2010 IEEE International Conference on Data Mining Workshops

Knowledge File System

A principled approach to personal information management

Kuiyu Chang
School of Computer Engineering
Nanyang Technological University
Singapore 639798
e-mail: kuiyu.chang@pmail.ntu.edu.sg

I Wayan Tresna Perdana, Bramandia Ramadhana, Kailash Sethuraman, Truc Viet Le, Neha Chachra
School of Computer Engineering
Nanyang Technological University
Singapore 639798

***Abstract*— The Knowledge File System (KFS) is a smart virtual file system that sits between the operating system and the file system. Its primary functionality is to automatically organize files in a transparent and seamless manner so as to facilitate easy retrieval. Think of the KFS as a personal assistant, who can file every one of you documents into multiple appropriate folders, so that when it comes time for you to retrieve a file, you can easily find it among any of the folders that are likely to contain it. Technically, KFS analyzes each file and hard links (which are simply pointers to a physical file on POSIX file systems) it to multiple destination directories (categories). The actual classification can be based on a combination of file content analysis, file usage analysis, and manually configured rules. Since the KFS organizes files using the familiar file/folder metaphor, it enjoys 3 key advantages against desktop search based solutions such as Google's Desktop Search, namely 1) usability, 2) portability, and 3) compatibility. The KFS has been prototyped using the FUSE (Filesystem in USErspace) framework on Linux. Apache Lucene was used to provide traditional desktop search capability in the KFS. A machine learning text classifier was used as the KFS content classifier, complimenting the customizable rule-based KFS classification framework. Lastly, an embedded database is used to log all file access to support file-usage classification.**

virtual file system; search engine; personal information management; indexing; classification

I. INTRODUCTION

Today, people are increasingly relying on their computers or mobile phones to manage their life, typically storing gigabytes of data that include personal emails, messages, documents, contacts, presentation slides, audios, videos, etc. However, due to the tremendous growth in the number of personal files, manually organizing these assets using the 40-year old folder/file metaphor is practically impossible. It is not surprising that many users nowadays simply can't find their files [1].

This shows that the existing manual mechanisms for manipulating files and directories are way out of touch with the explosive information growth personally faced by today's computer users. Keep in mind that the classical file/directory manipulation mechanisms were leftovers from systems of the past, where a disk was only a few kilobytes large that stored at most hundreds of mainly homogeneous text files. Anything more complicated has been traditionally stored in a database. We are thus at a critical junction in history where new solutions to this problem must be investigated.

Manually browsing through numerous directories is probably the simplest but yet most frustrating task when it comes to searching for a file. Technically savvy users may opt to install a desktop search engine such as Google Desktop Search, which to some extent reduces the severity of the problem but, on the other hand, it also brings in another set of problems.

First, searching is possible only if one knows what one is looking for, and is typically applicable to text content. There are times when a user is looking for a particular file that may contain too many generic words shared by other files, which leads to the second problem of returning too many hits. Going through the returned list of search results may be as frustrating as browsing though a set of hierarchical candidate directories. In fact, there are no simple strategies to rank a corpus of text documents without ready-available link/relationship information between documents; Google's Page Rank [2] strategy is powerless here. Third, a search simply finds the file, but does not help user organize it properly so that next time when he needs the same file he could navigate straight to the location instead of using a search engine all over again. Another consequence is that the information is forever tied to the search engine, i.e., no search engine, no organized information. This leads to the fourth problem; the non-portability of retrieved files, i.e., the search engine simply retrieves files but do nothing to organize them. If the corpus of documents is copied to a USB disk and moved to another system, the same problem will persist on that system unless it also has a desktop search engine installed. At the end of the day, using a search engine to find files may take equal or more time as going through the various hierarchically organized directories manually.

This last point above is the primary motivation for KFS, to "place" a file in as many appropriate directories as possible so that during manual navigation though the file system, a user is likely to encounter a hit at the very first few locations that come to his mind. To illustrate this, suppose a set of hierarchical directories have been painstakingly created by a user in the file system, and each

978-0-7695-4257-7/10 $26.00 © 2010 IEEE
DOI 10.1109/ICDMW.2010.119

copy of some email is automatically placed into multiple appropriate directories by KFS as shown in the example of Figure 1. With the same email placed in four possible locations, his chance of finding this email (recall) is immediately increased 4 times.

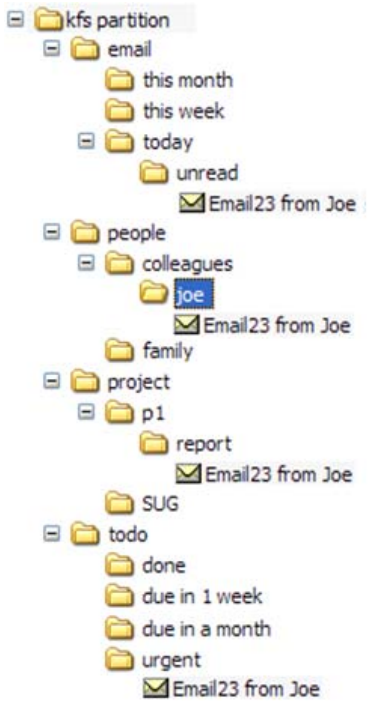

Figure 1. With "Email23 from Joe" automatically placed in 4 relevant folders, the user has a higher likelihood of locating this email later on.

II. Related Work

There exist many approaches to deal with the problem of organizing personal information. Here we review a few of the mainstream ideas.

Google Desktop Search [3] is a local desktop search engine made for various operating systems (Windows, Mac, and Linux). A similar desktop search tool offered by Apple is Spotlight [4], which works only under the OS X platform. Google Desktop Search and Apple Spotlight are two very efficient search engines that allow any file to be found fast and effectively. They support many types of files and their simple and intuitive interface is the main draw for most users. However, both of them are designed to return accurate search results without really organizing the files contained within. So all the limitations of the low level fix approach of search engines apply.

Gnome Storage [5] is an open-source effort to revolutionize the file system interface by storing everything in a relational database. It provides a virtual file system layer (GnomeVFS) for compatibility with existing applications. It is an ambitious effort that incorporates many advanced features like associations between objects, awareness of which application is opening an object, and revision tracking of objects.

Usenetfs is a stackable file system for large article directories [6]. It was developed to improve the file search on newsgroup servers. Similar to files on personal computers, files on servers typically reside in only a few commonly used directories over time, which makes searching inconvenient. Usenetfs changes the file structure by creating smaller directories containing fewer files, instead of the large existing flat file structures. This improves the processing rate of the articles on the server. Usenetfs is portable and imposes little overhead, thereby providing substantial performance improvements.

Microsoft has also put a lot of effort in this area. Microsoft's WinFS file system [7] is a commercial attempt to replace the file system with a relational database. Although well invested, it has been delayed many times since it was first announced in 2002. As of 2010, it has not been released commercially due to performance issues.

Another interesting research project carried out by Microsoft is MyLifeBits [8], which aims at recording everything in a person's life. With regard to computer use, every mouse click is recorded, every web page visited (not only the link) is stored, and every IM chat is logged. Moreover, a user wearing a sophisticated camera and GPS-tracking device as well as a set of sensors will have every event happening recorded and saved with time stamp and location information. In other words, MyLifeBits tries to create a "lifetime store of everything" [9]. While it seems very promising, the project is not yet available for public or commercial use.

A related but less ambitious approach is the "Stuff I've Seen" system by Dumais [10], which simply remembers all entities including files, web pages, emails, contacts, etc., that a user have come into contact on his computer. By revising this history, which is like a super-charged web browsing history, a user is able to find items guided mainly by his temporal recollection of the desired item..

MIT's haystack [11] is a different approach for information management. Haystack proposes a new concept of information management by organizing all types of information into one universal interface. Therefore, there is no need for separate applications to manage different types of data. By unifying the access to all types of information from within a single application, data from various applications can be easily associated and cross-referenced with one another. The main problem with this sort of half-revolutionary approach is that users have to rely on a single application, which might not be compatible with existing solutions.

NEPOMUK (Networked Environment for Personal Ontology-based Management of Unified Knowledge) [12] is a collaborative project between various European institutes (to name a few, German Research Center for AI, IBM Ireland, Thales SA – France, EDGE-IT – France, National University of Ireland, etc.). It ambitiously aims to turn the personal computer into a collaborative environment with state-of-the-art online collaboration and personal data management. The project seeks to augment the intellect of those involved by providing and organizing an enormous amount of information contributed by its

members from all over the world. In order to achieve this goal, the project introduces some new concepts and solutions, one of which is the Social Semantic Desktop. The Social Semantic Desktop is a new kind of desktop environment that enhances the traditional desktop by providing information with semantic meaning, thereby making information understandable by a computer. It also supports the interconnection and exchange of information with other desktops and users, i.e., it is social. Although the solution is original and praiseworthy, NEPOMUK's implementation is extremely complex and currently works only on the K Desktop Environment (KDE).

Rather than being a local personal information management of any sort, Freebase [13] is a vast public database that supports the categorization and sharing of information and knowledge. It attempts to create a global knowledge base that is structured, searchable, writable, and editable by a community of contributors. Freebase is free and open to anyone. It is powered by Metaweb, which allows enormous volumes of data to be collected, organized, connected, and modified. The data in Freebase are all linked together and can be collaboratively edited. Users of Freebase can contribute, structure, search, copy, and use data through either the Freebase.com web site or via the application program interface (API) for any commercial or non-commercial purposes. While the idea of building a public database for anyone to consume or contribute is laudable, it still has a long way to go before people can really benefit from this vast and organized source of information and knowledge.

The evolutionary approach adopted by KFS is different from all other approaches mentioned earlier. Its most related cousin is the GnomeVFS component of Gnome Storage. KFS makes changes at the file system level, not application level. Thus, the files are inherently organized on the storage media. The fact that KFS operates at file system level gives it more control over other tools that operate at the application level. This, however, does not restrict KFS to provide features commonly available in other desktop search tools. In fact, users can use existing desktop search or PIM tools on top of KFS. This kind of hybrid approach combines the powerful features of KFS with the features provided by any other PIMs or desktop search tools.

III. KNOWLEDGE FILE SYSTEM

The Knowledge File System (KFS) is a virtual file system that is designed to help users organize information using the familiar hierarchical file/directory metaphor. In other words, KFS offers a set of features that help alleviate the problem of manual information classification and retrieval. The main features of KFS include automatic classification of files, indexing, and logging of usage. Furthermore, KFS revamps the paradigm of hierarchical tree by explicitly allowing a file to reside in more than one directory. KFS is similar to Evolution's vFolders [14], but it is more flexible as it can classify any object that can be represented as a file.

KFS is designed as a stackable virtual file system (VFS) that works one layer above an underlying file system. The underlying file system does the actual work of storing and retrieving data. Being a virtual file system, we can view KFS as another file system like Ext4, except that it has many more special features. Typical VFS operations on the KFS like copy and move are simply passed on to the base file system. KFS comes with a bundle of features that are useful for users to organize their files. The major design features are described in this section.

A. Multiple Directories

A directory in the KFS partition can be considered as a category. A hierarchical directory structure mimics a hierarchical category or ontology. This is the dual of assigning multiple attributes to a file; we place it in multiple directories. In practice, a file can be semantically associated with more than one category, i.e., KFS allows a file to reside in one or more directory within a KFS partition without duplicating the data. This is a simple yet powerful mnemonic for information retrieval. For illustration, consider a video file of a 2006 basketball match between the San Antonio Spurs and Los Angeles Lakers, which could be logically placed into multiple subdirectories of a KFS partition, e.g., `/kfs/video`, `/kfs/sports`, `/kfs/nba`, `/kfs/lakers`, `/kfs/spurs`, `/kfs/2006/`. The multiple placements, facilitate effortless retrieval of the file at later times.

Clearly, multiple hard links can be created to achieve this, but hard links are difficult to manage and track. KFS provides a suite of user space tools to track and maintain such links via a central database. For instance, there are KFS tools that can list all hard links of a file, or delete a file including all its hard links at once. Internally, this is achieved by assigning a unique ID to every distinct file in KFS (which is similar to the inode concept in file systems). To facilitate efficient lookup, KFS stores the ID and filename association in an embedded database. To classify a file to a category, a user can simply link it under the directory/category. To de-classify a file from a category, user can delete the link from that directory. If a user would like to delete a file including all associated links from a KFS partition, a special tool called 'kfsremove' can be used. KFS will also physically delete a file from the KFS partition if there are no more links pointing to it, i.e., like dangling pointers.

B. Automatic Classification

KFS provides functions to perform automatic classification of files. Once a file is created, moved, or copied into a KFS partition, KFS automatically categorizes it into multiple pre-specified directories based on content similarity with respect to predefined classification ontology. This allows files to be automatically and logically organized in the partition, facilitating easy retrieval. Any suitable content or usage classifier can be used with the KFS. Out of the box, KFS comes with a built-in trained Support Vector Machines classifier from the libsvm library. The KFS can be reconfigured to use

other classifiers such as K-nearest neighbor or Naïve Bayes, but user must periodically retrain the classification models to ensure decent performance. Users can also write their own classification plug-ins and train a new classification model by supplying training documents for each category.

Currently, KFS supports automatic classification of text and HTML files via the content classifier and HTML KFS plug-ins. Support for other popular document formats like PDF and OpenOffice can be added by developing new KFS plug-ins. Similarly, classification of non-textual content such as images, sounds, can be added in the future. In addition to similarity based classification, users can create rule-based classification. Each rule is comprised of a set of regular expressions on the filename, location, and MIME type of the target file. A simple classification rule can be as follows: classify all PDF files with 'report' in its name to the 'report' directory. The following scenario illustrates the steps taken by Linux KFS to automatically classify a file:

1. File `sports.txt` is copied to `/kfs/data` directory.
2. KFS intercepts the copy operation and passes the file to the appropriate KFS plug-ins.
3. KFS creates a master copy of `sports.txt` in the backing store KFS directory, auto-renamed as some hashed valued filename `.store/0F0XA.txt`.
4. A hard link to `.store/0F0XA.txt` is created in the original user designated KFS target path `/kfs/data/sports.txt`.
5. KFS does the following in the background:

The KFS Classifier categorizes the file as a sports related file, and creates a hard link to `.store/0F0XA.txt` in `/kfs/category/sport/sports.txt`.

The KFS MIME Classifier determines that it is a text file and creates a hard link to `.store/0F0XA.txt` in `/kfs/.mime/text/sports.txt`.

C. Custom Classifier

The KFS classification framework allows user defined classifiers. Suppose a user added a classifier engine designed to classify a PDF document, then he can simply register the classifier with the KFS Classification framework. The user defined classifier will be called when a PDF file (as determined by the included Libmagic file type plug-in described below) needs classification. Before classifying a file, KFS must first determine its file type, whether it is plain text, HTML, image, binary, etc. This is important as to decide which classifier to invoke for classifying the file. Unlike Windows OS, there is no generally accepted concept of a file extension in POSIX systems. Moreover, file extensions are often unreliable for determining a file type. KFS employs the Libmagic library to guess the MIME type of a file.

D. Text Indexer

KFS comes with a text content indexer, which allows files to be searched by keywords. By default, KFS automatically indexes the entire KFS partition. Every file system change will be delegated to the KFS Indexing Engine for index updates. For example, when a file is copied to the KFS partition, the KFS Indexing Engine automatically indexes it. Likewise, the index is automatically updated whenever a file in the KFS partition is modified.

The KFS Indexing Engine is based on the CLucene [15] package, which is a high performance indexer implemented in C. KFS also supports incremental indexing, which means that files can be added and deleted without requiring a full re-index of all files in the system. In addition, the KFS Indexing Engine employs batch indexing to reduce the overall indexing time. KFS will index multiple files all at once instead of one by one. This is useful if user copies many files into the KFS partition at one go.

Before a file can be indexed, it must be pre-processed to create a pre-index document that contains the tokens to be indexed. For example, the initial processing stage will tokenize the data and get rid of some unimportant stopwords. The next step is to add the pre-index document to the index. The pre-processing step depends on the type of a file and its content language (for text). KFS provides a built-in mechanism (called a preprocessor) to pre-process a plain text file and produce a pre-index document that is ready to be indexed. A user-defined preprocessor can be attached to the KFS Indexing Framework to allow KFS to index other types of files. For example, users can create a preprocessor for PDF documents and register it with the KFS Indexing Framework. Then, whenever a PDF document needs indexing, the PDF preprocessor will be called to produce a pre-index text version of the file that is ready to be indexed. Once the pre-index text document is ready, the KFS Indexing Engine updates its index to include the new pre-index document. Using this modular framework, the KFS Indexing Engine can index any type of files provided the corresponding preprocessor is available.

E. Event Logger

Every operation on a KFS partition is monitored. Operations such as copying, moving, updating, and deleting of files are logged to the embedded database. Users can simply browse through the database records to retrieve the log information. Such information can be very useful for usage mining and auditing.

Related to the Event Logger, KFS allows users to register an external program (shared object library in practice) to be called whenever a specific KFS operation is logged. This comes in handy for situations where we wish to be notified whenever someone reads or writes to a common KFS sub-directory.

For example, suppose a user would like to automatically test and grade every programming assignment added to the `/submissions` directory. To do this, he can create a simple script to compile the submitted source files and run it with some predefined test cases and assign marks based on the testing results. After

that, the user can simply register it with the KFS Logging Framework.

F. Customizability of KFS

The KFS Indexer, KFS Classifier, and KFS Logger are all customizable. KFS provides a configuration file for each of the framework (namely Classification, Indexing, and Logging). On top of that, there is a master configuration file where the overall behavior of KFS is configured.

The classification configuration file lists the mappings between a set of rules to the corresponding classifier program. The set of rules comprises the filename, location, and MIME type of the file to be classified. Regular expressions can be used to refine each set of rules, thereby providing the user with a detailed level of customization of the KFS Classification Framework. For example, a user can create a rule targeting any file beginning with '`alaska`', whose MIME type is html, and which is located within `/kfs/website/pages` to be classified by a user defined classifier.

The Logging configuration file provides a similar function as the Classifier configuration file except that it defines mappings between a set of rules and external programs to be invoked when the corresponding rule is satisfied. The set of rules comprises the filename (there could be two files involved), location, MIME type of the file as well as the operation type. For instance, a user could log every link creation operation that points to any image file residing in `/kfs/data`.

The Indexer configuration file defines mappings between a set of rules and the corresponding preprocessor library to be called. The rule includes the filename, location, and MIME type. Users can specify which preprocessor library to be used for processing a particular type of file. For example, users can configure KFS to preprocess all Spanish HTML document prior to indexing.

G. Extensible Architecture

As described earlier, KFS can be easily extended by user-made plug-ins, which just needs to be attached to one of the KFS frameworks. Specifically, the KFS Classification Framework allows a user to add a new Classifier. This enables KFS to practically classify any type of file. The KFS Indexing Framework allows the user to add a new preprocessor, which enables KFS to index any type of file in any language. Lastly, the KFS Logging Framework allows user to monitor all accesses to the KFS partition by attaching a monitoring daemon.

IV. KFS VERSUS OTHER APPROACHES

Table I compares KFS with existing desktop search solutions. Clearly, neither Google Desktop nor Beagle supports file system organization via hard link management. This is expected as both of them work at the application level. On the other hand, both Google Desktop and Beagle clearly support a larger number of file types for indexing. Being extensible, additional KFS plug-ins can be written to increase the indexing capability of KFS. As for the automatic classification of files, only KFS provides this feature integrated with hard link management.

TABLE I. KFS VERSUS TWO DESKTOP SEARCH SOLUTIONS.

Features	KFS	Google Desktop	Beagle
File system organization	Automatic	No	No
Text Classification	English	No	No
Indexing	Text & HTML	Various	Various
Extensible	Yes	Yes	Yes

TABLE II. KFS VERSUS OTHER FILE SYSTEMS.

	KFS	UsenetFS	GnomeVFS
Virtual file system	Yes	Yes	Yes
Learning Curve	Low	Low	Low
Link management	Yes	-	Yes
Text Classification	English	No	No
Indexing	Text & HTML	Numbering based	Text, PDF, etc
Extensible	Yes	Yes	Yes
Maintains data consistency	Yes	Yes	Yes
Log user changes	Yes	No	Yes
Scalability	High	High	Moderate
File Browsing	Yes	flat	Yes
Stackable	Yes	Yes	No

TABLE III. KFS VERSUS TWO RADICAL APPROACHES.

	KFS	NEPOMUK	Freebase
Learning curve	Moderate	High	High
Application Compatibility	Yes	KDE only	Web & API
Portability	Yes	No	Yes
File System Compatibility	Yes	N/A	N/A
Personalizable	Yes	N/A	Yes
Requires Internet connection	No	Yes	Yes
Free	Yes	Yes	Yes
Open-source	Yes	Yes	Yes

Table II compares KFS with other file system approaches. According to Table II, all three systems are extensible meaning that users can add more functionality. Both KFS and Gnome Storage support indexing with Gnome Storage having a better support for file types. KFS is different from Gnome Storage both in philosophy and goals. Gnome Storage aims to be the revolutionary file system with lots of advanced features like arbitrary graph based associations between objects, and is based on a relational database. KFS, on the other hand, aims at maximizing the utility of the familiar file/directory hierarchical metaphor by providing tools to automatically manage hard links. The ultimate goal of KFS is to make the current file system metaphor manageable to end users without having them learn a whole new paradigm of

information management. We feel intuitively that the KFS is more acceptable to users of today, and should be much easier to use in the near term. Whether the KFS and file/directory metaphor will be viable in the long term remains to be seen.

Finally, Table III gives a rough comparison between KFS, NEPOMUK and Freebase (as mentioned earlier). Although they are of very different approaches, they all share the same goals and philosophy and are therefore worth comparing.

V. KFS FOR LINUX

KFS for Linux is implemented as 2 main parts that executes as two separate processes, KFS-FUSE as the virtual file system and KFS daemon, which contains the logger, indexer, and classifier. This approach follows the general rule of thumb to separate system (kernel) related part, which is the file system, and implement the remaining functionality in the other part (user space). The reasons to split the implementation are summarized as follows:

- Performance – the virtual file system component is very critical to the overall system performance because it relates to the operating system kernel. A clear separation will enable us to optimize the performance and minimize performance hit to the overall system. This still enables us to perform heavy processing such as classification in user space.
- Easy to debug and maintain – kernel programming and debugging is much more difficult than user space programming and debugging. The interface to the kernel is already provided by FUSE framework. Thus, we only need to take care of the file system implementation that resides in user space.
- More library support – by developing part of KFS in user space grants it access to a vast library of software unavailable in the kernel context. This greatly simplifies the development of KFS.
- Easily extensible – our approach allows third parties to easily write plug-ins for KFS in user space instead of the more complex kernel space.

A. KFS – FUSE

KFS is not a standalone file system, but instead a stackable file system. It sits on top of another underlying file system and utilizes it to store the actual data. This approach greatly reduces the amount of work needed to develop KFS as developing a high quality file system is a very complex process requiring many man years. Furthermore, this allows KFS to reap full benefits from years of work done in creating an efficient file system.

KFS resides between the user program and the underlying file system. This enables KFS to intercept any relevant file system calls and perform some tasks before or after delegating the call to the underlying file system.

For example, in the event that a file `data.txt` is updated in a KFS partition, the following string of events is expected:

- KFS FUSE is notified that a file is updated.
- KFS FUSE delegates the actual work to the underlying file system.
- KFS FUSE sends a message to user-space KFS that `data.txt` is being updated.

The KFS FUSE module is developed based on the FUSE (File System in User Space) [16] module. FUSE allows us to create our own file systems without editing the kernel code. This is achieved by running the file system implementation code in user space. FUSE provides a bridge to the actual kernel interfaces. FUSE was officially merged into the mainstream Linux kernel tree starting from version 2.6.14.

The KFS FUSE module resides both in kernel space and user space. It contains secure and efficient modules that reside in the kernel, as well as high performance communication interface between kernel and user space, and a set of APIs that can be used according to our needs. FUSE has been widely used to create file system drivers and interfaces. The FUSE kernel module and the FUSE library communicates via a special file descriptor that is obtained by opening `/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 file system.

For most of file operations such as move, delete, create directory, and so on, there is an equivalent FUSE API call that will be executed when such an operation is performed. Thus KFS can intercept the operation by simply intercepting the corresponding API call.

KFS FUSE supports multiple virtual file system, which means that the same module can be used to mount different partitions at the same time, and each virtual file system runs as an independent process. Figure 2 shows the KFS FUSE design on top of an Ext3 file system.

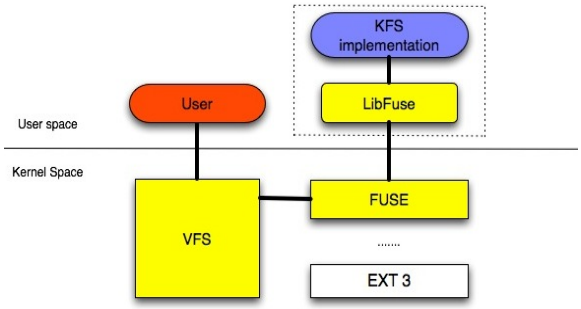

Figure 2. KFS FUSE.

B. User-space KFS

The user space component of KFS is responsible for the computationally intensive tasks including maintaining the KFS partition index, classifying files, updating of the KFS metadata database, and logging of user activities.

Being a complex framework, KFS does not rely on itself to perform all of its functionalities. Instead, it makes use of a number of existing technologies to build a complete and powerful file system. This minimizes the amount of code to be rewritten and avoids reinventing the wheel. For example, KFS relies on the CLucene framework for indexing and the libsvm for content classification. KFS also utilizes the Berkeley DB [17] embedded database. Berkeley DB was chosen because it is a lightweight database that runs in the same process as the calling application, with no context switching overhead.

The user space KFS daemon constantly listens to a predefined port for user space commands. This allows other process to communicate and alter the behaviors of KFS during runtime. For example, a user might wish to pause file indexing during peak periods.

The user space KFS is designed with scalability and extensibility in mind; it allows third party plug-ins to be attached to the system on the fly. In addition, some performance enhancement techniques such as batch indexing are also employed.

C. *Plug-ins and User-space Tools*

A number of user space tools were developed to fully utilize the KFS features are listed as follows.

- Mount, unmount, create, and delete KFS partition (multiple mounted KFS partitions possible).
- Search the Index for a keyword.
- Search for certain characters in filenames.
- Find all files that link to the same data.
- Delete a file and all of its references.
- Display usage log data.
- Pause or resume indexing.

In addition, a simple plug-in to strip HTML tags has been developed and attached to the KFS, which allows KFS to classify HTML files. For ordinary users, the set of command line tools provided above might be difficult to use, in which case they could use the included GUI java program to manage KFS partitions.

VI. PERFORMANCE EVALUATION OF KFS

Although the KFS is designed to perform classification and indexing of files in batch mode, we want to evaluate how this will impact the actual use, i.e., whether it will cause a noticeable delay to the user. Experiments were simulated on an Intel Pentium 4 Processor 3.40 GHz desktop with 2048 MB of memory running Ubuntu Linux 7.10 [18] operating system with Linux kernel 2.6.22-14 and the following libraries:

- FUSE 2.7.0
- CLucene 0.9.20
- Libsvm2
- C++ Berkeley DB 4.3.28
- Boost C++ Library 1.34.1

Various quantities (100 to 1000) of text files, each of average size 2.6 KB were copied to a KFS partition. The time taken to copy as well as to classify and index the files was measured using the Linux time command. Measurements are also taken for other file systems and tools. GnomeVFS-copy is measured using a simple looping script to copy the files one by one.

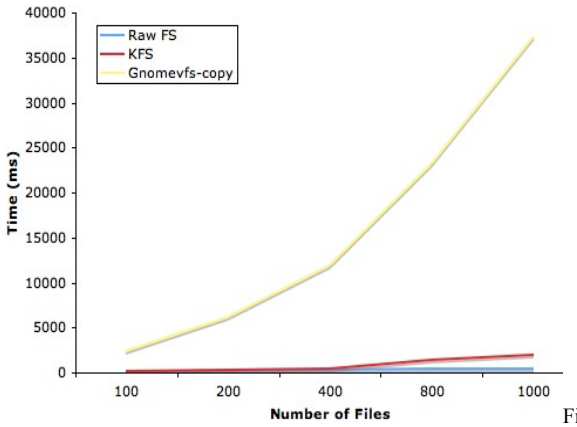

Figure 3. Time taken versus number of files for batch file copy.

From the results in Figure 3, the baseline raw file system performance is clearly the fastest among the three. Gnomevfs-copy was chosen because it implements the Gnome Virtual File System library [19]. Gnomevfs-copy is significantly slower than KFS. This is probably due to it been called by a looping script, since there is no way to specify a list of files for gnomevfs-copy.

KFS performs only marginally slower than the raw file system. The separation between the file system module and indexing/classification module enables the user to finish the file-copying process while KFS continue to index/classify in the background.

A. *Sample Classification*

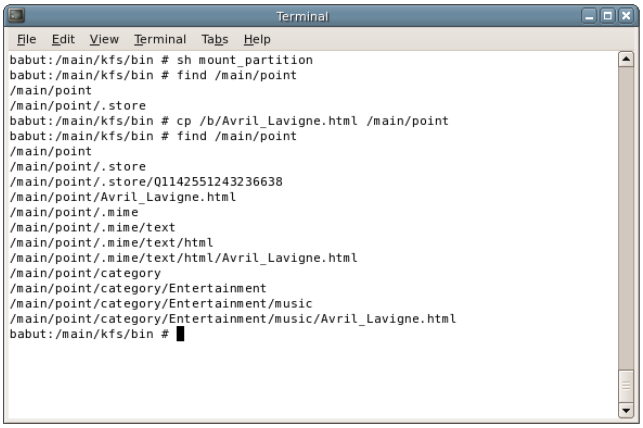

Figure 4. Sample KFS classification in console.

Figure 4 demonstrates the classification feature of KFS. When a file is copied to a KFS Partition, KFS automatically classifies the file and create hard links in the appropriate locations based on the classification result. In this example, the mounted KFS partition is `/main/point`, and `/main/point/.store` is where

the raw data file is located. We see that initially there is nothing in the KFS partition. However, after the file `Avril Lavigne.html` is copied to `/main/point`, it is indexed and classified by KFS, which results in the creation of two hard links to this new file, one in `.mime/text/html`, and the other in `category/Entertainment/music`.

VII. Conclusions

Currently, Linux KFS is still in its infancy. Its features are limited but could be greatly enhanced with its open and extensible architecture. Although we have only written a few plug-ins to improve the basic functionality of Linux KFS, in its current form Linux KFS could potentially be used to classify emails as individual files using the Mail Directory format of Courier's IMAP email system. The auto classification feature currently only works with English text and HTML files. This can be enhanced by providing classifier plug-ins that support non-English text content. In addition, an advanced translator plug-in is needed if we want to classify or index non-text files such as PDF or OpenOffice documents. Likewise, the indexing feature currently supports plain text only. This could be improved by writing a preprocessor library to preprocess a variety of non-text files. Linux KFS can help user organize their files without taking away any flexibility and life can go on as normal; as a file system, Linux KFS can seamlessly integrate with the vast majority of existing applications. Being extensible, Linux KFS can be built into a very sophisticated file system for managing personal information.

So far, we have not systematically evaluated user responses to the KFS. For future work, we need to test the KFS extensively on common day-to-day tasks like organizing emails and pictures, to truly measure its effectiveness in improving a user's recall when it comes to finding his file. Moreover, extensive end user testing is also very important to discover limitations and areas of improvement for the KFS approach to solving the information explosion phenomenon. We believe that the advent of huge portable storage in mobile wireless devices will make the KFS even more relevant as a unified paradigm to organize and retrieve information on the mobile devices.

Acknowledgements

This research was funded in part by NTU Startup Grant CE-SUG 11/03 and Singapore Ministry of Education's Academic Research Fund Tier 1 RG 30/09.

References

[1] C. A. N. Soules, G. R. Ganger, "Why can't I find my files? New methods for automating attribute assignment", Technical Report CMU-CS-03-116, School of Computer Science, Carnegie Mellon University, 2003.

[2] S. Brin, L. Page, "The Anatomy of a Large-Scale Web Search Engine", Proceedings of WWW, 1998.

[3] Google Desktop Search, http://desktop.google.com/en/, 2010.

[4] Apple Spotlight, www.apple.com/macosx/features/spotlight/, 2010.

[5] S. Nickell, "A Cognitive Defense of Associative Interfaces for Object Reference", Draft, http://people.gnome.org/~seth/storage/associative-interfaces.pdf, 2010.

[6] E. Zadok, I. Badulescu, "Usenetfs: A Stackable File System for Large Article Directories", Technical Report CUCS-022-98, Computer Science Department, Columbia University, 1998.

[7] R. Grimes, "Code Name WinFS: Revolutionary File Storage System Lets Users Search and Manage Files Based on Content", MSDN Magazine, Jan 2004.

[8] J. Gemmell, G. Bell, R. Lueder, "MyLifeBits: a personal database for everything", Communications of the ACM, vol. 49, Issue 1, pp. 88-95, Jan 2006.

[9] S. Cherry, "Total Recall", IEEE Spectrum, pp. 18-24, Nov 2000.

[10] Dumais, S.T., Cutrell, E., Cadiz, J., Jancke, G., Sarin, R., Robbins, D.C., "Stuff I've Seen: A system for personal information retrieval and re-use", ACM SIGIR, 2003.

[11] D.R. Karger, K. Bakshi, D. Huynh, D. Quan, V. Sinha, "Haystack: A customizable general-purpose information management tool for end users of semistructured data", in Proc. CIDR, 2005.

[12] N. Papailiou, C. Christidis, D. Apostolou, G. Mentzas, R. Gudjonsdottir, "Personal and Group Knowledge Management with the Social Semantic Desktop", in Proc. Collaboration and the Knowledge Economy: Issues, Applications and Case Studies, 2008.

[13] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, J. Taylor "Freebase: a collaboratively created graph database for structuring human knowledge", Proc. of SIGMOD, pp. 1247-1250, 2008.

[14] Gnome Evolution Project, http://projects.gnome.org/evolution

[15] CLucene: port of the popular Apache Lucene search engine, http://clucene.sourceforge.net/

[16] E. Zadok, I. Badulescu, "A Stackable File System Interface for Linux", in Proc. Linux Expo Conference, Raleigh, NC, pp. 141-151, May 1999.

[17] BerkeleyDB, http://www.sleepycat.com/

[18] Ubuntu Linux, http://www.ubuntu.com/

[19] Gnome File System library, http://library.gnome.org/devel/gnome-vfs-2.0/