Heuristic Horizontal XML Fragmentation

Hui Ma, Klaus-Dieter Schewe

Massey University, Information Science Research Centre
Private Bag 11 222, Palmerston North, New Zealand
[h.ma|k.d.schewe]@massey.ac.nz

Abstract. A challenging question is how XML can be used to support distributed databases. This leads to the problem of how to obtain a suitable, cost-efficient distribution design for XML documents. In this paper we sketch a heuristic approach to minimise query costs for the case of horizontal fragmentation. The approach is based on a cost model that takes the complex structure of queries on XML documents into account. We show that the minimisation of transportation costs is decisive, and that this can be achieved locally by either accepting or rejecting a horizontal fragmentation with a simple predicate that arises from one of the most frequent queries.

1 Introduction

Since a few years XML has attracted a lot of attention as a highly expressive datamodel. In particular, XML enables an integrated view on data at least within an organisation, and it is not unrealistic to set up an enterprise-wide XML-database. As large organisations are located at different places, a consequence of this view is that we have to deal with distributed databases. Thus, the problem arises to provide adequate distribution techniques for XML, and to show that these techniques will enable increased performance of the distributed systems in comparison to using centralised XML documents.

In the context of the relational datamodel (RDM) distribution design mainly addressed the problems of schema fragmentation and allocation of the fragments to the machines in a computer network [11]. Fragmentation is commonly divided into horizontal fragmentation, which splits a relation into disjoint unions, and vertical fragmentation, which projects a relation onto a subset of its attributes. Horizontal and vertical fragmentation have been discussed intensively in [3, 4, 7]. Several authors have approached the generalisation of fragmentation techniques to object oriented datamodels. For instance, horizontal fragmentation is discussed in [1, 5, 8], and vertical fragmentation in [2, 6, 10]. Due to the similarity between object oriented and semi-structured data, some of these techniques have also been adapted to XML [9].

In this paper we continue our work on distribution design for XML-based databases from [9], where we introduced fragmentation operations for XML. Finding a reasonable distribution is usually accompanied by a query performance cost model. The problem is to design fragments and to allocate them in such a

Proceedings of the CAiSE'05 Forum - O. Belo, J. Eder, J. Falcão e Cunha, O. Pastor (Eds.)

way that the overall performance of the distributed database system is better than the one of an equivalent centralised one.

This problem was only briefly sketched in [9] assuming that XML documents are stored in a relational database and that cost optimisation techniques known from the RDM would be applied. In this paper we present an alternative approach that does not assume a relational representation of the XML documents. Instead of this we develop a query cost model for XML. Then we present a heuristic approach to minimise query costs for the case of horizontal fragmentation. We show that the minimisation of transportation costs is decisive, and that this can be achieved locally by either accepting or rejecting a horizontal fragmentation with a simple predicate that arises from one of the most frequent queries.

2 A Query Cost Model

Let us now look briefly at query processing costs. For this we assume that the queries are expressed by some XML query algebra, so each query give rise to a query tree. We may then make the assumption that these query trees are optimised, which allows us to assume that the leaves of such trees are the input document(s), predecessors of leaves are selection nodes, which themselves have projection nodes as predecessors. In exceptional cases there are no such selection or projection nodes.

Crucial to the query costs are the sizes of documents that have to be built during query execution, as these documents have to be stored at secondary storage, retrieved from there again, and sent between the locations of a network. Therefore, we first approach an estimation of sizes adapting the work in [8].

In order to do so, we look at the elements ℓ and attributes a defined in the schema DTD, and estimate the size $s(\ell) = s(e)$, respectively, depending on the defining regular expression e for ℓ, the attributes of ℓ, and the types of these attributes. Then, if r is the root element, $s(r)$ is the (estimated) size of a document rooted at r.

Let s_i be the average size of elements in the set V_i – recall that $V_i = dom(b_i)$ for a base type b_i. In particular, let us assume $b_0 =$, i.e. s_0 is the average size of an identifier. This can be used to determine first the size $s(a)$ of an attribute a, i.e. the average space needed for it in storage. We obtain $s(a) = s_i$, if a is of type b_i, $s(a) = s_0$, if a is of an identifier or reference type, and $s(a) = r \cdot s_0$ if a is of mutiple reference type. In the last of these cases r is the average number of elements named ℓ' that are referenced in ℓ.

Now let $att(\ell)$ denote the set of attributes defined on element n. We can proceed inductively to define the size $s(e)$:

- If the defining expression e for ℓ is a base type b_i, then we obtain $s(e) = b_i$.
- If the defining expression e for ℓ is ϵ, we obtain $s(e) = \sum_{a \in att(\ell)} s(a)$.
- If the defining expression e for ℓ is an element name n, we obtain $s(e) = s(n) + \sum_{a \in att(\ell)} s(a)$.

- If the defining expression e for ℓ is an iteration $(e')^*$, we obtain $s(e) = q \cdot s(e') + \sum_{a \in att(\ell)} s(a)$, where q is the average number of successor elements with defining expression e'.
- If the defining expression e for ℓ is a sequence $e_1, \ldots, e_n$, we obtain $s(e) = \sum_{i=1}^{n} s(e_i) + \sum_{a \in att(\ell)} s(a)$.
- If the defining expression e for ℓ is a choice $e_1 \mid \cdots \mid e_n$, we obtain $s(e) = \sum_{i=1}^{n} p_i \cdot s(e_i) + \sum_{a \in att(\ell)} s(a)$, where p_i is the probability that the successor of ℓ is defined by e_i. In particular, we have $\sum_{i=1}^{n} p_i = 1$.

Horizontal fragmenentation corresponds to replacing a document x by some union $x_1 \upsilon_p \ldots \upsilon_p x_n$. The path p used herein just leads to a successor of the root element. The new documents x_i $(i = 1, \ldots, n)$ are all obtained by selections, and we may assume $n = 2$ by switching to horizontal fragmentation with [illegible], i.e. satisfiable conjunctions of simple selection formulae.

Another round of query optimisation might shift the selection $\sigma_{p_2,\varphi}$ and the projection $\pi_{p_1,e}$ inside the newly introduced union ν_{p_3}, but the "upper part" of the query tree would not be affected. Therefore, in order to optimise horizontal fragmentation, it is decisive and sufficient to consider the projection-selection subqueries above.

The size of a selection node $\sigma_{p',\varphi}$ is $p \cdot s$, where s is the size of the successor node in the query tree and p is the probability that an element that matches the path p will satisfy φ. The size of a projection node $\pi_{p',e}$ is $(1-p) \cdot s \cdot \frac{s_e}{s_o}$, where s is the size of the successor node in the query tree, s_o is the average size of an element reached by p, s_e is the average size of an element defined by e, and p is the probability that two elements x_1, x_2 matching p' coincide on their projection to e, i.e. $\pi_e^{e'}(x_1) = \pi_e^{e'}(x_2)$.

Fragmentation results in a set of fragments $\{f_1, \ldots, f_n\}$ of average sizes $s_1, \ldots, s_n$. If the network has nodes $N_1, \ldots, N_k$ we have to allocate these fragments to the nodes, which gives rise to a mapping $\lambda : \{1, \ldots, n\} \to \{1, \ldots, k\}$, which we call a [illegible].

However, the fragments only appear on the leaves of query trees. More generally, we must associate a node $\lambda(v)$ with each node v in each relevant query tree. $\lambda(v)$ indicates the node in the network, at which the intermediate query result corresponding to v will be stored.

Given a location assignment λ we can compute the total costs of query processing. Let the set of queries be $Q^m = \{Q_1, \ldots, Q_m\}$. Query costs are composed of two parts: [illegible] and [illegible]: $costs_\lambda(Q_j) = stor_\lambda(Q_j) + trans_\lambda(Q_j)$.

The storage costs give a measure for retrieving the data back from secondary storage, which is mainly determined by the size of the data. The transportation costs provide a measure for transporting between two nodes of the network.

The storage costs of a query Q_j depend on the size of the involved documents and on the assigned locations, which decide the storage cost factors. It can be expressed as $stor_\lambda(Q_j) = \sum_h s(h) \cdot d_{\lambda(h)}$, where h ranges over the nodes of the

query tree for Q_j, $s(h)$ are the sizes of the involved documents, and d_i indicates the storage cost factor for node N_i $(i = 1, \ldots, k)$.

The transportation costs of query Q_j depend on the sizes of the involved documents and on the assigned locations, which decide the transport cost factor between every pair of sites. It can be expressed by $trans_\lambda(Q_j) = \sum_h \sum_{h'} c_{\lambda(h')\lambda(h)} \cdot s(h')$. Again the sum ranges over the nodes h of the query tree for Q_j, h' runs over the predecessors of h in the query tree, and c_{ij} is a transportation cost factor for data transport from node N_i to node N_j $(i, j \in \{1, \ldots, k\})$.

For each query Q_j we get a value for its frequency $freq_j$. The total costs of all the queries in Q^m are the sum of the costs of each query multiplied by its frequency. It can be expressed by $cost_\lambda = \sum_{j=1}^{m} cost_\lambda(Q_j) \cdot freq_j$.

3 Optimisation of Horizontal Fragmentation

We now present a heuristic approach for the optimisation of horizontal fragmentation. The major objective of the approach is to base the fragmentation decision on the efficiency of the most frequent queries. We have already seen that we may concentrate on projection-selection subqueries (PSSs) and that horizontal fragmentation with follow-on algebraic query optimisation will only affect these subqueries. As each such subquery defines a set of simple selection formulae. Obviously, we should only be interested in horizontal fragmentation based on normal predicates that are defined by this set of simple selection predicates.

Let us first look at the effects to query costs of a fragmentation with a single simple predicate. Basically we have to adapt query trees, take into account which effect algebraic optimisation will have on the PSSs, and find a modified location assignment λ that would reduce the query processing costs, provided such a λ exists. We have to distinguish three cases for this scenario.

Case I. Assume that the selection formula φ in the PSS has the form $\varphi \equiv \psi \wedge \varphi_0$ and that we use φ_0 to fragment the document x. So, the leave x would be replaced by the fragment x_1 of x that is determined by φ_0, and its predecessor in the query tree would become $\sigma_{p,\psi}$. There are no other changes to the query tree.

Then the size associated with the projection node does not change. Therefore, if the location assignment λ was optimal before the fragmentation, it does not make sense to change it for any node other than the three nodes corresponding to the PSS. Unless the selection node is deleted from the query tree, its size remains unchanged. This implies that there is at most a small change to the storage costs. In particular, as transportation costs are larger anyway, we neglect this effect on the storage costs. The processing of the subquery itself does not require any transport of data. Therefore, we can assume that λ assigns the same network node N_i to these nodes in the query tree.

As a consequence, the fragmentation can only require the change of the assigned network node from N_i to some N_j. This corresponds to the allocation of the fragment x_1.

Case II. Assume now that the fragmentation with the selection formula φ_0 leads to two fragments x_1 and x_2. Then the PSS would be replaced by a subquery of the form $\pi_{p_1,e}(\langle p\rangle\sigma_{p_2,\varphi}(\langle p'\rangle x_1 \upsilon_{p_3} x_2\langle/\rangle)\langle/\rangle)$. Algebraic query optimisation will move the selection inside the union.

In this case the size associated with the projection node does not change. As we can assume that the union node is assigned to the same network node as the projection node, we obtain again marginal changes to the storage costs. Thus, we can concentrate on the transportation costs. In addition, the processing of the subquery would require sending the smaller selection result to the location of the larger one.

As a consequence, the fragmentation can only require that the network node assigned to the two new fragments are different and maybe also different from the network node that was assigned to x. Then the fragment with the larger size after selection will determine the location assignment for the union and the projection node.

Case III. The assumption for the third case is the same as for case II. However, we now assume that the sizes of $\sigma_{p_2,\varphi}(x_i)$ are almost equal for $i = 1, 2$, and that the projection has only a small impact on the size of the result of the PSS. In this case it is advantageous to take the predecessor node n of $\pi_{p_1,e}$ in the query tree, and to assign $\lambda(n)$ to the projection and the union node. This implies that we have to transport both selection results to the same network node $\lambda(n)$. As in the two other cases, we may neglect the changes to the storage costs.

Now consider all the PSSs. Let $\{q_1, \ldots, q_N\}$ be the set of these local queries. Each of these local queries q_i has a "target node" T_i, i.e. a network node to which the result of the query will be transported. Let s_i be the size of the result of q_i and let $freq_i$ be the frequency of q_i $(i = 1, \ldots, N)$. Let x be some document fragment. Then for each network node N_j the [illegible] x [illegible] N_h are $cost_h(x) = \sum_{j=1}^{N} \sum_{i,T_i=N_j} s_i \cdot freq_i \cdot c_{h,i}$.

We apply the [illegible], which allocates a fragment x to a network node N_h such that $cost_h(x)$ will be minimal. According to our discussion for the three basic cases of how fragmentation affects the query costs, the allocation of fragments to network nodes following the cost minimisation heuristics, already determines the location assignment, provided that an optimal location assignment for the queries was given prior to the fragmentation.

Now we observe that one of the two fragments resulting from horizontal fragmentation with a formula in Φ will reside at the same location as the document before fragmentation, whereas the other fragment will be moved to a new location. In the first case the transportation costs remain the same. In the second case the transportation costs will be reduced.

This suggests to take a total order on the elements of Φ according to their frequency, i.e. let $\Phi = \{\varphi_1, \ldots, \varphi_\ell\}$ with $freq(\varphi_i) \geq freq(\varphi_j)$ iff $i \leq j$. Then determine $\Psi = \{\varphi_1, \ldots, \varphi_{\ell'}\}$ such that fragmentation with elements in Ψ leads

to a re-allocation of a fragment, whereas the fragmentation with elements in $\Phi - \Psi$ does not add changes. Then ℓ' can be determined by binary search.

4 Conclusion

In this paper we continued our work on distribution design for XML-based databases from [9]. We introduced a detailed query performance cost model, and then addressed the problem to design fragments and to allocate them in such a way that the overall performance of the distributed system is better than the one of an equivalent centralised one. Our approach does not assume a relational representation of the XML documents. Instead of this we presented a heuristic approach to minimise query costs for the case of horizontal fragmentation. We showed that the minimisation of transportation costs is decisive, and that this can be achieved locally by either accepting or rejecting a horizontal fragmentation with a simple predicate that arises from one of the most frequent queries.

References

1. Bellatreche, L., Karlapalem, K., and Simonet, A. Algorithms and support for horizontal class partitioning in object-oriented databases. *Distributed and Parallel Databases 8*, 2 (2000), 155–179.
2. Chinchwadkar, G. S., and Goh, A. An overview of vertical partitioning in object oriented databases. *The Computer Journal 42*, 1 (1999).
3. Chu, P.-C. A transaction oriented approach to attribute partitioning. *Information Systems 17*, 4 (1992), 329–342.
4. Chu, P.-C., and Ieong, I. T. A transaction-based approach to vertical partitioning for relational databases. *IEEE Transactions on Software Engineering 19*, 8 (1993), 804–812.
5. Ezeife, C. I., and Barker, K. A comprehensive approach to horizontal class fragmentation in a distributed object based system. *Distributed and Parallel Databases 3*, 3 (1995), 247–272.
6. Ezeife, C. I., and Barker, K. Distributed object based design: Vertical fragmentation of classes. *Distributed and Parallel Databases 6*, 4 (1998), 317–350.
7. Lin, X., Orlowska, M., and Zhang, Y. A graph-based cluster approach for vertical partitioning in databases systems. *Data & Knowledge Engineeering 11*, 2 (1993), 151–170.
8. Ma, H. Distribution design in object oriented databases. Master's thesis, Massey University, 2003.
9. Ma, H., and Schewe, K.-D. Fragmentation of XML documents. In *Proceedings XVIII Simpósio Brasileiro de Bancos de Dados (SBBD 2003)* (Manaus, Brazil, 2003), pp. 200–214.
10. Malinowski, E., and Chakravarthy, S. Fragmentation techniques for distributing object-oriented databases. In *Conceptual Modeling - ER '97* (1997), D. W. Embley and R. C. Goldstein, Eds., vol. 1331 of *Lecture Notes in Computer Science*, Springer, pp. 347–360.
11. Özsu, M. T., and Valduriez, P. *Principles of Distributed Database Systems.* Alan Apt, New Jersey, 1999.