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.

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]. Hümer 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. $\text{dw-model}.xml$ that represents warehouse metadata, the XML graph representing warehouse metadata is denoted $G_{\text{dw-model}}$.

2. a set of $\text{facts}_d.xml$ documents that each store information related to set of facts $f$;

3. a set of $\text{dimension}_d.xml$ 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 $\text{facts}_d.xml$ 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 $G_{\text{facts}_d}$.

A $\text{dimension}_d.xml$ 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 $G_{\text{dimension}_d}$.

![Figure 1: facts_d.xml (a) and dimension_d.xml (b) graph structure](image)

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 vari-
ant, derived horizontal fragmentation, consists in partition-
ing 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 frag-
mented.

### 3.2 Data warehouse fragmentation

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

To improve ad-hoc query performance, Datta et al. ex-
plot a vertical fragmentation of facts to build the Cujo index
[8], while Golfarelli et al. apply the same fragmentation on
warehouse views [10]. Bellatreche and Boukhalfa apply hor-
izontal fragmentation to a star-schema [2]. Their fragmenta-
tion 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 rec-
ommend 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 de-
 fined 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 hor-
izontal fragmentation to reduce irrelevant data access rate
and efficiently process join operations across multiple rela-
tions [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 pred-
  icates [17]. Completeness means that two relation in-
  stances belonging to the same fragment have the same
  probability of being accessed by any query. Minimality
  guarantees that there is no redundancy in predicates.

- **Affinity-based fragmentation.** This method is an
  adaptation of vertical fragmentation methods to hor-
  izontal fragmentation [16]. It is based on the predicate
  affinity concept [25], where affinity defines query fre-
  quency. 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 sub-
element. The references are then added to the Document
Type Definition (DTD) defining the XML document. An-
drade et al. propose to apply fragmentation to an homoge-
neous XML collection [1]. They adapt traditional fragmen-
tation 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. How-
ever, in XML data warehouses, decision-support queries are
more complex: they involve multiple join operations over
multiple XML (fact and dimension) documents. Hence, pri-
mary horizontal fragmentation is not adapted in our con-
text. Relational data warehouse fragmentation approaches
recommend to use derived horizontal fragmentation (Sec-
tion 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 frag-
mentation to XML data warehouses (Definition 2).

**Definition 2.** In an XML data warehouse, derived hor-
izontal fragmentation first splits horizontally $G_{\text{dimension}_d}$
graphs with respect to a given workload $W$, and then par-
titions the $G_{\text{fact}_j}$ graphs with respect to $G_{\text{dimension}_d}$
fragments.

#### 4.2 General principle

In our fragmentation methodology, we first apply a pri-
mary 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, respec-
tively. Both help fragment $G_{\text{dimension}_d}$ 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 $G_{candidate}$ 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 $p := P_{a_k} \theta \text{id} = v | \theta \text{XPath}(P_{a_k}) | Q$, 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 $\theta \text{XPath}$ 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, $p_1 := \forall y/attribute[@id='c\_nation\_key']/@value='PROMOBURNISHED COPPER'$. These selection predicates form the set $P'_{\text{Horizontal}}$.

Example. $P'_{\text{Horizontal}}$ contains attributes $c\_nation\_key$. In $G_{bw-model}$, $c\_nation\_key$ 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.

$M_d = \{m_i | m_i = \land q_j \in P'_d\}$, where $q_j = q_i$ or $q_j' = \neg q_i$, $1 \leq j \leq n$, $1 \leq i \leq 2^n$ 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. $m_1 = p_1 \land \neg p_2$ is an example minterm predicate, where $p_1$ and $p_2$ are the sample selection predicates from Section 4.2.

Example. Let $P'_{customer} = \{\forall y/attribute[@id='c\_nation\_key']/@value='13', \forall y/attribute[@id='c\_nation\_key']/@value='15'\}$ be a complete and minimal set obtained by the COM-MIN algorithm for dimension $customer$. A minterm $m_1 = \forall y/attribute[@id='c\_nation\_key']/@value='13$ and $\forall y/attribute[@id='c\_nation\_key']/@value='15'$. $m_1$ contains attribute $c\_nation\_key$.

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 an $n \times n$ matrix, where $n$ represents
the number of selection predicates in \( P \) and \( m \) the number of queries in \( W \).

### Table 1: Sample predicate usage matrix

| query/predicate | \( q_1 \) | \( q_2 \) | \( q_3 \) | \( q_4 \) | \ldots | \( q_m \) |
|----------------|---------|---------|---------|---------|       |---------|
| \( p_1 \)      | 1       | 0       | 0       | 0       |       | 0       |
| \( p_2 \)      | 1       | 1       | 0       | 0       |       | 0       |
| \ldots         |         |         |         |         |       |         |
| \( p_n \)      | 1       | 1       | 0       | 0       |       | 1       |

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

### Table 2: Sample query frequency vector

| \( q_1 \) | \( q_2 \) | \( q_3 \) | \( q_4 \) | \ldots | \( q_m \) |
|---------|---------|---------|---------|       |---------|
| 10      | 20      |         |         |       |         |

### Table 3: Sample predicate affinity matrix

| \( p_1 \) | \( p_2 \) | \( p_3 \) | \( p_4 \) | \ldots | \( p_n \) |
|---------|---------|---------|---------|       |---------|
| 20      | 0       | 10      | 0       |       |         |
| 0       | 30      |         |         |       |         |
| \ldots |         |         |         |       |         |
| \( p_5 \) | 10      | 0       | 25      | 0       |         |
| \ldots |         |         |         |       |         |
| \( p_m \) | 0       |         |         |         | 0       |

Predicate clustering. This step exploits the graphical algorithm proposed by Navathe et al. [15] for vertical fragmentation, which has 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 \( c \) \( \subseteq \) \( C \) groups selection predicates sharing values in \( Aff \).

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

Compose predicate terms. Cycle set \( C \) is first evaluated to determine distinct common attributes in \( C \) 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 3 gives an example of predicate term schematic table. Predicates in cycle \( c_1 \) do not include attribute \( a_0 \). \( c_1 \) is hence divided to a set of sub-cycle \( c_{1, j} \). Each \( c_{1, j} \) sub-cycle contains predicates from \( c_1 \) and a predicate \( p_j \) that includes attribute \( a_0 \). 

<table>
<thead>
<tr>
<th>( a_1 )</th>
<th>( a_2 )</th>
<th>\ldots</th>
<th>( a_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_1 )</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>( c_2 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>\ldots</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( c_3 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\( a_0 \) 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 conjunction of all predicate terms. It is added to ensure fragmentation completeness. To ensure fragmentation disjunction, a set of minterms is also created (Section 4.3.2).

**Example.** \( t_1 \) and ELSE=\( ¬p_1 \) or \( ¬p_3 \) or \( ¬p_5 \) or \( ¬p_2 \) are predicate terms used to fragment \( G_{dimension_customer} \).

### 4.4 Fact fragmentation

The \( G_{fact\_s} \) graphs are finally fragmented according to horizontal fragments obtained by applying either the PC or AB method on dimensions. The fragmentation of \( G_{fact\_s} \) graphs is achieved by semi-join operations based on a virtual key reference. This key defines the relationships between \( G_{dimension\_s} \) and \( G_{fact\_s} \) 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 \( G_{dimension\_s} \) graphs correspond to nodes in \( G_{fact\_s} \) graphs.

We finally build an XML document that represents the fragmentation schema, \texttt{fragmentation_schema.xml}. Its corresponding graph, denoted \texttt{Schema}, is provided in Figure 5. The root node, \texttt{Scheme}, is composed of fragment elements.
The XWeB 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\_sale\_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 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.

<table>
<thead>
<tr>
<th>Facts</th>
<th>Number of cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sale facts</td>
<td>7000</td>
</tr>
<tr>
<td>Dimensions</td>
<td>Number of instances</td>
</tr>
<tr>
<td>Customer</td>
<td>1000</td>
</tr>
<tr>
<td>Supplier</td>
<td>1000</td>
</tr>
<tr>
<td>Date</td>
<td>500</td>
</tr>
<tr>
<td>Part</td>
<td>1000</td>
</tr>
<tr>
<td>Documents</td>
<td>Size (MB)</td>
</tr>
<tr>
<td>facts_sale_xml</td>
<td>2.14</td>
</tr>
<tr>
<td>dimension_Customer_xml</td>
<td>0.431</td>
</tr>
<tr>
<td>dimension_Supplier_xml</td>
<td>0.485</td>
</tr>
<tr>
<td>dimension_Date_xml</td>
<td>0.104</td>
</tr>
<tr>
<td>dimension_Part_xml</td>
<td>0.388</td>
</tr>
</tbody>
</table>

### Table 5: XWeB warehouse characteristics

<table>
<thead>
<tr>
<th>Config.</th>
<th>Number of facts</th>
<th>Number of queries</th>
<th>Number of join operations</th>
<th>Number of predicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Config. 1</td>
<td>800</td>
<td>13</td>
<td>22</td>
<td>20</td>
</tr>
<tr>
<td>Config. 2</td>
<td>800</td>
<td>19</td>
<td>35</td>
<td>30</td>
</tr>
<tr>
<td>Config. 3</td>
<td>4000</td>
<td>19</td>
<td>35</td>
<td>30</td>
</tr>
</tbody>
</table>

### 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-
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 experiments 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 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.

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


