Paper, presented at: Proceedings of the 12th International Conference on Extending Database Technology (EDBT/ICDT 2009), March 23-26 2009, Saint-Petersburg, Russia

BaseX & DeepFS
Joint Storage for Filesystem and Database

Alexander Holupirek* Christian Grün* Marc H. Scholl
University of Konstanz
Dept. of Computer & Information Science
Box D 188, 78457 Konstanz, Germany
{holupire, gruen, scholl}@inf.uni-konstanz.de

ABSTRACT
Mere storage of personal data in state-of-the-art filesystems is a markedly well done job in current operating systems. Convenient access to and information retrieval from such data, however, is crucial to leverage the stored information. Thereby database style query languages can be of great use. We demonstrate a user level filesystem implementation that is built on recent semi-structured database storage techniques. As such, it serves as a storage layer for the BaseX XQuery processor and, while it appears to the operating system as a conventional filesystem, a large part of its content can be queried using XPath/XQuery.

1. PROBLEM DESCRIPTION
Steadily increasing hard disk capacities lead to more and more personal data stored in filesystems on personal computers. While the mere storage is an easy-to-manage task, convenient access to and information retrieval from huge amounts of data is crucial to leverage the stored information. Recent operating systems come with integrated search capabilities (*e.g.*, Instant Search or the Spotlight Architecture) or can easily be equipped with a third-party desktop search application, such as Google's Desktop Search.

These tools clearly offer a smarter way to access personal information stored in the filesystem. However, the keyword-driven search approach, as it is used by today's search engines, is—while perfectly suitable for the user—inherently limited for applications. An additional support for query languages would be preferable.

We demonstrate a filesystem implementation in userspace, called DeepFS, which is built on semi-structured database storage techniques. Together with BaseX, an open-source query processor[1], it offers query capabilities beyond keyword search. Applications (or sophisticated users) may use XQuery and its Full-Text and Update W3C recommendations to interact with the filesystem.

*DFG Research Training Group GK-1042 "Explorative Analysis and Visualization of large Information Spaces".

2. SYSTEM ARCHITECTURE
In Figure 1 we give an overview of the system we like to demonstrate. The key element is the (joint) storage system on the right side of the illustration. It is a filesystem in userspace implementation and assembles all data of the filesystem (file hierarchy, filesystem metadata, as well as user data). It uses a storage format suitable for the BaseX XQuery processor to evaluate queries on the data. In fact, DeepFS is completely transparent to the query processor and appears as just another storage layer with a conventional database, holding a collection of documents.

2.1 Filesystems in USErspace (FUSE)
DeepFS is based on the FUSE framework, that allows to implement filesystems outside the operating system kernel in a separate protection domain in a user process. It was first implemented for and integrated into the Linux kernel [11]. Additional implementations exist for the Mac OS X [10], FreeBSD [5], NetBSD [6, 7] and the OpenSolaris [9] kernels.

The FUSE library interface closely resembles the in-kernel virtual filesystem interface. Function callbacks, which are registered by the user level implementations, 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 on to the mount syscall, to match up the descriptor with the mounted filesystem.

2.2 Joint Storage for FS and DB
From user level perspective, the system provides two access paths to the filesystem (depicted by the octagons on the left side). Conventional/legacy access can be achieved for any application. The virtual filesystem (VFS) operations initiated by the applications are looped back into userspace and captured by the functions registered with the callback interface of the FUSE user level library.

Since any FUSE implementation allows to organize the data the way it likes, the DeepFS implementation stores the file

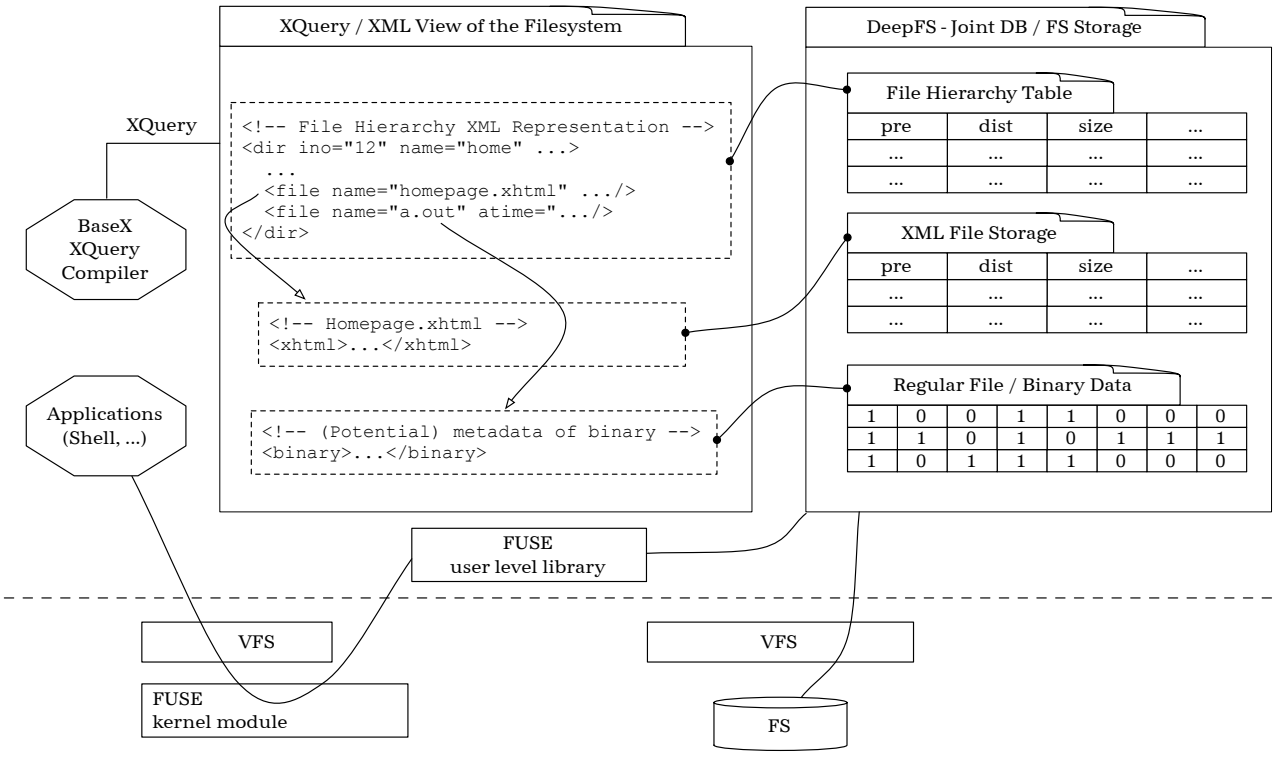


Figure 1: The user level filesystem serves as a joint storage for filesystem and database. Queries against an XML representation of the filesystem are possible as well as conventional filesystem access via the OS.

hierarchy by using the pre/distance/size encoding (see next section). It augments the table with some additional information, such as a unique `inode` number or the file type. However, the crucial point is that—while the storage is in first place optimized to communicate with the operating system kernel—it also communicates with the BaseX XQuery processor.

From the XQuery perspective, the FUSE system stores an XML representation of the filesystem, which is valid against a W3C XML Schema Definition. A DeepFS database instance consists of a file hierarchy representation and a collection of XML documents. Following the UNIX tradition there are block and character special, directory, fifo, symbolic link, socket, and regular file types. Filesystem metadata (access time, protection mode, file size, ...) is placed in a dedicated `http://www.deepfs.org/fs` namespace.

Traditionally, files are roughly classified as either text or binary. Motivated by the steadily increased dissemination of XML files, the DeepFS implementation adds XML as a third type. DeepFS uses the pre/distance/size encoding to store the file hierarchy—and its related metadata—and XML documents. As such, XML files are ready to be queried as an integrated part of the DeepFS document collection.

The data of regular files is stored, as before, by the underlying filesystem of choice. By default, only its metadata can be queried. However, on user's request, it can be transformed into a queryable representation by translator plugins. For instance, textual files with an inherent structure, such as e-mails, can be included with their structure exposed. The current e-mail translator produces a mapping, such as:

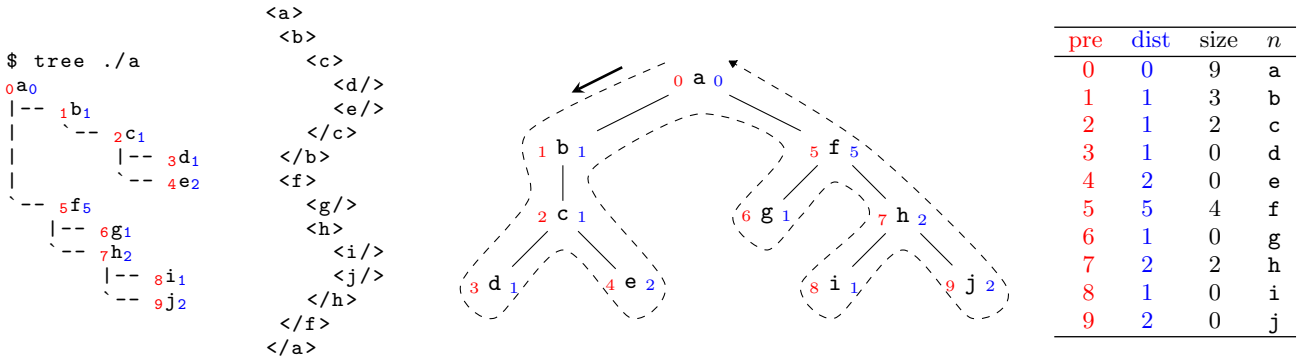
```
<Mail>
  <Subject>...</Subject>
  <From>...</From>
  <To>...</To>
  <Content-Type>...</Content-Type>
  <Section>...</Section>
  <Attachment ...>...</Attachment>
</Mail>
```

2.3 The pre/distance/size encoding

The BaseX XQuery processor operates on XML data stored in the pre/distance/size encoding. It is derived from the XPath Accelerator encoding [3], which is currently used in the MonetDB/XQuery system [8]. Those flat tree encodings have proven to show excellent query performance [2, 4].

Figure 2 shows a pre/distance/size encoded tree. The `pre` value is dense and ordered for the complete tree structure, and it is implicitly given by its position. `dist` defines the relative distance to the parent `pre` value, and `size` contains the number of descendants of a node.

To facilitate updates, the table structure is organized in disk blocks. A block directory references the first `pre` value of each block. The `dist` and `size` values have to be modified if deletions/insertions are performed: The `size` values are updated for all ancestor of that node—which means that a maximum of $log(n)$ nodes in the tree has to be accessed—and the `dist` values are updated for the following siblings



pre	dist	size	n
0	0	9	a
1	1	3	b
2	1	2	c
3	1	0	d
4	2	0	e
5	5	4	f
6	1	0	g
7	2	2	h
8	1	0	i
9	2	0	j

Figure 2: Storing trees (such as file hierarchies, XML documents) in the pre/distance/size encoding.

and the following siblings of the ancestor nodes. In comparison, *e.g.*, the storage of absolute parent references would demand a complete renumbering of all nodes in the tree table that follow a deleted/inserted nodes, yielding it as inapt for updates in filesystems.

2.4 The encoded file hierarchy

As the pre/distance/size encoding is basically a storage for tree structures, it can be seamlessly used to store the file hierarchy of a filesystem. The hierarchical mapping of filesystems is straight-forward, as illustrated in Figure 2. Together with the metadata (access time, protection mode, file size, ...) and any information relevant to operate a traditional filesystem, the file and directory structure is stored in the "File Hierarchy Table" (Figure 1) and accessible for the XQuery processor as well as for operating system requests.

3. QUERY THE FILESYSTEM

Once BaseX is told to operate on a filesystem database instance, it communicates with the DeepFS filesystem in userspace implementation and accesses the instantiated filesystem data structures. The filesystem appears to the BaseX XQuery processor as storage layer, providing access to the file hierarchy and a collection of XML documents. Many standard operations on files and directories can easily be represented in XPath/XQuery, as is shown in the following examples:

- the disk usage can be calculated with `du -s` or expressed in XPath with `sum(//file/@size)`
- files can be searched in the current directory with the command `find . -name find.me` or, by using XPath, with `.//file[@name = 'find.me']`
- text files can be deleted with `rm -r *txt` or, alternatively, `delete .//file[matches(@name, 'txt$')]`

Although the implemented mappings are straightforward, they externalize formerly hidden information. The leverage of tacit information, formerly encapsulated in various formats, leads to a standardized and easily accessible representation. This provides a basis to operate on filesystem data with query languages.

Think about finding an e-mail with a known sender, a big attachment and some keywords:

```
for $mail in //file/Mail
let $attach := $mail/Attachment
where $mail/From = 'jim.walker@mail.com'
  and $mail/Section
    ftcontains 'Hansson' ftand 'report'
  and $attach/@size > 3000000
return deepfs:path($attach)
```

Queries may combine filesystem metadata (such as file size, directory names) with file content and use both filesystem commands and languages for semi-structured data, such as XQuery, to request and manipulate the data. In the case of e-mails, comparable functionality is already offered by advanced e-mail applications. However, each application has to provide its own implementation, leading to highly redundant code for similar functionality. Our approach strives to provide such capabilities as a basic service of the filesystem layer. Furthermore, the search is not restricted to application defined communication paths (such as the often connected e-mail, calendar, address book applications), but can include any data stored in the filesystem.

4. DEMONSTRATION SCENARIO

We will present two FUSE enabled operating systems (*e.g.*, Linux, OS X, Free-/NetBSD ...) installed on a notebook. A preloaded database—containing a filesystem hierarchy and file data—is prepared and ready for operation. For a fresh start, an empty filesystem/database instance is mounted as well. The database instances are mounted as filesystem in userspace (system information will reveal that) and are as such ready to be used as any other conventional filesystem in a Unix OS. A Unix shell is opened in a terminal window, which allows for the navigation and operation of the filesystem with conventional Unix commands, such as `cd`, `mkdir`, `rmdir`, `find`, `grep`, etc. Arbitrary tools can be used to modify existing file contents, such as `vi` or `emacs` for text files. Next, the visual interface to BaseX (see Figure 3) is connected to the same database/filesystem instance. One can follow the manipulations done in the Unix shell by watching the database changes and vice versa. On the other hand, XQuery, XQuery Full-Text and XQuery Update requests can be performed on the file system, as described in the previous section. On a second workspace, additional verbose system information about the running processes will be displayed

Figure 3: BaseX provides visual access to query results. The user can browse and manipulate the results and further refine the result set by issuing further keyword-based or full-fledged (X)Queries.

(including a logging trace of the table accesses). We prepare numerous example queries (using BaseX as command line interpreter), so you do not have to rely on your XPath/XQuery knowledge. What you should experience from the demonstration is the parallel use of known, established and conventional filesystem interaction together with the query capabilities of an XQuery Processor. Recalling Figure 1, the two views on the filesystem/database instance are offered to the user and ready to be explored.

5. REFERENCES

[1] BaseX. Visual Exploration and Querying of XML Data. `http://www.basex.org/`.

[2] P. A. Boncz, T. Grust, M. van Keulen, S. Manegold, J. Rittinger, and J. Teubner. Pathfinder: XQuery - The Relational Way. In *Proc. of the 31st Int'l Conference on Very Large Databases (VLDB)*, Trondheim, Norway, 2005.

[3] T. Grust. Accelerating XPath Location Steps. In *Proc. of the ACM SIGMOD Int'l Conference on Management of Data*, Madison, Wisconsin, June 2002.

[4] T. Grust, M. Mayr, J. Rittinger, S. Sakr, and J. Teubner. A SQL:1999 Code Generator for the Pathfinder XQuery Compiler. In *Proc. of the ACM SIGMOD Int'l Conference on Management of Data*, Beijing, China, June 2007.

[5] C. Henk. FreeBSD Port of the FUSE Framework. `http://fuse4bsd.creo.hu/`, 2007.

[6] A. Kantee. puffs - Pass-to-Userspace Framework File System. In *Proc. of the 2nd Asia BSD Conference (AsiaBSDCon)*, 2007.

[7] A. Kantee and A. Crooks. ReFUSE: Userspace FUSE Reimplementation Using puffs. In *Proc. of the 6th European BSD Conference (EuroBSDCon)*, 2007.

[8] MonetDB. Query Processing at Light Speed. `http://monetdb.cwi.nl//`.

[9] OpenSolaris Project. Fuse on Solaris. `http://opensolaris.org/os/project/fuse/`, 2008.

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

[11] M. Szeredi. Filesystem in USErspace. `http://fuse.sourceforge.net/`.