Design Report for ErdosFS

March 21, 2014

Harrison Hunter
hhunter@mit.edu
Recitation Instructor: Katrina LaCurts

1 INTRODUCTION

This proposal describes the design of a new folders system for UNIX named ErdosFS. ErdosFS implements the ability for users to create special directories called smart folders. Smart folders are directories of symbolic links to all of the files that match the a given search query. A smart folder maintains all current matches to the given search query with low performance overhead by using the inotify system and a database to dynamically update the contents of each folder. This replaces the need to run the slow find operation every time one wants the up-to-date results of a certain search.

ErdosFS is space-efficient and provides all the functionality of the Unix File System (UFS) without significantly affecting overall performance. Since the folders are implemented as regular directories, the interface is familiar to the user, easy to use, and supports all regular UNIX operations.

2 DESIGN DESCRIPTION

To provide the functionality explained in the introduction and explained formally in 2.1, the smart folder system extends UFS using two main tools, inotify and persistent key-value stores which we will refer to as databases. The inotify system is used to alert the system of changes so that ErdosFS can update all the smart folders. The database system ensures that ErdosFS is able to update itself quickly and does not slow down the entire file system. ErdosFS has initialization, maintenance, and deletion methods to provide the desired functionality. ErdosFS exposes a set of OS methods (API) that allow the Unix file system and the end user to gain all of the benefits of smart folders.

2.1 FUNCTIONALITY

2.1.1 Smart Folder Contents

Smart folders contain symlinks to any regular file in the UFS.

2.1.2 Command Line Tool

ErdosFS can be accessed by the following command line inputs.

Create Operation

searchmount mountpoint searchpath [expression]

This creates a smart folder at the mount point with symlinks to all the files in searchpath that match the expressionm defined in section 2.1.3.

Delete Operation

searchmount -u mountpoint

This removes the smart folder at the location specified in mountpoint.

2.1.3 Expression

Expression is the specification of which files from searchpath to include in the smart folder. Expression is translated directly into a normal UNIX find expression. The results of this find are then piped into a new directory at mountpoint in the create operation.

2.1.4 Expression Definition

The smart folder supports the following filter types in the expression:

- File name string and substring matching
- File size ($<, >$, or $=$ given value)
- Time of file's creation, last modification, or last use (relative to given time)
- File's user or group ID ($=$ or $\neq$ given value)
- File's permission bits match a specified value exactly, or at least one of the specified bits are set, or all of the specified bits are set
- Boolean combinations of the above.

2.1.5 Expression Example

Example expression: -name example
This matches all files named example. This will be translated to the following find command:
find searchpath -name example -type f

2.2 Inotify

ErdosFS uses the inotify system to monitor system events to maintain the correct contents in each folder. The inotify system is applied to the entire file system as explained in 2.4. Inotify events are written to a buffer. The buffer is an in-memory queue that holds the file name of every file that triggers an inotify. The buffer is continuously read by the dynamic update system defined in 2.5 to maintain the correctness of the contents of each folder.

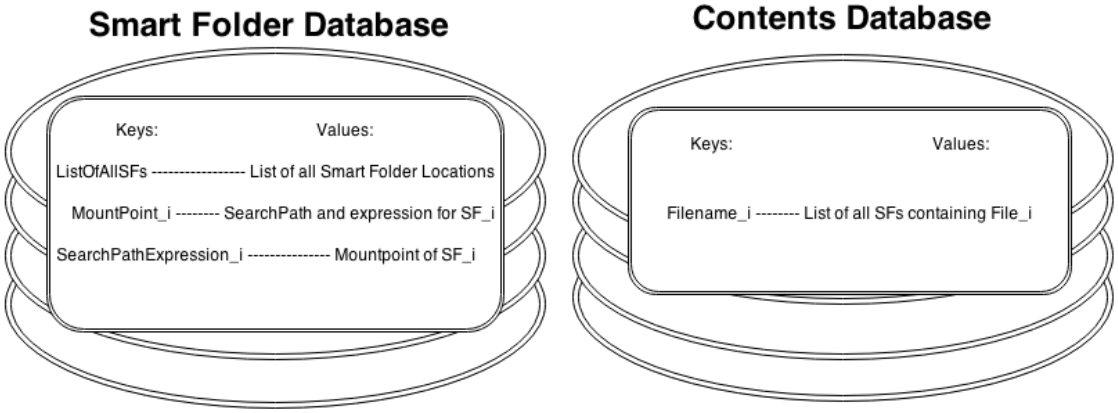

Figure 1: Figure of the types of key-value pairs in each database. Here SF_i or File$_i$ refer to any smart folder or file in the system, i.e. SF$_i$ means there is one key-value pair for every smart folder.

2.3 Database

ErdosFS uses databases to speed the update process ensuring the correctness of the contents of each folder. Each database is a persistent key-value store with put, get, and remove operations.

2.3.1 Smart Folder Database

The smart folder database contains a list of all smart folders. This is a single key-value pair defined by (key=listOfAllSFs, value=[array containing all smartfolder mountpoints]). It also contains a bijective mapping between a smart folder's searchpath-expression pair and mountpoint. A mountpoint uniquely defines a smart folder because the mountpoint is the full path to the smart folder, and only one folder with a given path can exist. The expression-searchpath pair defines the contents of a search folder. The bijective mapping is achieved by having both (key=mountpoint, value=searchpathexpression) and (key=searchpathexpression, value=mountpoint) key-value pairs in the database.

2.3.2 Contents Database

The contents database includes all the files that are contained in a smart folder as keys. The value for each folder is the smart folders that contain that folder. Thus each key-value pair is defined as (key=filename, value=[array of all smart folders containing the file]). Storing the smart folders that contain each file may seem redundant because the system

already stores the contents of each smart folder on disk, but, as discussed in the design and analysis sections, this enables updates to be completed in m steps, where m is the number of smart folders, instead of m*n steps, where n is the number of files in the smart folder system. Analysis found that memory usage with this system was sufficiently small, and that the speed up of not completing m*n steps in updating each file justified the redundant storage.

2.4 Initialization

2.4.1 System Initialization

The smart folder and contents databases are initialized once with the contents defined in 2.3. The inotify system is applied to the root directory and every subdirectory, thereby encompassing the entire file system. This inotify system is initialized in the background every time the system boots. The inotify system usage is described more fully in section 2.5.

2.4.2 Individual Folder Initialization

Individual smart folders are initialized by the create method defined in 2.1.2.

To create a smart folder for a given expression at mountpoint, the system makes a new directory at mountpoint via the normal Unix file system functionality mkdir. It the finds and creates symlinks in our new folder to all files in searchpoint that match the expression and updates or creates their entry in the database to include the current smartfolder. Finally, it adds the smart folder to the root database as defined above.

2.5 Dynamic Update System

To maintain the correctness of all the smart folders ErdosFS employs an inotify system. As defined in the initialization, all subdirectories in the root directory as well as the root directory itself are in the watch list. When any file system events occur, they are added to our buffer, which our smartfolder dynamic update application reads from continuously.

Upon reading an event from the buffer, ErdosFS gets the list of all smart folders that contain that file from the database and checks that the file still meets the criteria defined by the search path and expression for each folder and that the file path is still the same (i.e. that the symlink is still valid), if not, it removes the symlink from that folder and removes that folder from the files list of matching folders. Then ErdosFS gets the list of all other smart folders from the database and checks if the newly updated file meets the criteria defined. If so, then it adds a symlink to that file to the folder and adds that folder to the files list of matched folders in the database. If not, it is ignored.

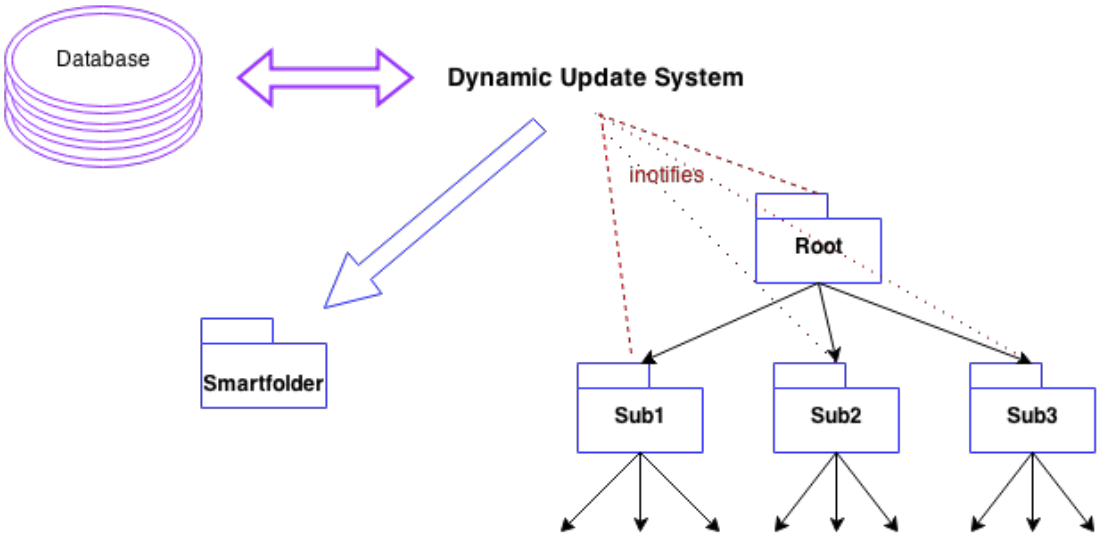

Figure 2: Visual representation of smart folder dynamic update system. Files and folders communicate with the dynamic update system via inotfies, and the dynamic update system reads and changes the database and updates the smart folders.

2.6 Deletion

A smart folder can be deleted via the API detailed in 2.1.2.

This command is executed by finding all the symlinks in the smart folder for each file, removing the smart folder from the database list of smart folders containing the file, deleting the folder from the master list of smart folders, and removing the folder and all symlinks from the disk.

2.7 ErdosFS OS methods

ErdosFS exposes all its functionality to Unix and the user via the following operating system methods.

2.7.1 createDirectory

createDirectory(name, searchPath, expression)

This method can call into the operating system to create a smart folder that exposes the four methods described below to search its contents. This method is implemented by the UFS mkdir command, because the system uses the standard UFS commands to create smart folders.

2.7.2 getFirstDirectoryEntry

DirEntryPointer getFirstDirectoryEntry(name)

Called by the operating system when it wants to begin iterating over the files in the smart folder. The smart folder system returns a pointer to the first entry in the directory with the given name, or null if there are no files in the directory. This method, along with methods 2.7.3 and 2.7.4 are implemented by UFS because the smart folder is implemented as a regular unix folder.

2.7.3 readNextDirectoryEntry

DirEntryPointer readNextDirectoryEntry(DirEntryPointer previousFile)

The operating system uses readNextDirectoryEntry to iterate through the files in a directory. ErdosFS returns a pointer to the next directory entry after previousFile in the directory containing said file, or null if there are no more files in the directory.

2.7.4 fileName

String fileName(DirEntryPointer entry)

Called when the operating system wants to get the name to display for a file listed with getFirstDirectoryEntry or readNextDirectoryEntry. ErdosFS returns the name of the file represented by directory entry entry.

2.7.5 readSymbolicLink

String readSymbolicLink(DirEntryPointer entry)

Called when the operating system wants to get the path of the file that the directory entry points to. ErdosFS returns the target of a symbolic link stored in the directory entry pointed to by entry. This method is implemented by the regular Unix functionality of symbolic links because the symbolic links are stored as regular Unix symbolic links.

3 ANALYSIS

3.1 Space

ErdosFS stores information in memory via the database and on disk via the folders themselves.

3.1.1 In Memory Space

ErdosFS stores n+3m items in memory where n is the total number of files stored in smart folder and m is the number of smart folders. We store n items because the system

User	Files	Memory Used	Notes
Personal laptop	1.5M	162MB	Stored every file in file system (not including afs mounts) in 24 different smart folders of varying size
Home computer	.8M	96MB	Stored every file in file system in 20 different smart folders
Work Computer	1.3M	170MB	Stored every file in file system (not including afs and block storage mounts) using 12 different smart folders

Figure 3: Experimental results of real space usage of storing all the files on 3 different computers in an in-memory data store. These tests were implemented using Redis, an in-memory persistent key-value store, as a proxy for our database defined in 2.3.

stores the location of each file that is in a smart folder. We store 3m items because the system stores a list of all smart folders, and then the bijective mapping of the location of the smart folder with its descriptor (searchmount and expression). The size of each item is at most 256 bytes because Unix file names are limited to 256 bytes, but in reality filenames are often much shorter. In a sampling of 10 potential users(5 MIT students, 2 non software industry adults, 2 software industry adults, and 1 high schooler) the average number of files on a users computer was 1.1 million files. If a user creates 10 smart folders that each independently encompass a random 10% of the users computer's total number of files ErdosFS would take up at worst (1,100,000+3*10)*256*c_1 + c_2 bytes on disk, where the constants are the overhead of maintaining the database.

In reality filenames are much shorter and people are unlikely to have a number of files on the order of the entire hard drive be stored in ErdosFS. For a typical use case, people will likely be storing tens to hundreds of files in a folder and have tens to hundreds of smartfolders. In this case the in memory storage is on the order of single Mbs which is very reasonable considering all 10 people surveyed had more than 4 GB RAM. Even in the case of storing the entire file system in ErdosFS, the memory usage is feasible.

3.1.2 Disk Space

ErdosFS stores n+m items on disk, where n is the number of files in the smart folder system and m is the number of smart folders. The n items are symlinks, which take up the same amount of space as the file name that they point to. Therefore our disk usage is O(in-memory disk usage) which we showed to be sufficiently small. In addition, disk storage is abundant on modern computers, and the addition of several or even hundreds

of megabytes of additional data is not a concern to the system or user. Operating systems typically consume gigabytes of data, making the addition of ErdosFS largely irrelevant.

3.2 Time and processor time

3.2.1 Folder Creation

Folder creation uses the unix find command and run time is dominated by the find. Simple benchmarks showed that find operations can take 1-5 minutes when run on the root directory. An alternative design that avoids this bottleneck consists of indexing the entire file system so any arbitrary find runs faster. Analysis of this proposal showed this took a prohibitive amount of space and processor overhead to maintain.

The find bottleneck is permissible for ErdosFS because running a find is unavoidable in practice, in line with user expectations (the user isn't expecting ErdosFS to improve the speed of its default Unix operations) and performs much better for most use cases. Finds on smaller search directory performed much better, frequently at or below a few seconds which is permissible for ErdosFS.

Benchmarking database methods with Redis (described above) showed that $\tilde{3}$50k inserts/second could be completed, meaning that almost any smart folder operation can be done in less that one second after the initial find, and in the worst case - a smart folder was created containing every file on a sample user's hard drive - the database part of the creation process would still run in less than 5 seconds. Overall performance is sufficiently fast because most smart folders will have a small fraction of this many files.

3.2.2 Folder Maintenance

Folder maintenance consists of triggering the inotify events, reading from and processing the buffer, and updating the database. Maintenance takes at worst m operations per file changed. These operations are run on a thread in the background and thereby do not block for the user. They do take processor time and will be continually running in the background if the user is continuously updating files. If the buffer is empty, the thread will suspend activity and stop taking processor time and simply poll the buffer until new events are written. Typical computer use cases do not change many files per second, so the processor usage will likely be negligible and have an imperceivable effect on the user.

3.2.3 Folder Access

Smart folder access performs identically to normal Unix folder access because smart folders are stored as regular folders of symlinks on disk.

Use Case	Space	Time	Notes
User creates smart folder in small directory n = 300	~300*p Items in Memory, ~300*p Items on disk	Find operation (<1 second) + 1 disk write of 300*p items (low ms) + ~300*p DB puts (low ms)	Low space (items max 256 bytes, expected ~30 bytes), time dominated by find operation, UX feels similar to regular search
User creates smart folder in root directory n = 10e5	10e5*p items in memory, 10e5*p items on disk	Find operation (low minutes) + 1 disk write of 10e5*p items(ms) + 10e5*p DB puts (low seconds)	Time dominated by find operation, space is approx 2 * 10e5*p items, but only 10e5*p items of expected size 30 bytes in memory, UX feels similar to regular search
Users mounting search folders automatically at startup or login	N/A	N/A	The advantage of using real directories over other models is that this case is automatically handled because search folders do not need to be recreated, only the inotify system needs to be redeployed upon startup

Figure 4: Analysis of common and limiting uses of ErdosFS where p is the percent of items that match the search criteria

3.3 User experience

ErdosFS achieves its goal of providing functionality to the end user without compromising their overall computing experience. RAM usage is sufficiently low, disk usage is negligible, and smart folders perform similarly to any other folder on disk, which is what the user will expect from their system. ErdosFS does use processor time, but this usage is believed to be imperceivable for most cases. In addition, leveraging UFS provides a familiar interface for the user to use ErdosFS and doesn't require learning any complicated new interfaces.

4 CONCLUSION

ErdosFS provides all the functionality outlined in the memo and section 2.1 in a space efficient and performance sensitive way. The user will benefit by having an intuitive way of getting immediate results for repeated searches.

5 ACKNOWLEDGMENTS AND REFERENCES

5.1 Acknowledgments

Thank you to Jessie Stickgold-Sarah and Pratiksha Thaker for reviewing my design memo and proposal and providing helpful feedback.

5.2 references

1. (2014, March 21). Linux manual pages. Retrieved from https://www.kernel.org/doc/man-pages/

2. D.M. Ritchie, K. Thompson, The UNIX Time-Sharing System, in Communications of the ACM, vol. 17, no. 7, 1974.

3. J. H. Saltzer and M. F. Kaashoek, M. Frans, Principles of Computer System Design. Waltham, Massachusetts: Morgan Kaufmann, 2009.

Word count = 2488