LiFSBrowse: a Visual, User Environment for the Linking File System

Technical Report UCSC-SSRC-07-08
August 2007

Sasha Ames
sasha@cs.ucsc.edu

Storage Systems Research Center
Baskin School of Engineering
University of California, Santa Cruz
Santa Cruz, CA 95064
http://www.ssrc.ucsc.edu/

Note: this report is a copy of a Masters project report, originally submitted for review in June 2005.

LiFSBrowse: A Visual, User Environment for the Linking File System

Sasha Ames
sasha@cs.ucsc.edu
Master's Project Report
Department of Computer Science
University of California, Santa Cruz

Abstract

The Linking File System introduces a new storage paradigm for enhanced user productivity through relationships between files, yet it opens up new challenges in usability. LiFSBrowse is a GUI for LiFS that attempts to meet those challenges through giving customizable graphical views of the file system. LiFSBrowse supports interaction through link manipulation and file system querying. We describe the layout of LiFSBrowse in detail and give some examples of its usability through a sample file system view.

1. Introduction

The Linking File System (LiFS) introduces a new paradigm in file system organization. Using attributed links, users of the file system have the potential for huge gains in productivity, such as through the ability to provide context for their documents. Links allow users to express relationships between documents, where they previously lacked the ability. However, LiFS is only the infrastructure. LiFS itself does not have an interface, thus, determining the true usability of LiFS shall be left up to those who can provide such an interface.

In this paper, I present LiFSBrowse. LiFSBrowse is a graphical user interface to the Linking File System. The main goal of LiFSBrowse is to provide users with a better experience in interacting with LiFS over command-line tools. LiFSBrowse should allow the user to browse her files and links, list the attributes on links and files, and display the file when possible. To enhance browser usability, browse displays should be customizable to the user's liking through a series of view options. Additional functionality should give the user the ability to interact with LiFS, such as through creating, modifying, or deleting links. Finally, the user should have the ability through LiFSBrowse to query LiFS for a specific set of files and links. This paper should demonstrate that LiFSBrowse can meet these functional goals, thus enhancing the usability of the Linking File System.

This paper is organized as follows. The next section gives an overview of LiFS, a description the text-based LiFS interface, and the advantages to using a GUI. Section 3 describes the GUI features and layout. Section 4 discusses the LiFSBrowse implementation. Section 5 gives some examples of the usability of LiFSBrowse. Sections 6 and 7 discuss related and future work, and section 8 concludes.

2. Background and Motivation

2.1. Overview of LiFS

In this section I give an overview of the major features of LiFS. We describe LiFS in more detail here [1], but I wish to convey the features that best distinguish it from conventional file systems.

LiFS contains a new file system feature, called a *relational link*. These relational links express relationships between any two files. Thus, in LiFS a regular file may point or refer to another. Relational links are different than symbolic links which allow a user to create additional path names for files or directories, thus becoming "linked" to additional places other than the original path. Moreover, relational links

may contain metadata in the form of associated attributes. These attributes take the form of key/value pairs where on a given link, each key must be unique. LiFS also allows attributes placed on files directly, as is done using the extended attribute API in the Linux kernel 2.6.x [9].

The existence of links between files leads us to a loss of distinction between files and directories. Hence, having files refer to other files allows files to "contain" others, a property that was previously reserved for directories. Thus, directories become, in essence, simply zero byte files that exist for organizational purposes and backwards compatibility with conventional file system structures. Moreover, this implies that a path to any given file shall occur through a series of files, which themselves may also contain data and can be opened.

LiFS additionally allows multiple links between files. This is so a single or multiple users may wish to have different sets of metadata to show separate relationships between the two files. I will present an example of this feature in section 5. The caveat to this feature is that links must be uniquely identifiable through some combination of their attributes. A simple identification scheme, but one not required by the file system, is to have a unique name attribute on each link.

The proposed implementation of LiFS is to have all metadata for files, links and attributes reside in magnetic RAM. MRAM is a new form of high speed, non-volatile RAM. It should greatly increase metadata access performance over conventional reads and writes to disk for metadata.

An example related to this research where LiFS might be used is for a the working directory of a software project. Binaries may be linked to object files, which in turn may be linked to the source files that generated them, or vice-versa. In addition, we can compile source code for different hardware architectures by following the appropriate link to the complier configured for the target architecture, where a link exists to a compiler for each architecture. The resulting object code could be linked to the source with attributes indicating which architecture it should run on.

```
[sasha@mram1 mount]$ ~/cvs/linkingfs/src/ln2
Usage:  to add links:
  ln2 [-s] <source file> <dest files> <attributes>
       to set attributes on existing links:
  ln2 -set <source> <dest> <old attrib> <new attrib>
       to get attributes for a link:
  ln2 -get <source> <dest> <attribs to match>
       to remove a link:
  ln2 -rm <source> <dest> <attribs to match>
[sasha@mram1 mount]$
```

Figure 1. Usage for the `ln2` command.

```
[sasha@mram1 mount]$ ls
nohup.out  projectA  projectB  projectC  usr
[sasha@mram1 mount]$ ls2
usr
    -> include
       - follow=true
projectA
    -> randomgen.c
    -> randomgen.h
projectB
    -> /nohup.out
       - my=newlink,userid=500
    -> linkedlist.c
    -> linkedlist.h
    -> /projectA/randomgen.c
projectC
    -> hashtable.c
    -> hashtable.h
    -> /projectB/linkedlist.c
[sasha@mram1 mount]$
```

Figure 2. Example output from the `ls2` command.

2.2. Interface to LiFS

Like any file system with advanced features, LiFS requires a user interface to expose those features. Application programmers may use the LiFS API, but that is less valuable to an end user of the file system. To initially make those features available we introduced the `ln2` and `ls2` command line utilities. These utilities allow for the utilization of LiFS' features beyond what available through conventional tools run through a shell.

`Ln2` is an adjunct utility to the `ln` program. Whereas ln allows a user to create hard and soft/symbolic links, ln2 enables the creation of relational links in LiFS. Figure 1 shows the usage for this utility. The source and target files must be existing

files to be linked. In addition, the utility requires the specification of some attribute metadata for the new relational link.

Ln2 has other features too that are relevant to linking. The utility lets a user modify the attributes, on a relational link, see all the attributes, or delete a link. In such cases, the link in question must be uniquely identified by a set of attributes. Queries that match more than a single return an appropriate error to inform the user of such.

The functionality of `ls2` is meant to augment `ls` for LiFS relational links and attributes. When ls2 lists the files under a given directory, it displays each file and all links emanating from the files. It displays attributes on the link as well. Figure 2 shows some files with their links. It lists the target name and attributes as comma-delimited strings in the format "key=value". At the top of the figure we can see the output from `ls` for comparison.

Additionally, `ls2` allows a user to query for specific links returned based on a combination of attributes. At present, the query capability is very simple. It allows only a conjunction of key/value pairs to match links.

2.3. Advantages of a GUI for LiFS

While these tools give an end user some ability to browse in LiFS, they do impose somewhat of a hindrance on the user. To achieve desired results, users may be required to perform a series of command line operations that include the commands described above. Alternatively, we may turn to a graphical user interface for LiFS to provide better usability and functionality.

In many ways, a GUI for LiFS can be more advantageous than conventional file browsers are for conventional file systems. Conventional file browsers do provide views of hierarchical directory tree structures, conventional metadata, and even extended attributes on files. However, all those capabilities are available on a command line. Seeing multiple levels of hierarchy may require multiple operations via command line, but should be a relatively straightforward process.

With LiFS, the GUI now can represent, in addition to a tree, the graph structures created by relationships between the files given by the addition of relational links. Using our text-based approaches, trying to realize a graph structure though various file lists of source nodes in a graph can become very cumbersome. However, the graph visualization grants the user instant knowledge of the relationships that she is concerned with.

Another advantage of using a GUI for LiFS is that it has the ability to present multiple scales of view for the file system. This becomes more important when links allow files to refer to one another that may appear located in disparate directories. While we can get a good local or partial view of the file system, showing a limited number of files, we obtain the addition of a more global view where there are more complex relationships present.

Here is an example of on an operation using several steps over a command line. We would like to change the attribute on a link, but we are not precisely sure how to identify the link. At first, we may guess and attempt to make the modification in one step using `ln2`, in which case we must provide a correct source path, target, and attributes for identifications. Trial and error come in to play here. At this point, either we may be correct, or we fail to identify the link correctly or uniquely. We may try again or try to locate the link using one or more calls to `ls2`. If the source path is correct, we then may look through the output from `ls2` for the link in question. Once identified, we can correctly use `ln2` to make the attribute modification for that link.

As we can see, it takes several operations and considerable human effort to complete a process that should be simple. To contrast, let us consider what is involved in using a GUI. We will have to locate the source file for the link, but once in view, we should be able to see all its child links and identify the one we want. Then, we may change the existing attributes, either modifying one or adding using a modification pane, and we are complete. Shortcuts to query the GUI for the correct source file or by attributes to help find the link in question even faster make this an even substantially improved process.

Thus, it is the goal of LiFSBrowse is to grant the user many of the above listed advantages. The following section will describe what features of LiFSBrowse have been included to satisfy meeting that goal.

3. GUI design

The LiFSBrowse GUI is designed to be able to display file relationships via links, select link metadata via attributes, and select file content all in one window. In order to accomplish this, the main LiFSBrowse window is divided into three sections, one for plotted file and links, one for attributes, and one for file content. Additionally, the LiFSBrowse window atop has a menu bar and below has a status bar, as is standard for most GUI applications.

The pane comprising the left side of the application contains LiFSBrowse's representation of the files and links from within LiFS. The GUI represents files by blue circles with a black outline and links by black arrows. It labels each link using the name component of the link's target file path. A user may select a link by clicking on the label or the link itself. When selected, a link and its label changes color to the hilight color, which is red by default. Having both the link and label hi-lighted together is especially helpful in identifying the selected link when the user views a file system with many links and labels plotted in tight proximity of one another. Additionally, the content in the content pane and attributes shown in the attribute pane may change when a user selects a new link.

The GUI plots files on the display as they are found using a depth-first search of the file system. It plots "deeper" files, i.e. children to the right of a parent file. Files with a "sibling" relationship are plotted top to bottom along the same vertical axis. There is a maximum depth setting (default of 10) that controls when to stop exploring for new links from a starting source file. As the DFS operation is called recursively on a child link, the value is decremented until it reaches 0.

LiFSBrowse offers the user a number of viewing options for customizing the file and link display. There are three general on-off options to be switched between, resulting in eight potential views. First is the structural drawing mode. This option is mainly cosmetic, affecting only the look of the output display. The first option for this mode are right angle links, where the first link from a source file shall be drawn as a horizontal arrow to the right of the plotted file. All subsequent files are drawn down and to the right, in an "L" shaped fashion with a longer horizontal component, this mimicking many file browsers that use tress to display directory hierarchies. Labels appear directly below the horizontal component of the link they identify. The other alternative is the "fan-out" view, where the child links of a source file are drawn out still to the right of the file, but at different angles, starting close to the top of the circle for the file, working its way to the rightmost point, and to the bottom. Having both these views gives the user alternatives, as some file systems may be geared to appear better in a particular mode.

The second option is tree versus graph view modes. Under tree mode, ALL child links of a source file are drawn, until the max depth is reached. This means that target files and links are drawn even if they are redundant, i.e. already drawn elsewhere in the display for the file system. In such cases, a selected link redundantly drawn shall be hi-lighted in all places that it was drawn. Additionally, any cycles created by files referring to one another are drawn out (redundant display) until the maximum depth is reached. Graph mode eliminates all such redundancy and cycles. In this mode, new child files are plotted as discovered in the search algorithm. However, for a file in question if one of its children has already been plotted as child of a another previously plotted file, then we draw a link directly from the newer source to that target child.

In "L" plotting mode combined with graph mode: if a case arises perhaps a link may be drawn through a file representation, LiFSBrowse draws the link as an arc. Additionally, in both plotting modes, LiFSBrowse draws arc-shaped links for situations where files lie on the same horizontal row, save for those initially discovered with a parent-child relationship. Otherwise, links are drawn as straight lines from the source file to the target. Some files are not protected by the provision for arc drawing in graph mode that exists for tree mode. Having these two different mode choices gives the user a great deal of power for viewing through a choice between focusing on source/target relationships for individual files, as done in tree mode, or all relationships, shown in graph mode.

The third option is whether or not LiFSBrowse should automatically expand all files to plot their child links. Turning "expand off" forces the user to manual click on files that concern her to view their children. This may make more easily readable file system dis-

plays by limiting the amount of data plotted. Moreover, it may often be the case that a user is only concerned with a limited set of links and files to be plotted at any one time. "Expand on", of course, provides the convenience of having all links and files plotted automatically.

Additionally, a user may modify the view by changing the size of the drawn components. Consequently, this feature behaves similarly to the zoom tool in many graphics application. Presently there are 5 size modes to choose between, ranging from "Tiny" to "Bigger." The chosen size affects the length and width of links, size of the file circles, and the font size of the labels. This mode is what should be primarily responsible for providing the functionality of switching between small and large file system views.

The user may add new links, modify or delete existing ones through the main pane. Adding a link requires the user to click on the source file, the target file, and fill in the attribute data (discussed later). To modify a link, the user simply must select the link she wishes to modify and change the attributes. Deletions require the user select the link to be deleted and confirm[1]. At present, Link sources and target files may not be modified in a single LiFSBrowse operation. If a user wishes to do so, she may delete the link and add a new link with a new desired source and target, but the same attributes as the deleted link.

The attribute pane, located in upper-right quadrant of the application frame, displays attributes of the current selected link. Rows hold each attribute key/value pair, with the keys in the left-hand column and values on the right. In cases of multiple links between a pair of files, a user may discern different attribute sets for each through having the background of the attribute text be painted alternating colors. The attribute data rows from the first link have a red background followed by the rows from the second link contain a blue background, and so on, Additionally, attributes on the target file of the links will have a third unique background color, set to green by default.[2]

Furthermore, the attribute pane also incorporates the functionality for attribute entry and querying. When the user creates a new link or selects a link for modification, the pane switches from attribute display mode to attribute entry. The user may then add new attributes or modify the existing keys and/or values. A submit button finalizes the user's changes. Moreover, the attribute entry view allows the user to enter queries. These queries come in a set of key/value pairs to match attributes on links in the file system. Once submitted, the query will return a new display of files and links in the main display pane on the left.

Below the attribute pane is the content display pane. In this pane, a user may see the content of the target file of the currently selected link. Presently, LiFSBrowse supports plain text and some image files. LiFSBrowse requires that a link targeting the file or the file itself be tagged with the attribute "file_type=" in order to display its content. The values "text" or "image" support those particular file types.

Finally, the bottom of the LiFSBrowse application frame contains the status bar. The status bar gives various messages to the user. These include the current link source and target paths, instructions to the user for a pending operation, and error operation successful messages. An example of an instruction is when a user selects "Add" from the "Link" menu, the status bar instructs the user to select a source file.

4. Implementation

I have implemented LiFSBrowse in the Python scripting language. I chose this because it is high level, interpreted and may lead to more rapid development than lower level languages such as "C" that are more appropriate for actual file system development. Moreover, I selected to use the tkinter GUI toolkit for this implementation as it is considered the de-facto standard for user interface tasks within Python [14].

The version of LiFS I used is not the one discussed above (section 2.1) that will utilize MRAM, but a prototype to serve as a proof of concept for LiFS. This

[1] LiFSBrowse running over the prototype PostgreSQL backed LiFS will not delete links that result in removing the target file permanently. Moreover, the next version of LiFS will let you remove the file when the final link targeting it is removed (reference count reaches 0), unless it contains child links of its own. Those links and all subsequent children must be removed first.

[2] Attributes on files were not available to read from the LiFS prototype. Hence we cannot demonstrate this feature, but I look to correct it with the next version of LiFS.

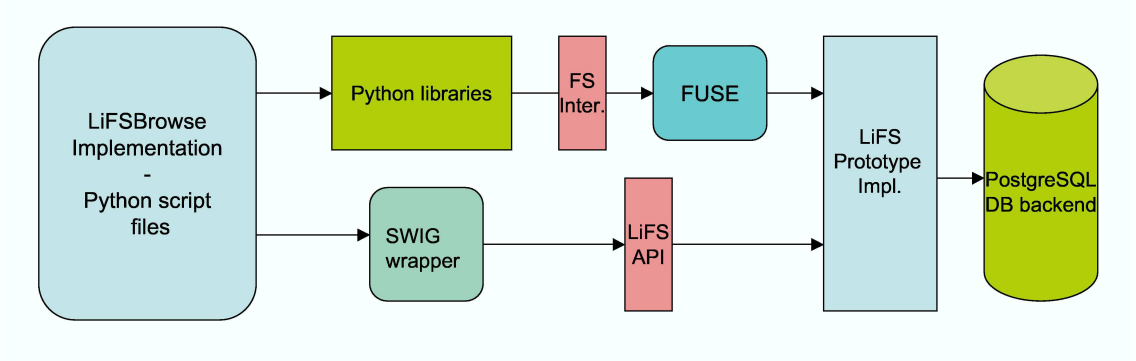

Figure 3. The architecture of the LiFS prototype and how LiFSBrowse accesses it

prototype runs with a PostgreSQL database [18] as the backing store for LiFS' metadata. The interface for standard file system operations comes through FUSE (file systems in user space) [17]. The python system libraries allow LiFSBrowse to access that functionality that goes through FUSE. A separate channel to the file system implementation provides additional API to LiFS calls that involve access and manipulation of relational links. This API I have made available for use in programs written in Python via SWIG wrapper code [16].

One implementation challenge came through handling a file system work around introduced in the prototype version of LiFS. As allowing the use of files and directories interchangeably breaks POSIX semantics, we decided to use the "@" character as an escape to force either to be a directory. Thus, in all paths, the directory names has the character appended within a path. In the implementation of LiFSBrowse, the code to handle content display uses the standard file system library exposed through FUSE as opposed to the linking and attribute related functionality which use the LiFS API. In that module's calls to the file system, the pathnames had to include the "@" character in order to properly open files for display purposes.

5. Usability Examples

Figure 5 shows a collection of files with various links between the files. In this case, each file stores the image of a musical artist or group and thus represents the artist or group. Thus, the links suggest relationships between each artists, with relationship categories such as influence, rivalry, etc.

The one exception to relationship types of those links, we may find in the link colored red. The red

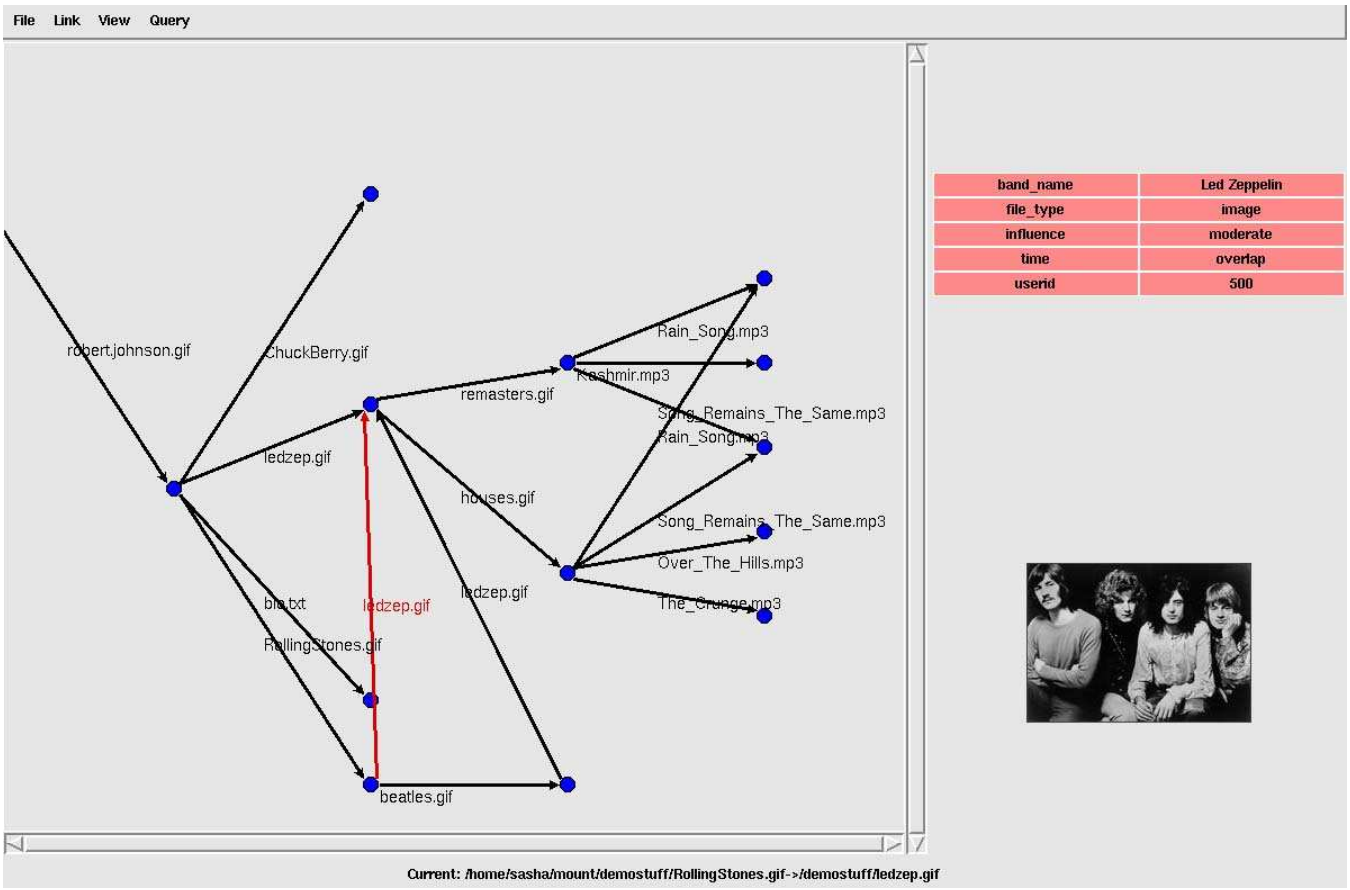

Figure 6. Resulting graph after addition of a new link. Some additional links and files are shown by expansion

The user has just successfully added a link, and we can see the resulting graph displayed in Figure 6. The newly added link is from the file "RollingStones.gif" to "ledzep.gif", and is hi-lighted. We may notice that layout of the graph has changed since the last figure. While this may appear disconcerting to the user, it can be beneficial because the new layout may better arrange the files and links for the new graph configuration than the previous layout. Moreover, we should notice that the font, file and link sizes have all grown. This is because our user must have changed the view size mode for LiFSBrowse to a larger size. An additional change to the display is apparent in the status bar, which now displays the source and destination files of the hi-lighted link.

The hi-lighted link in this figure has different metadata describing the relationship between the two files than shown in the previously considered view. This link suggests a relationship of influence between the two bands, and we may see that the level of influence is "moderate" In addition, we are given a sense that the bands may overlap in their years active, given the attribute for "time=overlap" Finally, we see that the file type is that of an image in this case, and the image (band photo) is displayed instead of text.

The file system graph plot in Figure 6 also contains additional files and links that were not present in the previous view. These files and links are for releases (albums or compilations) and songs for Led Zeppelin. In this case there are two releases which between the two of them point to a total of six song files. Where the release files point to the same song shows that that song appears in both release. Since the track number for that song may vary between release, an attribute given that information can be found on the link.

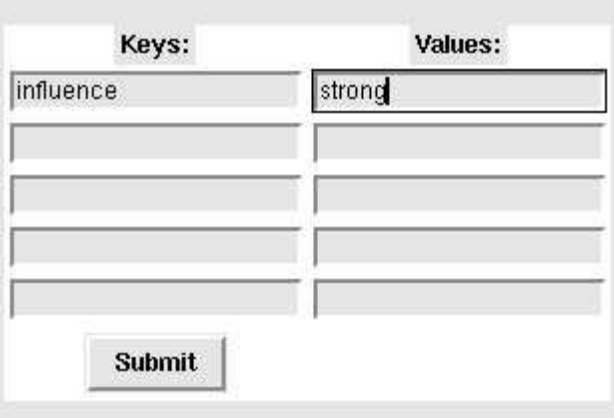

Figure 7. Panel to input a query to LiFS-Browse. A similar panel is used for attribute entry and modification

Suppose we wish to perform a query in LiFS-Browse, that is, create a display of files and links based on a search traversing links that match some key/value pair input. Figure 7 is the panel display for inputing attributes to form our queries. In our case, the user has entered a key of "influence" and the value "strong into the query box. Additionally, this display doubles in use for entering attributes when creating or modifying links.

Figure 8 shows the results of the user's query. The number of links between the files has been paired down to only those that denote a relationship of strong influence. The user has also set the view size to "Bigger": the largest size available in LiFSBrowse.

The user has selected a link pointing to a file for Chuck Berry, and this brings us to an interesting change in the attribute display area. There are now two sets of attributes, each with separate colors. Two of attributes means that the hi-lighted link represents two links in the underlying linking file system. In this case the user query has resulted in both links displayed because they have the query attribute in common. However, aside from "userid", the other attributes differ between the two links. The first has more information about the artist. The second seems to convey that this link's purpose is to convey the relationship between the two files.

Figure 9 shows much of the sample file system in "Tiny" size. This mode gives a more complete picture of what elements comprise the file system because more links and files can be plotted in the window.. Hence, we see many files starting with the file system root shown with a "/" to the "leaf" files that appear within the display and on the right edge of the graph.

The figure also show what tree mode of the browser looks like. In this view there are many files and links that are redundantly plotted. For example, examine the series of links that start with "robertjohnson.gif" on the left and end with "ledzep.gif" following each horizontally. That series denotes the chronological succession of influence. Additionally, we notice that the files along that chain of links have the child link for "ledzep.gif". Each of those files has the same set of decedent links and files, all of which are redundant. The redundancy is further exemplified by the hi-lighted link. In this case it is for the link to "houses.gif". Each set of redundant links and files has the same link hi-lighted. This makes it easy for a user of LiFSBrowse to identify redundant information when using tree views.

6. Related Work

McGuffin, *et al.* take an interesting approach to file system visualization [10]. Their system, Expand-Ahead places files and folders within folders to express containment relationships. Files may be listed by name or content may be displayed within. The system also features zoom in, auto drill-down, and will automatically expand various folders to size in order to best display as much data on screen as possible. LiFSBrowse, likewise attempts to show a user an overview of relationships within the file system and allows the user to zoom in on particulars. However, Expand-Ahead appears to not handle the non-tree edges that may be found through LiFS relational links without redundancy.

In scientific network visualization, CiteSpace focuses on identifying and tracking thematic trends. It features not only finding highly cited clusters, but also considers the trails that are pivotal points in transitions within a knowledge domain, perhaps a paradigm shift. Its visualization features graph display of citation and term links between documents, with additional rings on notes to mark highly cited documents, color to denote "pivotal" points, and age of the link. LiFSBrowse

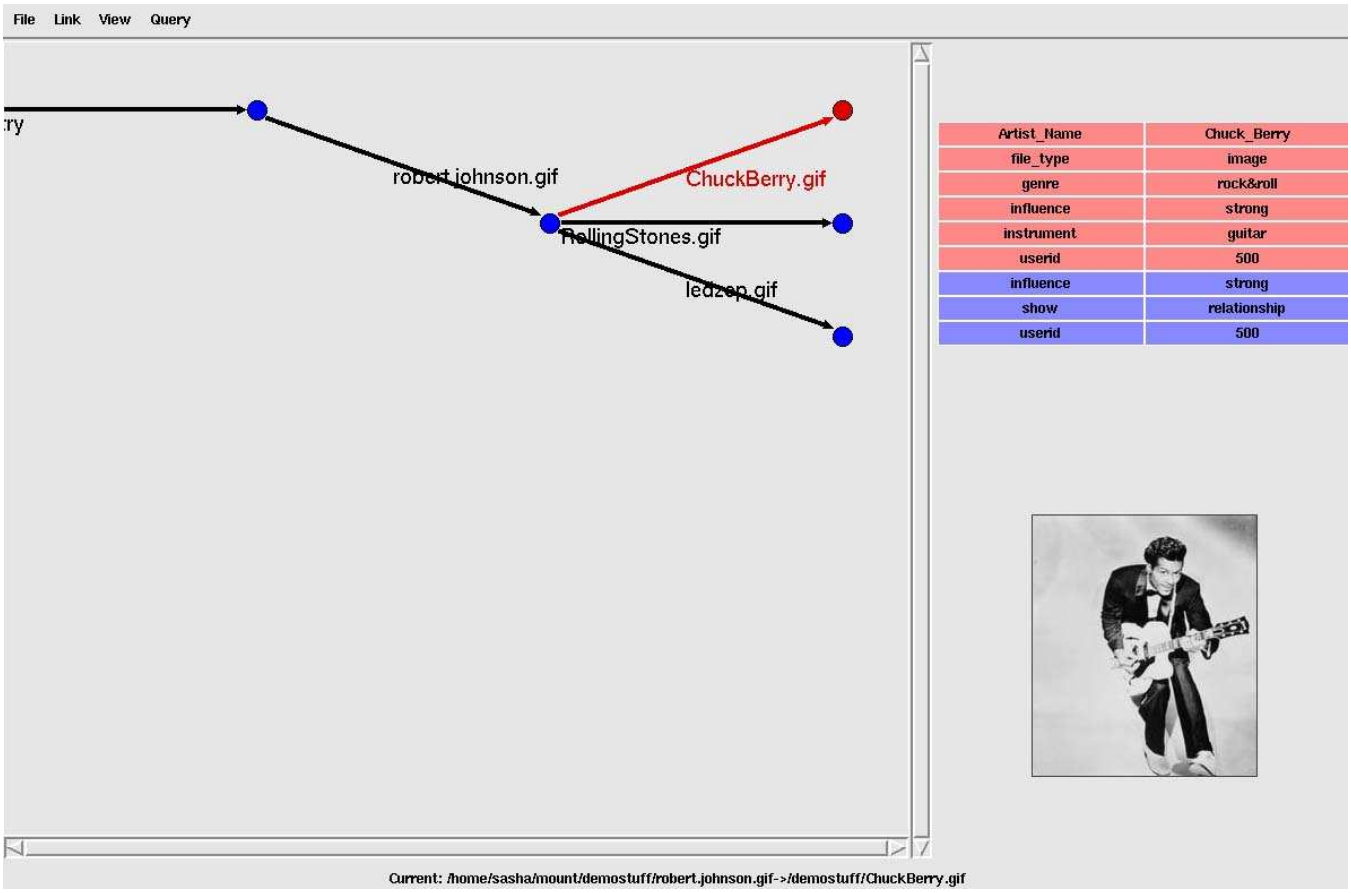

Figure 8. The result of the query. The highlighted link represents two links with distinct attributes.

could very well display citation information in graphical form for a file system organized by citation [4].

There have been a number of attempts to present advanced visualizations of the World Wide Web [2]. Natto [15] initially places nodes and links on a horizontal plane, but allows the user to manually raise nodes to "disentangle" them from the plane to improve the structure's visibility. Initial node placement is based on web page attributes, such as name and size. Another approach to web visualization is the construction of three-dimensional hyperbolic space represented inside a sphere. This view shows a focus on some nodes by plotting those near the center the largest and reducing the size of others as the plot moves towards sphere's edge, which is conceptually an infinite distance from the center [12].

The Presto system is an alternative to the hierarchical document spaces found in conventional file systems. Presto allows users users to manage documents in terms of assignable attributes. Vista, the Presto browser, plots related document collections inside ovals that can be opened an closed. The collections are based on sets of attributes which they themselves can be plotted on the Vista desktop. LiFSBrowse supports attribute based queries for documents like Presto/Vista. However, Vista does not display relationships between documents possible through relational linking [7]

There have been various attempts to visualize the Internet, a considerably large graph space to investigate. This was once a hot topic of discussion, where a major focus was on making the visualizations more useful to the average end usercitevisint95, visint96. As described in [6], a major goal of the CAIDA organization is to create Internet visualizations. One such visualization of Internet topology at a macroscopic level used an Internet Control Message Protocol to probe the Internet. The result was they could obtain snapshots and display in polar coordinates with local ISPs at the center, working its way out to further

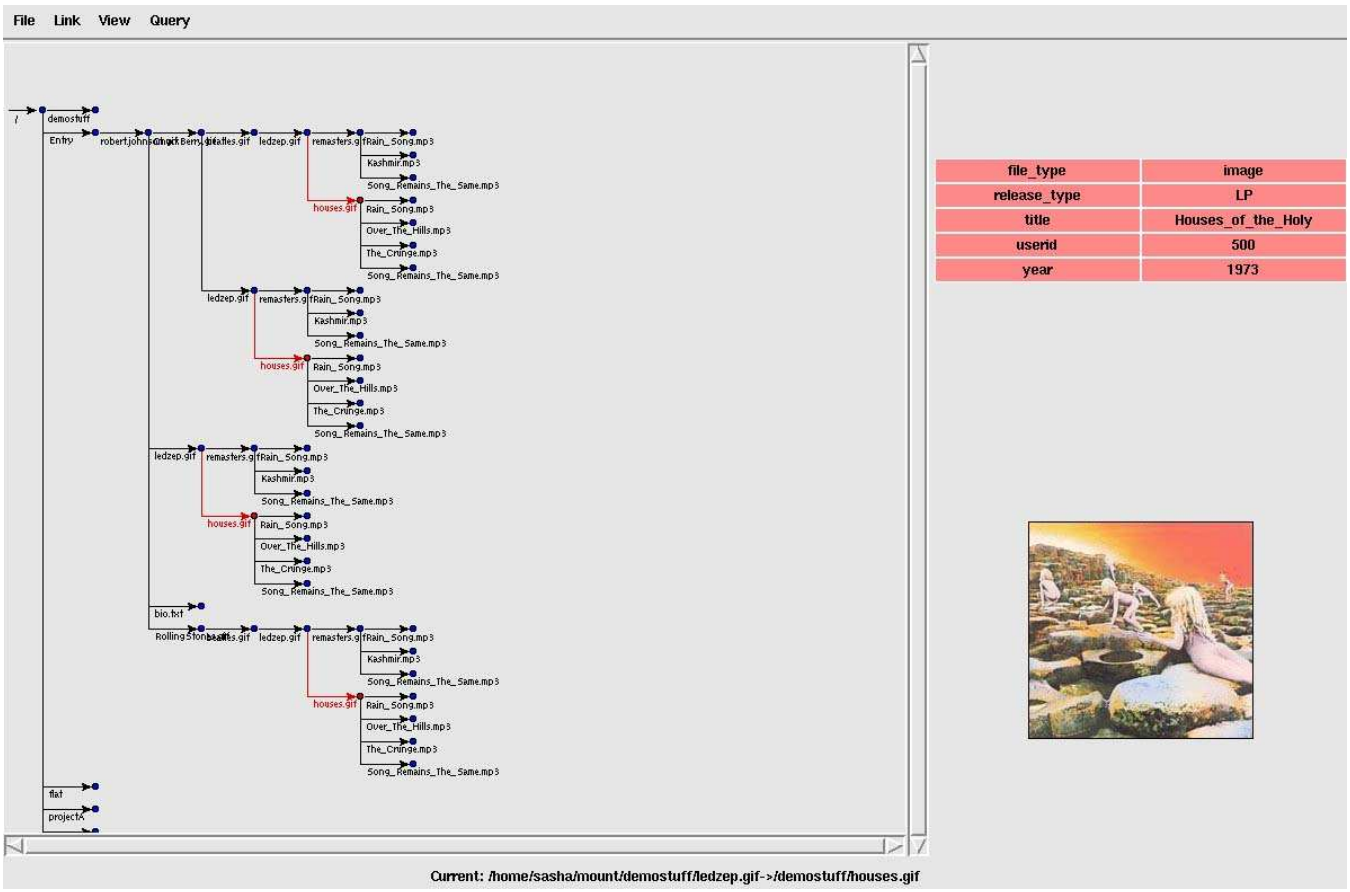

Figure 9. A view of the file system in tree mode. A link targeting the file for the *Houses of the Holy* album image is highlighted red in multiple locations showing the redundancy.

and further hops. CAIDA also features a number of Internet visualization tools, including Walrus which can plot million node directed graphs in three dimensional, non-euclidean hyperbolic space [3] (based on Munzner's work mentioned above).

Cheswick, *et al.* attempted to visualize the Internet though collecting routing paths from a test host. They used a spring-force algorithm to lay out the graphs from the collection databases which resulted in around 88,000 nodes and 100,000 edges for the whole Internet. Their resulting displays focused on producing minimal spanning trees that take the appearance of a "peacock on the windshield: or "koosh balls", thus smaller subgraph plots were more useful [5]. Furthermore, Munzner, *et al.* attempted to visualize the MBone Global Topology [13]. Their visualizations plotted the MBone tunnels as arcs over three dimensional spherical projections of the whole and segments of the earth, and of the United States. Arc height represented longer tunnels (latency) between points on the surface as to make them more prominent for attempts to analyze global congestion caused by longer paths.

Finally, The Navigational View Builder was a tool developed to integrate node and link topology with underlying information space in hypermedia systems [11]. This tool featured clustering of related nodes and fish-eye views to insert a focus on a node. These serve as good ideas for future work for LiFS-Browse.

7. Future Work

LiFSBrowse presented the opportunity to explore many possible alternatives for plotting file and link data but there is much more to accomplish. First and

foremost, LiFSBrowse should be switched to use to the next version of LiFS. This should correct some problems and simplify some steps in the implementation. Some aspects that will be fixed include the ability to retrieve attribute metadata on files, delete operations, and the pathname fudge using the "@" character.

One feature to be added to LiFSBrowse that could potentially benefit a user is a link or file focus. This would magnify the link or file in question, and then draw other child links and files in diminishing size as you get further removed from the focus. This could provide additional emphasis on a local context for a user.

Another additional feature of value to a LiFSBrowse user is a browse history of visited links. Including a history would give the user a sense of what she has already seen in browsing the file system. This could be implemented by painting all links a certain color or use different shading to give an idea of how long ago the link was visited.

Users may encounter in their graph displays situations where a group of files all refer to one another with relational links. The consequence of this is a quite messy display with many plotted links that become difficult to read. A potential solution to this is a "Cluster" display, where the group of files is plotted as a single unit. The unit can then be opened up with information on each file and the links within the cluster.

A final worthy goal for further development on LiFSBrowse is to improve the graph plotting. A simple improvement would be to allow users to define their own file positions. Each file could be grabbed and repositioned. All links with that particular file as a source or target would then be redrawn. Another possible improvement would be to try a different search algorithm for exploring the file system graph space. I chose DFS because it yielded simple handling of plot recursion for drawing multiple levels. The consequence of using only that algorithm is that all graphs are plotted not even close to optimally. However, a modified version of BFS could have been considered – maybe only for graph mode – as an alternative. Finally, graph layout is a well explored area, resulting in a number of free and commercial products to handle to the problem [8]. Thus, I should strongly consider plugging one into LiFSBrowse to determine where to plot files and links.

8. Conclusion

LiFSBrowse expands the value of the Linking File System through giving its users improved display and interaction. Command line interaction with LiFS will limit users' abilities to interface with LiFS, necessitating more advanced alternatives. Being able to see the relationships that exist between files firsthand should greatly improve understanding of their purpose and serve as reminders for context long forgotten. We hope to improve upon LiFSBrowse with the advent of the stable version of LiFS. It is progressing, and should perform much better, hopefully resulting in improved stability for LiFSBrowse.

Acknowledgments

I would like to express my deepest thanks to my advisor Ethan Miller for his many suggestions towards the work. Additional thanks go out to Nikhil Bobb, Scott Brandt, Adam Hiatt, Carlos Maltzann, and Alisa Neeman for influential discussions we've had. Research for LiFS was funded in part by National Science Foundation grant 0306650.

References

[1] A. Ames, N. Bobb, S. A. Brandt, A. Hiatt, C. Maltzahn, E. L. Miller, A. Neeman, and D. Tuteja. Richer file system metadata using links and attributes. In *Proceedings of the 22nd IEEE / 13th NASA Goddard Conference on Mass Storage Systems and Technologies*, Monterey, CA, Apr. 2005.

[2] S. Benford, I. Taylor, D. Brailsford, B. Koleva, M. Craven, M. Fraser, G. Reynard, and C. Greenhalgh. Three dimensional visualization of the world wide web. *ACM Comput. Surv.*, 31(4es):25, 1999.

[3] CAIDA. Walrus - graph visualization tool. http://www.caida.org/tools/visualization/walrus/, 2005.

[4] C. Chen. The centrality of pivotal points in the evolution of scientific networks. In *IUI '05: Proceedings of the 10th international conference on Intelligent user interfaces*, pages 98–105, New York, NY, USA, 2005. ACM Press.

[5] B. Cheswick, H. Burch1, and S. Branigan. Mapping and visualizing the internet. In *Proceedings of the 2000 USENIX Annual Technical Conference*, pages 1–12, San Diego, California, June 2000.

[6] K. C. Claffy. Caida: Visualizing the internet. *IEEE Internet Computing*, 5(1):88, 2001.

[7] P. Dourish, W. K. Edwards, A. LaMarca, and M. Salisbury. Presto: an experimental architecture for fluid interactive document spaces. *ACM Trans. Comput.-Hum. Interact.*, 6(2):133–161, 1999.

[8] J. Ellison and S. North. Graphviz - graph visualization software. http://www.graphviz.org/, 2004.

[9] The linux kernel archive. http://www.kernel.org/, 2005.

[10] R. McGuffin ; Davison, G.; Balakrishnan. Expand-ahead: A space-filling strategy for browsing trees, October 2004.

[11] S. Mukherjea and J. D. Foley. Navigational view builder: a tool for building navigational views of information spaces. In *CHI '94: Conference companion on Human factors in computing systems*, pages 289–290, New York, NY, USA, 1994. ACM Press.

[12] T. Munzner and P. Burchard. Visualizing the structure of the world wide web in 3d hyperbolic space. In *VRML '95: Proceedings of the fi rst symposium on Virtual reality modeling language*, pages 33–38, New York, NY, USA, 1995. ACM Press.

[13] T. Munzner, E. Hoffman, K. Claffy, and B. Fenner. Visualizing the global topology of the mbone. In *INFOVIS '96: Proceedings of the 1996 IEEE Symposium on Information Visualization (INFOVIS '96)*, page 85, Washington, DC, USA, 1996. IEEE Computer Society.

[14] Tkiniter - pythoninfo wiki. http://wiki.python.org/moin/TkInter, 2005.

[15] H. Shiozawa and Y. Matsushita. Www visualization giving meanings to interactive manipulations. In *Advances in Human Factors/Ergonomics 21B (HCI International p 97)*, pages 791–794, San Francisco, CA, August 1997.

[16] Simplified wrapper and interface generator. http://www.swig.org/, 2005.

[17] M. Szeredi. File System in User Space README. http://www.stillhq.com/extracted/fuse/README, 2003.

[18] J. C. Worsley and J. D. Drake. *Practical PostgreSQL*. O'Reilly, 1st edition, 2002.