A Relative Cost Model for XQuery

Soichiro Hidaka
National Institute of Informatics
Tokyo Japan
hidaka@nii.ac.jp

Hiroyuki Kato
National Institute of Informatics
Tokyo Japan
kato@nii.ac.jp

Masatoshi Yoshikawa
Kyoto University
Kyoto Japan
yoshikawa@i.kyoto-u.ac.jp

ABSTRACT

XQuery is a functional query language for XML. We propose a relative XQuery cost model that is able to estimate the performance gain during source level transformation. This research facilitates the evaluation of various rewriting techniques without introducing real engines. The cost model consists of simple recursive functions based on functional language constructs. They are determined using formal semantics and other known efficient algorithms. Analytic comparison of costs between expressions before and after transformation is possible in an engine-independent manner. The relativity of the model allows uninterpreted components within, which do not affect the mathematical proof of the comparison. Moreover, it can be tailored to reflect engine specific evaluation strategies such as the order of evaluation of operands.

1. PROBLEM STATEMENT AND CONTRIBUTION

In the database optimization research field, performance evaluation is often conducted by experiments using real engines. However, as they are conducted on their own engines ([3] for example), applicability to other engines is not guaranteed for the results. Since our target language XQuery[4] is relatively new, many of the engines are still in the research stage. Therefore, stable reference engines available to everyone do not yet exist. In addition, these engines make constant progress, which makes these results quickly obsolete. Installations are also hard tasks. Our goal is to make a cost model that can play a role of a sort of "virtual engine" as an evaluation infrastructure for rewriting optimization research. Our contributions of XQuery cost model include (i) simple, static, provable estimation of relative cost gain/loss in an engine-independent manner, (ii) ability to incorporate diversity in evaluation strategy so that the cost model can tell us which rewriting to choose. Please refer to [6] for more complete list of (semantically-validated) transformation rules, cost definition derivations, proved cost change calculationsas well as evaluation of the cost model using real engines.

SAC'07 March 11-15, 2007, Seoul, Korea

2. COST MODEL

Since the model should represent entities that produce correct results, (1) it is basically derived from formal semantics [4]. However, (2) for expressions whose efficient evaluation strategies are known, that strategy is used instead. In case of element construction expressions whose cost description can be found in neither of (1) or (2), our original assumptions apply (3).

The cost model consists of the following three basic constituents. **Cost function** $\mathtt{c}(e)$ is recursively defined by the cost, size, and probability functions of the subexpressions. **(Auxiliary) size function** $\mathtt{s}(e)$ is defined in terms of size and probability functions of the subexpressions. It doesn't depend on the cost of the subexpressions. **(Auxiliary) probability function** $\mathtt{p}(e)$ is defined in terms of the probability and size functions of the subexpressions. It is also independent of the costs of the subexpressions.

Summary of cost functions can be found in Table 1.

Estimation for case (1): The cost of **for** expression (CFOR) is determined using formal semantics. In its *Dynamic Evaluation* section, the number of judgments in the premise part are equal to the number of items in the input sequence s. We obtain (CFOR) assuming *uniform* cost for each evaluation of **return** clause ($f(\$x)$).

Although the evaluation strategies for each operator (we refer to those in Table 1 rather than physical operators) may depend on each real engine, this diversity can be incorporated to obtain an engine-specific cost model. For example, (CSOME) models engines that stops evaluation of **some** expression as soon as it finds a binding which makes its body yield true.

Estimation for case (2): Path expression is one of the language constructs that can characterize the behavior of implementations because there is a lot of optimization opportunities. Based on the notion of data and query complexity [5] and our simple benchmarking, we have chosen (CGSTEP) for engines that evaluate polinomially with respect to both query and data size such as eXist[1], and (CCSTEP) for those evaluate exponentially for query size and polinomially for data size such as Galax 0.4.0[2].

Estimation for case (3): Specification requires that an element constructor make a whole copy of its content. By

[1] `http://exist.sourceforge.net/`
[2] `http://www.galaxquery.org/`

expression	description	cost	label	size/probability	label
for $\$x$ **in** s **return** $f(\$x)$	for expr.	$\mathtt{c}(s) + \mathtt{s}(s) \cdot \mathtt{c}(f)$	CFOR	$\mathtt{s}(s) \cdot \mathtt{s}(f)$	SFOR
literal	literal	constant		(uninterpreted)	
$\langle x \rangle \{e\} \langle / x \rangle$	elem. constructor	$\mathtt{c}(e) + (\mathtt{s}(e) + \mathtt{s}(e//*)) \cdot C_e$	CDEC	1	SDEC
e/x	relative path expr.	$\mathtt{c}(e) + \mathtt{s}(e)^c \cdot k$	CCSTEP	$k \cdot \mathtt{p}(\text{nodename} == x) \cdot \mathtt{s}(e)$	SCSTEP
$e/q_1/q_2/\ldots/q_N$	trailing path expr.	$\mathtt{c}(e) + \mathtt{s}(e)^c \times N^d$	CGSTEP		
$e//x$	descendant path expr.	$\mathtt{c}(e) + k \cdot \mathtt{s}(e//*)$	CDSTEP	$\mathtt{s}(e//*) \cdot \mathtt{p}(\text{nodename} == x)$	SDSTEP
e_1, e_2	sequence	$\mathtt{c}(e_1) + \mathtt{c}(e_2)$	CSEQ	$\mathtt{s}(e_1) + \mathtt{s}(e_2)$	SSEQ
$\$x$	variable reference	C_{vr}	CVR	$\mathtt{s}(\text{dereference}(\$x))$	SVR
e_1 **and** e_2	logical expr.	$\mathtt{c}(e_1) + \mathtt{c}(e_2)$	CNAND	1	SAND
		$\mathtt{c}(e_1) + \mathtt{p}(e_1) \cdot \mathtt{c}(e_2)$	CAND	$\mathtt{p}(e_1) \cdot \mathtt{p}(e_2)$	PAND
some $\$x$ **in** s **satisfies** $p(\$x)$	quantified expr.	$\mathtt{c}(s) + \mathtt{s}(s) \cdot \mathtt{c}(p)$	CNSOME	1	SSOME
		$\mathtt{c}(s) + \frac{1-(1-\mathtt{p}(p))^{\mathtt{s}(s)}}{\mathtt{p}(p)} \mathtt{c}(p)$	CSOME	$1 - (1 - \mathtt{p}(p))^{\mathtt{s}(q)}$	PSOME

Meanings of symbols in the table: e, f: expressions, s: expression as a sequence, p: expression as a predicate, C_e,C_{vr},c, d, k: some constants, x: symbols for elements and variables, dereference($\$x$): an expression bound to $\$x$, nodename: name of the context node, $f(\$x)$: expressions that depend on variable $\$x$. Labels start with C, S and P for (resp.) cost, size and probability functions.

Table 1: Summary of cost functions for langage constructs

intuition, the cost will be proportional to the size of the contents of the element constructor. Consequently, we derived (CDEC) and (SDEC) for the element construction costs, where $\mathtt{s}(e//*)$ is recursively defied by $\mathtt{s}(e/*)+\sum_{x \in e/*} \mathtt{s}(x//*)$. Our benchmark on Galax also confirmed this model.

3. COST CHANGE ESTIMATION AND IMPLICATION

Let us consider the following associative law for quantified expression that has **where** clauses:

some $\$y$ **in** (**for** $\$z$ **in** q **where** $h(\$z)$ **return** $g(\$z)$) **satisfies** $f(\$y)$ = **some** $\$z$ **in** q **satisfies** ($h(\$z)$ **and** (**some** $\$y$ **in** $g(\$z)$ **satisfies** $f(\$y)$))

Cost change estimation using Table 1 concludes cost reduction under (CSOME) and (CAND), while increase under (CSOME) and (CNAND). The intuitive explanation of the cost reduction is that in the left hand side, although **some** may determine the result without scanning every element of its **in** clause, evaluation of the **for** expression for every element in the input sequence q is necessary. This is because the model is based on strict evaluation strategy. Lazy evaluation strategy would lead to the same cost for both sides. In other words the right hand side (partially) simulates lazy evaluation. Consequently, the cost model can simulate the lazy evaluation strategy by the application of this kind of transformation. As for cost increase, we were able to observe it for Galax 0.3.5 and eXist snapshot-20041119[3], since both operands of **and** expressions are always evaluated in both these implementations. This is an example that the cost model told us some specific pitfalls in existing engines with very sophisticated features. For these implementations, the law can be expressed in terms of **if** expressions by applying an additional transformation e_1 **and** $e_2 =$ **if** (e_1) **then** e_2 **else** `fn:false()` to avoid this cost increase.

4. RELATED WORKS

For implementations in which XML data is stored in RDB and XQuery is transformed to SQL [2], the cost models are discussed based on those of SQL engines [8]. Cost model in IBM DB2 XML [1] is also highly integrated with cost model for RDB. XQuery is translated into a set of alternative physical plans including index access, for which costs are estimated based on statistic information such as numbers of documents as well as average numbers of nodes with same name under specific elements. TIMBER [7] translates queries including XQuery into their tree algebra, in which cost estimator using selectivity in a path expression and the size function plays an important role. Our cost model works at logical level rather than physical plan-selection level, where concrete value estimated are compared and the plan with the least cost is chosen for execution. Full-fledged DB like [1] have rewriting phase, although the rules are based on heuristics. Our model is expected to provide a basis to the heuristics with respect to relative cost gain.

[3] `http://prdownloads.sourceforge.net/exist/eXist-snapshot-20041119.jar`

5. REFERENCES

[1] A. Balmin, T. Eliaz, J. Hornibrook, L. Lim, G. M. Lohman, D. Simmen, M. Wang, and C. Zhang. Cost-based optimization in DB2 XML. *IBM Syst. J.*, 45(2):299–319, 2006.

[2] P. Bohannon, J. Freire, P. Roy, and J. Siméon. From XML Schema to Relations: A Cost-Based Approach to XML Storage. In *ICDE*, pages 64–75, 2002.

[3] A. Deutsch, Y. Papakonstantinou, and Y. Xu. The NEXT Logical Framework for XQuery. In *VLDB*, pages 168–179, 2004.

[4] D. Draper, P. Fankhauser, M. Fernández, A. Malhotra, K. Rose, M. Rys, J. Siméon, and P. Wadler. XQuery 1.0 and XPath 2.0 Formal Semantics. W3C Candidate Recommendation, June 2006.

[5] G. Gottlob, C. Koch, and R. Pichler. Efficient Algorithms for Processing XPath Queries. In *VLDB*, pages 95–106, 2002.

[6] S. Hidaka, H. Kato, and M. Yoshikawa. An XQuery Cost Model in Relative Form. Technical Report NII-2005-016E, National Institute of Informatics, Nov. 2005.

[7] H. V. Jagadish, S. Al-Khalifa, A. Chapman, L. V. S. Lakshmanan, A. Nierman, S. Paparizos, J. M. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, and C. Yu. TIMBER: A native XML database. *VLDB Journal*, 11(4):274–291, 2002.

[8] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and extensible algorithms for multi query optimization. In *SIGMOD*, pages 249–260, 2000.