May 2003, Vol.18, No.3, pp.403–406 J. Comput. Sci. & Technol.

Managing Very Large Document Collections Using Semantics

WANG GuoRen (王国仁)[1], LU HongJun (陆宏钧)[2], YU Ge (于 戈)[1]
and BAO YuBin (鲍玉斌)[1]

[1] *Department of Computer Science, Northeastern University, Shenyang 110004, P.R. China*
[2] *Department of Computer Science, Hong Kong University of Science and Technology, P.R. China*
E-mail: wanggr@mail.neu.edu.cn
Received August 31, 2001; revised July 12, 2002.

Abstract In this paper, a system is presented where documents are no longer identified by their file names. Instead, a document is represented by its semantics in terms of *descriptor* and *content vector*. The *descriptor* of a document consists of a set of attributes, such as date of creation, its type, its size, annotations, etc. The *content vector* of a document consists of a set of terms extracted from the document. In this paper, a semantic document management system XBASE is designed and implemented based on the semantics and the functions of three main modules, X-Loader, X-Explorer and X-Query.

Keywords semantic document, multidimensional exploring, document querying

1 Introduction

With the rapid development of the Internet and the World Wide Web (WWW), very large amount of information is available and ready for downloading, most of which is free of charge. At the same time, secondary storage device, the hard disk with large storage capacity is available at affordable prices. Disk space seems no longer a concern even for personal computer users. Most of us nowadays often dump a large number of various types of documents into our computers without much thinking. On the other hand, the file system has not been changed too much for the past decades. Although there are various types of file systems, most of them organize files in directories that form a tree structure, and a file is identified by its name and pathname in the directory tree. Finding a particular file on the disk with dozens of gigabytes data becomes never an easy task.

The problem has attracted attentions from both the academic researchers and industrial vendors. One possible solution is to introduce semantics of files into file systems[1]. By semantics of files, it means any higher-level information related to the file. Vasudevan and Pazandak listed the following examples of such information[2]: (1) *definitional*: file extension and magic numbers that define file type; (2) *associative*: keywords in file that characterize content; (3) *structural*: physical and logical organization of the data, including intra- and inter-file relationships; (4) *behavioral*: viewing and modification semantics, change management; and (5) *environmental*: creator, revision history. With such built-in information, tools that use such information can be built to provide users with more flexible way to access file other than file names and paths in directories.

The work reported in this paper is along the same line: managing large document collections using semantics. Among the semantic information mentioned above, two types of information, *descriptive* and *associative*, are used. The rational is that when a user searches for a document stored in her/his computer, she/he most likely has some vague idea about the document. On the other hand, the most difficult thing to remember is the name of the file that contains the document. Generally speaking, semantics of a document include any information regarding the document and its relationship with other documents. Among various kinds of semantic information about a document, we are in particular interested in two types of semantics, *definitional* information and *associative* information because (1) they can help to retrieve documents more efficiently; and (2) they are easy to remember by users, though may not be precise.

This work is supported by the Teaching and Research Award Programme for Outstanding Young Teachers in Higher Education Institutions by the Ministry of Education, China (TRAPOYT), and the National Natural Science Foundation of China (Grant Nos.60173051 and 60273079).

2 System Architecture

The system, as shown in Fig.1, is based on the client/server paradigm. The server side consists of an *X-Server*, the document databases, metadata, and indexes. We use prefix X because XML is used as a common language to specify the documents and metadata where data exchange is involved. Documents are stored in a document database system implemented on the top of an object database system[3] conforming to the ODMG Standard. Definitional semantics are stored as metadata, while associative semantics are stored in the form of inverted index to provide associative retrieval. The *X-Server* consists of modules for storing a document into database, inserting semantic information into metadata and index, processing user queries, and interfacing with the client side.

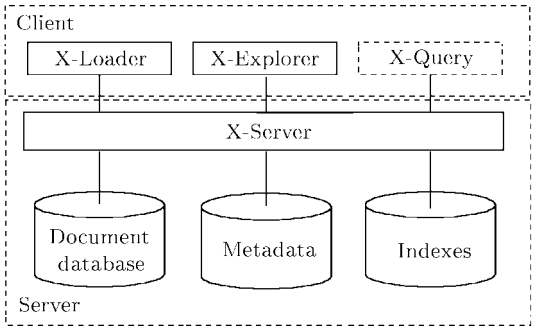

Fig.1. System architecture of XBASE.

The client side consists of *X-Loader* and querying facilities, *X-Explorer* and *X-Query*. X-Loader is responsible for loading documents and extracting their semantics. *X-Explorer* is a multidimensional document explorer to help users browse and locate documents they want. Keyword-based queries can also be input from X-Explorer. *X-Query* is used to query XML-style documents through an XML-based query language.

3 X-Loader

Fig.2 depicts the working process of X-Loader. The loader interface interacts with the user and Internet Browser which is the default browser of the system. When a user drags a link pointing to a document of her/his interests from the Internet Explorer to the X-Loader, the anchor text (treated as the title) and URL of the link are captured using the Microsoft OLE drag-drop technology. The document is then downloaded from the source site. There are two major steps after the documents are downloaded and before they are stored in the database.

The first step preprocesses a document. In case that the document is compressed, it will be decompressed first for semantic information extraction. The type of the document is recognized (analyzed), currently based on the file extensions. Documents can be categorized into three groups: XML documents (semi-structured documents), text-based documents, and non-text-based documents such as postscript and .PDF files. For semi-structured documents, schema information is extracted. For non-text-based documents, text extraction is applied.

The second step, *semantics extraction*, generates semantic information for the document. Most scalar attribute values, such as creation time, size, etc., are easy to obtain. In order to obtain the associative semantics of documents, the text representation of a document goes through the traditional term extraction process. Those terms form the content vector of the document. This term extraction process consists of typical procedures such as removing stop words, stemming, term selection, etc. For semi-structured documents, such as XML documents, the schema information is extracted.

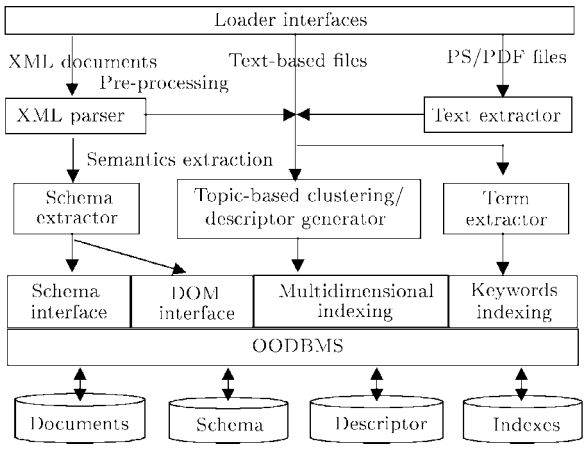

Fig.2. Working process of X-Loader.

Two types of semantic information, the document descriptor (DD) and content vector (CV) are indexed to support fact access. Based on the properties of the values of DD and CV, two indexing structures are supported. Four scalar attributes, *createdTime, fileSize, fileKind*, and *clusterNo*, are indexed using R^*-tree[4]. R^*-tree is an efficient multidimensional indexing method when the number of dimensions is not too large (3–5). Terms in string attributes in DD, *title, source*, and *memo*, together with the content vector, are indexed us-

ing a B^+-tree. The leave nodes of the tree contain entries of terms associated with a list of IDs of documents in which the term appears.

4 X-Explorer

Our system supports three ways for retrieving documents from the database: using query languages, using keywords, and explorative browsing. X-Explorer provides both keywords-based querying and visual multidimensional explorative browsing of the stored documents. Fig.3 is the screen dump of the interface of X-Explorer. The screen is divided into three areas, filtering window, document exploring window, and document list window.

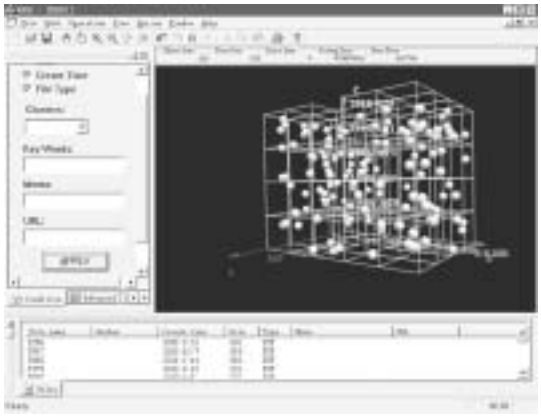

Fig.3. Interface of X-Explorer.

The filtering window allows users to select dimensions to explore and enter search conditions. Users can also set up complex Boolean search conditions on the document descriptor in the advanced filtering window. After the search conditions are set up, the system searches for the documents from the database and displays the results both in the document exploring window and the document list window. In the document list window, users can see the values of attributes in the document descriptor of the found documents. A particular document can be viewed by clicking corresponding entry in the document list window.

The search results are also displayed in the exploring window, where documents are represented as objects in a multidimensional space defined by the chosen attributes. To facilitate visual exploring in large volume of results, the following operations are provided: rotating, zooming, scaling, moving, undo/redo.

5 X-Query

The interface of X-Query is shown in Fig.4, which consists of three windows: document selection window (DSW), *DTD displaying window* (DDW), and path expression construction window (PECW). Users first can select a document they want to query from the *document selection window*. Once the document is selected, the DTD corresponding to the selected document is shown in the *DTD displaying window*, based on which users can construct an RPE-based query through the *path expression construction window*. After the query is constructed, it can be executed by clicking the *search* button on the right-bottom and the result is shown in a pop-up window.

Fig.4. Interface of X-Query.

In general, there are a root element and a target element in an RPE query and the target element is usually the result of the RPE query. An RPE query may involve four basic operators: *parent-child* (/), *ancestor-descendant* (//), *closure* ($*$ or $+$), and *predicate filtering* operators. In X-Query, an RPE query containing ancestor-descendant operators is rewritten as an equivalent RPE query only containing the other three basic operators based on the schema information about the database. For example, for RPE query "/site//homepage" it can be rewritten as query "/site/people/person/homepage" using the schema information. And then the *PathShorten* policy is used to rewritten RPE query. The procedure of the PathShorten policy is described as follows. 1) An RPE query is first rewritten as an equivalent RPE query not containing ancestor-descendant operators using the DTD information. For a given RPE containing ancestor-descendant operators, a set of RPEs not containing ancestor-descendant operators may be obtained by rewriting the RPE query, called Positive Path Set (PPS) of the given

RPE query. 2) Using the DTD information about the database, the set of all RPEs from the root element to the target element of the given RPE query can be figured out, named as the Universal Path Set (UPS) of the given RPE query. The Supplementary Path Set (SPS) of the given RPE query can be computed by *UPS-PPS*. It is easy to prove that the result of *PPS* is equal to the result of (*ext* (*target element*)− *SPS*). That is to say, the given RPE query can be evaluated through two ways. One is to use *PPS* and the other is to use *SPS*. Assume that $PPS = \{PPS_1, PPS_2, \ldots, PPS_n\}$. If *cost*(*PPS*) is less than *cost*(*SPS*), then *PPS* is chosen to compute the given RPE query. Otherwise, *SPS* is chosen. This is referred to as *PathComplementing* rule. 3) Assume that the *SPS* is chosen for computing the given RPE query (the situation is similar to *PPS*). For each SPS_i in the set of *SPS*, assume that $SPS_i = E_1/E_2/\cdots/E_n$. Examine to see if each element E_i $(1 \leqslant i \leqslant n)$ is a key element. If there is only one path via an element from the root element to the target element, then this element is referred to as a key element. It is obvious that if element E_k $(1 < k \leqslant n)$ is a key element, then $E_1, E_2, \ldots, E_k$ are also key elements. Assume that $E_1, \ldots, E_k$ $(1 \leqslant k \leqslant n)$ are the key elements in the path SPS_j. Then, path nodes $E_1, \ldots, E_{k-1}$ can be shortened since there is only one path from the root element to the target element E_n. That is to say, expression $E_1/E_2/\cdots/E_n$ is equivalent to expression $E_k/E_{k+1}/\cdots/E_n$. Thus, the number of parent-child operators in the path can be reduced by $k-1$. This means $(k-1)$ join operations can be dropped in this case if join operations are used to deal with parent-child operators. This is called *PathShorten* rule. 4) For each shortened path, it is evaluated based on the *extent-join* way.

6 Conclusions

This paper describes a system that manages very large amount of documents based on semantics. Three types of semantics are employed: definitional semantics, associative semantics, and schema information. These semantics about documents are stored with a multidimensional index and an inverted index. In addition, the functions of the system, including system architecture, *X-Loader*, *X-Explorer* and *X-Query*, are introduced in detail in this paper.

References

[1] Gifford D K, Jouvelot P, Sheldon M A *et al.* Semantic file systems. In *Proc. the Thirteenth ACM Symposium on Operating System Principles*, California, USA, 1991, pp.16–25.

[2] Vasudevan V, Pazandak P. Semantic file systems. Technical Report of Object Services and Consulting, Inc. http://www.objs.com/survey/OFSExt.htm, 1997.3.

[3] Yu G, Kaneko K, Bai G, Makinouchi A. Transaction management for a distributed object storage system WAKASHI-design, implementation and performance. In *Proc. the Twelfth International Conference on Data Engineering*, New Orleans, Louisiana, 1996, pp.460–468.

[4] Beckmann N, Kriegel H P, Schneider R *et al.* The R^*-tree: An efficient and robust access method for points and rectangles. In *Proc. the 1990 ACM SIGMOD Conference*, Atlantic City, NJ, 1990, pp.322–331.