Towards Recommendations for Horizontal XML Fragmentation

Tatiane Lima da Silva[1], Fernanda Baião[2], Jonice de Oliveira Sampaio[1], Marta Mattoso[3], Vanessa Braganholo[4]

[1] PPGI/Universidade Federal do Rio de Janeiro, Brazil
tatiane.lima@ufrj.br, jonice@dcc.ufrj.br
[2] NP2Tec/Universidade Federal do Estado do Rio de Janeiro (UNIRIO), Brazil
fernanda.baiao@uniriotec.br
[3] COPPE/Universidade Federal do Rio de Janeiro, Brazil
marta@cos.ufrj.br
[4] IC/Universidade Federal Fluminense, Brazil
vanessa@ic.uff.br

Abstract. The large amount of XML data available on the web and inside organizations makes the performance of query processing a big concern. Several techniques can be applied to improve query processing performance, including indexing and data distribution. The increasing popularity of clouds, clusters and grids makes data distribution a feasible alternative. In these approaches, data is fragmented and distributed to several nodes, and queries submitted by users are processed in parallel, thus improving performance. However, the problem of how to fragment an XML database has not been adequately addressed. There are lots of definitions for XML fragments in the literature, but few proposals focus on how to use those definitions to actually fragment the database – a process called fragmentation design. Both the relational and the object-oriented models have solid methodologies for database fragmentation design. Inspired by them, the main objective of this article is to study and propose guidelines to be used in a fragmentation design algorithm for XML databases, aiming at increasing query processing performance.

Categories and Subject Descriptors: H.2 [**Database Management**]: Systems - Distributed Databases

Keywords: database fragmentation design, heuristics, horizontal fragmentation, XML

1. INTRODUCTION

Due to the large volume of data predominantly stored in XML databases, there is great concern about the performance of query processing and, consequently, several work in this area [Andrade et al. 2006; Gang and Rada 2007; Moro et al. 2009; Kling et al. 2011; Figueiredo et al. 2010]. The increasing popularity of clouds, clusters and grids makes data distribution a feasible alternative. In these approaches, data is fragmented and distributed to several nodes. Queries submitted by users can then be processed in parallel, thus improving performance [Ozsu and Valduriez 2011].

There are two distinct ways of using fragmentation to improve query performance: physical fragmentation and virtual fragmentation. Physical fragmentation approaches [Ozsu and Valduriez 2011] physically break the data, generating what we call "fragments", and allocates the fragments into different nodes in a network. On the other hand, virtual fragmentation [Rodrigues et al. 2011] requires data to be replicated over network nodes, thus requiring more disk space. With respect to physical data fragmentation (our focus in this article), the potential performance gain is obtained depending on the location (proximity) of the data when the query is segmented into parts and sent to different

The authors would like to thank CNPq and FAPERJ for partially supporting this work.

nodes that runs in parallel on a smaller volume of data in each node. However, physical fragmentation can also degrade the query performance [Figueiredo et al. 2010; Silva et al. 2012]. This occurs, for example, when the query execution on the fragmented database requires processing joins to perform reconstructions that were not needed in the original query, among other reasons. Therefore, fragmentation design requires a careful analysis on the most frequent queries, so that fragmentation can provide performance gains for most of them.

The process of fragmentation design can be divided into three stages [Ozsu and Valduriez 2011]: (i) extraction of relevant data (frequent queries, database schema, access frequency of each attribute, among others); (ii) analysis, where information extracted in the previous phase is evaluated to decide which type of fragmentation should be applied; and (iii) fragmentation itself. The analysis phase is, thus, the most important phase in the fragmentation design. It is crucial for the performance of the applications that will run over the fragmented database.

The ideas of fragmentation and distribution proposed for the relational [Ozsu and Valduriez 2011] and object-oriented [Baião et al. 2004] models have influenced several work on XML query processing in distributed environments. In fact, several approaches propose XML fragmentation techniques and algorithms to generate them [Bremer and Gertz 2003; Ma and Schewe 2003; Andrade et al. 2006; Abiteboul et al. 2009]. One of the most explored aspects in the literature is precisely the definition of what an XML fragment is [Bremer and Gertz 2003; Ma and Schewe 2003; Andrade et al. 2006], and how queries can be processed over distributed and fragmented XML databases [Figueiredo et al. 2010]. The problem of how to fragment an XML database has also started to receive some attention recently [Pagnamenta 2005; Kurita et al. 2007; Bonifati and Cuzzocrea 2007; Mahboubi and Darmont 2009; Kling et al. 2010], but they lack on providing a methodology to fragment XML databases. To the best of our knowledge, there is no methodology for the distribution design of XML data that analyzes which fragmentation techniques should or should not be applied in a given scenario, and also no experimental results that can be used to derive this methodology. This has a direct impact on performance of applications that run over distributed databases. Proposals in the literature assume the designer already knows how the database should be fragmented [Bremer and Gertz 2003; Abiteboul et al. 2009].

The main goal of this article is to give a step forward towards a methodology for XML fragmentation design. In particular, we focus on the analysis stage. We contribute by providing a set of recommendations for horizontal XML fragmentation. We start by horizontal fragmentation since it is the first step to solve the general problem of the analysis phase. Our recommendations are based on a series of experimental evaluations over three XML databases of different sizes. Specifically, this article extends our previous work [Silva et al. 2012] by analyzing a larger dataset.

The remaining of this article is structured as follows. In Section 2 we discuss related work. Section 3 presents concepts related to horizontal XML fragmentation that are used in our analysis. Section 4 describes our experimental evaluation, which was used as the foundation for deriving recommendations for horizontal XML fragmentation, presented in Section 5. Finally, we conclude in Section 6.

2. XML FRAGMENTATION DESIGN

There are many approaches in the literature that present definitions of XML fragments [Bremer and Gertz 2003; Andrade et al. 2006; Kling et al. 2010] and XML approaches for fragmentation design in general [Bremer and Gertz 2003; Ma and Schewe 2003; Pagnamenta 2005]. However, regarding the analysis stage of the fragmentation design, there is no work that details the criteria that need to be taken into consideration in that stage to generate a good fragmentation design (a good fragmentation design in one that improves query performance for the most frequent application queries). Thus, there lacks a consistent decision-making method to assess which type of fragmentation is more applicable in each scenario, causing fragmentation to be *ad-hoc*, typically based on the designers' experience.

Ma and Schewe [2003] emphasize the importance of considering the frequent queries in the definition of the fragments. Moreover, in their work, they describe heuristics for horizontal XML fragmentation. The heuristics are based on a cost model, where the biggest offender of horizontal XML fragmentation is the transport time of pieces of the query answer between nodes in the network. However, Ma and Schewe [2003] do not provide experimental results to evaluate the efficiency of the proposed heuristics.

Bonifati and Cuzzocrea [2007] describe a structure-driven fragmentation methodology. That is, issues such as size, width and depth of the XML subtrees are considered in the fragmentation process. The authors propose a set of heuristics for XML fragmentation, called SimpleX. These heuristics seek to determine the maximum values for the variables size, width and depth of the subtrees, and fragmentation is done so not to overcome these values.

Kurita et al. [2007] propose a method for query processing over distributed XML databases. The method focuses on XML data fragmentation and dynamic relocation of fragments on the network nodes. The authors consider that for an efficient large-scale XML query processing system, we must follow four steps: data partitioning, data distribution, distributed query processing and dynamic data reallocation. However, their work is restricted to vertical fragmentation only. During the fragmentation, the authors do consider the frequent queries, but the fragment size instead. Their goal is to obtain homogeneous fragment sizes.

Mahboubi and Darmont [2009] propose to apply horizontal XML fragmentation over a data warehouse. They explore two methods for horizontal fragmentation: predicate construction and affinity-based fragmentation. The authors mention that several studies suggest that derived horizontal fragmentation is the best way to shred XML data. However, according to them, this statement does not necessarily apply to the XML data warehouse architecture. As a conclusion of their experiment, the authors present fragmentation affinity attributes as being the best way to shred an XML data warehouse.

Kling et al. [2011] focus on exploring the distribution in the context of XML database systems as a way to solve the problem of effectiveness and efficiency in accessing large-scale XML data. Their proposed solution addresses two problems of XML query processing in distributed environments: localization, which is the conversion of a single query execution plan into several ones, so they can be executed in a distributed environment; and pruning, which eliminates parts of the execution plan that not contribute to the query result. In other words, only the fragments that contain relevant data to the query would be accessed. Specifically, for horizontal fragmentation, the authors use the same technique they apply in the relational model. That is, the fragmentation is obtained by composing minterms extracted from the frequent queries.

In summary, there are several different approaches to fragment documents horizontally, vertically or in a hybrid way. However, none of them focuses on the analysis phase, i.e. they do not give any hints or suggestions to help the designer to choose a horizontal or vertical fragmentation, for example. All related work is focused on phases (i) extraction of relevant data and phase (iii) fragmentation itself, lacking a direction to the analysis phase.

3. BACKGROUND: HORIZONTAL XML FRAGMENTATION

Our work uses the concept of XML fragment proposed by Andrade et al. [2006]. This is because among the definitions we found in the literature [Bremer and Gertz 2003; Ma and Schewe 2003; Pagnamenta 2005; Kling et al. 2011] that is the closest to the definition of fragments in the relational model [Ozsu and Valduriez 2011]. This choice is essential, since we want to take advantage of the fragmentation design proposed for the relational model [Ozsu and Valduriez 2011]. Andrade et al. [2006] define three types of fragments: horizontal that uses selection predicates to separate documents into different fragments; vertical, which "cuts" the data structure through projections; and hybrid, which combines selection and projection operations in its definition.

The definition of fragment uses collections as its base structure. A collection C of XML documents is a set of data trees. We say it is homogeneous if all the documents in C satisfy the same XML type. If not, we say the collection is heterogeneous. Given a schema S, a homogeneous collection C is denoted by the expression $C := < S, \tau_{root} >$, where τ_{root} is a type in S and all instances of C satisfy τ_{root} [Andrade et al. 2006]. A horizontal fragment F of a collection C is defined by the selection operation (σ) applied over documents in C, where the predicate of σ is a boolean expression with one or more simple predicates. Thus, F has the same schema of C, where C is a collection of XML documents [Andrade et al. 2006]. Horizontal fragmentation should be applied to Multiple Document collections (MD) [Yao et al. 2004], since the selection operation acts over entire documents instead of parts of documents. Single Document (SD) repositories can be horizontally fragmented through a hybrid fragmentation process, where we first apply a vertical fragmentation and then a horizontal one.

Based on the definition of fragments provided by Andrade et al. [2006], Figueiredo et al. [2010] developed a methodology for processing queries on distributed and fragmented XML databases. The prototype developed by Figueiredo et al. [2010] takes care of distributing the query to the relevant fragments. The prototype includes a mediator, which is responsible for query processing, going from the query decomposition until the consolidation of the final results. Each node of the network, in turn, has an adapter that receives the sub queries sent by the mediator and runs them on the local node. We have adapted this prototype to improve its performance and to adjust it to run on clusters, and used it in our experimental evaluation, which we describe in the next section.

4. EXPERIMENTAL EVALUATION

Our goal in this article is to derive a set of recommendations for horizontal XML fragmentation based on experimental results. The main goals of our experimental evaluation are as follows.

(1) To compare the performance of queries on a centralized environment and on a distributed environment, testing different scenarios while running the same set of queries in all of them. These scenarios simulate situations where the fragmentation takes the frequent queries into account, and others that do not.
(2) To evaluate the performance of queries that benefit from fragmentation and queries that do not benefit from fragmentation.

Our experiments used the XBench benchmark [Yao et al. 2004]. The benchmark data were systematically fragmented according to various strategies, inspired by the literature [Baião et al. 2004; Ozsu and Valduriez 2011] and the queries behavior was assessed in each scenario. The analysis was made by comparing the average response times of queries in the different scenarios. Each query was executed 10 times. We discarded the first execution time, and then calculated the average response time using the remaining 9 runs. The experiments were performed on 9 nodes of a homogeneous cluster. Each node consists of two Intel Xeon quad core processors (8 cores), with 16 GB of RAM and local hard disk of 160 GB. One of them acted as the mediator, which is responsible for submitting the queries, generating the sub queries and consolidating the final results. An instance of the adapter component runs in each of the eight remaining nodes. Those nodes are responsible for the local execution of sub queries. Each instance of the adapter uses the local disk of the node where it was allocated, where the native XML database Sedna was installed. The scenarios are described next. Table I presents a summary of the criteria we used for fragmentation and allocation of the fragments in our experiment.

Scenario 0: Execution in a centralized environment, using only one node to run all queries.

Scenario 1: Implementation of horizontal fragmentation, using attributes of frequent queries. In our experiment, we considered the benchmark queries to be the frequent queries. We used two

Table I. Fragmentation criteria and allocation of fragments in each scenario

Scenario	Fragmentation Criteria	Allocation
1.1.1	Frag1: total >= 11000	Frag1: Node 1
	Frag2: total <= 7000	Frag2: Node 2
	Frag3: total > 7000 and total < 11000	Frag3: Node 3
1.1.2	Frag1: total >= 11000	Frag1: Node 1
	Frag2: total <= 7000	Frag2: Node 2
	Frag3: total > 7000 and total < 11000	Frag3: Node 3
1.2.1	Frag1: total <= 1000	Frag1: Node 1
	Frag2: total > 1000	Frag2: Node 2
1.2.2	Frag1: total <=1000	Frag1: Node 1
	Frag2: total > 1000 and total <=7000	Frag2: Node 2
	Frag3: total >7000 and total <=11000	Frag3: Node 3
	Frag4: total >11000	Frag4: Node 4
1.2.3	Frag1: total <=250	Frag1: Node 1
	Frag2: total >250 and total <=500	Frag2: Node 2
	Frag3: total >500 and total <=1000	Frag3: Node 3
	Frag4: total > 1000 and total <=7000	Frag4: Node 4
	Frag5: total >7000 and total <=11000	Frag5: Node 5
	Frag6: total >11000	Frag6: Node 6
1.2.4	Frag1: total <=250	Frag1: Node 1
	Frag2: total >250 and total <=500	Frag2: Node 2
	Frag3: total >500 and total <=1000	Frag3: Node 3
	Frag4: total > 1000 and total <=5000	Frag4: Node 4
	Frag5: total > 5000 and total <=7000	Frag5: Node 5
	Frag6: total >7000 and total <=9000	Frag6: Node 6
	Frag7: total >9000 and total <=11000	Frag7: Node 7
	Frag8: total >11000	Frag8: Node 8
3.1.1	Frag1: transaction_country_id <=23	Frag1: Node 1
	Frag2: transaction_country_id > 23 and transaction_country_id <=46	Frag2: Node 1
	Frag3: transaction_country_id > 46 and transaction_country_id <=69	Frag3: Node 2
	Frag4: transaction_country_id > 69	Frag4: Node 2
3.1.2	Frag1: transaction_country_id <=23	Frag1: Node 1
	Frag2: transaction_country_id > 23 and transaction_country_id <=46	Frag2: Node 2
	Frag3: transaction_country_id > 46 and transaction_country_id <=69	Frag3: Node 3
	Frag4: transaction_country_id > 69	Frag4: Node 4
3.1.3	Frag1: transaction_country_id <=12	Frag1: Node 1
	Frag2: transaction_country_id > 12 and transaction_country_id <=23	Frag2: Node 2
	Frag3: transaction_country_id > 23 and transaction_country_id <=46	Frag3: Node 3
	Frag4: transaction_country_id > 46 and transaction_country_id <=69	Frag4: Node 4
	Frag5: transaction_country_id > 69 and transaction_country_id <=81	Frag5: Node 5
	Frag6: transaction_country_id > 81	Frag6: Node 6
3.1.4	Frag1: transaction_country_id <=12	Frag1: Node 1
	Frag2: transaction_country_id > 12 and transaction_country_id <=23	Frag2: Node 2
	Frag3: transaction_country_id > 23 and transaction_country_id <=39	Frag3: Node 3
	Frag4: transaction_country_id > 39 and transaction_country_id <=46	Frag4: Node 4
	Frag5: transaction_country_id > 46 and transaction_country_id <=58	Frag5: Node 5
	Frag6: transaction_country_id > 58 and transaction_country_id <=69	Frag6: Node 6
	Frag7: transaction_country_id > 69 and transaction_country_id <=81	Frag7: Node 7
	Frag8: transaction_country_id > 81	Frag8: Node 8

sub-scenarios: (1.1.1) three fragments distributed in two nodes; (1.1.2) three fragments distributed in three nodes.

Scenario 2: Implementation of horizontal fragmentation, using the number of nodes available for allocation and domain data. We evaluated the queries and extracted the attribute used in most of the selections. The goal is to generate fragments covering the domain of this attribute. We then generated four sub-scenarios: (2.1.1) two fragments in two nodes; (2.1.2) four fragments in four nodes; (2.1.3) six fragments in six nodes; (2.1.4) eight fragments in eight nodes.

Scenario 3: Implementation of horizontal fragmentation, without using the attributes of frequent queries. We executed four sub-scenarios classified as follows: (3.1.1) two fragments in two nodes; (3.1.2) four fragments in four nodes; (3.1.3) six fragments in six nodes; (3.1.4) eight fragments into eight nodes.

For evaluating the behavior of each scenario, we executed 19 queries of XBench. Furthermore, we evaluated the queries performance on three databases of different sizes (4MB, 40MB and 400MB). The summary of these queries and the selection predicates used in each of them are presented in Table II. By analyzing these queries, it is possible to observe that the most frequent selection attribute is

Table II. Queries performed in the experiments and their respective attributes selection

Query	Predicate
Q1	count(/order/order_lines/order_line) >= 5
Q2, Q12	id = 1
Q3	id = 3
Q4	id = 5
Q5	count(/order/order_lines/order_line) = 1
Q6	id = 6
Q7	total > 7000 and count(/order/order_lines/order_line) >= 5
Q8, Q9	total > 7000
Q10	total < 2000
Q11, Q13, Q15, Q16	total > 11000
Q14	id = 2
Q17, Q18	total > 10000
Q19	total > 7000 and total < 8000

"total". Therefore, scenario 1 evaluates the results when the databases are fragmented by "total" while scenario 2 uses this same attribute, but also explores its domain.

We now compare the average response times of queries in the different scenarios, using the centralized scenario (Scenario 0) as a baseline. The goal of these experiments is to verify if the fragmentation allows better results when compared to the centralized scenario. Additionally, we want to verify that fragmentations that do not consider frequent queries lead to poor performance when compared to a scenario that takes them into consideration. Figures 1 and 2 present a comparative response time analysis between the centralized environment (Scenario 0) and the best scenario analyzed in the experiments with two database sizes: 4MB and 40MB, respectively.

By analyzing Table II, Figures 1 and 2, we can see that 12 queries have benefited from fragmentation when compared to the centralized environment. In the 4MB experiment, these queries are Q6, Q7, Q8, Q9, Q10, Q11, Q13, Q15, Q16, Q17, Q18, and Q19. In the 40MB experiment, query Q6 did not benefit from the fragmentation, but on the other hand, query Q12 did. In the 4MB experiment, only queries that did not have a selection on the "total" attribute did not benefit from the fragmentation (Q1, Q2, Q3, Q4, Q5, Q12, and Q14). The same behavior can be observed in the 40MB experiment with only a small variation in the set of queries that have not benefited from the fragmentation (Q1, Q2, Q3, Q4, Q5, Q6, and Q14). In the 4MB experiment, query Q16 achieved a performance gain of almost 70% (dropping from 175.1ms in the centralized environment to 99.5ms in scenario 2.1.3), while query Q10 obtained a performance gain of approximately 260% in the 40MB experiment (decreasing from 467.68ms in the centralized environment to 128.8ms in scenario 1.1.1). The experiments in the 4Mb and 40MB databases do not show much difference in their results. However, query Q10 performed much better in the 40MB (when compared to the centralized scenario). On the other hand, queries Q15 and Q16 reduced their gains on that database. Furthermore, queries Q12 (4MB experiment) and Q6 (40MB experiment) benefited from the fragmentation, despite of not accessing the frequent attribute (total). This occurred because scenario 3.1.2 generated fragments with uniform sizes, thus improving load balance during query execution.

We then increased the database size and checked the queries behavior. For the experiment on the 400MB database, only two queries (Q1 and Q5) did not present performance gains when compared to the centralized environment, as described in Figure 3. The remaining 17 queries (Q2, Q3, Q4, Q6, Q7, Q8, Q9, Q10, Q11, Q12, Q13, Q14, Q15, Q16, Q17, Q18, and Q19) presented percentage gains ranging from 55% to 1654%. Figure 3 shows the average response time of the centralized scenario and the best distributed scenario. Please note the logarithmic scale in the figure.

Summarizing, in the 400MB experiment several queries benefited from the fragmentation: ***scenario 1.1.1***, 17 queries; ***scenario 1.1.2***, 16 queries; ***scenario 2.1.1***, 11 queries; ***scenario 2.1.2***, 15 queries; ***scenario 2.1.3***, 15 queries; ***scenario 2.1.4***, 15 queries; ***scenario 3.1.1***, 14 queries;

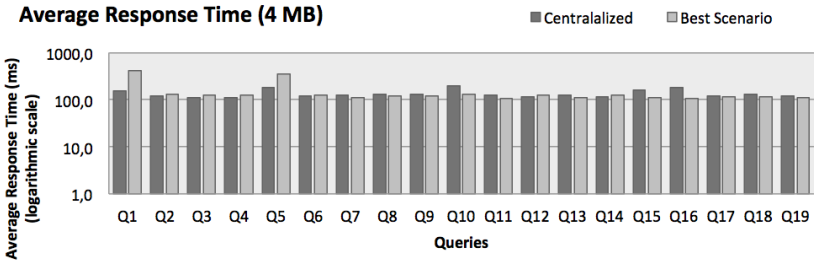

Fig. 1. Average query response times on the 4MB database, in the centralized and the best of the analyzed scenarios

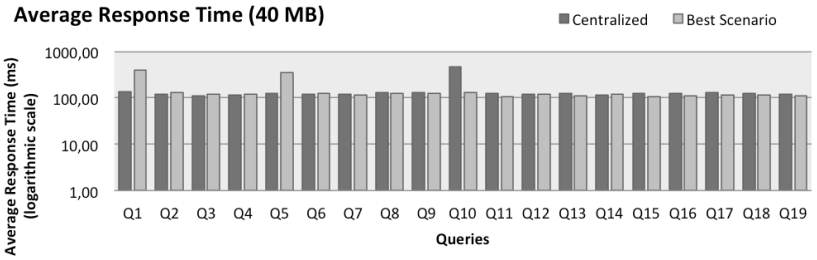

Fig. 2. Average query response times on the 40MB database, in the centralized and the best of the analyzed scenarios

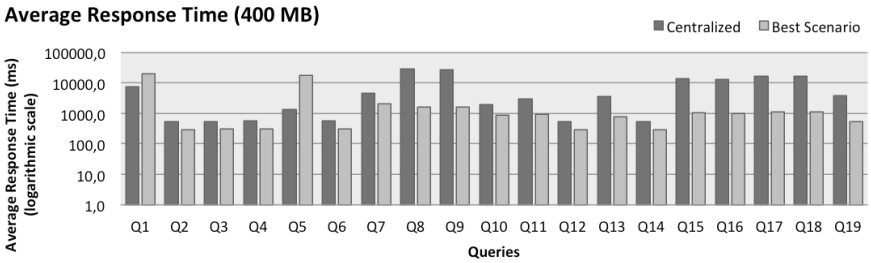

Fig. 3. Average query response times on the 400MB database, in the centralized and the best of the analyzed scenarios

scenario 3.1.2, 14 queries; ***scenario 3.1.3***, 14 queries; ***scenario 3.1.4***, 14 queries. When we compare scenarios 1, 2 and 3 for these 17 queries that have benefited from the fragmentation, 6% of the queries achieved the best response time in scenario 1; 1, 6% in scenario 2; and 88% in scenario 3. Recall that scenario 3 is the one that do not use the frequent attribute in the fragmentation.

After presenting the experimental results, we now proceed to a discussion of the results we obtained with the three database sizes. Initially, it is important to remember that in the 4MB and 40MB experiments there were 11 queries that have benefited from the fragmentation, while in 400MB experiment there were 17 queries that have benefited from the fragmentation.

Queries Q1 and Q5 did not benefit from the fragmentation in any of the three proposed experiments. For these queries, in all scenarios, all fragments needed to be accessed during query processing. Moreover, they had large partial results, and the transfer time between the local nodes and the mediator had great influence in the total query processing time.

Queries Q2, Q3, Q4, Q6, Q12, and Q14 did not benefit from the fragmentation in the 4MB and 40MB experiments, but did in the 400MB one. These queries needed to access all fragments in all of the 10 evaluated scenarios. However, in the 400MB experiment, although they still had to access all fragments, the number of documents in each fragment was much smaller than in the centralized database. Of course this is also true for the other two database sizes, but here, the large size of the database (400MB) had a great negative impact in the processing times of the centralized database, much more than in the 4MB and 40MB databases. Another point worth noting in the 400MB ex-

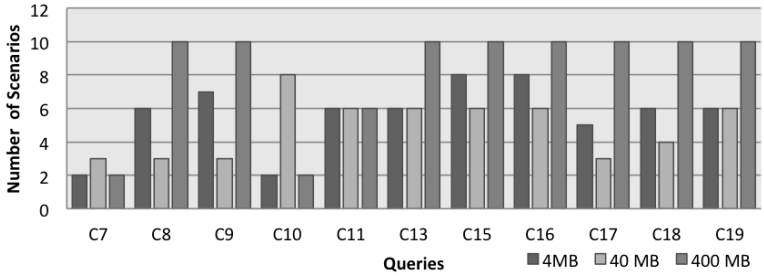

Fig. 4. Comparison between the scenarios that have benefited from the fragmentation

periment is that the best performance for these queries took place in scenario 3 (specifically, scenario 3.1.2) in which the database was fragmented by an infrequent attribute. Queries in scenario 3.1.2 performed better than in other scenarios because its fragments contained approximately the same numbers of documents (uniform distribution of data), which contributed for load balance. In fact, handling large XML documents has proven to be a hazard in performance [Ferraz et al. 2010]. The smaller the documents, the better the access performance.

As for the queries that used the frequent attribute "total", Figure 4 shows, the number of scenarios in which the fragmentation was beneficial. Note that the 400MB experiment has, in general, a larger range of beneficial scenarios in each of the queries.

As a conclusion, in the experiments with the 4MB and 40MB databases, performance gains were only obtained in queries that had "total" as its selection predicate. These queries did handle significantly large size data, and so issues such as seek time, latency and throughput were not as significant as in the larger dataset. Generalizing the results, we may suppose that larger datasets would benefit from the fragmentation even more. However, we do not have experimental results to show this for now.

5. RECOMMENDATIONS FOR HORIZONTAL XML FRAGMENTATION

XML fragmentation design must consider several criteria, similar to what was done for other data models that preceded it. This section presents some of these criteria, which were inspired by the relational [Ozsu and Valduriez 2011] and object-oriented [Baião et al. 2004] models. Our criteria consider the experimental results presented in the previous section.

There are important issues that must be evaluated on a distribution design in order to choose the best fragmentation alternative to a given collection. This work focuses on three points that have proven to have great influence on the quality of the data distribution:

Application Characteristics: operations that are performed on the data;

Semantic of the Database: represented by attributes and relationships of the schema;

Quantitative information: instances of collections and their sizes.

The analysis stage consists in determining which type of fragmentation should be performed in the database. In literature, there are no studies defining points that should be considered in this step. In most cases, one assumes that the designer knows what kind of fragmentation is to be applied. In this article, we define some factors that allow us to consistently choose the type of fragmentation to be applied on XML databases.

The application information is fundamental to understand its behavior, i.e., to identify, for example, the actual use of the database by the user. This makes it possible to assess how it is being used so that it can be fragmented to bring faster answers to the user. The analysis stage collects three kinds of data: ***access operations***, which is composed of information relating to the operations applied on queries; ***logical schema***, that includes the XML schema and XML documents, and finally, ***quantitative***

information, that regards information about the data volume and type of database that will be fragmented. We now detail the information extracted at this stage.

Frequent Queries. We analyze frequent queries to obtain data access statistics. The main statistics we collect at this stage are the following.

Classification of Operations: For each frequent query, we classify operations as selection or projection.

Operation Frequency: We extract the frequency of each operation type, including projected attributes and selection predicates.

Attribute Cardinality: We evaluate the cardinality of the attributes to which each operation is being applied. This assessment is particularly important in the decision towards vertical fragmentation. This type of fragmentation cannot be applied on attributes with cardinality greater than one, as this compromises the reconstruction of the fragments.

Frequency of attributes: We analyze the frequency of attributes on frequent queries. Also, we check the affinities of these attributes, i.e., we compare all occurrences of the attributes in the same query.

Database Type. We evaluate the database that will be fragmented. Depending on the database type, there are limitations on the fragmentation type that can be applied. A database with a single document repository can only be vertically or hybridly fragmented.

Data Volume. We measure the database size both in terms of number of documents and in terms of disk space occupied. This type of analysis allows us to verify the feasibility of fragmenting the database. For this work we classify the databases in three sizes, which are: ***small***: $<$ 10MB; ***medium***: 10MB to 100MB; and ***large***: $>$ 100MB.

Based on the aforementioned information and on the experiments we performed, we now present recommendations to the XML fragmentation design. It is important to note that this study presents recommendations for horizontal fragmentation only, i.e., we do not consider vertical fragmentation or hybrid fragmentation in our recommendations. Additionally, the development of tools and algorithms to support the fragmentation design is out of the scope of this article. However, future implementations could use our recommendations in algorithms for XML data fragmentation.

—Horizontal fragmentation can be applied over a collection that is stored in a MD repository.

—Large-sized collections should be fragmented.

—Horizontal fragmentation should only be applied over a collection C when the majority of the queries over C contains selection predicates on C.

—A small- or medium-sized collection C should only be fragmented when the majority of the queries over C contains simple predicates over the same attribute/element.

—Horizontal fragments of a collection C should have homogeneous sizes.

—A collection C should only be fragmented when queries over C do not contain aggregation functions.

It is important to not that for large collections, the fragment size does not significantly influence the query response times. Additionally, for small- and medium-sized collections, the query processing time is proportional to the number of fragments that is accessed during query processing. These characteristics should also be taken into account when fragmenting a database.

6. CONCLUSIONS

In this article, we have broadened the aspects that should be considered during the fragmentation design. We performed experiments on XML databases of 4, 40 and 400MB seeking to analyze the behavior of query execution times in various scenarios. By using these database sizes, we aimed at analyzing how data growth impacts in the distributed query processing performance.

Our experiments presented performance gains for frequent queries that have benefited from the fragmentation process, when compared to the results obtained in the centralized environment. Additionally, in the larger database we obtained gains even for queries that did not benefitted from the fragmentation. From these results we were able to define a set of recommendations for horizontal fragmentation of XML data that help in choosing the best type of horizontal fragmentation to be applied to a particular database.

As future work, we propose the implementation of these scenarios on larger databases to evaluate whether the increased volume of data implies in different results. Moreover, we plan to expand the recommendations in this work to other fragmentation types: vertical and hybrid.

REFERENCES

Abiteboul, S., Gottlob, G., and Manna, M. Distributed XML Design. In *Proceedings of the Symposium on Principles of Database Systems*. Providence, USA, pp. 247–257, 2009.

Andrade, A., Ruberg, G., Baião, F., Braganholo, V., and Mattoso, M. Efficiently Processing XML Queries over Fragmented Repositories with PartiX. In *Proceedings of the International Workshop on Database Technologies for Handling XML Information on the Web*. Munich, Germany, pp. 150–163, 2006.

Baião, F., Mattoso, M., and Zaverucha, G. A Distribution Design Methodology for Object DBMS. *Distributed Parallel Databases* 16 (1): 45–90, 2004.

Bonifati, A. and Cuzzocrea, A. Efficient Fragmentation of Large XML Documents. In *Proceedings of the International Conference Database and Expert Systems Applications*. Regensburg, Germany, pp. 539–550, 2007.

Bremer, J. and Gertz, M. On Distributing XML Repositories. In *Proceedings of the International Workshop on the Web and Databases*. San Diego, USA, pp. 73–78, 2003.

Ferraz, C. A., Braganholo, V., and Mattoso, M. ARAXA: Storing and Managing Active XML documents. *Web Semantics: Science, Services and Agents on the World Wide Web* 8 (2-3): 209–224, 2010.

Figueiredo, G., Braganholo, V., and Mattoso, M. Processing Queries over Distributed XML Databases. *Journal of Information and Data Management* 1 (3): 455–470, 2010.

Gang, G. and Rada, C. Efficiently Querying Large XML Data Repositories: A Survey. *IEEE Transactions on Knowledge and Data Engineering* 19 (10): 1381–1403, 2007.

Kling, P., Ozsu, M., and Daudjee, K. Generating Efficient Execution Plans for Vertically Partitioned XML Databases. *PVLDB* 4 (1): 1–11, 2010.

Kling, P., Ozsu, M., and Daudjee, K. Scaling XML Query Processing: Distribution, Localization and Pruning. *Distributed and Parallel Databases* 29 (5): 445–490, Oct., 2011.

Kurita, H., Hatano, K., Miyazaki, J., and Uemura, S. Efficient Query Processing for Large XML Data in Distributed Environments. In *Proceedings of the International Conference on Advanced Networking and Applications*. Washington, USA, pp. 317–322, 2007.

Ma, H. and Schewe, K.-D. Fragmentation of XML Documents. In *Proceedings of the Brazilian Symposium on Databases*. Manaus, Brazil, pp. 200 – 214, 2003.

Mahboubi, H. and Darmont, J. Enhancing XML Data Warehouse Query Performance by Fragmentation. In *Proceedings of the ACM Symposium on Applied Computing*. Hawaii, pp. 1555–1562, 2009.

Moro, M., Braganholo, V., Dorneles, C., Duarte, D., Galante, R., and Mello, R. XML: Some Papers in a Haystack. *SIGMOD Record* 38 (2): 29–34, 2009.

Ozsu, M. and Valduriez, P. *Principles of Distributed Database Systems*. Prentice Hall, 2011.

Pagnamenta, F. *Design and Initial Implementation of a Distributed XML Database*. Ph.D. thesis, University of Dublin, Irlanda, 2005.

Rodrigues, C., Braganholo, V., and Mattoso, M. Virtual Partitioning ad-hoc Queries over Distributed XML Databases. *Journal of Information and Data Management* 2 (3): 495–510, 2011.

Silva, T., Baião, F., Sampaio, J., Mattoso, M., and Braganholo, V. Recomendações para Fragmentação Horizontal de Bases de Dados XML. In *Proceedings of the Brazilian Symposium on Databases*. São Paulo, Brasil, pp. 145–152, 2012.

Yao, B., Ozsu, M., and Khandelwal, N. XBench Benchmark and Performance Testing of XML DBMSs. In *Proceedings of the IEEE International Conference on Data Engineering*. Boston, United States, pp. 621–632, 2004.