Towards Full-fledged XML Fragmentation for Transactional Distributed Databases

Rebeca Schroeder[1], **Carmem S. Hara (supervisor)**[1]

[1]Programa de Pós Graduação em Informática
Universidade Federal do Paraná (UFPR)
Caixa Postal 19.081 – 81.531-980 – Curitiba – PR – Brazil

{rebecas,carmem}@inf.ufpr.br

Level: PhD
Admission: 2010
Qualifying Exam: September 2012 (expected)
Conclusion: February 2014 (expected)
Steps completed: literature review and preliminary solution
Future steps: complete solution and evaluation

Abstract. *In data distribution design, fragmentation has been widely applied to provide scalable and available database services. Recently, the advent of cloud computing and the dissemination of large-scale distributed systems have shown that the traditional solution for data distribution is not suitable. In this paper we tackle the fragmentation problem for transactional databases which are highly distributed. Our specific goal is to develop a fragmentation approach that avoids distributed transactions and, consequently, improve the system throughput and maximize storage. To this purpose, we analyze a transactional workload in order to pack the most correlated data in the same fragment or in a set of few fragments. We are particularly focused on the XML model since it is a flexible model to support several applications, including systems in the cloud. This paper presents the current state of our work, preliminary results of our contribution and future directions we intend to pursue.*

keywords: *distribution design, fragmentation, XML, distributed transaction, cloud datastore, graph partitioning.*

1. Introduction

Cloud computing is transforming several aspects of computing and, mainly, providing new ways to deliver and handle services. Customers are attracted to on-demand services which may be available in real time and ready-to-go. Different service types are provided, from infrastructure to application services, including database as a service (DaaS). However, providing a full-fledged data management service for the cloud has not yet become a reality [Stonebraker et al. 2007, Agrawal et al. 2010]. The prospect of infinite storage capacity and processing power provided by the cloud has presented interesting new challenges to the database community and several open issues related to requirements of fundamental applications to the database industry [Abadi 2009].

The primary approach to leverage a scalable database service is through data fragmentation. By partitioning data and workload across a large number of sites, it is possible to speed up query processing, especially for OLAP systems where the goal is to maximize the intra-query parallelism. Many of these analytical systems have adopted cloud DBMSs that give up ACID guarantees in favor of availability and scalability [Stonebraker et al. 2007]. On the other hand, OLTP systems also need to be scalable but cannot give up transactions with strong consistency [Pavlo et al. 2012]. For these systems, the ability to provide transactions with ACID guarantees is closely related to the existence of a data fragmentation design which minimizes the execution of distributed transactions. Otherwise, it is hard to maintain strong consistency over data distributed across distant geographic sites. Moreover, distributed transactions add network messages, increase latency and, consequently, decrease throughput [Curino et al. 2010]. Even through it is not a new problem, the dissemination of cloud database systems has shown that the traditional solutions for data fragmentation are not enough [Pavlo et al. 2012, Yang et al. 2012].

In this paper, we tackle the fragmentation problem for XML data stored at distributed sites. XML may support several types of applications, including RDF-based systems [Abiteboul et al. 2011] and systems in the cloud [Abiteboul et al. 2008, 28msec 2012]. Most of the research efforts present heuristics for horizontal fragmentation of XML documents, especially for OLAP systems. They exploit simple selection predicates derived from the workload in order to generate fragments which maximize the intra-query parallelism [Cuzzocrea et al. 2009, Figueiredo et al. 2010]. They are based on partitioning methods where a fragment could be a document or a document subgraph. In general, XML documents are data-centric, and the semantics of data depends on the nested element structure. Thus, partitioning XML data involves fragmenting tightly-coupled documents which represent both the instance and the schema. We believe a finer-grained partitioning strategy must be provided to support more complex accesses over the schema, for example, structural joins. This kind of access pattern characterizes the main transactional workload for the database industry.

We propose a workload-based approach for XML schema fragmentation. This schema-based strategy allows fragmenting data at design level, while avoiding exhaustive analysis at instance level. We adopt a graph-based technique where database items are represented by nodes and transactions are represented by edges connecting items accessed together by transactions. The goal of our fragmentation method is to find a fragmentation schema which minimizes the weight of cut edges while maximize storage. Intuitively, the cost of the cut edges is related to the number of distributed transactions. Hence, we

aim to minimize the execution of distributed transactions by placing the most related data in the same fragment. As our first effort, we present an approach to cluster related data items through a similar reasoning applied by vertical partitioning of relational databases. In addition, we state the future challenges we intend to pursue towards full-fledged XML fragmentation for transactional systems in highly distributed databases.

The remainder of this paper is organized as follows. Section 2 highlights fragmentation issues and presents our proposal for XML fragmentation. In addition, we present preliminary results and work in progress for the thesis related to this work. Section 3 is dedicated to a comparative analysis of related work. We conclude in Section 4 by summarizing our preliminary contribution and future assignments.

2. XML Fragmentation

In general, data distribution models support horizontal and vertical fragmentation. In relational databases, the horizontal approach generates fragments which have a subset of relation tuples, and the vertical fragmentation produces fragments which contain a subset of relation attributes. Most of the data distribution models for XML are analogous to the models for relational data. While vertical fragments are usually subtrees of an XML schema, there is no consensus about XML horizontal fragmentation. In [Figueiredo et al. 2010] and [Kling et al. 2010], horizontal fragments are sets of documents from an homogeneous collection of XML documents, whereas in [Gertz and Bremer 2003] and [Ma and Schewe 2010] they may be instances of elements or instances of subgraphs of the XML document.

Fragmentation techniques are defined for specific workloads. For example, fragmentation approaches for analytical queries are focused on allowing parallel query scans over large datasets, especially through horizontal fragmentation [Cuzzocrea et al. 2009]. On the other hand, transactional workloads are usually supported by finer-grained partitioning to avoid distributed transactions. However, such fine-grained fragmentation has been supported by recent approaches that perform an exhaustive analysis on database instances [Curino et al. 2010, Pavlo et al. 2012, Yang et al. 2012]. Analyzing the whole database in a cloud infrastructure is hard and may be prohibitive. We are also focused on providing a fine-grained partitioning approach, however, our intention is to avoid such exhaustive process. We believe that an XML fine-grained fragmentation may be considered through a hybrid fragmentation process, where both vertical and horizontal techniques are applied. In this paper, we present our first effort in providing a suitable fragmentation approach for XML transactional systems in this context. Thus, our starting point is a vertical partitioning approach for XML schemas.

Our vertical approach defines how an XML schema must be fragmented in order to cluster data items accessed together by transactions. This in turn determines how each XML document must be partitioned. Although an XML schema could represent a database with a few large XML documents, we are considering a context which is commonly found in enterprise applications: an XML database with many small XML documents [Chen et al. 2012]. It is also common in RDF datasets which usually have many small RDF files. Before introducing our vertical fragmentation approach, we present a definition for XML schema structures and the workload characterization method in the next section.

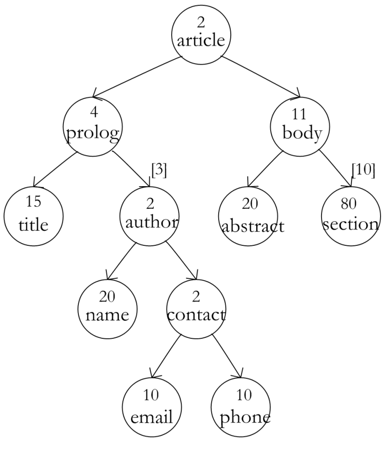

Figure 1. Structural Summary

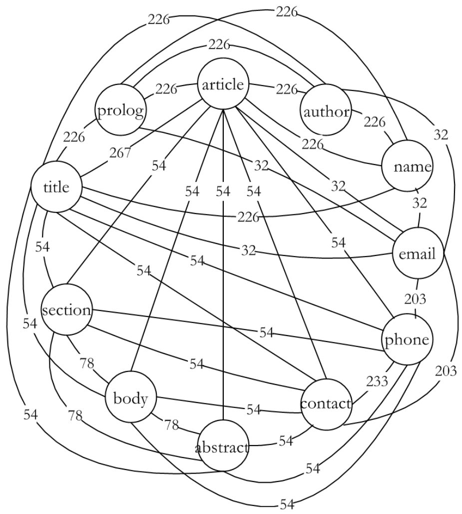

Figure 2. Affinity graph

2.1. XML Structure and Workload

An XML schema is represented by a structural summary in which a directed cyclic graph models the set of XML elements and attributes. Figure 1 presents a structural summary where internal nodes represent XML complex elements and leaf nodes are attributes or simple elements. The parent-child relationships among nodes define the hierarchical structure for the schema, where *article* is the root element. We also annotate the graph with the storage size required for each node and the average number of occurrences expected for a subelement within its parent. For example, *author* is a multi-valued subelement of *prolog*, and each *prolog* instance contains 3 *authors* in average. The storage size of a leaf node consists of the expected size of their values. As an example, for the node *email* the expected storage size is 10. For an internal node, the size is given by the structure of its children, but not the actual values stored by them. In the example, *contact* consists of the storage size required for keeping a structure composed of *email* and *phone*. In our fragmentation approach, the number of occurrences and storage size are considered to pack data items in a given fragment size.

We have defined a workload characterization method in order to relate data items according to a workload. These workload correlations are represented by an affinity graph. To this end, a node is created for each data item of the XML structure, where edges represent the usage of data items within a transaction. Thus, an edge connects two nodes if they are accessed by the same transaction. Edge weights denote the number of times the pair is accessed together. Figure 2 shows an affinity graph for the XML structure summary for a given workload. For a complete definition of our workload characterization method, please refer to [Schroeder et al. 2012].

Our goal is to generate an XML fragmentation schema by cutting an affinity graph. Figure 3 shows an example of fragmentation schema, where each fragment holds a subset of nodes of the affinity graph depicted in Figure 2. The strategy discussed in the next section heuristically minimizes the cost of the graph cut, while satisfying a storage threshold which represents the storage capacity for each fragment. Given that the cost of a graph cut is computed by the sum of the weights of the inter-fragment edges, our ultimate goal

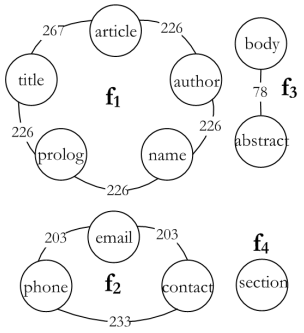

Figure 3. Fragmentation schema

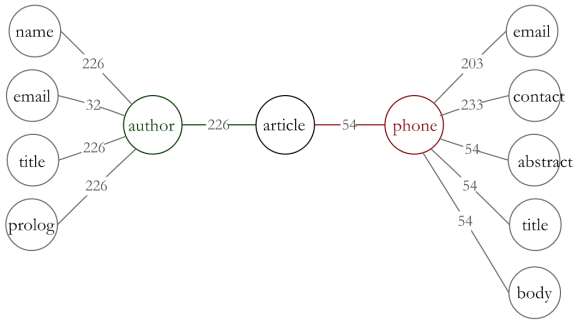

Figure 4. Strongly correlated set

is to pack the most correlated data items in the same fragment according to the workload.

2.2. Vertical Fragmentation for XML

We propose a greedy algorithm where an XML structural summary is divided into multiple and disjoint fragments. Such algorithm, namely *xAFFrag*, has been proposed in [Schroeder et al. 2012] and introduces the concept of a *strongly correlated set* (scs) of nodes in order to identify suitable fragments. More specifically, $scs(n)$ is defined for a node n in the XML structure and it determines which nodes have stronger affinity with n than with others in the graph. As an example, consider the connections of *article* with *author* and *phone* in Figure 4. According to our approach, *author* is in $scs(article)$ because the edge *(article,author)* has the highest weight among all connections of *author*. On the other hand, *phone* is not in $scs(article)$ given that its correlations with *email* and *contact* are stronger than with *article*. We denote by scs^+ the transitive closure of the scs relation.

Computing scs for nodes in the schema is the basis of our fragmentation strategy. Given an affinity graph, we start processing edges in descending order of affinity weight. For example, if the graph of Figure 2 is considered as input, we must start by processing the edge $(article, title)$ since it is the one with the highest weight. Thus, *article* and *title* are inserted in the first fragment, and we keep inserting nodes which are in $scs^+(article)$. However, before inserting new nodes, we must check whether the fragment size exceeds the storage capacity for fragments in the system. We refer to this storage capacity as a *storage threshold* that is given as input to our process. In order to exemplify our algorithm, we consider that the *storage threshold* is set to 100.

The size of a fragment is given by the sum of the expected number of occurrences of nodes multiplied by their sizes. For example, the size of the first fragment is set to 22 after the inclusion of *article*, *title* and *prolog*. Since we keep considering the $scs^+(article)$, we insert *author* and *name* and the size of the fragment is increased by 6 and 60, respectively. Observe that the multiple occurrence for *author* is considered to compute the storage size required for both nodes. Hence, the size for the first fragment is set to 88 and we stop to add nodes given that all nodes in $scs^+(article)$ were inserted. The same reasoning is applied to place all nodes in some fragment according to their scs and the storage threshold. After this initial step, the fragmentation schema generated is represented by Figure 3.

Observe that even through *section* is in $scs^+(body)$, it is not assigned to the third

fragment. This is because the storage size required by *section*(800) would exceed the storage threshold. Thus, we assume that *section* generates a fragment by itself and each instance is assigned to a distinct fragment.

Given that the query performance on distributed datastores has a direct correspondence with the number of data requests issued across the network, we also intend to minimize the number of fragments generated. In order to do so, the final step of *xAFFrag* algorithm maximizes storage by merging fragments if their sizes lie within the storage threshold. In our example, the size of fragments $\{contact, phone, email\}$ and $\{body, abstract\}$ are 66 and 31, respectively. Thus, they are merged and the final fragmentation schema generated is $\{\{article, title, prolog, author, name\}, \{contact, phone, email, body, abstract\}, \{section\}\}$.

2.3. Preliminary Results

We have conducted an experimental study in a distributed datastore in order to determine the effect of our fragmentation approach by comparing *xAFFrag* to close related approaches. The first approach, namely *MakePartition* [Navathe and Ra 1989], is a traditional solution for vertical partitioning of relational databases which applies a similar reasoning based on affinity graphs. The second one, called *XS*, is a close related algorithm for fragmenting XML documents [Bordawekar and Shmueli 2008]. We apply the fragmentation approaches on the XML schema and workload provided by the XBench benchmark. From each approach, the XBench dataset was fragmented and stored in a cloud datastore. We execute XBench using six different cluster sizes of eight Amazon EC2 nodes allocated in a single region.

In order to compare *xAFFrag* with *XS* and *MakePartition*, we have measured the system throughput and the query response time of a set of 11 XBench queries. The results showed that *xAFFrag* can improve query performance by 55% compared to *XS*. It shows that our approach is more effective in clustering related data in the same fragment. Besides, the system throughput achieved by *xAFFrag* is 22% higher than *MakePartition*, since our strategy to maximize storage decreases the number of requests issued across the network. Detailed information on the experiments can be found in [Schroeder et al. 2012].

2.4. Horizontal Fragmentation for XML

The horizontal partitioning is considered as our future step. The big challenge is to provide a non-exhaustive process to identify and represent selection operations in a big data environment. In order to support such operations, we intend to extend our graph model to represent affinities among sub-sets of data items. Thus, we can represent range of items accessed together and how they are related to other subsets of items. Besides, we believe that horizontal partitioning methods are more sensitive to changes in workload than vertical strategies. This is because changes in the volume of data accessed is more frequent than changes in the access pattern. To this end, we intend to provide an incremental solution to support such dynamic workloads.

Besides the horizontal fragmentation, we are currently investigating graph cut approaches proposed in the literature in order to derive an analytical model for the problem

addressed in this paper. Given that the optimal solution must be achieved by this model, our goal is to provide an analytical method to evaluate our solution and compare it with related works.

Many heuristics have been proposed for the general problem of partitioning graphs. Specially, the partitioning presented by this paper is closely related to the *k-cut* problem[Vazirani 2003]. There are several approximated algorithms proposed to solve this problem. However, the fragmentation problem adds complexity to the existing solutions, by considering storage maximization and XML constructs. We are currently investigating how to consider these constraints in order to optimize our algorithm.

3. Related Work

The problem of finding an optimal fragmentation schema for a distributed database is known to be NP-hard [Kling et al. 2010]. The traditional search process used to find the optimal fragmentation schema is the *what-if* method [Agrawal et al. 2004, Pavlo et al. 2012]. *What-if* is an exhaustive process which compares several potential solutions with a cost model to estimate how well a DBMS will perform. Hence, the search space to find a suitable solution tends to be larger and heuristic-based approaches become more attractive, specially for very large databases.

Horizontal partitioning is the common way to fragment a database and achieve a scalable service. However, the best approaches often depend on the application type. It is not different to fragment XML databases. For example, there are approaches providing heuristics for fragmenting XML analytical databases [Cuzzocrea et al. 2009, Figueiredo et al. 2010]. On the other hand, fragmentation methods for transactional databases are quite different. They usually require a complete analysis of the workload in order to identify access pattern and generate finer-grained fragments. Schism [Curino et al. 2010] addresses requirements to partition relational database with OLTP workloads. Similar to our approach, they intend to avoid distributed transactions. However, the entire database must be evaluated to identify tuples which are accessed together by transactions. We are also interested in generating fine-grained fragments according to the workload. Nevertheless, we target the XML model and avoid such exhaustive process.

The current state of our method is similar to the traditional vertical partitioning algorithm *MakePartition* [Navathe and Ra 1989] proposed for relational databases. They are also based on affinity graphs to create fragments. However, the number of fragments generated for a given dataset tends to be larger given that they do not focus on maximizing the storage capacity of the fragments. Our preliminary results show that it decreases systems performance since the number of requests issued across the network is also larger. To the best of our knowledge, the work most related to ours is the XS algorithm [Bordawekar and Shmueli 2008]. They also work on a graph partitioning problem, however, our approach is more effective to cluster related data in the same fragment. It happens because their analysis only considers the affinity among elements in parent-child, previous-sibling and next-sibling relationships.

4. Conclusion

We have presented the current state of our work to provide an effective fragmentation approach for highly distributed databases. Our workload analysis is applied to derive a

vertical fragmentation strategy for XML schemas which defines how to partition a set of XML documents. We make contributions for data-centric XML databases which are populated with many small XML documents. According to preliminary experiments, our approach is effective to improve the performance of distributed datastores by reducing the execution of distributed transactions, compared to close related approaches.

There are several issues that deserve further investigation. As future work we intend to extend our solution for considering horizontal fragmentation techniques through a feasible and incremental solution for dynamic workloads. In this context, we also plan to consider a load balancing solution in the presence of skewed workloads.

Acknowledgements: This work was partially supported by CNPq (Proc. 484366/2011-4-Ed.Universal) and by AWS in Education research grant award.

References

28msec (2012). Sausalito: a scalable xml database designed for the cloud. Available at: http://www.28msec.com/.

Abadi, D. J. (2009). Data management in the cloud: Limitations and opportunities. *IEEE Computer Society Technical Committee on Data Engineering*, 32:3–12.

Abiteboul, S., Manolescu, I., Polyzotis, N., Preda, N., and Sun, C. (2008). Xml processing in dht networks. In *Proceedings of the IEEE 24th ICDE*, pages 606–615.

Abiteboul, S., Manolescu, I., Rigaux, P., Rousset, M.-C., and Senellart, P. (2011). *Web Data Management*. Cambridge University Press.

Agrawal, D., El Abbadi, A., Antony, S., and Das, S. (2010). Data management challenges in cloud computing infrastructures. In *6th International Conference on Databases in Networked Information Systems*, pages 1–10. Springer-Verlag.

Agrawal, S., Narasayya, V., and Yang, B. (2004). Integrating vertical and horizontal partitioning into automated physical database design. In *Proceedings of ACM SIGMOD*, pages 359–370.

Bordawekar, R. and Shmueli, O. (2008). An algorithm for partitioning trees augmented with sibling edges. *Inf. Process. Lett.*, 108(3):136–142.

Chen, L. J., Bernstein, P., Carlin, P., Filipovic, D., Rys, M., Shamguvov, N., Terwilliger, J., Todic, M., Tomasevic, S., and Tomic, D. (2012). Mapping xml to a wide sparse table. In *ICDE'12*.

Curino, C., Jones, E., Zhang, Y., and Madden, S. (2010). Schism: a workload-driven approach to database replication and partitioning. *Proc. VLDB Endow.*, 3:48–57.

Cuzzocrea, A., Darmont, J., and Mahboubi, H. (2009). Fragmenting very large xml data warehouses via kmeans clustering algorithm. *Int. J. Bus. Intell. Data Min.*, 4:301–328.

Figueiredo, G., Braganholo, V. P., and Mattoso, M. (2010). Processing queries over distributed xml databases. *Journal of Information and Data Management*, 3(1):455–470.

Gertz, M. and Bremer, J. (2003). Distributed xml repositories: Top-down design and transparent query processing. Technical report, TR CSE-2003-20. Dep. of Computer Science, U. of California, USA.

Kling, P., Özsu, M. T., and Daudjee, K. (2010). Distributed xml query processing: Fragmentation, localization and pruning. Technical report, University of Waterloo.

Ma, H. and Schewe, K.-D. (2010). Fragmentation of xml documents. *Journal of Information and Data Management*, 1(1):21–34.

Navathe, S. and Ra, M. (1989). Vertical partitioning for database design: a graphical algorithm. *ACM SIGMOD International Conference on Management of Data*, 18:440–450.

Pavlo, A., Curino, C., and Zdonik, S. B. (2012). Skew-aware automatic database partitioning in shared-nothing, parallel oltp systems. In *SIGMOD'12*, pages 61–72.

Schroeder, R., Mello, R. S., and Hara, C. S. (2012). Affinity-based xml fragmentation. In *15th International Workshop on the Web and Databases (WebDB 2012)*, Scottsdale, Arizona, USA.

Stonebraker, M., Madden, S., Abadi, D. J., Harizopoulos, S., Hachem, N., and Helland, P. (2007). The end of an architectural era: (it's time for a complete rewrite). In *VLDB '07*, pages 1150–1160.

Vazirani, V. V. (2003). *Approximation Algorithms*. Berlin: Springer.

Yang, S., Yan, X., Zong, B., and Khan, A. (2012). Towards effective partition management for large graphs. In *SIGMOD'12*, pages 517–528, New York, NY, USA. ACM.