HybFS: Adding multiple organizational views through a virtual overlay file system.

Stefania Victoria Costache
Email: stefania.costache@gmail.com

Dan Pintilei
Email: dan.pintilei@gmail.com

Abstract

Traditional hierarchical file systems offer a single organizational view of the data, thus making file searching and browsing cumbersome in situations when only incomplete information is available. In a semantic file system, the navigation is based on the additional metadata associated with files, therefore offering multiple views of the same files, but the original hierarchy is not kept. In this paper we present a virtual overlay file system that integrates the capabilities of semantic file systems with the original hierarchical file systems. This is done by tagging files with additional metadata, like tags that can have associated values. The user can organize files in a hierarchic way through the original file system operations but in the same time having access to advanced search capabilities and operations on multiple sets of files. The results of the search are listed through virtual directories which are generated at runtime.

1. Introduction

Semantic navigation proves to be usefull when dealing with files that are great in number and spread in a multitude of directories on the disk. In a hierarchic file system, files can be categorized in only one way, by specifying their location. However, this approach is not well suited in all situations anymore. Deciding how to organize files in the classic hierarchy is frustrating when there can be many possibilities to categorize the same file. Documents, music, images and movies are files that are always hard to manage. For example, a picture can be organized by the date and the place, by its content (it describes the city, or the landscape, or maybe it's a portrait), or its context (a long holiday, a weekend, or maybe a relaxing afternoon) but the user can choose only one file path to express all these attributes. To address this problem, semantic file systems and user space applications for file indexing offer a navigation based on the additional metadata associated with files, offering multiple views for the same files. But the hierarchical organization of a file system is not something to be eliminated. One can still need to organize the files in a hierarchical way and this is a real problem for a `full` semantic file system, where the user has no ideea of how the files are stored, how are they organized. A hierarchical view of files remains a logical and in the same time an unique way of identifying a file. That is why the file organization method imposed by the semantic file systems can be seen more as an additional extension of the existent file system.

In this paper we explore an approach for file organization that keeps the original hierarchical view and adds multiple organizational views for the information storage in the same time. This solution combines concepts used in previous approaches to the same problem. We implemented HybFS as an user-mode overlay file system, thus adding the multi-dimensional views as an extension to the current one. The file system can be mounted over a directory specified at mount time and, when accessing the mount point, the user can assign keywords to files and/or modify them, or issue search queries. Also, common tasks like extracting metadata directly from files or transfering the metadata between two HybFS mount points are accesible through an additional application.

We maintain the hierarchical representation because it is a good way of identifying files both uniquelly and logically, and in the same time we don't modify the original file organization from the underlying file system. Moreover, this could be considered as an extra precaution in case something happens to the metadata storage to permit the user to continue working easily with the files.

This paper is organized as follows. The next section outlines the previous work on which we based our design while Section 3 presents the HybFS features. Section 4 deals with the design and implementation details and Section 5 describes the evaluation of our solution. In the end, we draw the conclusions and some future directions of development in Section 6.

2. Related Work

In this Section we will present some of the more important attemps of adding a semantic organization to the file storage. These solutions are not only at the file system level, but there are also desktop applications, each of this category having its own advantages and drawbacks.

Beagle [1] is a desktop search application that supports not only the indexing of file content and format but it also provides the cappability to search inside IM conversations, mails, documents and webpages. It includes a GUI tool, an API for integration with other applications and a deamon for a real-time indexing. For this purpose it uses the *inotify* interface provided by the Linux kernel to update the information about the indexed files. Another well-known and used desktop application is *Spotlight* [3], a feature provided by Mac OS X. Spotlight it is based on a Metadata Server daemon that monitors the file system and responds to the clients search requests. Besides the basic file information, the deamon can also index the file content based on additional plugins. Also, queries that support the boolean operators can be performed.

Desktop applications are non-intrusive because most of them don't affect the file storage and they keep the additional information separate. Moreover, they can be accesible for the user, allowing advanced indexing configurations but in the case you change the application, all the work organizing the files is lost. Also, any other software from the system not only that will not be advantaged by this extension, but interfacing it with the desktop application will require changing parts of the software itself. In parallel with the development of such desktop applications, efforts have been made to design file systems that provide the same capabilities.

The first concept file system that implemented the idea of semantic file systems was described by David Gifford et. al. [10]. This led to a large amount of derivative work, including the semantic file systems and desktop applications recently developed. The main concept was the adding of attribute-value pairs to the files from the hierarchical file system, attributes extracted from files with the help of *transducers*. The search result is returned as a virtual directory, a directory that does not exist on the storage, but is created on the fly. However, a real problem is the fact that when creating a new file, the user has no control of the real path. Another interesting paradigm was the *logic file system* [13], which treats a path like a logical formula and files are indexed at creation. Other file systems that preceeded them, like Tagfs [8], Semfs [12], Orion [6], Insight [11], Tagsistant [4] start from the same ideas and don't take the original file path in consideration when performing queries, and not allowing the search request to be issued in an arbitrary directory from the file system (thus completly disconsidering the original hierarchical organization). Another solution, more oriented to a relational data storage approach was Windows Future Storage (WinFS) [5]. It was designed to allow users to find files using a structured query language, and it relies on a SQL Server layer. And since tagging files with additional attributes doesn't explain the relationship between different set of files, the Linking File System (LiFS) [7] came with support for attributed links between files. A different approach to semantic file systems is presented by Deepak Garg and et. al. in their technical report [9], when they introduce the file path as a separate attribute-value pair. However, their implementation was incomplete and allowed only manual tagging by the user.

3. HybFS Semantics

With HybFS one can have access to tag information from the interface of a file system and can specify paths for multiple directories and what plugins to use for parsing, by using a separate application. As an example, you can use the EXIF plugin for the images directory and the MP3 plugin for the music directory. However, the file indexing is not automated, for now, the user must specify the file or the directory to be parsed

The additional attributes are often called *tags*, or *keywords* and can be added to existent files or to the newly created ones. A tag can describe some content-related information of the file, when they are extracted directly from the file content, or a more general information about it. The tags can be simple, or can have an associated value, that is used to granulate even more the description. In this case, the attribute represents the criteria of description and the value - the subcategory. If the tag has no associated value, it will be associated with a special value, *null*. Any combination of multiple tags, and/or tag-value pairs is seen as a virtual directory that has the same name as the search query itself. We call it *virtual directory* because it only exist as an abstraction exported by HybFS and created at runtime. An entry in the virtual directory can be seen as a symbolic link to a file that match the current search pattern, or it can be a virtual directory that describes the other tags, or *tag:value* pairs assigned to the file entry, for further refining the query. In order for the user to have a hierarchic view, we define a directory with the name *"path:"* for the original mount point. This is desired for a better granularity. As an example, one can organize the pictures in directories based on the location where they were taken and refine the search with the aid of tags by date, people names, camera model and other.

The common file system operations are the same as in a normal file system but without support for symbolic and hard links yet. HybFS also supports operations like add, replace or remove for the file tags. The navigation in HybFS resembles the navigation through a normal hierarchic file system, except that the file path can be also based on the tag information associated with the files. From the application point of view, the result of a query is seen as a directory.

3.1. Query syntax

A query can be composed from:

- A tag, or a tag and value pair: $(picture)$, $(picture : autumn)$
- A conjunction, disjunction or negation of tags: $(picture : autumn + myself)$, $(picture : autumn|picture : winter)$ or $(!myself)$
- One or many conjunctions, disjunctions or negations of queries: $(q_1 + q_2)$, $(q_1 \mid q_2)$ or $(!q_1)$, where q_1 and q_2 are simple or composed queries.

3.2. HybFS operations

We describe the common HybFS features in terms of shell commands because they are user oriented and well known.

- cd : The equivalent of a change directory is a *refine query*. The resulted query is applied to the files that match the current query, or, if the path has a real component, to the files from the resulted path.
- ls : This command will list all the files that match the conjunction from the current query and the specified query, or, if a real path is specified, all the files that are in that path and match the query. When listing the root directory, we will see all the tags and values from the file system, and the special directory *path:*. Also, one of the possible drawbacks of full semantic file systems is that one doesn't receive any suggestions when navigating through the virtual directories and in many times it has to know what is looking for. We solved this by adding virtual folders to refine the results of the navigation. For example, when listing all the pictures that have been taken in the year "2008" and are in the real directory "My Pictures", or in its sub-directories, virtual directories are created from additional tags of the files already found that are not included in our query:

```
ls 'path:/My Pictures/(year:2008)'

IMG_6577.JPG    IMG_6590.JPG
IMG_6578.JPG    IMG_6639.JPG
IMG_6579.JPG    IMG_6662.JPG
IMG_6581.JPG    IMG_6683.JPG
IMG_6583.JPG    IMG_6684.JPG
IMG_6584.JPG    IMG_6685.JPG
IMG_6587.JPG    (picture:autumn)
IMG_6589.JPG
```

- rm : This keeps the original syntax, it will delete the file from the file system. Also, it can be used to delete *sets* of files that match certain queries, by specifying virtual directories.
- mv : The move operation has a special syntax, because it can be used for changing the set of tags for a file, or a set of files. The renaming of files keeps the original syntax. If the destination file path contains a virtual directory, then it will try to do a tag operation based on the queries that form the virtual directory path. Also, the queries that specify the new tags to be assigned must be based on conjunctions only. This is happening because a different syntax will be too ambiguous. If there is more than one query in the path, then the operations will happen in the specified order. The syntax of the destination query can be as follows:
  - $(tag_0 + tag_1 + tag_2 : value_2 + ...)$: all the tags for the files depicted by the source query and/or directory are removed and the tags $tag_0, tag_1, ...$ are added. If a tag doesn't have a value specified, then the "null" value will be assigned.
  - $(|tag_0{+}tag_1{+}tag_2 : value_2{+}...)$ All the tags for the files resulted from the source query and/or directory are kept, the new tags are added.
  - $(!tag_0{+}tag_1 : value_1{+}...)$ It removes all the pairs $tag_i : value_i$ for the files resulted from the source query. If the value is not specified, than it removes the pair, indifferent of the tag value.

For example, the command $mv\ 'q_0/*'\ '/path : /dir\_path/q_1'$ will execute the operation on tags described by q_1 for the files that match the query

q_0 and in the same time the files will be moved in the directory "$/path : /dir\_path/q_1$".

- cp : The copy of files will not assign the existent tags of the source file to the destination file. However, tag assignment can be explicitly specified, by adding to the destination path the conjunction query that contains them. To properly copy a file in the HybFS file system, or between two HybFS mount points, we designed a similar tool to *cp*.

Other file system common operations that are applied to files can be valid only when the path has a real component (the path can have a virtual component also, when the files are the result of a query). The *create* operation is somehow different, because it will interpret the virtual component of the path as a request to add new tags to the new file. Therefore, the query must be formed only from conjunctions, to prevent an ambiguous request.

4. Design and Implementation

The implementation of HybFS consists of three modules as described in Figure 1: the user-space file system, a library comprising of an uniform interface to the metadata storage and an application that allows loading of multiple plug-ins for extracting metadata from files, getting statistics about tags for testing purposes and transferring files between two HybFS mount points, together with their metadata. The plug-in support is represented through a generic interface that can be extended by each new plug-in sub-module. For further developing purposes, the library interface supports multiple instances of the back-end interface, called *'virtual directory branches'*, each one of them having its own database connection. However, for now, this can be useful only from the HybFS application.

4.1. Front-end Module

The front-end is represented by the HybFS file system interface. We implemented our file system with the FUSE [2] toolkit for user-level file systems in Linux, which is in turn based on the Virtual File System Layer (VFS). In this way, we structured HybFS as a layer of extended-content around an already existent location on the file system. The HybFS file system application implements the common file system operations needed to provide basic functionality. Also, it is responsible for initializing the back-end interface for the mounted directory, parsing the paths and passing the results to it. The abstract flow of a simplified request is described as follows:

1) A file system operation is issued (e.g. create, rename, unlink) by a user application and it is passed by the VFS to the FUSE file system driver and then to the HybFS file system.
2) In HybFS, the provided file path is parsed and the resulted queries and the real file path (if any) are packed in an internal representation used afterwards to issue operations to the metadata database.
3) In the case of a tag operation or a query, the metadata from the database is accessed and, possibly, modified.
4) The results are returned to the HybFS core.
5) Based on the results obtained, and if needed, the HybFS interface passes the operation to the underlying file system.

4.2. Backend Module

The backend module is represented by a library used to access the metadata database from the file system interface and our application. This provides a uniform interface to the query parsing and the database access internal methods, allowing the future development of other metadata indexing solutions.

For indexing the semantic attributes of the files, we use a relational database. This allows any search query to be directly mapped on a SQL query. The database is implemented using Sqlite, which is linked in the application as a library. The database contains three tables with information about the tagged files. The first table contains the file inode number, the file mode and the real path relative to the mounted directory. Also we keep all the tags assigned for a file in a separate field, for issuing search queries faster. The second table is for storing the tag and value pairs, together with a unique id and the last one keeps the association between a file id and a tag:value id. The last two tables can be used for tag statistics from the HybFS Control Application. For simple tags (that don't have a value), we assign the "null" value. The metadata directory is set in each mount point, with the name ".hybfs" . The main database is kept in the file ".hybfs_main.db", thus all the tables are in the same file.

4.3. HybFS Control Application

The HybFS Control application provides an interface to define tagging behaviors for different mounted directories. This will allow the user to tag automatically files based on their types and the existing supported tagging modules. For now we support tag extraction for MP3's and JPEG files. The application allows specifing

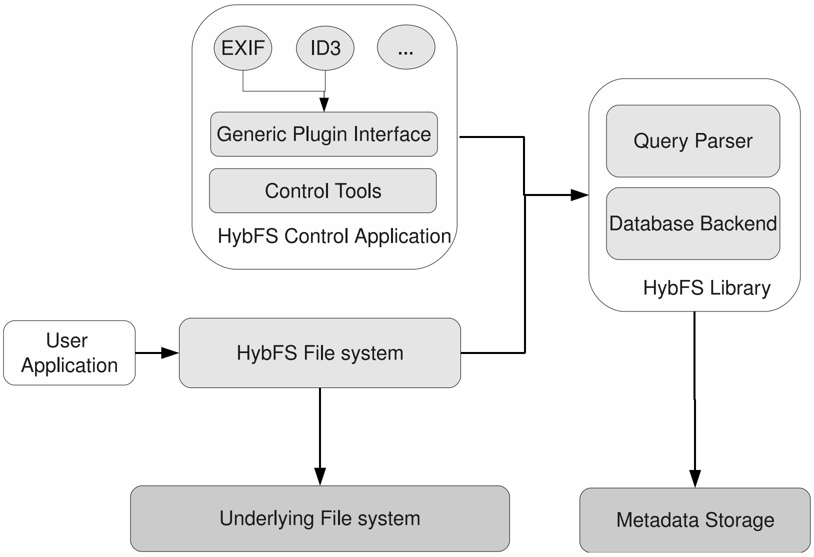

Figure 1. HybFS Design

multiple HybFS mount points and for each mount point a different set of plugins can be loaded. For example if we have the /fs1 and /fs2 as the directories in which HybFS was mounted, we can load the MP3 plugin for /fs1 and both the MP3 and EXIF plugins for /fs2. When indexing a directory from /fs1 only the mp3 files are parsed and their information loaded into the appropriate database while from a directory from /fs2 both mp3 and picture files are processed.

The application can also be used for special file operations like copy and move. When files are copied between different HybFS mount points, their tags are also transfered between the two databases. Even if there wasn't any HybFS file system mounted for a specific location, that location can be defined as a mount point in the application and a new database is created in order to permit copy/move operations to/from the new defined location. This allows the transfer of files together with their additional tags. The operation needed for this implies the defining of the directory where the files will be copied as a mount point using the HybFS Control Application on both the source and the destination. When defining an additional directory as a mount point, the application will initialize the database storage for tagging information, if it is not defined already.

5. Evaluation

HybFS is an overlay user mode file system implemented with FUSE which means that it will imply an additional overhead for file operations. Almost for each file operation a query to the database is required. As seen in [5] and [6], for the metadata an external database is used. In order to simplify the process and obtain better timing we chose to use Sqlite3, which is loaded as a library and accessed through the back end module of HybFS. This also absolves the user of running a database manager all the time as a separate application thus being an extra burden.

We evaluated HybFS in terms of user efficiency and we measured the overhead imposed by it. First, we measured the average time for locating a file and copy files which are identified based on a criteria. There are cases in which the tag navigation is not necessary. For instance, there are files that are well identified by their name and the directory structure in which they are found is very simple and logic. This usually happens when dealing with file types that are weakly represented and all the files are well grouped in very few locations. But even for these type of files, a certain improvement can be obtained by combining the hierarchical approach with the semantic one. If the user knows where a certain type of files are located, he can navigate to that location in the classic way and then use tags that provide a more insightful description.

The average time obtained for the two operations are presented in the followind table along with a *usecase* for them. The usecase is the best way to emphasize the advantages of using HybFS for the every day work.

test	file type	HybFS	EXT3
Locate file	mp3	13s	10s
Locate file	image	15s	20s
cp files	mp3	15s	3m
cp files	image	14s	2m30s

The tests were made on a 6 level directory structure (150 images, 200 melodies). A common use case is as follows:

One of your very good friends celebrates her birthday, so you have decided to make her a personalized birthday card (lets name her Anca). For that you've decided that you need one picture with her and after refining your birtday card, some pictures with flowers. You know exactly what picture with her you want. For what you are concerned, you know that the picture is somewhere in `photos/France/` so you follow the next steps (in our example are represented as shell commands, but the same scenario can be followed form any file browser):

```
$cd path:/photos/France/
$ls '(sena+character:anca)'
(camera:Canon_DIGITAL_IXUS_75)
(paris:null)
(type:image)
(month:april)
Paris/Anca/Sena.JPG
(year:2008)
```

With HybFS, after you entered in the directory `photos/France/`, you can refine your search using the semantic property of HybFS. You also know that the picture has been taken in Paris, near the Sena river, so you issue the command `ls '(sena+character:anca)'`. Now all you have to do is to copy this file somewhere, lets say `temp` from the overlaid directory. This is done with the `cp '(sena+character:anca)'/* path:/temp` command. For this picture you could have looked using a hierarchic file system too, and this in about the same amount of time. But you need to locate more than one image, using a hierarchic file system would search in more than one directory, if the files were scattered on the disk. In HybFS you can list the files with the following steps:

```
$cd '(flowers)'
$ls
 photos/19_may/Vannes/Vannes&Auray2.jpg
 photos/2007_contry_side/P1010078.JPG
 photos/France/Paris/Anca/Flori.JPG
 photos/France/Paris/Dan/Mar/1.JPG
 (location:vannes)
 (location:dersca)
 (location:paris)
 (location:marseille)
 (camera:C750UZ)
 (author:dan)
 (color:blue)
 (color:yellow)
 ...
```

These are all your images with flowers alongside their other tags. If you want a picture with blue flowers, you add a new filter: `cd '(color:blue)'` and you obtain:

```
$ls
 photos/France/Paris/Dan/Mar/1.JPG
 photos/France/Paris/Anca/Flori.JPG
 (author:dan)
 (location:marseille)
 (location:paris)
```

Then you can copy the images, or insert them in the birthday card.

6. Conclusions

In this paper we presented an virtual overlay file system that provides semantic capabilitites to normal hierarchic file systems, thus allowing advanced file searching and tagging from any user application. All the additional metadata is kept sepparate and with this aproach we don't modify the original file structure and we allow reverting to the old view at any time.

For now, to use the HybFS Control Application, the user must run the application and issue commands from the console. Further work is being done to split the application in two modules: a deamon that runs in background and takes care of the module loading and file parsing and a set of tools so that one can have access to special HybFS functions from anywhere. Also the application can be improved by automating the parsing of files and adding a possible integration with indexing solutions used by popular applications. How the solution was tested on Linux, the next steps will be porting it to other operating systems. We plan to replace of the FUSE interface with a similar solution for Windows and Mac OS X. In the end, allowing collaborative file tagging and searching by integrating HybFS with a peer-to-peer file system will be the next challenge.

References

[1] Beagle. http://beagle-project.org/Main_Page.

[2] FUSE: Filesystem in Userspace . http://fuse.sourceforge.net/.

[3] Spotlight. http://en.wikipedia.org/wiki/Spotlight_(software).

[4] Tagsistant. http://www.tagsistant.net/.

[5] WinFS. http://en.wikipedia.org/wiki/Winfs.

[6] F. Shaikh A. Joshi and S. Todwal. Orion File System : File-level Host-based Virtualization.

[7] A. Ames, C. Maltzahn, N. Bobb, EL Miller, SA Brandt, A. Neeman, A. Hiatt, and D. Tuteja. Richer File System Metadata Using Links and Attributes. In *Mass Storage Systems and Technologies, 2005. Proceedings. 22nd IEEE/13th NASA Goddard Conference on*, pages 49–60, 2005.

[8] Stephan Bloehdorn. Tagfs - tag semantics for hierarchical file systems. In *In: Proceedings of the 6th International Conference on Knowledge Management (I-KNOW 06*, pages 6–8, 2006.

[9] S. PanditA D. Garg, V. Mehta and M. De Rosa. Writable Semantic File System.

[10] David K. Gifford, Pierre Jouvelot, Mark A. Sheldon, and James W. O'toole. Semantic file systems. In *Communications of the ACM*, pages 16–25, 1991.

[11] David Ingram. Insight: A semantic file system.

[12] P. Mohan, S. Venkateswaran, M. Raghuraman, A. Siromoney, and I. Chennai. SemFS: A Semantic approach to File Systems.

[13] Y. Padioleau and O. Ridoux. A logic file system. In *USENIX Annual Technical Conference*, 2003.