Available online at www.sciencedirect.com
ScienceDirect
Information Systems 32 (2007) 344–364
www.elsevier.com/locate/infosys

Schema-conscious XML indexing

Krishna P. Leela[1], Jayant R. Haritsa*

Database Systems Lab, SERC/CSA, Indian Institute of Science, Bangalore 560012, India

Received 4 May 2003; received in revised form 31 October 2005; accepted 31 October 2005

Recommended by Y. Ioannidis

Abstract

User queries on extensible markup language (XML) documents are typically expressed as regular path expressions. A variety of indexing techniques for efficiently retrieving the results to such queries have been proposed in the recent literature. While these techniques are applicable to documents that are completely schema-less, in practice XML documents often adhere to a schema, such as a document type descriptor (DTD). In this paper, we propose schema-conscious path-hierarchy indexing of XML (SphinX), a new XML indexing scheme that utilizes the schema to significantly enhance the search process. SphinX implements a persistent index structure that seamlessly combines the schema information with standard B-tree technology, resulting in a simple and scalable solution. A performance evaluation over a variety of XML documents, including the Xmark benchmark, indicates significant benefits with regard to both index construction and index access.

Keywords: XML; Index; DTD; B-tree

1. Introduction

Extensible markup language (XML) [1], by virtue of its self-describing and textual nature, has become extremely popular as a flexible medium of data exchange and storage, especially on the Internet. Correspondingly, there has been significant research activity, in both academia and industry, on the development of languages for specification of user queries on XML document repositories. Many query languages have been proposed in the literature, including Lorel [2], XML-QL [3], XPath [4], etc. from which XQuery [5] has emerged as the standard. These proposals have significant differences in their syntax and formulation, but at their core, they all support *regular path expressions* (RPEs), an elegant and powerful mechanism for specifying traversal of graph-based data such as XML. For example, the RPE /bib[//book|article]/author/lastname specifies (in XPath syntax) the retrieval of the lastnames of all authors of books or articles that are reachable from the bib root element.[2]

In order to process RPE queries efficiently, a variety of XML *indexing* techniques have been

*Corresponding author.
E-mail addresses: krishna@yahoo-inc.com (K.P. Leela), haritsa@dsl.serc.iisc.ernet.in (J.R. Haritsa).
[1]Currently with Yahoo! India.

[2]The standard output of XPath expressions is "node-sets" (i.e. a set of node-ids); however, since our focus is on XPath *queries*, we assume that the expected output is the actual data (i.e. the *sub-trees*) associated with the result nodes.

0306-4379/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.is.2005.10.002

proposed in the literature. These techniques include the classical Lore [6] and T-index [7], as well as more recent followups such as ToXin [8], XISS [9], Index Fabric [10], APEX [11], F&B-Index [12], Holistic Twig Joins [13], Barashev and Novikov [14] and ViST [15]. The methodology of the majority of these techniques is to first construct a graph-based equivalent of the original XML document, and then to create indexes on this graph representation.

We move on in this paper to considering situations where the XML document that is to be indexed additionally conforms to a *schema*, such as a *Document Type Descriptor* (DTD) [16]. Such situations are quite common in practice—for example, BioML [17] and MathML [18] are DTDs specified for information exchange by the genomics and mathematics communities, respectively. XML schemas have been used in the prior literature for deriving relational database schemas [19,20], for query pruning [21] and minimization [22], for gathering document statistics [23], etc. However, using schema information to enhance *indexing efficiency* has not been considered before to the best of our knowledge.[3]

1.1. The SphinX index

We propose here a new indexing mechanism, called *SphinX* (schema-conscious path-hierarchy indexing of XML), intended for schema-conforming XML documents. While SphinX can, in principle, be used with any kind of schema (e.g. XML Schema [25]), including those derived *post-facto* from the data (e.g. Data Guide [6]), in this paper we focus specifically on DTDs, which are an extremely popular and common schema representation in real-world applications.

Given an XML document and its associated DTD, stored either on a file system or native XML engine,[4] SphinX proceeds as follows: the document is first converted into its equivalent graph representation, called the *document graph*. A special feature is that this graph is created with *bi-directional* links in order to facilitate both top-down and bottom-up traversal of the graph. Then, the DTD is also converted into a graph-based representation, called the *schema graph*. The leaves of the schema graph contain, for all the paths in the document on which indexes have been built, pointers to the roots of B^+*-trees* (hereafter simply referred to as B-trees). Each B-tree indexes directly into the corresponding atomic values in the document graph. A pictorial overview of the SphinX system is shown in Fig. 1.

A crucial point to note here is that although the native XML documents may be highly complex and large in size, their associated DTDs are very *compactly* described through the use of regular expressions. For example, the DTD for *Xmark*, the popular XML benchmark [26], is less than 5 kB in size. This compactness is also captured in our schema graph representation of the DTD—the schema graph takes up only about 50 kB. This means that it can be reasonably assumed that although the schema graph is a persistent structure, it can always be loaded in its entirety into memory.

The schema graph supports the efficient determination of exactly those paths in the document graph that are *relevant* to a given query—this feature is extremely important since path identification forms the basis of answering RPE queries. That is, the schema graph ensures the *selection* of only the relevant paths making it both "precise" and "complete". In contrast, indexing techniques on schema-less documents require, from a candidate set of paths, the explicit evaluation and subsequent *rejection* of paths not relevant to the query (i.e. elimination of "false-positives"). The rejection process may require arbitrarily long traversals of the document graph, which is typically a large

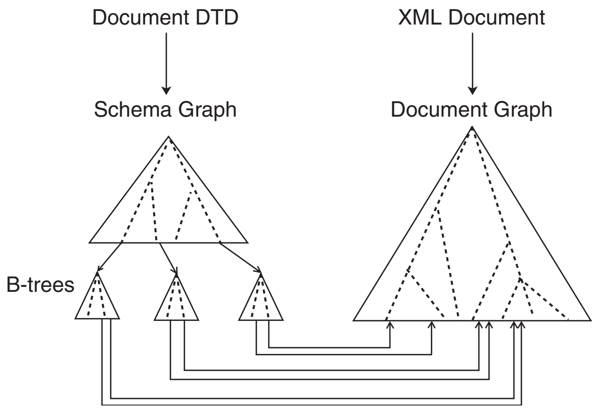

Fig. 1. Overview of SphinX system.

[3]Industrial products such as Tamino [24] and eXcelon [37] do use schemas for a variety of purposes, including determining the storage of XML objects and specifying the indexing properties of individual elements and attributes, but not for supporting indexing per se (personal communication with the technical personnel of these products).

[4]When XML documents are stored on RDBMS backends (e.g. [19,20]), the standard indexing mechanisms of relational engines have to be used.