Data Mining-based Fragmentation of XML Data Warehouses

Hadj Mahboubi, Jérôme Darmont

HAL Id: hal-00336980
https://hal.science/hal-00336980

Data Mining-based Fragmentation
of XML Data Warehouses

Hadj Mahboubi
University of Lyon (ERIC Lyon 2)
5 avenue Pierre Mendès-France
69676 Bron Cedex – France
hadj.mahboubi@eric.univ-lyon2.fr

Jérôme Darmont
University of Lyon (ERIC Lyon 2)
5 avenue Pierre Mendès-France
69676 Bron Cedex – France
jerome.darmont@univ-lyon2.fr

ABSTRACT

With the multiplication of XML data sources, many XML data warehouse models have been proposed to handle data heterogeneity and complexity in a way relational data warehouses fail to achieve. However, XML-native database systems currently suffer from limited performances, both in terms of manageable data volume and response time. Fragmentation helps address both these issues. Derived horizontal fragmentation is typically used in relational data warehouses and can definitely be adapted to the XML context. However, the number of fragments produced by classical algorithms is difficult to control. In this paper, we propose the use of a k-means-based fragmentation approach that allows to master the number of fragments through its k parameter. We experimentally compare its efficiency to classical derived horizontal fragmentation algorithms adapted to XML data warehouses and show its superiority.

Categories and Subject Descriptors

H.2 [**Database Management**]: Physical Design

General Terms

Performance

1. INTRODUCTION

XML data sources that are pertinent for decision-support are becoming increasingly common with XML becoming a standard for representing complex business data [6]. However, they bear specificities (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. Hence, many efforts toward XML data warehousing have been achieved in the past few years [13, 35, 39], as well as efforts for extending the XQuery language with near On-Line Analytical Processing (OLAP)

capabilities, such as advanced grouping and aggregation features [6, 21, 37].

In this context, performance is a critical issue, since XML-native database systems currently suffer from limited performances, both in terms of manageable data volume and response time for complex analytical queries. These issues are typical of data warehouses and can be addressed by fragmentation. Fragmentation consists in splitting a data set into several fragments such that their combination yields the original warehouse without loss nor information addition. Fragmentation can subsequently lead to distribute the target data warehouse, e.g., on a data grid [16]. 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 [5]. Several approaches have also been proposed for XML data fragmentation, but they do not take multidimensional architectures (i.e., star-like schemas) into account.

In derived horizontal fragmentation, dimensions' primary horizontal fragmentation is crucial. In the relational context, two major algorithms address this issue: the predicate construction [32] and the affinity-based [31] strategies. However, both are automatic and the number of fragments is not known in advance, while it is crucial to master it, especially since distributing M fragments over N nodes with $M > N$ can become an issue in itself. Hence, we propose in this paper the use of a k-means-based fragmentation approach that allows to control the number of fragments through its k parameter. Our idea, which adapts a proposal from the object-oriented domain [14] to XML warehouses, is to cluster workload query predicates to produce primary horizontal dimension fragments, with one fragment corresponding to one cluster of predicates. Primary fragmentation is then derived on facts. Queries including given predicates are executed over the corresponding fragments only, instead of the whole warehouse, and thus run faster.

The remainder of this paper is organized as follows. We first discuss existing research related to relational data warehouse, XML data and data mining-based fragmentation in Section 2. Then, we present our k-means-based XML data warehouse fragmentation approach in Section 3. We experimentally compare its efficiency to classical derived horizontal fragmentation algorithms adapted to XML data warehouses and show its superiority in Section 4. Finally, we conclude this paper and present future research directions in Section 5.

2. RELATED WORK

2.1 Fragmentation Definition

There are three fragmentation types in the relational context [23]: vertical fragmentation, horizontal fragmentation and hybrid fragmentation.

Vertical fragmentation splits a relation R into subrelations 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.

2.2 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 [15], while Golfarelli *et al.* apply the same fragmentation on warehouse views [19]. Munneke *et al.* propose a fragmentation methodology for a multidimensional database [30]. Fragmentation consists in deriving a global data cube into fragments containing a subset of data. This process is defined by the slice and dice operation. The authors also define another fragmentation strategy, server, that removes one or several dimensions from a hypercube to produce fragments with fewer dimensions than the original data cube. Bellatreche and Boukhalfa apply horizontal fragmentation to a star-schema [5]. Their fragmentation strategy is based on a query workload and exploits a genetic algorithm to select a partitioning schema. This algorithm aims at choosing an optimal fragmentation schema that minimizes query cost. Finally, Wu and Buchmaan recommend to combine horizontal and vertical fragmentation for query optimization [38]. A fact table can be horizontally partitioned with respect to one or more dimensions. It can also be vertically partitioned according to its dimensions, i.e., all the foreign keys to the dimension tables are partitioned as separate tables.

To distribute a data warehouse, Noaman *et al.* exploit a top-down strategy that uses horizontal fragmentation [32]. 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 [36]. 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 static derived horizontal fragmentation to reduce irrelevant data access rate and efficiently process join operations across multiple relations [5, 32, 36]. In the literature, the prevalent methods used for derived horizontal fragmentation are the following [23].

- **Predicate construction**. This method fragments a relation by using a complete and minimal set of predicates [32]. 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 [31]. It is based on the predicate affinity concept [40], 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.

2.3 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 [7, 18, 25] or to distribute or exchange XML data over a network [9, 10].

To fragment XML documents, Ma *et al.* define a new fragmentation type: *split* [24, 25], 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. The authors extend the DTD and XML-QL languages. Bonifati *et al.* also propose a fragmentation strategy for XML documents that is driven by structural constraints [7, 8]. This strategy uses both heuristics and statistics. Andrade *et al.* propose to apply fragmentation to an homogeneous XML collection [2]. They adapt traditional fragmentation techniques to an XML document collection and base their proposal on the Tree Logical Class (TLC) algebra [33]. The authors also evaluate these techniques and show that horizontal fragmentation provides the best performance.

Gertz and Bremer introduce a distribution approach for an XML repository [18]. They propose a fragmentation method and outline an allocation model for distributed XML fragments in a centralized archirecture. Gertz and Bremer also define horizontal and vertical fragmentation for an XML document. A fragment is defined with a path expression language, called *XF*, which is derived from XPath. This fragment is obtained by applying an *XF* expression on a graph *RG* representing XML data. Moreover, the authors define exclusion expressions that ensure fragment coherence and disjunction.

Bose and Fegaras use XML fragments for data exchange in a peer-to-peer network (P2P), called XP2P [10]. 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 [9]. 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 addition, the authors associate to each fragment two path expressions:

super fragment and *child fragment*. These paths ensure the identification of fragments and relationships.

In summary, these proposals adapt classical static fragmentation methods to split XML data. An XML fragment is defined and identified by a path expression [9] or an XML algebra operator [2]. Fragmentation is performed on a single XML document [24, 25] or on an homogeneous XML collection [2]. Note that XML data warehouse fragmentation has not been addressed yet, to the best of our knowledge.

2.4 Data Mining-based Fragmentation

Although data mining has proved useful for selecting physical data structures that enhance performance, such as indexes or materialized views [3, 4], few approaches exploit data mining for fragmentation.

Gorla and Betty exploit association rules (by adapting the Apriori algorithm [1]) for vertical fragmentation approach in relational databases [20].

Darabant and Campan propose the horizontal fragmentation method for object-oriented distributed databases based on k-means clustering we inspire from [14]. This method clusters object instances into fragments by taking all complex relationships between classes into account (aggregation, associations and links induced by complex methods).

Finally, Fiolet and Toursel propose a parallel, progressive clustering algorithm to fragment a database and distribute it over a grid [17]. It is inspired by the CLIQUE sequential clustering algorithm that consists in clustering data by projection.

Though in limited number, these studies clearly demonstrate how data mining can be used for vertical and horizontal fragmentation, through association rule mining and clustering, respectively. They are also static, though.

3. K-MEANS-BASED FRAGMENTATION

Although XML data warehouse architectures from the literature share a lot of concepts (mostly originating from classical data warehousing), they are nonetheless all different. Hence, we proposed a unified, reference XML data warehouse model that synthesizes and enhances existing models [29], and on which we base our fragmentation work. We first recall this model before detailing our fragmentation approach.

3.1 XML Warehouse Reference Model

XML warehousing approaches assume that the warehouse is composed of XML documents that represent both facts and dimensions. All these studies 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 [12]. Moreover, this representation also allows to model constellation schemas without duplicating dimension information. Several fact documents can indeed share the same dimensions. Hence, we adopt this architecture model. More precisely, our reference data warehouse is composed of the following XML documents:

- *dw-model.xml* that represents warehouse metadata;

- a set of $facts_f.xml$ documents that each store information related to set of facts f;

- a set of $dimension_d.xml$ documents that each store a given dimension d's member values.

The *dw-model.xml* document (Figure 1) defines the multidimensional structure of the warehouse. Its root node, *DW-model*, is composed of two types of nodes: *dimension* and *FactDoc*. A *dimension* node defines one dimension, its possible hierarchical levels (*Level* elements) and attributes (including their types), as well as the path to the corresponding *dimension_d.xml* document. A *FactDoc* element defines a fact, i.e., its measures, references to the corresponding dimensions, and the path to the corresponding $facts_f.xml$ document.
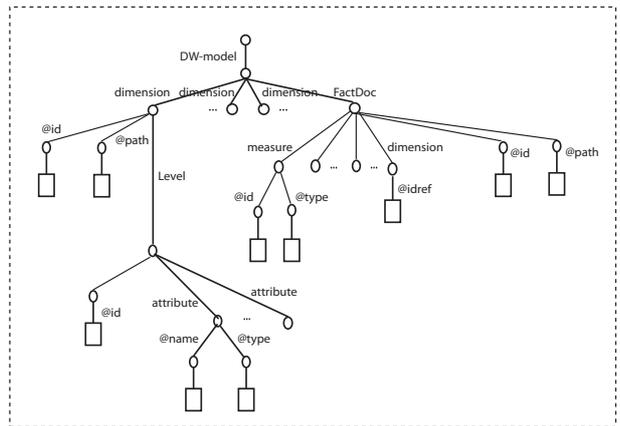

Figure 1: *dw-model.xml* graph structure

A $facts_f.xml$ document stores facts (Figure 2(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.

Finally, a *dimension_d.xml* document helps instantiate one dimension, including any hierarchical level (Figure 2(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.
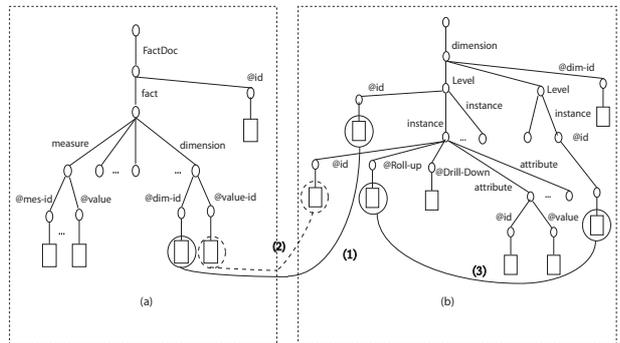

Figure 2: $facts_f.xml$ **(a) and** $dimension_d.xml$ **(b) graph structures**

3.2 Fragmentation Approach

3.2.1 Principle

Since the aim of fragmentation is to optimize query response time, the prevalent fragmentation strategies are workload driven [5, 7, 18, 31, 32]. More precisely, they exploit selection predicates found in workload queries to derive suitable fragments. Our approach also belongs to this family. Its general principle is summarized in Figure 3. It outputs both a fragmentation schema (metadata) and the fragmented XML warehouse. It is subdivided into three steps that are detailed in the following sections:

1. selection predicate extraction from the query workload;

2. predicate clustering with the k-means method;

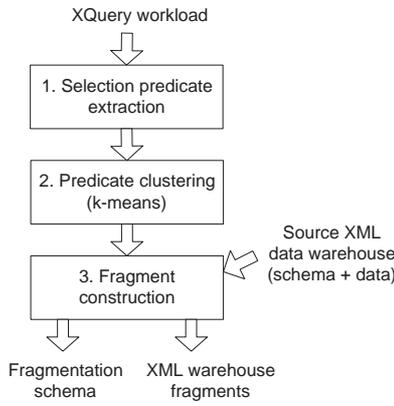3. fragment construction with respect to predicate clusters.

Figure 3: K-means-based fragmentation principle

3.2.2 Selection Predicate Extraction

Selection predicate set P is simply parsed from workload W. For example, let W_S be the sample XQuery workload provided in Figure 4 and P_S the corresponding predicate set. $P_S = \{p_1, p_2, p_3, p_4, ...\}$, where:
$p_1 = \$y/attribute[@id = "c\_nation\_key"]/@value>"15"$,
$p_2 = \$y/attribute[@id = "c\_nation\_key"]/@value = "13"$,
$p_3 = \$y/attribute[@id = "p\_type"]/@value = "PBC"$ and
$p_4 = \$y/attribute[@id = "d\_date\_name"]/@value = "Sat."$.
For example, p_2 and p_3 are selection predicates obtained from query $q_2 \in W_S$.

Parsed predicates are then coded in a query-predicate matrix QP whose general term QP_{ij} equals to 1 if predicate $p_j \in P$ appears in query $q_i \in W$, and to 0 otherwise. For example, the QP_S matrix corresponding to W_S and P_S is featured in Table 1.

3.2.3 Predicate Clustering

Our objective is to derive fragments that optimize data access for a given set of queries. Since horizontal fragments are built from predicates, clustering predicates with respect to queries achieves our goal. Intuitively, we ideally seek to build rectangles (clusters) of 1s in matrix QP. We chose the widely-used k-means algorithm [26] for clustering. This algorithm inputs vectors of object attributes (columns of QP

```
q1    for $x in //FactDoc/Fact,
      $y in //dimension[@dim-id="Customer"]/Level/instance
      where $y/attribute[@id="c_nation_key"]/@value>"15"
      and $x/dimension[@dim-id="Customer"]/@value-id=$y/@id
      return $x

q2    for $x in //FactDoc/Fact,
      $y in //dimension[@dim-id="Customer"]/Level/instance,
      $z in //dimension[@dim-id="Part"]/Level/instance
      where $y/attribute[@id="c_nation_key"]/@value="13"
      and $y/attribute[@id="p_type"]/@value="PBC"
      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 //dimension[@dim-id="Customer"]/Level/instance,
      $z in //dimension[@dim-id="Date"]/Level/instance
      where $y/attribute[@id="c_nation_key"]/@value="13"
      and $y/attribute[@id="d_date_name"]/@value="Sat."
      and $x/dimension[@dim-id="Customer"]/@value-id=$y/@id
      and $x/dimension[@dim-id="Part"]/@value-id=$z/@id
      return $x
```

Figure 4: XQuery workload snapshot W_S

	p_1	p_2	p_3	p_4	...
q_1	1	0	0	0	
q_2	0	1	1	0	
...					
q_{10}	0	0	1	1	

Table 1: Sample query-predicate matrix QP_S

in our case). It attempts to find the centers of natural clusters in source data by minimizing total intra-cluster variance $\sum_{i=1}^{k} \sum_{x_j \in C_i} (x_j - \mu_i)^2$, where $C_i, i = 1, ..., k$ are the k output clusters and μ_i is the centroid (mean point) of points $x_j \in C_i$. Let C be the set of all clusters C_i.

Usually, having k as an input parameter is viewed as a drawback in clustering. In our case, this turns out to be an advantage, since we want to limit the number of clusters/fragments, typically to the number of nodes the XML data warehouse will be distributed on.

In practice, we used the Weka [22] SimpleKMeans implementation of k-means. SimpleKMeans uses the Euclidean distance to compute distances between points and clusters. It directly inputs matrix QP (acually, the p_j vectors) and k, and outputs set of predicate clusters C. For example, on matrix QP_S with $k = 2$, SimpleKMeans outputs:

$$C_S = \{\{p_1\}, \{p_2, p_3, p_4\}\}.$$

3.2.4 Fragment Construction

The fragmentation construction step is itself subdivided into two substeps (Figure 5). First, predicate cluster set C is joined to warehouse schema (document *dw-model.xml*) to produce an XML document named *frag-schema.xml* that represents the fragmentation schema (Figure 6). Its root node, *Schema*, is composed of *fragment* elements. 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 the predicates used for fragmentation. For example, the fragmentation schema *frag-schema_S.xml* corresponding to cluster set C_S is provided in Figure 7.

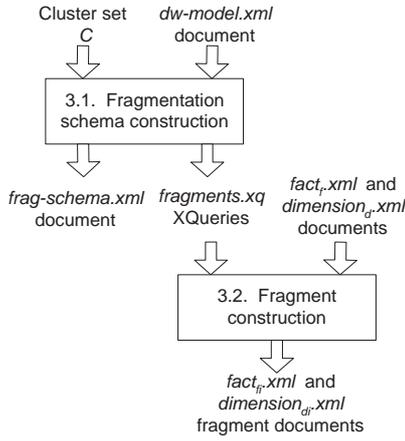In this process, we also output a set of XQueries (the
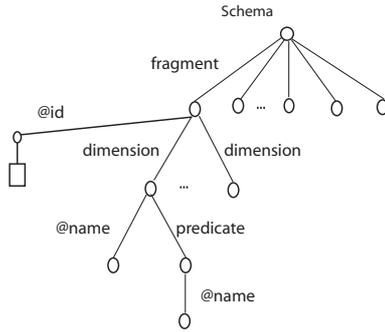
Figure 5: Fragment construction substeps

Figure 6: *frag-schema.xml* graph structure

fragments.xq script) that, applied to the XML data warehouse (i.e., the whole set of $facts_f.xml$ and $dimension_d.xml$ documents), produces the actual fragments, which we store in a set of $facts_{f_i}.xml$ and $dimension_{d_i}.xml$ documents, $i = 1, ..., k+1$. As fragments, these documents indeed bear the same schema than the original warehouse. The $(k+1)^{th}$ fragment is based on an additional predicate, denoted $ELSE$, which is the negation of the conjunction of all predicates in P and is necessary to ensure fragmentation completeness (Section 2.2). In our running example, $ELSE = \neg(p_1 \wedge p_2 \wedge p_3 \wedge p_4)$.

Figure 8 provides an excerpt from the $fragments_S.xq$ script that helps build fragment $f2$ from Figure 7. Dimension fragments are generated first, one by one, through selections exploiting the predicate(s) associated to the current dimension (three first queries from Figure 8). Then, fragmentation is derived on facts by joining the original fact document to the newly-created dimension fragments (last query).

4. EXPERIMENTS

Since derived horizontal fragmentation is a NP-hard problem [11] solved by heuristics, we choose to validate our proposal experimentally.

4.1 Experimental conditions

We use XWeB (XML Data Warehouse Benchmark) [27] as a test platform. XWeB is based on the reference model defined in Section 3.1, and proposes a test XML data warehouse and its associated XQuery decision-support workload.

```
<Schema>
    <fragment id="f1">
        <dimension name="Customer">
            <predicate name="p1" />
        </dimension>
    </fragment>
    <fragment id="f2">
        <dimension name="Customer">
            <predicate name="p2" />
        </dimension>
        <dimension name="Part">
            <predicate name="p3" />
        </dimension>
        <dimension name="Date">
            <predicate name="p4" />
        </dimension>
    </fragment>
</Schema>
```

Figure 7: Sample *frag-schema$_S$.xml* document

```
element dimension{ attribute dim-id{Customer}, element Level{
attribute id {Customers},
for $x in document("dimension_Customer.xml")//Level
where $x//attribute[@id="c_nation_key"]/@value="13"]
return $x }
}
element dimension{ attribute dim-id{Part}, element Level{
attribute id {Part},
for $x in document("dimension_Part.xml")//Level
where $x//attribute[@id="p_type"]/@value="PBC"]
return $x }
}
element dimension{ attribute dim-id{Date}, element Level{
attribute id {Date},
for $x in document("dimension_Date.xml")//Level
where $x//attribute[@id="d_date_name"]/@value="Sat."]
return $x }
}
element FactDoc {
for $x in //FactDoc/Fact,
$y in document("dimension_Customer_f2.xml")//instance,
$z in document("dimension_Part_f2.xml")//instance,
$t in document("dimension_Date_f2.xml")//instance
where $x/dimension[@dim-id="Customer"]/@value-id=$y/@id
and $x/dimension[@dim-id="Part"]/@value-id=$z/@id
and $x/dimension[@dim-id="Date"]/@value-id=$t/@id
return $x
}
```

Figure 8: Excerpt from sample *fragments$_S$.xq* script

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 2.

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. Due to space constraints, our workload is only available on-line[1].

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[2] to store and query the warehouse. Our code is written in Java and connects

---

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

Facts	Maximum 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 2: XWeB warehouse characteristics

to X-Hive and Weka through their respective Application Programming Interfaces (APIs). It is available on demand.

4.2 Fragmentation Strategy Comparison

In this first series of experiments, we aim at comparing our k-means-based horizontal fragmentation approach (denoted KM) to the classical derived horizontal fragmentation techniques, namely by predicate construction (PC) and affinity-based (AB) primary fragmentation (Section 2.2), which we adapted to XML data warehouses [28]. We also record performance when no fragmentation is applied (NF), for reference.

4.2.1 Query Response Time

This experiment measures workload execution time with the three fragmentation strategies we adopted. For KM, we arbitrarily fixed $k = 8$, which could correspond to a computer cluster's size. The fragments we achieve are stored in distinct collections to simulate data distribution. Each collection can indeed be considered to be stored on a distinct node 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 *frag-schema.xml* document. Then, we execute the query over each fragment and save execution time. To simulate parallel execution, we only consider maximum execution time.

Figure 9 plots workload response time with respect to data warehouse size (expressed in number of facts). It clearly shows that fragmentation significantly improves response time, and that KM fragmentation performs better than PC and AB fragmentation when the warehouse scales up. Workload execution time is indeed, on an average, 86.5% faster with KM fragmentation than with NF, and 36.7% faster with KM than with than AB. We believe our approach performs better than classical derived horizontal fragmentation techniques because these latter produce many more fragments (159 with PC and 119 with AB *vs.* 9 with KM). Hence, at workload execution time, queries must access many fragments (up to 50 from our observations), which multiplies query distribution and result reconstruction costs. The number of accessed fragments is much lower with KM (typically 2 fragments in our experiments).
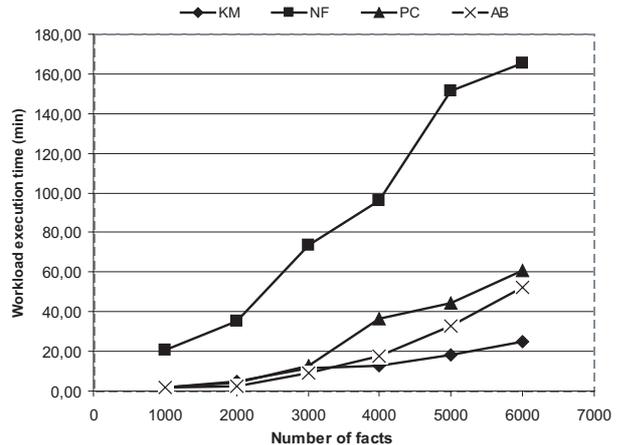

Figure 9: Fragmentation efficiency comparison

4.2.2 Fragmentation Overhead

We also compare the PC, AB and KM ($k = 8$) fragmentation strategies in terms of overhead (i.e., fragmentation algorithm execution time). When assessing performance, it is indeed necessary to find a fair trade-off between gain and overhead. Table 3 summarizes the results we obtain for an arbitrarily fixed data warehouse size of 3,000 facts. It shows that KM clearly outperforms AB and PC, which is in line with these algorithms' complexities : $O(|P|)$, $O(|P|^2)$ and $O(2^{|P|})$, respectively. While AB and PC would have to run off-line, KM could on the other hand be envisaged to run on-line.

	PC	AB	KM
Execution time (h)	16.8	11.9	0.25

Table 3: Fragmentation overhead comparison

4.3 Influence of Number of Clusters

In this experiment, we fixed data warehouse size (to 4,000 and 5,000 facts, respectively) and varied KM parameter k to observe its influence on workload response time. Figure 10 confirms that performance improves quickly when fragmentation is applied, but tends to degrade when the number of fragments increases, as we explained in Section 4.2.1. Furthermore, it hints that an optimal number of clusters *for our test data warehouse and workload* lies between 4 and 6, making us conclude that over-fragmentation must be detected and avoided. Note that, on Figure 10, $k = 1$ corresponds to the NF experiment (this one fragment is the original warehouse).

5. CONCLUSION

In this paper, we have introduced an approach for fragmenting XML data warehouses that is based on data mining, and more precisely clustering and the k-means algorithm. Classical derived horizontal fragmentation strategies run automatically and output an unpredictable number of fragments, which is nonetheless crucial to keep under control. By contrast, our approach allows to fully master the
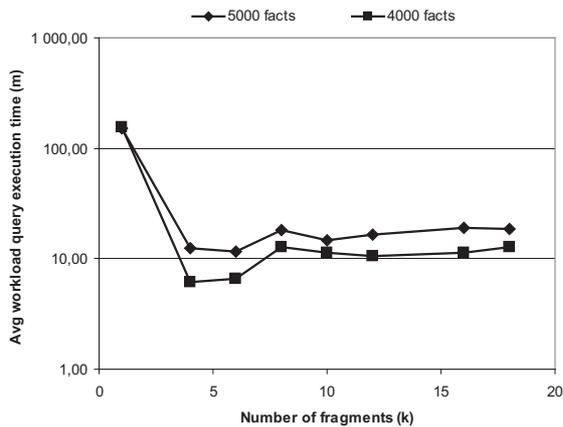
Figure 10: Influence of number of clusters

number of fragments through the k-means k parameter.

To validate our proposal, we have compared our fragmentation strategy to XML adaptations of the two prevalent fragmentation methods for relational data warehouses. Our experimental results show that our approach, by producing a lower number of fragments, outperforms both the others in terms of performance gain and overhead.

Now that we have efficiently fragmented an XML data warehouse, our more direct perspective is to distribute it 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.

Finally, in a continuous effort to minimize the data warehouse administration function and aim at autoadministrative systems [3, 4], we plan to make our data mining based-fragmentation strategy dynamic. The idea is to perform incremental fragmentation when the warehouse is refreshed. This could be achieved with the help of an incremental variant of the k-means algorithm [34].

6. ACKNOWLEDGMENTS

7. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *20th International Conference on Very Large Data Bases (VLDB 94), Santiago de Chile, Chile*, pages 487–499. Morgan Kaufmann, 1994.

[2] 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 Workshops PhD, DataX, IIDB, IIHA, ICSNW, QLQP, PIM, PaRMA, and Reactivity on the Web, Munich, Germany*, volume 4254 of *LNCS*, pages 150–163. Springer, 2006.

[3] K. Aouiche, P.-E. Jouve, and J. Darmont. Clustering-based materialized view selection in data warehouses. In *10th East-European Conference on Advances in Databases and Information Systems (ADBIS 06), Thessaloniki, Greece*, volume 4152 of *LNCS*, pages 81–95. Springer, September 2006.

[4] S. Azefack, K. Aouiche, and J. Darmont. Dynamic index selection in data warehouses. In *4th International Conference on Innovations in Information Technology (Innovations 07), Dubai, United Arab Emirates*. IEEE Computer Society, November 2007.

[5] 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 *LNCS*, pages 115–125. Springer, 2005.

[6] 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.

[7] A. Bonifati and A. Cuzzocrea. Efficient Fragmentation of Large XML Documents. In *18th International Conference on Database and Expert Systems Applications (DEXA 07), Regensburg, Germany*, volume 4653 of *LNCS*, pages 539–550. Springer, 2007.

[8] A. Bonifati, A. Cuzzocrea, and B. Zinno. Fragmenting XML Documents via Structural Constraints. In *10th East European Conference on Advances in Databases and Information Systems (ADBIS 06), Thessaloniki, Greece, Local Proceedings*, pages 17–29, 2006.

[9] 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.

[10] 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.

[11] K. Boukhalfa, L. Bellatreche, and P. Richard. Fragmentations Primaire et Dérivée : Etude de Complexité, Algorithmes de Sélection et Validation sous Oracle 10g. In *4èmes journées francophones sur les Entrepôts de Données et l'Analyse en ligne (EDA 08), Toulouse, France*, volume B-4 of *RNTI*, pages 123–139. Cépaduès, June 2008.

[12] 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.

[13] 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 *LNCS*, pages 39–54. Springer, 2006.

[14] A. S. Darabant and A. Campan. Semi-supervised learning techniques: k-means clustering in OODB

Fragmentation. In *Second IEEE International Conference on Computational Cybernetics (ICCC 04), Vienna, Austria*, pages 333–338. IEEE Computer Society, 2004.

[15] 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.

[16] R. L. de Carvalho Costa and P. Furtado. Data Warehouses in Grids with High QoS. In *8th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 06), Krakow, Poland*, volume 4081 of *LNCS*, pages 207–217. Springer, 2006.

[17] V. Fiolet and B. Toursel. Progressive clustering for database distribution on a grid. In *4th International Symposium on Parallel and Distributed Computing (ISPDC 05), Lille, France*, pages 282–289. IEEE Computer Society, 2005.

[18] 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.

[19] 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.

[20] N. Gorla and P. W. Y. Betty. Vertical Fragmentation in Databases Using Data-Mining Technique. *International Journal of Data Warehousing and Mining*, 4(3):35–53, 2008.

[21] M. Hachicha, H. Mahboubi, and J. Darmont. Expressing OLAP operators with the TAX XML algebra. In *3rd International Workshop on Database Technologies for Handling XML Information on the Web (DataX-EDBT 08), Nantes, France*, March 2008.

[22] G. Holmes, A. Donkin, and I. Witten. Weka: A machine learning workbench. In *Second Australia and New Zealand Conference on Intelligent Information Systems, Brisbane, Australia*, 1994.

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

[24] 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.

[25] H. Ma, K.-D. Schewe, S. Hartmann, and M. Kirchberg. Distribution Design for XML Documents. In *3rd International Conference on Electronic Commerce Engineering (ICECE 03), Hangzhou, China*, pages 1007–1012. International Academic Publisher/World Publishing Corporation, 2003.

[26] J. MacQueen. Some Methods for classification and Analysis of Multivariate Observations. In *5th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley, USA*, pages 281–297. University of California Press, 1967.

[27] 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.

[28] H. Mahboubi and J. Darmont. Enhancing XML Data Warehouse Performance by Fragmentation. Technical report, Technical report, ERIC, University of Lyon 2, France, 2008.

[29] H. Mahboubi, M. Hachicha, and J. Darmont. *XML Warehousing and OLAP*. Encyclopedia of Data Warehousing and Mining, Second Edition. IGI Publishing, August 2008.

[30] D. Munneke, K. Wahlstrom, and M. K. Mohania. Fragmentation of multidimensional databases. In *10th Australasian Database Conference (ADC 99), Auckland, New Zealand*, pages 153–164, 1999.

[31] 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.

[32] 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.

[33] 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.

[34] D. Pham, S. Dimov, and C. Nguyen. An Incremental K-means algorithm. *Journal of Mechanical Engineering Science*, 218(7):783–795, 2004.

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

[36] 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.

[37] N. Wiwatwattana, H. V. Jagadish, L. V. S. Lakshmanan, and D. Srivastava. X^3: A Cube Operator for XML OLAP. In *23rd International Conference on Data Engineering (ICDE 07), Istanbul, Turkey*, pages 916–925, 2007.

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

[39] J. Zhang, W. Wang, H. Liu, and S. Zhang. X-warehouse: building query pattern-driven data. In *14th international conference on World Wide Web (WWW 05), Chiba, Japan*, pages 896–897. ACM, 2005.

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