries are more complex: they involve multiple join operations over multiple XML (fact and dimension) documents. Hence, primary horizontal fragmentation is not adapted in our context. Relational data warehouse fragmentation approaches recommend to use derived horizontal fragmentation (Section 3.2), which is more adapted to analytical queries. In addition, there are, to the best of our knowledge, no XML data warehouses fragmentation works in the literature. In consequence, we propose to adapt horizontal derived fragmentation to XML data warehouses (Definition 2).

Definition 2. *In an XML data warehouse, derived horizontal fragmentation first splits horizontally $G_{dimension_d}$ graphs with respect to a given workload W, and then partitions the G_{fact_f} graphs with respect to $G_{dimension_d}$ fragments.*

4.2 General principle

In our fragmentation methodology, we first apply a primary horizontal fragmentation onto warehouse dimensions using either the predicate construction method, denoted PC, or the affinity-based method, denoted AB (Section 3.2). Both these methods input selection predicates (Definition 3) from W (Section 4.3.1). AB also exploits data access frequencies. Our adaptations of PC and AB to the XML context are described in Sections 4.3.2 and 4.3.3, respectively. Both help fragment $G_{dimension_d}$ graphs. Note that we consider both the PC and AB methods to compare their efficiency, which has never been addressed in the literature

as far as we know. Based on these fragments, we then fragment the G_{facts_f} graphs and build a fragmentation schema for the whole XML data warehouse. This process is detailed in Section 4.4.

DEFINITION 3. *A selection predicate is defined by expression* $p := P_{a_k}\theta[value \mid \emptyset_{XPath}(P_{a_k}) \mid Q]$, *where* P_{a_k} *and* Q *are path expressions and* P_{a_k} *is defined on attribute* a_k, $\theta \in \{=, <, >, \leq, \geq, \neq\}$, $value \in D_k$ *where* D_k *is the domain of* a_k, *and* $\emptyset_{XPath}$ *is any XPath function* [1].

4.3 Primary horizontal fragmentation

4.3.1 Selection predicate extraction

The set P of selection predicates used to fragment the $G_{dimension_d}$ graphs is identified by parsing W. For example, $p_1 := \$y/attribute[@id =' c_nation_key']/\ @value >' 15'$ and $p_2 := \$y/attribute[@id =' p_type']/\ @value =' PROMOBURNISHEDCOPPER'$ are selection predicates obtained from query q_1 in the sample XQuery workload provided in Figure 2.

```
q1   for $x in //FactDoc/Fact,
     $y in //dimensions[@dim-id='Customer']/Level/instance
     $z in //dimensions[@dim-id='Part']/Level/instance
     where $y/attribute[@id='c_nation_key']/@value='13'
     and $y/attribute[@id='p_type']/@value='PROMO
     BURNISHED COPPER'
     and $x/dimension[@dim-id='Customer']/@value-id=$y/@id
     and $x/dimension[@dim-id='Part']/@value-id=$z/@id
     return $x

...
q10  for $x in //FactDoc/Fact,
     $y in //dimensions[@dim-id='Customer']/Level/instance
     $z in //dimensions[@dim-id='Date']/Level/instance
     where $y/attribute[@id='c_nation_key']/@value>'15'
     and $y/attribute[@id='d_date_name']/@value='Saturday'
     and $x/dimension[@dim-id='Customer']/@value-id=$y/@id
     and $x/dimension[@dim-id='Part']/@value-id=$z/@id
     return $x
```

Figure 2: Workload snapshot

4.3.2 PC primary horizontal fragmentation

Principle

Based on selection predicate set P and metadata from $G_{dw-model}$, PC identifies candidate $G_{dimension_d}$ graphs for fragmentation. A candidate dimension graph $G_{candidate_d}$ is a $G_{dimension_d}$ graph targeted by workload queries.

For each candidate dimension and its corresponding selection predicate set $P_d \subset P$, a set of complete and minimal selection predicates $P_d^{'}$ is generated with the COM-MIN algorithm [18] that guarantees completeness and minimality (Section 3.2). PC finally builds from $P_d^{'}$ a set of minterms that horizontally fragment the $G_{candidate_d}$ graphs.

Fragmentation methodology

1. **Attribution of selection predicates to dimension XML graphs**. This step affects to each dimension graph $G_{dimension_d}$ its corresponding selection predicate set $P_d \subset P$. P_d is identified from $G_{dw-model}$, which stores for each dimension its corresponding attributes. Hence, we can identify candidate dimension graphs ($G_{candidate_d}$) for horizontal fragmentation.

   Example. Predicate p_2 contains attribute *c_nation_key*. In $G_{dw-model}$, *c_nation_key* is a member of the *customer* dimension. Hence, we identify $G_{dimension_{customer}}$ as a candidate dimension for fragmentation.

2. **Selection predicate completeness and minimality**. In this step, we apply the COM-MIN algorithm, which inputs P_d and outputs a set of complete and minimal predicates $P_d^{'}$. Given $P_d^{'}$, the set M_d of minterm predicates is then constructed.
   $M_d = \{m_i | m_i = \wedge_{q_j \in P} q_j^*\}$, where $q_j^* = q_j$ or $q_j^* = \neg q_j$, $1 \leq j \leq n$, $1 \leq i \leq 2^n$ and n represents the number of selection predicates. A minterm predicate $m_i \in M_d$ is the conjunction of all predicates from $P_d^{'}$, taken in natural or negative form. $m_1 = p_1 \wedge \neg p_2$ is an example minterm predicate, where p_1 and p_2 are the sample selection predicates from Section 4.2.

   Example. Let $P_{customer}^{'} = \{\$y/attribute[@id = 'c_nationkey']/@value = 13, \$y\ /attribute[@id = 'c_nationkey']/@value > 15\}$ be a complete and minimal set obtained by the COM-MIN algorithm for dimension *customer*. A minterm m_1 is $\$y/attribute[@id =' c_nationkey']/@value = 13$ and $\$y/attribute[@id =' c_nation\ key']/@value <= 15$.

3. **Candidate graph fragmentation**. This step builds primary horizontal fragments from $G_{candidate_d}$. A fragment is obtained by associating to each minterm predicate $m_i \in M_d$ the set of nodes in $G_{candidate_d}$ that verifies it.

   Example. Minterm m_1 is used to fragment $G_{candidate_{customer}}$.

4.3.3 AB primary horizontal fragmentation

Principle

AB uses query frequency to build horizontal fragments by exploiting specific matrices (predicate usage and affinity matrices). It clusters selection predicates from P by exploiting a graphical algorithm. A cluster is defined as a selection predicate cycle and forms a fragment of a $G_{dimension_d}$ graph.

Fragmentation methodology

1. **Predicate usage matrix construction**. The predicate usage matrix, PUM, is built based on P. It defines predicate usage of each query $q_i \in W$. Matrix lines represent workload queries and columns simple selection predicates from P. General term $PUM(i, j)$ is set to one if q_i includes predicate p_j and to zero otherwise. In addition, the usage frequency of each query q_i is stored in a vector $Freq$.

   Example. Tables 1 and 2 provide examples of PUM matrix and query frequency vector, respectively.

2. **Predicate affinity matrix construction** The predicate affinity matrix, Aff, is built from the PUM matrix (Table 3). It is a $n \times n$ matrix, where n represents

[1] http://www.w3.org/TR/xpath-functions/

query/predicate	p_1	p_2	p_3	p_4	...	p_n
q_1	1	0	0	0		0
q_2	1	1	0	0		0
...						
q_m	1	1	0	0		1

n represents the number of selection predicates in P and m the number of queries in W.

Table 1: Sample predicate usage matrix

q_1	q_2	...	q_m
10	20	...	5

Table 2: Sample query frequency vector

the number of selection predicates in P. Aff matrix cells can contain numeric or nonnumeric ($\Rightarrow$, $\Leftarrow$ and $*$) values. A numeric value of an $Aff(i,j)$ cell gives the frequency sum of all queries referencing both predicates p_i and p_j. A value "$\Rightarrow$" indicates that predicate p_i implies predicate p_j; a value "$\Leftarrow$" indicates that predicate p_j implies predicate p_i; and a value "$*$" indicates that predicates p_i and p_j are similar. Two predicates p_i and p_j are similar if: (1) they are defined on the same attribute; (2) there exists a query q_i that uses predicates p_i and p_c and another query q_j which uses predicates p_j and p_c; and (3) p_c is a selection predicate that is defined on another attribute than the p_i and p_j predicates [16].

Example. Table 3 shows a example of affinity matrix.

	p_1	p_2	...	p_5	...	p_n
p_1	20	0		10		0
p_2	0	30				$\Leftarrow$
...						
p_5	10	0		25		0
...						
p_n	0	$\Rightarrow$		0		5

Table 3: Sample predicate affinity matrix

3. **Predicate clustering**. This step exploits the graphical algorithm proposed by Navathe *et al.* [15] for vertical fragmentation, which has been been adapted for horizontal fragmentation [16]. This algorithm inputs Aff and considers it as a complete graph, G_{Aff}. Then, it forms a linearly connected spanning-tree. A tree node represents a selection predicate p_i in $Aff(i,j)$ and an edge $e(p_i, p_j)$ an affinity value. The algorithm detects and extracts a set of cycles C, where a cycle $c_i \in C$ groups selection predicates sharing values in Aff.

   Example. Figure 3 shows a sample G_{Aff} that is build from $Aff(i,j)$. $C = \{c_1, c_2, ..., c_z\}$, where z represents the number of cycles. $c_1 = \{p_1, p_3, p_5\}$.

4. **Compose predicate terms**. Cycle set C is first evaluated to determine distinct common attributes in C

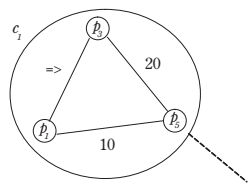

Figure 3: Predicate clustering example

predicates and construct a specific table called predicate term schematic table. This table stores attribute usage for each c_i. Based on this table, predicate terms t_i are constructed. A predicate term t_i constitutes an horizontal entry in the predicate term schematic table and covers all common attributes.

Example. Table 4 gives an example of predicate term schematic table. Predicates in cycle c_1 do not include attribute a_2. c_1 is hence divided to a set of sub-cycle c_{1j}. Each c_{1j} sub-cycle contains predicates from c_1 and a predicate p_j that includes attribute a_2. $t_1 = p_1 \wedge p_3 \wedge p_5 \wedge p_2$ for $j = 2$ is an example of predicate term.

	a_1	a_2	...	a_r
c_1	1	0		1
c_2	1	1		1
...	0	0		1
c_z	1	1		1

a_0 represents an attribute from dimension d and r is the number of attributes in C.

Table 4: Predicate term schematic table

5. **Candidate graph fragmentation**. Each obtained predicate term and an additional predicate, called ELSE, form an horizontal fragment. The ELSE predicate is the negation of the conjonction of all predicate terms. It is added to ensure fragmentation completeness. To ensure fragmentation disjonction, a set of minterms is also created (Section 4.3.2).

   Example. t_1 and ELSE=$\neg p_1$ or $\neg p_3$ or $\neg p_5$ or $\neg p_2$ are predicate terms used to fragment $G_{dimension_{customer}}$.

4.4 Fact fragmentation

The G_{facts_f} graphs are finally fragmented according to horizontal fragments obtained by applying either the PC or AB method on dimensions. The fragmentation of G_{facts_f} graphs is achieved by semi-join operations based on a virtual key reference. This key defines the relationships between $G_{dimension_d}$ and G_{fact_f} graphs. It is explicitly defined by the join qualification expression provided in Figure 4 and consists of a conjunction of two path expressions. These path expressions check whether nodes in $G_{dimension_d}$ graphs correspond to nodes in G_{facts_f} graphs.

We finally build an XML document that represents the fragmentation schema, *fragmentation_schema.xml*. Its corresponding graph, denoted *Schema*, is provided in Figure 5. The root node, *Schema*, is composed of *fragment* elements

```
document(facts_f.xml)/FactDoc/dimension[@dim-id=
document(dimension_d.xml)/dimension/Level/@id]
and
document(facts_f.xml)/FactDoc/dimension[@value-id
=document(dimension_d.xml)/dimension/Level[@id
=@dim-id]/instance/@id]
```

Figure 4: Join qualification

describing the obtained fragments. Each fragment is identified by an *@id* attribute and contains *dimension* elements. A dimension element is identified by a *@name* attribute and contains *predicate* elements that store minterms used for fragmentation.

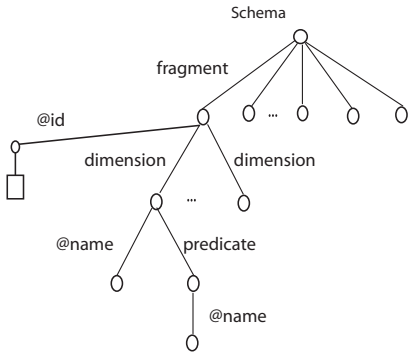

Figure 5: Fragmentation schema

5. EXPERIMENTS

5.1 Experimental conditions

In order to validate our proposal experimentally, we use XWeB (the XML Data Warehouse Benchmark) [14]. XWeB is based on the reference model defined in Section 2.2, and proposes a test XML data warehouse and its associated XQuery decision-support workload.

XWeB's warehouse consists of *sale* facts characterized by the amount (of purchased products) and quantity (of purchased products) measures. These facts are stored in the $facts_{sales}.xml$ document and are described by four dimensions: *Customer*, *Supplier*, *Date* and *Part* stored in the $dimension_{Customer}.xml$, $dimension_{Supplier}.xml$, $dimension_{Date}.xml$ and $dimension_{Part}.xml$ documents, respectively. XWeB's warehouse characteristics are displayed in Table 5.

XWeB's workload is composed of queries that exploit the warehouse through join and selection operations. We extend this workload by adding queries and selection predicates in order to obtain a significant fragmentation. Our workload is available on-line [2]. We ran our tests on a Pentium 2 GHz PC with 1 GB of main memory and an IDE hard drive under Windows XP. We use the X-Hive XML native DBMS [3] to store and query the warehouse.

5.2 Experiments

Our experiments measure workload execution time, with and without using fragmentation and separately evaluate

[2] http://eric.univ-lyon2.fr/~hmahboubi/Workload/workload.xq
[3] http://www.x-hive.com/products/db/

Facts	**Number of cells**
Sale facts	7000
Dimensions	**Number of instances**
Customer	1000
Supplier	1000
Date	500
Part	1000
Documents	**Size (MB)**
$facts_{sales}.xml$	2.14
$dimension_{Customer}.xml$	0.431
$dimension_{Supplier}.xml$	0.485
$dimension_{Date}.xml$	0.104
$dimension_{Part}.xml$	0.388

Table 5: XWeB warehouse characteristics

the PC and AB primary fragmentation strategies (Section 4.3.2 and 4.3.3, respectively). The fragments we achieve are stored in distinct collections to simulate data distribution. Each collection can indeed be considered as a distinct node/site and can be identified, targeted and queried separately. To measure query execution time over a fragmented warehouse, we first identify the required fragments with the *Schema* graph. Then, we execute the query over each fragment and save execution time. To simulate a parallel execution, we only consider the maximum execution time. We conducted two series of experiments.

5.2.1 *First series of experiments*

This series of experiments helps observe the impact of data warehouse size and workload characteristics on fragmentation quality. For this purpose, we exploit three warehouse and workload configurations (Table 6) in which we vary warehouse size (i.e., the number of facts) and the number of workload queries and selection predicates.

	Config. 1	**Config. 2**	**Config. 3**
Number of facts	800	800	4000
Number of queries	13	19	19
Number of join operations	22	35	35
Number of predicates	20	30	30

Table 6: Warehouse and workload configurations

Experiment results for configurations 1, 2 and three are showed on Figures 6, 7 and 8, respectively. In these figures, the X axis represents workload queries and the Y axis features query execution time when no fragmentation is applied on the warehouse, and when derived horizontal fragmentation is applied with PC and AB primary fragmentation.

For configuration 1, we obtain an average gain over no fragmentation of 72.95% with PC and 76.32% with AB. For configuration 2, in which the number of queries is increased over configuration 1, PC improves query execution time by 74.53% and AB by 78.32% on average. Finally, in configuration 3, we increase the number of facts and obtain an average gain of 62.59% with PC and 80,17% with AB.

These results confirm that fragmentation improves query performance. They also show that AB provides more bene-

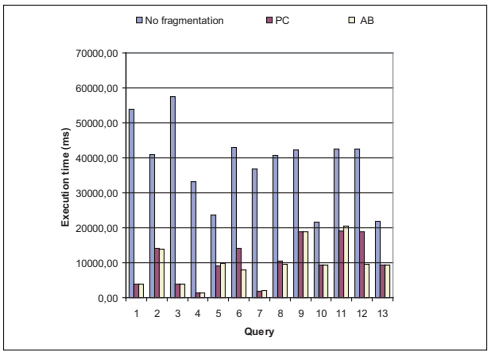

Figure 6: Configuration 1 results

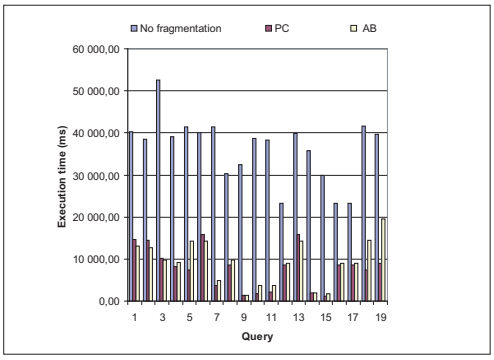

Figure 7: Configuration 2 results

fit than PC in all our test cases. We think this is thanks to AB's use of query frequencies to group in the same fragment all dimension instances and facts needed to perform a given join operation. In addition, we notice that PC fragmentation gain significantly declines in configuration 3, i.e., when warehouse size increases. We think this is due to the number of fragments produced by PC, which is greater than that obtained with AB (159 and 119, respectively). To further investigate this issue, we conduct a second series of experiment where we further observe PC and AB gain variation with respect to warehouse size.

5.2.2 *Second series of experiments*

This series of experiments helps aim at observing the effect of warehouse size on fragmentation gain. We vary warehouse size from 1000 to 5000 facts and measure the fragmentation gain achieved when using PC and AB primary fragmentation. The results of these experiments are plotted in Figure 9, whose X axis represents the number of facts and Y axis the corresponding gains obtained by PC and AB primary fragmentation.

Experiment results show that fragmentation gain declines when warehouse size decreases with both primary fragmentation methods. This is expected, since fragments become bigger and bigger, inducing a higher and higher scan cost when performing join operations. However, we also observe that performance degradation is reasonably slow for AB, while it is much steeper for PC. We believe that this is because AB builds fragments containing data required to perform the most frequent join operations in W, while storing less frequently accessed data in the ELSE fragment. PC

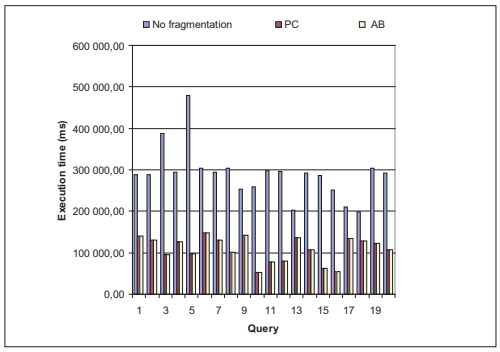

Figure 8: Configuration 3 results

does not take this aspect into account. It just groups in the same fragment data accessed by one or more queries simultaneously. It also uses minterms that may distribute data required to answer a single query in different fragments, which multiplies reconstruction joins when accessing data.

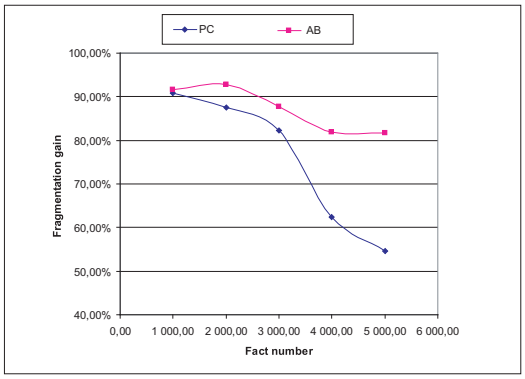

Figure 9: Fragmentation gain vs. warehouse size

6. CONCLUSION

In this paper, we have adapt, to XML context, and compare the two prevailing primary horizontal fragmentation methods from the relational world, namely predicate construction and affinity-based fragmentation. We have experimentally confirmed that derived horizontal fragmentation helped improve query response time significantly. Moreover, we also showed that affinity-based fragmentation clearly outperformed predicate construction in all our experiments, which had never been demonstrated before as far as we know, even in the relational context.

The natural follow-up of this work is to distribute fragmented XML warehouses on a data grid. This raises several issues that include processing a global query into subqueries to be sent to the right nodes in the grid, and reconstructing a global result from subquery results. Properly indexing the distributed warehouse to guarantee good performance shall also be very important.

7. REFERENCES

[1] A. Andrade, G. Ruberg, F. A. Baião, V. P. Braganholo, and M. Mattoso. Efficiently Processing

XML Queries over Fragmented Repositories with PartiX. In *Current Trends in Database Technology, EDBT 2006 PhD Workshop, Munich, Germany*, volume 4254 of *Lecture Notes in Computer Science*, pages 150–163. Springer, 2006.

[2] L. Bellatreche and K. Boukhalfa. An Evolutionary Approach to Schema Partitioning Selection in a Data Warehouse. In *7th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 05), Copenhagen, Denmark*, volume 3589 of *Lecture Notes in Computer Science*, pages 115–125. Springer, 2005.

[3] K. S. Beyer, D. D. Chamberlin, L. S. Colby, F. Ozcan, H. Pirahesh, and Y. Xu. Extending XQuery for Analytics. In *ACM SIGMOD International Conference on Management of Data (SIGMOD 05), Baltimore, Maryland*, pages 503–514. ACM, 2005.

[4] A. Bonifati, U. Matrangolo, A. Cuzzocrea, and M. Jain. XPath lookup queries in P2P networks. In *6th ACM CIKM International Workshop on Web Information and Data Management (WIDM 04), Washington, USA*, pages 48–55. ACM, 2004.

[5] S. Bose and L. Fegaras. XFrag: A query Processing Framework for Fragmented XML Data. In *8th International Workshop on the Web and Databases (WebDB 05), Baltimore, Maryland*, pages 97–102, 2005.

[6] D. Boukraa, R. BenMessaoud, and O. Boussaïd. Proposition d'un Modèle physique pour les entrepôts XML. In *Atelier Systèmes Décisionnels (ASD 06), 9th Maghrebian Conference on Information Technologies (MCSEAI 06), Agadir, Morocco*, 2006.

[7] O. Boussaïd, R. BenMessaoud, R. Choquet, and S. Anthoard. X-Warehousing: An XML-Based Approach for Warehousing Complex Data. In *10th East-European Conference on Advances in Databases and Information Systems (ADBIS 06), Thessaloniki, Greece*, volume 4152 of *Lecture Notes in Computer Science*, pages 39–54. Springer, 2006.

[8] A. Datta, K. Ramamritham, and H. M. Thomas. Curio: A Novel Solution for Efficient Storage and Indexing in Data Warehouses. In *25th International Conference on Very Large Data Bases (VLDB 99), Edinburgh, UK*, pages 730–733. Morgan Kaufmann, 1999.

[9] M. Gertz and J.-M. Bremer. Distributed XML Repositories: Top-down Design and Transparent Query Processing. Technical report, Departement of Computer Science, University of California, USA, 2003.

[10] M. Golfarelli, D. Maio, and S. Rizzi. Vertical fragmentation of views in relational data warehouses. In *Settimo Convegno Nazionale su Sistemi Evoluti Per Basi Di Dati (SEBD 99), Como, Italy*, pages 19–33, 1999.

[11] W. Hümmer, A. Bauer, and G. Harde. XCube: XML for data warehouses. In *6th International Workshop on Data Warehousing and OLAP (DOLAP 03), New Orleans, USA*, pages 33–40. ACM, 2003.

[12] A. Koreichi and B. L. Cun. On data fragmentation and allocation in distributed object oriented databases. Technical report, PRiSM, Versailles University, France, 1997.

[13] H. Ma and K.-D. Schewe. Fragmentation of XML Documents. In *XVIII Simpósio Brasileiro de Bancos de Dados, Manaus, Amazonas, Brasil*, pages 200–214. UFAM, 2003.

[14] H. Mahboubi and J. Darmont. Benchmarking XML data warehouses. In *Atelier Systèmes Décisionnels (ASD 06), 9th Maghrebian Conference on Information Technologies (MCSEAI 06), Agadir, Morocco*, 2006.

[15] S. Navathe and M. Ra. Vertical partitionning for database design: A graphical algorithm. In *ACM SIGMOD International Conference on Management of Data (SIGMOD 89), Portland, Oregon*, pages 440–450, 1989.

[16] S. B. Navathe, K. Karlapalem, and M. Ra. A Mixed Fragmentation Methodology for Initial Distributed Database Design. *Journal of Computer and Software Engineering*, 3(4), 1995.

[17] A. Y. Noaman and K. Barker. A Horizontal Fragmentation Algorithm for the Fact Relation in a Distributed Data Warehouse. In *1999 ACM International Conference on Information and Knowledge Management (CIKM 99), Kansas City, USA*, pages 154–161. ACM, 1999.

[18] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems, Second Edition*. Prentice-Hall, 1999.

[19] S. Paparizos, Y. Wu, L. V. S. Lakshmanan, and H. V. Jagadish. Tree Logical Classes for Efficient Evaluation of XQuery. In *SIGMOD International Conference on Management of Data (SIGMOD 04), Paris, France*, pages 71–82, 2004.

[20] B.-K. Park, H. Han, and I.-Y. Song. XML-OLAP: A Multidimensional Analysis Framework for XML Warehouses. In *7th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 05)*, volume 3589 of *Lecture Notes in Computer Science*, pages 32–42. Springer, 2005.

[21] J. Pokorný. XML Data Warehouse: Modelling and Querying. In *5th International Baltic Conference (BalticDB&IS 06), Tallin, Estonia*, pages 267–280. Institute of Cybernetics at Tallin Technical University, 2002.

[22] L. I. Rusu, J. W. Rahayu, and D. Taniar. A Methodology for Building XML Data Warehouse. *International Journal of Data Warehousing and Mining, 1(2)*, pages 67–92, 2005.

[23] P. Wehrle, M. Miquel, and A. Tchounikine. A Model for Distributing and Querying a Data Warehouse on a Computing Grid. In *11th International Conference on Parallel and Distributed Systems (ICPADS 05), Fuduoka, Japan*, pages 203–209. IEEE Computer Society, 2005.

[24] M.-C. Wu and A. P. Buchmann. Research Issues in Data Warehousing. In *Datenbanksysteme in Buro, Technik und Wissenschaft*, pages 61–82, 1997.

[25] Y. Zhang and O. Orlowska. On fragmentation approaches for distributed database design. *Information Sciences*, 1(3):117–132, 1994.