Publ. in: Proceedings of the 20. GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken) : May 13 - May 16, 2008, Apolda, Germany / International University in Germany, School of Information Technology. Hagen Höpfner ... - Bruchsal : School of Information Technology, 2008. - S. 31-35

An XML Database as Filesystem in Userspace

Alexander Holupirek Marc H. Scholl

University of Konstanz
Dept. of Computer & Information Science
Box D 188, 78457 Konstanz, Germany
holupire|scholl@inf.uni-konstanz.de

Abstract

More and more data is stored and converted to XML, and with the upcoming ability of databases to handle semi-structured data efficiently it seems reasonable to rethink database-supported filesystems. This paper describes an early prototype of a filesystem implementation in userspace using a database with XML/XQuery support as back-end. Traditionally files are roughly classified as either text or binary. We add XML as a third type and expose formerly hidden content of files to the system and the user with both, its structure and content. The implementation establishes a link between database and operating system and allows the use of XML processing languages, such as XPath and XQuery, on the data. Since the database is mounted as a conventional filesystem by the operating system kernel, access via the established (virtual) filesystem interface as well as database enhanced access to the same data is provided. The implementation uses MonetDB/XQuery, a well-known relational database system with XQuery and XQuery Update Facility (XQUF) support, which integrates the Pathfinder XQuery compiler [2] and the MonetDB kernel [1].

1 Motivation

Since the beginning of database management systems there is a desire to store any data in a database and have it ready to be queried. Several industrial and research efforts, such as WinFS or the Be Filesystem have been undertaken to push the filesystem into a database. None made it to technical production quality. Offshoots, like Microsoft's Instant Search or Apple's Spotlight Architecture, however, can be found in any of the recent operating system variants, and a users' demand for products helping to find relevant content can be derived from the increasing popularity of Desktop Search Engines, such as Google's Desktop Search or Yahoo's X1. While these tools offer a smarter way to access personal information stored in the filesystem, the keyword driven search approach, as it is used by today's search engines, is inherently limited. An additional support of database query languages would be preferable.

2 Relational XML storage as jump start for database filesystems

Despite the fact that several database-driven filesystem attempts have already failed, the advent of XML brought some significant enhancements to RDBMS which inspired us to dare another attempt. A major problem of storing files in a DBMS (apart from using BLOBs) has been the basic necessity to provide a schema first. With an unmanageable amount of file formats this is an impossible mission. Schema-oblivious storage techniques rendered it possible for XML data to be stored in the database without previous knowledge of its interior structure[1].

[1] An additional XML Schema specification for the file type may be of advantage to formulate queries against the document, but is not mandatory.

Since more and more applications store their own data as XML (OOXML, OpenDocument-Format), those files are already prepared to be handled with database technology. From our point of view, these documents are nothing but serialized database instances. In consequence, they are not only stored as plain text, but directly shredded[2] into the DBMS. Legacy application may still process them conventionally by serializing them to their textual representation, however they are ready to be processed using XPath/XQuery in direct communication with the database. Applications may access their data in a more granular way with direct support from the RDBMS.

Imposed by the tree-based XML model relational storage and processing techniques for hierarchically structured data evolved and RDBMSs have learned to handle tree-shaped data. This is of direct benefit as the hierarchic nature of filesystem organization can now consistently be mapped to the relational storage and leverage the associated algorithms (an elaborate discussion of relational XML storage and algorithms can be found in [13, Chap. 2]).

In the following we describe an early implementation stage of how a database with XML/XQuery support can be used as a userlevel filesystem. The paper is structured as follows: The implementation is using the MonetDB/XQuery database system and the Filesystem in USErspace (FUSE) framework. These will be introduced shortly in Section 3 and 4. Section 5 describes our system architecture. After a short description of what we like to demonstrate during the workshop, we finish with a summary.

3 MonetDB/XQuery and the Pathfinder XQuery compiler

Pathfinder is an XQuery compiler implementation backed by relational database kernels. It is based on the transformation of XML document collections into relational tables [4] and emits relational algebra plans which consequently can be executed on any RDBMS [6]. In combination with the MonetDB kernel [1] it results in the open source MonetDB/XQuery system, which has proven to be one of the fastest and most scalable XQuery implementations available today [3]. RDBMS incorporate the knowledge and research efforts of years. They have proven to store and query large data sets efficiently and can be considered as mature technology. Using the power of relational database technology in combination with the recent findings in the domain of semi-structured data processing make such systems a promising choice for the implementation of a userlevel filesystem with query capabilities. Besides, the Pathfinder XQuery compiler appears particularly well suited due to the following aspects:

Scalability. It is able to handle huge amounts (up to 10GB) of XML data in an efficient manner.

Targets multiple back-ends. The Pathfinder XQuery compiler has already been enhanced by a code generator that emits SQL. This code generator targets off-the-shelf relational database systems (e.g., DB2) and turns them into efficient and scalable XQuery processors [6, 5].

Integrates a XML information retrieval system. PF/Tijah [8] is a text search system which is integrated with the Pathfinder compiler. It includes out-of-the-box solutions for common tasks such as index creation and result ranking. We expect that to be of great benefit for the stored textual and XML files.

4 Filesystems in USErspace (FUSE)

FUSE is a framework for implementing filesystems outside of the kernel in a separate protection domain in a user process. It was first implemented for and integrated into the Linux operation system kernel [12]. There are reimplementations for the Mac OS X [11], FreeBSD [7], and NetBSD [10, 9] kernels. The FUSE library interface closely resembles the in-kernel virtual file

[2] A terminus technicus used to indicate the conversation of XML in its textual representation to an internal format used by the database. Read it as “import”.

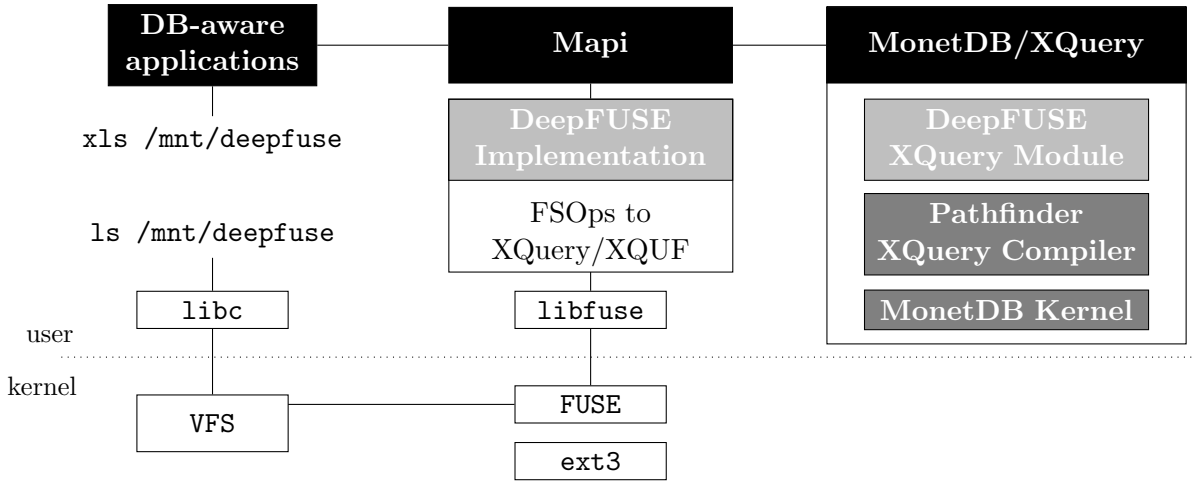

Figure 1: Establishing a link between OS and DBMS by implementing the DBMS as filesystem in userspace. Conventional as well as database supported access to the filesystem data is achieved.

system interface. The userlevel implementations are able to register function callbacks which get executed once a corresponding request is issued by the OS kernel. The FUSE kernel module and the FUSE library communicate via a special file descriptor: `/dev/fuse`. This file can be opened multiple times, and the obtained file descriptor is passed to the mount syscall, to match up the descriptor with the mounted filesystem. Our implementation is build on top of the `libfuse` library as depicted in Figure 1.

5 System Architecture

From a user's perspective, the system provides two access paths to the filesystem. Conventional/ legacy access for any application can be achieved as exemplified by the `ls` command. The (virtual) filesystem operations relevant for the listing of directory contents are looped back into userspace and captured by the functions registered with the callback interface of the `libfuse` library. The DeepFUSE implementation[3] is responsible for translating the filesystem operations into according XQuery/XQUF requests. The MonetDB/XQuery server offers a well-defined and easy to use API (encapsulated in the Mapi library) to leverage its functionality. DeepFUSE communicates as a client with a running server using a database driver and a textual protocol. The most frequently used functions called by the FUSE implementation have been encapsulated in an XQuery Module. This is most advantageous as it allows MonetDB/XQuery to prepare a query plan for those functions. A much faster execution is the consequence.

The userlevel filesystem implementation expects to operate on a filesystem XML representation valid against a W3C XML Schema Definition. A DeepFUSE XML instance is, of course, a (possibly empty) collection of files. Following the UNIX tradition there are block and character special, directory, fifo, symbolic link, socket and regular file types. Filesystem metadata, the `stat` information (access time, protection mode, file size ...) and any information relevant to operate a traditional filesystem is placed in the `http://www.deepfs.org/2008/fs` namespace. This namespace encapsulates the information needed to operate the database as a filesystem in userspace. It is stored in a separate XML collection inside the MonetDB/XQuery system.

The second access path (as depicted with the `xls /mnt/deepfuse` command) goes along

[3]We call it DeepFUSE, because operations on the file hierarchy do not necessarily end at the file level, but can continue on the file's inherent structure. The navigation along the file hierarchy in our implementation is basically the same as the navigation inside the XML files.

the conventional database interface. The filesystem data is stored in the database and functions implemented in the XQuery module are ready to be used. However, this is not the crucial point. We expect applications which use XML as their storage format to query their data for partial content or to only update "dirty" nodes instead of reading, processing and writing back complete XML files in their textual format as it is the case when stored in filesystems.

6 Demonstration during the Workshop

During the workshop we would like to give a demonstration of the system showing how filesystem operations propagate into requests in the DBMS. In the process we will work on the shell as a common UNIX user and issue some basic commands. Simultaneously, we will trace the resulting calls in the VFS and examine the corresponding generated queries.

7 Summary

While filesystems provide an easy and well-understood interface to the data, they lack important and demanded features like the ability to query the data. Applications such as desktop search engines or personal information management tools often use a keyword-based search approach which undeniably provides a user-friendly information discovery mechanism. There is no need to know the structure or format of the data let alone a query language. Ideally however, all retrieval strategies, i.e., with no, partial or full knowledge about structure, format and content of the data, should be supported. Existing tools usually index plain text content and thus have started to break the file content out of its black box. That means, while traditional filesystems do not care about the content of a file, the supporting userlevel applications which provide the missing search and retrieval functionality take it into account. A consistent further development is to provide means to expose the inherent file structure to allow content and structure queries.

RDBMSs are the right choice to query vast amounts of structured data. Data stored in the filesystem, however, is very heterogenous. With the rise of XML, the database community has been challenged by semi-structured data processing, enhancing their field of activity. Ongoing research efforts made RDBMSs, such as MonetDB/XQuery, capable of operating on tree-shaped data. As more and more applications save their data as XML, those files are ready to be directly saved into a DBMS. To go through with the concept, the file hierarchy tree along with its metadata is to be stored in the database in the same fashion.

We contribute an implementation that establishes the link between OS and DBMS. The DBMS is finally mounted as a filesystem by the OS and offers its data to arbitrary applications via the established (virtual) filesystem interface as well as through its database interface. As such we demonstrate the possibility of providing legacy filesystem access while storing the data in the database and have it ready to be queried.

8 Acknowledgment

This research is supported by the German Research Council (DFG) Research Training Group GK-1042 "Explorative Analysis and Visualization of Large Information Spaces".

References

[1] Peter A. Boncz. *Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications*. Ph.d. thesis, Universiteit van Amsterdam, Amsterdam, The Netherlands, May 2002.

[2] Peter A. Boncz, Torsten Grust, Maurice van Keulen, Stefan Manegold, Jan Rittinger, and Jens Teubner. Pathfinder: XQuery - The Relational Way. In Klemens Böhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson, and Beng Chin Ooi, editors, *VLDB*, pages 1322–1325. ACM, 2005.

[3] Peter A. Boncz, Torsten Grust, Maurice van Keulen, Stefan Manegold, Jan Rittinger, and Jens Teubner. MonetDB/XQuery: A Fast XQuery Processor Powered by a Relational Engine. In Surajit Chaudhuri, Vagelis Hristidis, and Neoklis Polyzotis, editors, *SIGMOD Conference*, pages 479–490. ACM, 2006.

[4] Torsten Grust. Accelerating XPath Location Steps. In Michael J. Franklin, Bongki Moon, and Anastassia Ailamaki, editors, *SIGMOD Conference*, pages 109–120. ACM, 2002.

[5] Torsten Grust, Manuel Mayr, Jan Rittinger, Sherif Sakr, and Jens Teubner. A SQL:1999 Code Generator for the Pathfinder XQuery Compiler. In Chee Yong Chan, Beng Chin Ooi, and Aoying Zhou, editors, *SIGMOD Conference*, pages 1162–1164. ACM, 2007.

[6] Torsten Grust, Sherif Sakr, and Jens Teubner. XQuery on SQL Hosts. In Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley, and K. Bernhard Schiefer, editors, *VLDB*, pages 252–263. Morgan Kaufmann, 2004.

[7] Csaba Henk. FUSE for FreeBSD. `http://fuse4bsd.creo.hu/`.

[8] Djoerd Hiemstra, Henning Rode, Roel van Os, and Jan Flokstra. PF/Tijah: text search in an XML database system. In *Proceedings of the 2nd International Workshop on Open Source Information Retrieval (OSIR)*, 2006.

[9] Antti Kantee. puffs - Pass-to-Userspace Framework File System. In *Proceedings of the Asia BSD Conference (AsiaBSDCon) 2007*, 2007.

[10] Antti Kantee and Alistair Crooks. ReFUSE: Userspace FUSE Reimplementation Using puffs. In *Proceedings of the 6th European BSD Conference (EuroBSDCon)*, 2007.

[11] Amit Singh. A FUSE-Compliant File System Implementation Mechanism for Mac OS X. `http://code.google.com/p/macfuse/`.

[12] Miklós Szeredi. Filesystem in USErspace. `http://fuse.sourceforge.net/`.

[13] Jens Teubner. *Pathfinder: XQuery Compilation Techniques for Relational Database Targets*. Ph.d. thesis, Technische Universität München, Munich, Germany, Oct 2006.