Semantic File Retrieval in File Systems using Virtual Directories

Prashanth Mohan, Raghuraman, Venkateswaran S and Dr. Arul Siromoney
{*prashmohan, raaghum2222, wenkat.s*}*@gmail.com and asiro@vsnl.com*

***Abstract*— Hard Disk capacity is no longer a problem. However, increasing disk capacity has brought with it a new problem, the problem of locating files. Retrieving a document from a myriad of files and directories is no easy task. Industry solutions are being created to address this short coming.**

We propose to create an extendable UNIX based File System which will integrate searching as a basic function of the file system. The File System will provide Virtual Directories which list the results of a query. The contents of the Virtual Directory is formed at runtime. Although, the Virtual Directory is used mainly to facilitate the searching of file, It can also be used by plugins to interpret other queries.

***Index Terms*— Semantic File System, Virtual Directory, Meta Data**

I. Introduction

FILE and directory management is an essential and inevitable part of everyday computer usage. With hard disks growing in size by leaps, we are faced with the problem of locating files. Conventional file systems impose a hierarchical structure of storage on the user – a combination of its location and filename [2]. Features like 'symbolic links' allow a file to be accessed through more than one path. However, a strict enforcing of the path which does not necessarily depict the meaning of the file itself still exists for all files.

The world has come to realize that the need for efficient and effective file retrieval methods and thus the industry has responded with software like Google's Desktop Search[1], Apple's Spotlight[2], Beagle[3], Microsoft's proposed WinFS[4]. Locating a file on a large hard disk is tough unless we know exactly where the file is located.

A. Semantic Structure

The problem with the present heirarchial storage system is that the 'semantic' i.e. the meta data information of the file is not given adequate importance. The main semantic of the stored file is the directory in which it is stored in. To cite from *O. Gorter's* thesis [2], let us take a example of '`/home/user/docs/univ/project/file`'. Now, the property associated with the file is that of 'project' but not as much of univ or of documents. A listing of the `/home` directory does not list the file but a listing of the '`/home/user/docs/univ/project`' directory lists the file. The 'Database File System' [2], encounters this problem by listing recursively all the files within each directory. Thereby a funneling of the files is done by moving through sub-directories.

The objective of our project (henceforth reffered to as *SemFS*) is to give the user the choice between a traditional heirarchial mode of access and a query based mechanism which will be presented using virtual directories [1], [5]. These virtual directories do not exist on the disk as separate files but will be created in memory at runtime, as per the directory name. The name of the directory itself will be a query which the File System driver will parse and populate the virtual directory.

II. General Information

SemFS provides an intuitive way of browsing the file system. It lets one move within the file system based on the file's meta-data and attributes. The meta-data of files will not be common. All files will possess the attributes 'owner' and 'last modified date', but a JPEG file would also choose EXIF data like height and width as it's meta-data, while an MP3 file would choose id3 data like length, artist, album, etc as it's meta-data.

A. Features

SemFS provides:

- Searching as a basic function of the File System
- Browsing the File System based on the file's meta-data and attributes
- An easy and intuitive way to retrieve your required file (It is only natural that one remembers what the file is and not where the file is)
- Use of logical operators to filter results – ^ is AND, | is OR, ! is NOT
- An API to create 'views', which can be thought of as a persistent virtual directory. It will be updated automatically and the clients notified in case of any updates

B. Usage

A typical way of using SemFS would be:

1) Mount the File System using the driver
2) Chdir into the mounted directory
3) Do '`cd type:mp3^len>3m^artist:Mike`'

Dr. Arul Siromoney is with the College of Engineering, Guindy in India

[1] `http://desktop.google.com/`
[2] `http://www.apple.com/macosx/tiger/`
[3] `http://beagle-project.org/Main_Page`
[4] `http://msdn.microsoft.com/data/ref/winfs/`

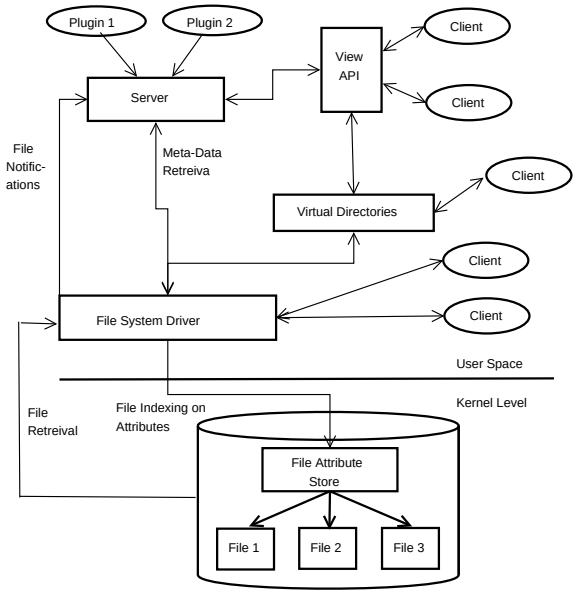

Fig. 1. Design of SemFS

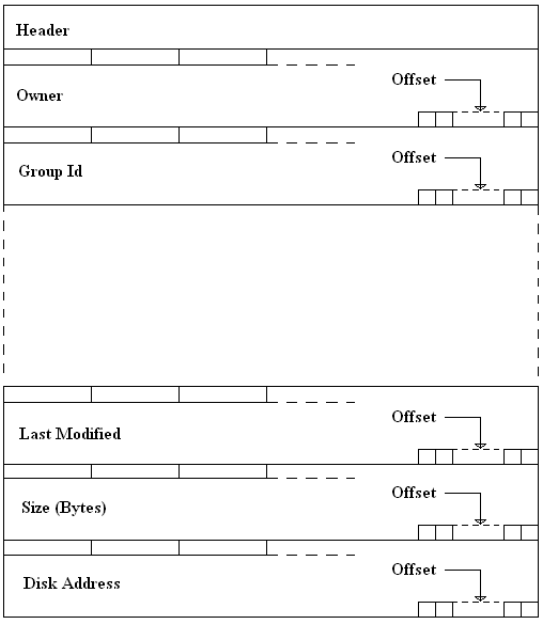

Fig. 2. Modified Inode Structure

4) The user is now chdired into a virtual directory which lists all MP3 files sung by 'Mike' of length greater than 3 minutes. (The queries will also support indexing based on file attributes like owner, file size, etc)

III. Design

The design of SemFS is meant to be easily 'extendable' or 'pluggable'. SemFS consists of 3 main components – The File System Driver, the SemFS daemon server and the SemFS API. SemFS works on the Server-Client architecture (Refer Figure 1).

A. File System Driver

SemFS will be a user space file system which can make use of the FUSE[5] or LUFS[6] libraries. We also considered the use of GNU/HURD translators [13], however, considering the wide use of FUSE and its active development, we decided to go ahead with FUSE. FUSE has bindings for a number of languages (including 4^{th} Generation languages like Python).

Apart from the user space daemon, SemFS also plans to change the storage of file attributes in the File System layout in order to optimize the indexing of files based on it's attributes (Refer §III-B.1). Hence, a kernel module will also be involved. However, the File System will also work without the kernel module, thereby keeping the project portable across most UNIX based Operating Systems. In the case that the kernel module is not installed, then the default ext3 based storage of files is used. In such a case, the indexing of file based on its attributes will not be optimized.

[5] http://fuse.sourceforge.net/
[6] http://directory.fsf.org/all/lufs.html

B. File Store

The files and directories itself will be stored in a partition or a special device file. The file system will support journaling (reusing ext3's data storage mechanism) and will be stored in a heirarchial fashion. Upon storing the file each time, it's meta-information is updated in the databases.

This meta information could either be stored in a in-kernel database [12], [14] (the KBDBFS project aims to maintain a Berkeley Data Base inside the kernel) or maintained in user space. We store some of the meta data in a user level database (a SQLite database should suffice). We store the File Attributes as usual in the inodes of the files. There will be no redundancy of file attributes, thereby doing away with a lot of race conditions. This will improve the look up time for such meta information.

1) File Attribute Storage: The internal representation of the file is given by an inode, which contains a description of the data layout of the file data. When a process refers to a file name, the kernel parses the file name – one component at a time, checks that the process has permission to search the directory or access the file, and eventually retrieves the inode for the file.

The problem in the current structure of the inode list is that when a query is executed wherein we index the files based on its attributes, we have to look into the irrelevant data of the inodes of all files. This naturally slows down querying since we would be wasting a lot of cycles due to unnecessary disk reads. Say, we want to query a set of files based on its file size, we would still need to read the other attributes like last modified date, time, etc although what we want to read is only the file size field. If however, we were able to read the size information of all the files in a single (or more) block access, the query speed would increase multi fold.

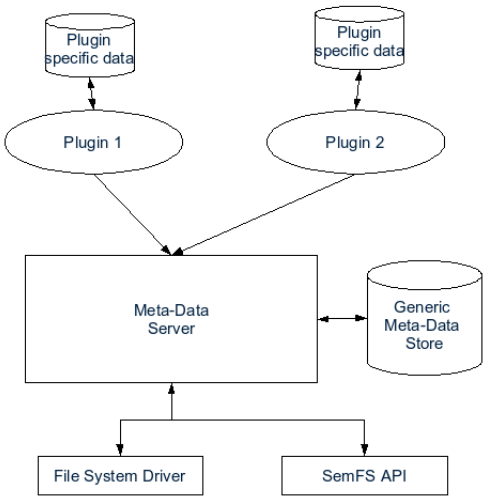

Fig. 3. Design of the SemFS Server

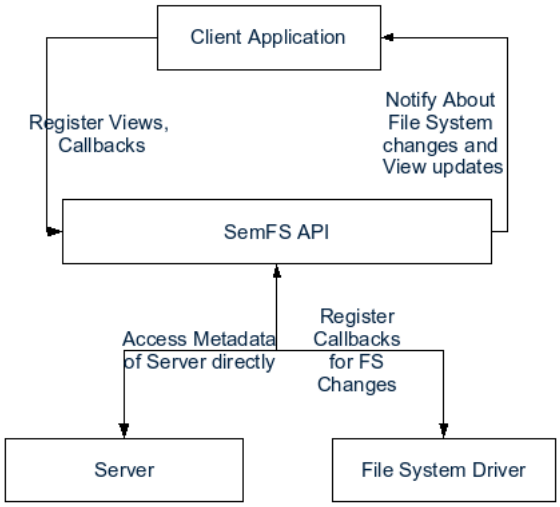

Fig. 4. Design of the SemFS API

In order to read the relevant data in fewer block access, the structure of an inode has been modified. Figure 2 shows the modified inode list. Searching based on a particular field of the inode is optimized since, the same field of all the files are grouped together in a minipage [15]. A minipage is a logical separation of similar attributes in the table. The rest of the file system structure including super block and directories remain the same as the UNIX file system layout. The advantages of using such a structure are:

- It maximizes inter-record spatial locality within each column in the page, thereby eliminating unnecessary requests to main memory without incurring space penalty
- Incurs a minimal record reconstruction cost
- It is orthogonal to other design decisions because it only affects the layout of data stored on a single page

The offset at the end of a minipage are actually binary bits denoting the availability of data in each record. If an item is present bit '1' will be present and '0' if it is deleted.

2) Data Storage: The existing storage mechanism of a typical UNIX file system like ext3 is reused. Most of the code base will be reused but for modifications where necessary, i.e. in places where inodes are referenced.

C. Server

The Server (Refer Figure 3) is the core of the Semantic File System. The server will translate the queries for the virtual directories into file listings. The server is also responsible for maintaining the 'Views'.

1) User Space Meta Data Store: As previously explained in §II-A, the meta-data information differs from file to file. The file specific meta-data is extracted and maintained in special databases by the Plugins for the server (Refer §III-C.2). We can store the meta-data in two ways:

- Inside the File System (Refer §III-B.1) – Within the inode List
- Outside the File System – In databases, which can be accessed by all applications

By storing the meta-data information outside of the file system, we will suffer a small performance hit. So, we will store the common file meta-data on disk in order to improve performance, while the file specific meta information will either be stored at the Server Database or databases maintained by the individual plug-ins.

2) Plugins: The SemFS daemon or server supports plugins which can define the logic for the query which is translated into the virtual directories. The plugins can also (usually) maintain their own database which will store the plugin specific meta-data. The work of the plugins include:

- Registering the plugins to the server
- Process and provide logic for the respective queries
- Register call backs for specific file modifications

D. SemFS API

The SemFS API will provide 'views' to the applications which are similar to Virtual Directories but they tend to be persistent in nature. The design of the SemFS API (Refer Figure 4) supports for:

- Support for Views
- Notify clients in case of view updates
- Shield the user from the database

E. Clients

Any application which makes use of the Semantic File System is a client. The use of the virtual directories is extended to all applications. In the case of the Database File System [2] and GLS Cube [5], the access to the meta data based searching is available only by recompiling the applications to use their custom APIs.

The SemFS API only makes the usability experience even better by providing features like automatic updating of 'Views'. All applications will still be able to access the Virtual Directories which offer a limited amount of Semantic information.

IV. Application of User Space Semantics

Hierarchical directory systems are very useful for organizing files, but they can only help to a certain point. SemFS provides the mechanism for easily accessing files based on its meta-data from all applications. Apart from these features, there can be certain extensible features that would highlight on the semantic structure storages.

A. File Tagging

Tagging is a feature that adds a user's logical perception about the file. Although tags do not necessarily define the semantics of the data, they are interpreted by the end-user as being related to a subset of his data, a subset that he logically creates. And contrary to the traditional directories, they are not monotonous. It also represents a relation between files. By this way the end-user can retrieve all documents related to a specific title.

The tags can be either added to a particular file or a group of files. The tag corresponding to a file are stored in the user space database along with SemFS server (which holds some of the file's meta-data).

B. File Versioning

Versioning is one of the feature that SemFS can support:

- Checkpoints and
- Versioning of files

Each time a file is modified, the server will be notified by the SemFS driver. The server checks if any of the plugins are waiting for the event that the file has been modified. If a call back exists, then the function corresponding to the plugin (i.e. the call back function) is initialized. This call back could retrieve the new file and store the diff between the two versions.

This can be used to restore the file to previous content after a certain period of time.

V. Related Work

There are a number of projects already working in similar areas. The need for creating a new architecture is to identify the problems in the current implementations and increase efficiency of retrieval.

A. GLS[3]

The GNU/Linux Semantic Storage System [5] is a solution designed to facilitate the management and retrieval of the data semantically. However, issues regarding consistency, stability and error-recovery exist. It does not offer any sort of error recovery. The inconsistency comes from the fact that GLScube is defined wholly in user-space, and thus, file system events may occur that GLScube does not record.

B. Leaftag

Leaftag[7] is a new project that facilitates the tagging of files. It does not use the file system's extended attributes. For example, when moving a tagged file, tagutils will index it again. This could potentially have side effects. It also lacks RDF features (the tag themselves cannot be nodes) and there is no way of expressing relations other than those in the tag.

C. Beaglefs

Beaglefs implements a file system representing a live Beagle query. The file system represents query hit results as symlinks to the targets. It provides constant time operation using extended attributes and supports many file operations.

References

[1] R. Pike, D. Presotto, and S. D. et al, "Plan 9 from bell labs," AT & T Bell Laboratories, Murray Hill, NJ, Tech. Rep., 1995.

[2] O. Gorter, "Database file system - an alternative to hierarchy based file systems," Master's thesis, University of Twente, August 2004.

[3] D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. W. O. Jr, "Semantic file systems," in *Proceedings of 13th ACM Symposium on Operating Systems Principles*. Association for Computing Machinery SIGOPS, Oct. 1991, pp. 16–25.

[4] Z. Xu, M. Karlsson, C. Tang, and C. Karamanolis, "Towards a semantic-aware file store," in *HotOS IX: The 9th Workshop on Hot Topics in Operating Systems*. USENIX Association, May 2003, pp. 145–150.

[5] A. Salama, A. Samih, A. Ramadan, and K. M. Yousef, *GNU/Linux Semantic Storage System*, 2006.

[6] D. P. Bovet and M. Cesati, *Understanding the Linux Kernel, Third Edition*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2005.

[7] N. Murphy, M. Tonkelowitz, and M. Vernal, "The design and implementation of the database file system," Nov. 2001.

[8] N. H. Gehani, H. V. Jagadish, and W. D. Roome, "Odefs: A file system interface to an object-oriented database," in *VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994, pp. 249–260.

[9] D. Mazieres, "A toolkit for user-level file systems," in *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2001, pp. 261–274.

[10] J. Nielsen. (1996, Feb.) The death of file systems. [Online]. Available: www.useit.com/papers/filedeath.html

[11] H. Reiser. (2001, Jan.) Name spaces as tools for integrating the operating system rather than as ends in themselves. [Online]. Available: www.namesys.com/whitepaper.html

[12] A. Kashyap, "File system extensibility and reliability using an in-kernel database," Master's thesis, Stony Brook University, Dec. 2004.

[13] Debian. (2006) Translators. [Online]. Available: www.debian.org/ports/hurd/hurd-doc-translator

[14] C. P. Wright, "Extending acid semantics to the file system via ptrace," in *Proceedings of FAST 05: 4th USENIX Conference on File and Storage Technologies*. USENIX Association, Dec. 2005.

[15] J. Zhou and K. A. Ross, "A multi-resolution block storage model for database design," in *Database Engineering and Applications Symposium*, July 2003, pp. 22 – 31, 16 – 18.

[16] Y. Padioleau and O. Ridoux, "A logic file system," in *Proceedings of FAST 03: 2nd USENIX Conference on File and Storage Technologies*. USENIX Association, Mar. 2003.

[7] http://www.chipx86.com/wiki/Leaftag