Graffiti: A Framework for Testing Collaborative Distributed File System Metadata[*]

C. MALTZAHN
University of California at Santa Cruz, U. S. A.

N. BOBB
University of California at Santa Cruz, U. S. A.

M. W. STORER
University of California at Santa Cruz, U. S. A.

D. EADS
University of California at Santa Cruz, U. S. A.

S. A. BRANDT
University of California at Santa Cruz, U. S. A.

E. L. MILLER
University of California at Santa Cruz, U. S. A.

Abstract

Managing storage in the face of relentless growth in the number and variety of files on storage systems creates demand for rich file system metadata as is made evident by the recent emergence of rich metadata support in many applications as well as file systems. Yet, little support exists for sharing metadata across file systems even though it is not uncommon for users to manage multiple file systems and to frequently share copies of files across devices and with other users. Encouraged by the surge in popularity for collaborative bookmarking sites that share the burden of creating metadata for online content [21] we present Graffiti, a distributed organization layer for collaboratively sharing rich metadata across heterogeneous file systems. The primary purpose of Graffiti is to provide a research and rapid prototyping platform for managing metadata across file systems and users.

Keywords

File System Metadata, Distributed Systems, Information Management

[*]This research was funded in part by National Science Foundation grant 0306650, and Network Appliance, Inc. Additional support for the Storage Systems Research Center was provided by Hewlett Packard Laboratories, Hitachi Global Storage Technologies, IBM Research, Intel, Microsoft Research, Rocksoft, Symantec, and Yahoo.

1 Introduction

A December 2006 Pew survey has found that 28% of internet users have applied keywords to (or “tagged”) content online [21]. This trend started with collaborative bookmarking sites [14] and has since extended to a variety of documents, including photos, music, movies, email, and scholarly articles (for an overview see [16]). This recent success is widely attributed to a form of content categorization that (1) allows users to tag objects using their own vocabulary instead of a controlled vocabulary, and that (2) results in vocabularies which converge sufficiently to be useful for indexing [18, 10]. Users prefer to use their own vocabulary because categorizing items with a controlled vocabulary is cognitively hard [17] and requires an amount of coordination that does not scale to very large systems like the Web. The success of collaborative tagging for organizing content on the Web raises the question whether collaborative tagging could play a similar role in storage management, especially given the wide availability of rich metadata support.

Loosely defined, rich file system metadata is data about files and relations between files that is derived from file content, file usage, or manually added by the user. Many applications and recently designed file systems now support rich metadata which is evidence for our assumption that rich metadata is useful for managing the ever growing amount of stored data. Personal music or photo collections are examples where most users rely on applications and application-specific metadata to organize their collections. Apple’s Spotlight is an example of the recent addition of rich metadata capabilities to a modern file system [4].

While rich metadata helps users manage large volumes of information, metadata is often confined to a single application or single file system. The metadata that shows up alongside a particular file in one application (e.g. a photo in a photo library program) may not appear in another because of differences in data organization and formats. Moving files from one file system to another might destroy some or all of a file’s metadata because of a mismatch of file system metadata capabilities. All of this diminishes the value of metadata and deters users from investing time and effort to fully utilize the capabilities of a rich metadata system.

Thus, in spite of the need for rich metadata there is little known about how rich metadata is used. This is exacerbated by the fact that file systems are hard to change, and that poorly tested file system modifications run the risk of corrupting or loosing user data. Yet, usage of rich metadata is best obtained from users working with real data.

We propose Graffiti, a distributed layer that sits on top of file systems across networked computers. The primary purpose of Graffiti is to provide a research and prototyping platform for managing metadata across file systems and users. The simplicity and file system independence of Graffiti allows for rapid iterative design and evaluation of metadata structures without affecting any file systems. We believe Graffiti will evolve into a useful service that by separating metadata management from applications frees application developers from recreating possibly inefficient metadata storage, search, and retrieval, and makes these metadata functions available across applications and systems.

In the next section, we present Graffiti's design, including a detailed description of its components, metadata structures, and metadata management mechanisms. We then give an overview of our prototype implementation of Graffiti. This is followed by a section on our initial experience of using this prototype. We conclude the paper with related work, ongoing work and research, and summary.

2 Design

2.1 Clients and Servers

Graffiti consists of a network of loosely coupled clients and servers. Clients run on user machines and associate metadata with files. There is one client for each user/machine pair. Servers allow users to share metadata across Graffiti clients, no matter whether these clients run on machines owned by the same user or by different users.

For this sharing mechanism to be useful, users need to be able to control the proliferation of their metadata. We accomplish this by allowing users (1) to explicitly select one or more Graffiti server for each client, including servers they created themselves, and (2) to control what metadata is shared with which of these selected servers. For example, a user might connect to three servers, one dedicated to an enterprise, one to a department, and one to his/her household. The metadata shared with the home server might not overlap with the metadata shared with the work server, but some of the metadata shared with the home server might also be shared with the public server. Each server enforces a particular set of policies on who may connect to it and how metadata may proliferate. Servers can peer with other servers and exchange metadata based on individual data exchange agreements. Users can decide for themselves how much they trust a particular server and adjust metadata sharing accordingly.

There is one Graffiti client for each user and for each machine, i.e. a machine with multiple users runs multiple clients, and a user with multiple machines uses one Graffiti client on each machine. The client provides a metadata update and retrieval API to local applications, tracks local file system changes to maintain file/metadata associations, and synchronizes metadata with servers. A client also functions when disconnected. In this respect a Graffiti client resembles an IMAP [7] client that supports synchronization with multiple email accounts.

Graffiti clients periodically synchronize with servers over secure HTTP. This allows connections across firewalls and keeps communication private. Users are required to have a password-protected account on every server with which they communicate. Whether users are allowed to open new accounts on a particular server is part of the server's individual policy.

2.2 Files

When we associate metadata to files we need to define the identity of a file (how we name the file) and identify those cases where metadata is propagated between copies of files, i.e. the files that we consider the "same". We refer to the combination of name space and metadata propagation as "file identity concept".

Traditionally, users identify a file within a file system by its path consisting of the directory path and its basename (the final portion of the path). In hierarchical (non-flat) file systems the basename is generally not sufficient to fully identify a file, and the directory path is specific to a particular file system. In a networked environment files are typically identified by their path and the name of a host that exports the file. This name space is sufficient when users prefer to keep metadata specific to one file system and if all move and copy operations between file systems copy the metadata as well.

There are however common cases where the same file can appear in multiple file systems without any such copy or move operation, e.g. retrieving an attached file while checking email on multiple machines, or downloading a file from the Internet to multiple machines. In these cases we would like to associate metadata to a file only once, and have that metadata propagate to all copies of the file that we consider the "same" file.

In Graffiti we strive to accommodate the whole spectrum between local and global identity concepts but focus on the more interesting case where users associate metadata to a file on one machine and expect to see the same metadata for the "same" file on other machines. We are exploring this global file identity concept based on content and file name: we associate metadata to the content of files and update metadata associations when the content changes. Copies of files inherit metadata associations by reference. As long as the content of the copies remain the same, metadata changes to one copy apply to all copies. Files that change content inherit metadata by copy. Subsequent metadata changes to those files apply to only copies of the same (now changed) content and are not propagated back to previous versions.

We accomplish this scheme by associating metadata to file content checksums, rather than to the location of the file data. This use of checksums as a global file identification allows the system to share metadata on matching files across multiple file systems. A drawback of this approach is that any change to a file will change the checksum and therefore break the file's association with Graffiti-maintained metadata. An important task of the Graffiti client is therefore to keep track of whatever file identity concept is enforced and to update metadata associations accordingly. For example, changing the content of a file will trigger the client to create a new checksum of the file, create a new mapping between the file's metadata and the new checksum, and update the local mapping between file checksums and file paths.

The client has to also account for common behaviors of applications that rename, and create files on every file close. The user's expectation in this case is that the newly created file inherits all metadata from the renamed file. A Graffiti client catches this case by creating a brief timeout (e.g. one second) after each rename within which the client remembers the original path of the renamed file. Any new file created at the original path within the timeout inherits the metadata of the renamed file.

Graffiti clients rely on file system event notification which are available in most modern file systems. The alternative is to poll the file system for changes and reconstruct event patterns from modification times, such as the rename/create pattern we just described. The advantage of event notifications over polling is that file system changes instantly update indices, and that one can track file deletes. We view polling as a fall-back mechanism in case event notification drops events or fails entirely.

A Graffiti server only deletes an out-of-date checksum if during synchronization none of its clients contain a matching file. For this the server collects checksum reference counts from its clients. A client only deletes an out-of-date checksum and its associated tags if none of the user's other clients contain a file that matches that checksum. Thus, as part of a synchronization clients first submit changed reference counts to servers and then collects the reference counts for the current user from the server.

2.3 Tags and Links

Graffiti's metadata consists of keywords or "tags". A tag is a string and as such can represent attribute/value pairs, URLs, file paths, or access rights. A tag is always associated with a user and one or two files. We call tags associated with two files "links" (see below). The association of tags with users is to allow users to tag files according to their individual understanding and language preference. For widely shared files this also allows Graffiti to analyze tag patterns and generate tag recommendations. Tag recommendations have proven very popular in existing collaborative tagging services on the Web such as Del.icio.us [6].

Links represent directional relations between two files. Typical usage examples of links between files are formal relationships such as previous versions, software dependencies, neighborhoods in large data sets, and provenance. A similar concept was introduced in [8, 9] with the difference that each of those links can carry an arbitrary set of attribute/value pairs instead of one tag and one user.

Relationships between files can also be represented by common tags (as opposed to links), e.g. the set of all files that are associated with the tag "birthday-pic". The difference is that in the case of tags it is the responsibility of users to keep different relationships distinct by using appropriate tags or tag combinations. Links allow this responsibility to be delegated to the system because a link is not only defined by its tag but also by its source and destination. So the meaning of a tag in a link can be “overloaded” depending on some property of its source or destination file.

2.4 Tag Interpreters, Indices, and Sharing

Generally, the meaning of tags depends on individual users or some consensus among users and are not further interpreted by Graffiti. However, Graffiti provides a plug-in infrastructure that allows the insertion of tag interpreters. The mechanism of tag interpreter plug-ins is a convenient way to introduce new functionality that is invoked and controlled by tags. Graffiti uses built-in tag interpreters to allow users to define indices and to control sharing of metadata. Users define new indices by specifying tag sub-strings, e.g. all tags that begin with “path:”.

Users control the sharing of metadata by adding “sharing tags” to a file of the form “sync:” followed by the server name. These tags instruct the Graffiti client to share metadata with the specified server. The scope of a sharing tag – that is the set of all tags, links, and file checksums included in the shared metadata – includes in the current design all tags of the associated file, including incoming or outgoing links. Sharing tags can also be links in which case the scope includes all tags (and links) of the source and target files. We are aware that finer-grained scoping is possible, and maybe even desirable, but decided to start out with simple file-level scoping and introduce finer grained scoping in a future version of Graffiti if necessary.

2.5 Tag Sharing

Users frequently share files via email or instant messaging. To allow two Graffiti users (let us call them sender and receiver) who don’t necessarily have accounts on a common Graffiti server to also share the metadata on such communicated files, Graffiti provides a tag sharing protocol: The sender accompanies the file (within an email or instant message) with a URL that represents a ticket which allows the receiver to import the metadata to that file to his or her own Graffiti client. The receiver can request the URL and receive the shared metadata during the next synchronization unless the sender and receiver do not share the same Graffiti server. In that case, requesting the URL will return a metadata document that the Graffiti client can import and associate with the received file.

2.6 Client/Server Synchronization

Graffiti clients either on command or periodically synchronize all file checksums and tags with Graffiti servers according to sharing tags. Synchronizations conflicts can only occur when the same user changes metadata on the same file checksum on multiple Graffiti clients. Based on the assumption that this is a rare occurrence, conflicts are resolved by taking the union of all submitted tags. Deleting a file on one machine and creating it on another (i.e. moving the file from one machine to another) does not cause a conflict since checksums of deleted files are only pruned when they disappeared from the entire Graffiti system.

3 Implementation

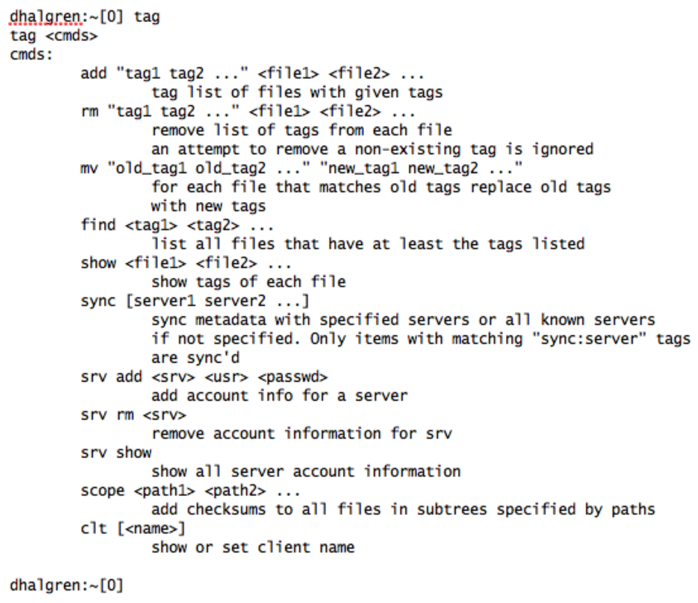
We implemented a prototype of Graffiti. The client is implemented in Java1.5 (J2SE 5.0) on Apple Mac OS 10.4 and consists of about 3,000 lines of code. We use fslogger [23] to make file system events available to the client, and Apache Derby [1] to store metadata at the client. The client prototype is designed to be portable across other platforms and to work with other file system notification services. The Graffiti server is implemented in Python (version 2.3.4) using the Twisted server framework (version 2.1.0) [5] and PostgreSQL (version 7.4.11) and consists of about 1,000 lines of code.

3.1 Client User Interface

We created a simple user interface primarily to get users to use Graffiti – our focus is the underlying system and not a user interface study. The user interface presented below is influenced by the Del.icio.us [6] Web site design and our experience using Graffiti. The three design goals were (1) be both script- and user-friendly, (2) rely as much as possible on the design of established Web interfaces such as Del.icio.us, e.g. show all tags, provide tag expansions, and search by conjunctions of tags, and (3) tightly integrate tagging with searching. All user interface functionality except for synchronization is based on data stored at the local client only. This prototype currently does not support links or indices, nor are the mechanisms for tag interpreter plugins or for tag sharing fully implemented.

To support tagging a Graffiti clients provides a command line interface and a graphical user interface. Figure 1 shows the usage information of the command `tag` which allows the user to manipulate tags of one or more files. The commands are designed to work well in shell scripts environments so they can take advantage of features such as file expansion and pipes. The commands are also optimized for speed by support for manipulating tags of many files with a single command. This is particular important for workflow applications in cases where the state is represented by tag combinations and a change of state involves a large number of files. In particularly the command `mv` allows the match-and-replace of a conjunction [1] of tags. A useful sequence of commands might be to select all files that match a conjunction of tags, using `find`, process selected files, and change the tags of files that have been processed using `mv`. For performance reasons we also

[1]Note that a list of tags is always interpreted as a conjunction of tags (which is commutative, of course) even though they are represented as strings in some commands.



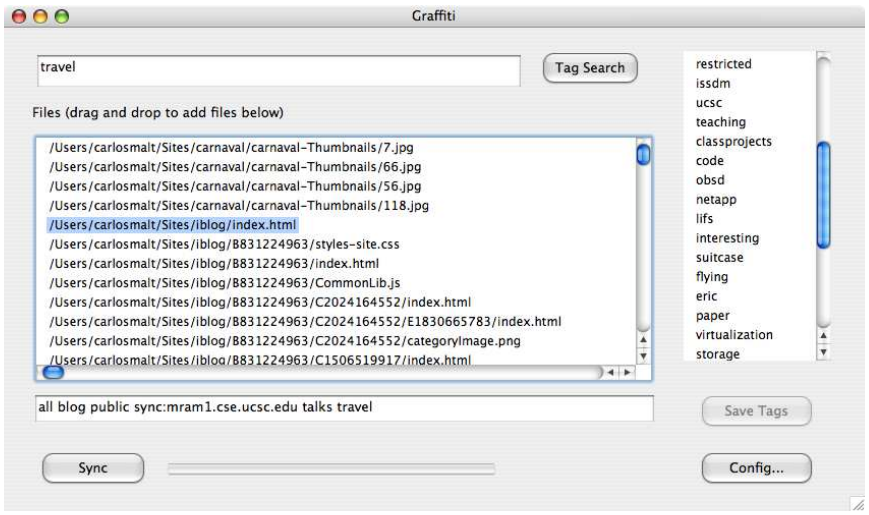
```
dhalgren:~[0] tag
tag <cmds>
cmds:
        add "tag1 tag2 ..." <file1> <file2> ...
                tag list of files with given tags
        rm "tag1 tag2 ..." <file1> <file2> ...
                remove list of tags from each file
                an attempt to remove a non-existing tag is ignored
        mv "old_tag1 old_tag2 ..." "new_tag1 new_tag2 ..."
                for each file that matches old tags replace old tags
                with new tags
        find <tag1> <tag2> ...
                list all files that have at least the tags listed
        show <file1> <file2> ...
                show tags of each file
        sync [server1 server2 ...]
                sync metadata with specified servers or all known servers
                if not specified. Only items with matching "sync:server" tags
                are sync'd
        srv add <srv> <usr> <passwd>
                add account info for a server
        srv rm <srv>
                remove account information for srv
        srv show
                show all server account information
        scope <path1> <path2> ...
                add checksums to all files in subtrees specified by paths
        clt [<name>]
                show or set client name

dhalgren:~[0]
```

Figure 1: Usage of the command `tag` which provides a command line interface for the Graffiti client. Many files can be processed with a single command.

included the command `scope` which limits checksumming of files to a subtree of the entire directory tree.

While the command interface is optimized for manipulating tags over a large number of files, the graphical user interface (Figure 2) attempts to reduce the cognitive overhead of adding tags to a particular file or searching for files that have already been tagged. The top entry field allows one to enter tags. The right panel shows all tags or all expansions of the text typed into the top entry field (since the last white space or the beginning of the entry field). Selecting will either replace an incompletely entered tag or add a new tag to the entry field. Hitting the “Tag Search” button displays all file paths in the center field that match the conjunction of the tags in the entry field. Files can also be “drag-and-dropped” into the center field which replaces any center field contents with the path name list of the dropped-in files. Selecting one file in the center field displays the tags of that file in the bottom entry field (note the synchronization tag `sync:mram1.cse.ucsc.edu`). Selecting multiple files displays the intersection of their tags. The bottom entry field allows editing of file tags. Hitting the “Save” button applies changes to all selected files. If multiple files are selected the content of the entry field only replaces the intersection of existing tags (duplicates are always ignored). The “Sync” button synchronizes the client with one or more servers depending on the client’s configuration, and the progress bar to



travel

/Users/carlosmalt/Sites/carnaval/carnaval-Thumbnails/7.jpg
/Users/carlosmalt/Sites/carnaval/carnaval-Thumbnails/66.jpg
/Users/carlosmalt/Sites/carnaval/carnaval-Thumbnails/56.jpg
/Users/carlosmalt/Sites/carnaval/carnaval-Thumbnails/118.jpg
/Users/carlosmalt/Sites/iblog/index.html
/Users/carlosmalt/Sites/iblog/B831224963/styles-site.css
/Users/carlosmalt/Sites/iblog/B831224963/index.html
/Users/carlosmalt/Sites/iblog/B831224963/CommonLib.js
/Users/carlosmalt/Sites/iblog/B831224963/C2024164552/index.html
/Users/carlosmalt/Sites/iblog/B831224963/C2024164552/E1830665783/index.html
/Users/carlosmalt/Sites/iblog/B831224963/C2024164552/categoryImage.png
/Users/carlosmalt/Sites/iblog/B831224963/C1506519917/index.html

restricted
issdm
ucsc
teaching
classprojects
code
obsd
netapp
lifs
interesting
suitcase
flying
eric
paper
virtualization
storage

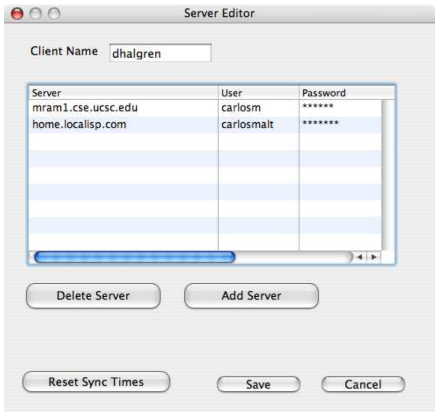
all blog public sync:mram1.cse.ucsc.edu talks travel

Figure 2: The graphical user interface for the Graffiti client attempts to reduce the cognitive overhead of adding tags to files or searching for tagged files. The design is based on the Del.icio.us social bookmarking service.

the right of the button gives an estimate of what fraction of the process is completed. Pressing the “Config” button pops up a dialogue window (Figure 3) that allows the management of Graffiti server accounts, setting the client’s name, and resetting either the client or the server synchronization state.

3.2 Server

The Graffiti server implementation allows users to share metadata. Direct exchange of metadata between servers is not implemented yet. The server consists of a relational database back-end and an API that clients access through secure HTTPS calls. The server has two primary roles. The first role of the server is to enable the collaboration of metadata across multiple machines. This is accomplished through the database and the API. The second role of the server is to collect usage data about collaborative metadata. This is done through event logging at the server and database levels. The relational database provides a persistent data-store for collaborative metadata using the schema in Figure 4.



Client Name: dhalgren

Server	User	Password
mram1.cse.ucsc.edu	carlosm	******
home.localisp.com	carlosmalt	*******

Figure 3: The dialog window allows users to manage multiple Graffiti server accounts. Special synchronization tags control how file metadata is shared with servers, e. g. the synchronization state of the client or any of the servers can be reset.

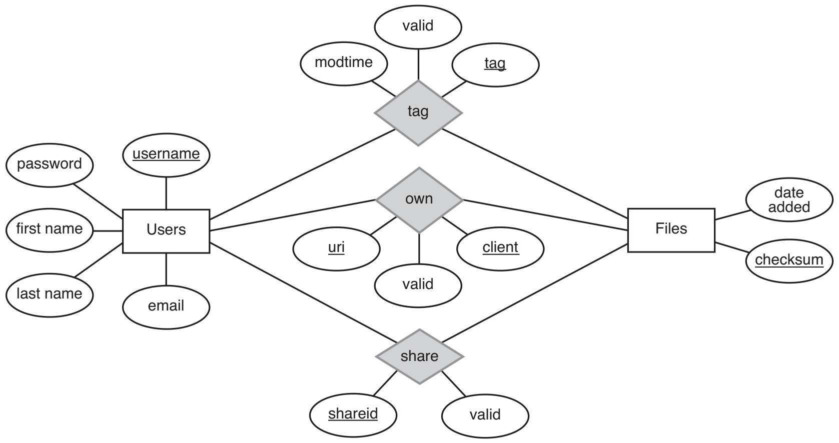


Figure 4: Entity-relationship diagram of the Graffiti server database schema. The database has four sets of data to manage. The first is the set of user accounts for that server. The second is the set of files, identified by checksum, that are owned by users. Third, the database tracks the tags that have been placed by users on files. Finally the server is able to manage metadata that users choose to share.

4 Experience

One of the advantages of Graffiti is that it can be readily used on existing file systems without the danger of corrupting or losing file data. This in combination with Graffiti's small code base often allowed us to quickly evolve the prototype based on user experiences. Our key insights were:

Tags do not make hierarchical directories obsolete. We found that directory hierarchies are only limiting if they are the only mechanism to organize files. We tried to organize files by tags only while keeping them all in one directory. This works pretty well as long as that directory only contains files that are of interest to the user. However, file systems almost always contain a plethora of system files that one wants to be kept out of the way by keeping them in system-managed directories as opposed to user-managed directories. Furthermore, we found that even within user-managed directories there are large sets of files that are managed by some other system, e. g. a versioning system or a music library application. In all these cases, we found it is significantly more convenient to use directories, and that tagging individual files contained in those directories is unnecessarily tedious. This led us to the insight that directories are good for hiding (and for keeping local name spaces small), while tags are good for finding. A file system is almost always shared by multiple agents, i. e. multiple users, applications, and the operating system. These agents have different perspectives on what needs to be hidden and what needs to be found. For example, we found it useful to be able to configure Graffiti to ignore files in some subdirectories.

Tagging directories is useful. In cases where we found that files are better managed by subdirectories, it was useful to tag the those subdirectories so that a search returns file names and directory names. This turns directories into a useful abstraction and scalability mechanism that allows the aggregate tagging of a large number of files without the overhead of tagging each file individually and keeping track of changes to those files. An interesting question is then how to share directory tags across file systems, i. e. how to give a directory a file system independent name. One approach is to take the checksum of the directory tree listing. Another approach is to specify just enough of a suffix of the directory's full path to fully identify copies in other file systems but to not include file system specific path prefixes. The first approach has the advantage that it fits well into the existing infrastructure of Graffiti. Not all file system notification services include directory updates, but they can be easily derived from file creations, renames, and deletions. The disadvantage is the same as representing files as checksums of their content: each change to a directory tree requires metadata updates. Limiting the depth of a directory tree listing might help, depending on how large upper level directories are and how often they change. The second approach requires no change tracking and therefore scales better than the first approach but we have not sufficiently investigated this approach to determine whether path suffixes are sufficiently unique to be used across file systems.

Uncovering duplication has great potential. One side surprisingly useful effect of managing files with the Graffiti client is that it makes file duplication visible. Graffiti revealed a surprising amount of duplication in home directories we used for testing, and motivates an extension to Graffiti which (1) shows which machine replicates a given file, and (2) provides a “de-duplication” service that can be used in a variety of storage management tasks, e. g. for reducing backup overhead or for filtering out duplicates in search results.

Tagging, searching, and file browsing are frequently interleaved. A commonly perceived limitation of the Graffiti user interface is that it does not provide the full functionality of a modern file browser such as the Finder in Mac OS X. For example, the Graffiti client allows one to find files but then one has to find the file again in the file browser in order to preview, open, or do anything else with them. Or, while working in the file browser or some other application displaying files, one cannot readily add tags to those files. Confirming this observation, Microsoft recently made available the personal information manager Phlat [11] which provides an innovative interface that allows for seamless switching between browsing, tagging, and searching.

5 Related Work

We have already mentioned popular tagging services on the Web [16] that influenced Graffiti’s design. Many existing file systems provide mechanisms for rich metadata but fall short in supporting collaborative maintenance of metadata [4, 15, 13]. The Linking File System (LiFS) introduces rich file system metadata that includes relational links that can carry arbitrary sets of attribute/value pairs [8, 9]. A number of systems provide support user-specific views on distributed file systems including standing queries based on rich metadata [12, 19]. A first step towards sharing rich metadata is a feature announced for Mac OS X.5 (Leopard) which allows for access of metadata on remote machines [3].

A number of infrastructures for collaborative metadata exist for particular applications but lack the generality necessary for all-purpose file systems. Perhaps the most famous example is the Compact Disk Database (CDDB) [2] where listeners gain access to CD metadata by submitting the fingerprint of the CD based on the ordered list of track durations (see [20] for an overview on other methods of matching metadata to CDs). If the fingerprint doesn’t exist, the listener can submit the track information to the database. Another example is the Scientific Annotation Middleware (SAM) [22] which uses WebDAV [24] servers to collaboratively maintain and share metadata of items of scientific data.

6 Ongoing Work and Conclusions

We continue to collect and analyze Graffiti workloads to increase our understanding of how users use rich metadata in file systems. We are currently working on the next version of Graffiti which will support all Graffiti design features as described in section 2, including links, user-defined indices, tag sharing, and tag interpreter plugins.

We are also developing a system event registration service. This service will allow users to specify metadata production rules activated by certain system events. An example of such a rule would be whenever a user copy-and-pastes content from one file to another, a link is installed between those files. Such rules will allow users to accumulate usage information about their file systems and will provide valuable data sets for research on contextual information management.

We are also investigating directory identity concepts that will allow us to share directory metadata across file systems. In section 4 we alluded to two such concepts with different costs and benefits.

Due to the content-based file identity concept Graffiti clients and servers reveals duplicates across different file systems and users. Duplicate information is valuable for ranking of search results (list the most convenient file reference only), for archival (duplicates are archived as references only), and for reliability (ensuring that a minimum number of replicas exist across different machines).

Graffiti metadata is tied to the identity of users. This provides opportunities and challenges. Among the opportunities is the possibility to build recommendation services on top of Graffiti that will further lessen the cognitive effort of tagging and might point users to interesting content. But sharing metadata raises privacy issues which we addressed by allowing users to connect to different Graffiti servers depending on trust and organizational context. We are also investigating alternative schemes that would provide privacy in alternative architectures such as a single central Graffiti server or a peer-to-peer architecture.

In summary, Graffiti offers a way to share three kinds of metadata: links, tags and indices. We presented a metadata sharing framework, and its potential applications are vast and include applications such as distributed data management, distributed indexing, and search.

References

[1] Apache derby project. http://db.apache.org/derby/.

[2] Cddb. http://en.wikipedia.org/wiki/CDDB.

[3] Mac os x.5 leopard sneak peek. http://www.apple.com/macosx/leopard/.

[4] Spotlight. http://www.apple.com/macosx/features/spotlight/.

[5] Twisted. http://twistedmatrix.com/trac/.

[6] del.icio.us. http://del.icio.us, Nov 2005.

[7] The imap connection. http://www.imap.org/, 2007.

[8] AMES, A., BOBB, N., BRANDT, S. A., HIATT, A., MALTZAHN, C., MILLER, E. L., NEEMAN, A., AND TUTEJA, D. 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).

[9] AMES, S., BOBB, N., GREENAN, K. M., HOFMANN, O. S., STORER, M. W., MALTZAHN, C., MILLER, E. L., AND BRANDT, S. A. LiFS: An attribute-rich file system for storage class memories. In *Proceedings of the 23rd IEEE / 14th NASA Goddard Conference on Mass Storage Systems and Technologies* (College Park, MD, May 2006), IEEE.

[10] BARONCHELLI, A., FELICI, M., CAGLIOTI, E., LORETO, V., AND STEELS, L. Sharp transition towards shared vocabularies in multi-agent systems. *Statistical Mechanics*, P06014 (June 2006).

[11] CUTRELL, E., ROBBINS, D. C., DUMAIS, S. T., AND SARIN, R. Fast, flexible filtering with phlat – personal search and organization made easy. In *In Proceedings of CHI'06, Human Factors in Computing Systems* (Montreal, Quebec, Canada, April 2006), ACM Press, pp. 261–270.

[12] DOURISH, P., EDWARDS, W. K., LAMARCA, A., AND SALISBURY, M. Presto: An experimental architecture for fluid interactive document spaces.

[13] GIFFORD, D. K., JOUVELOT, P., SHELDON, M. A., AND O'TOOLE, JR., J. W. Semantic file systems. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles (SOSP '91)* (Oct. 1991), ACM, pp. 16–25.

[14] GOLDER, S. A., AND HUBERMAN, B. A. Usage patterns of collaborative tagging systems. *Journal of Information Science 32*, 2 (2006), 198–208.

[15] GOPAL, B., AND MANBER, U. Integrating content-based access mechanisms with hierarchical file systems. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI)* (Feb. 1999), pp. 265–278.

[16] HAMMOND, T., HANNAY, T., LUND, B., AND SCOTT, J. Social bookmarking tools (part 1): A general review. *D-Lib Magazine 11*, 4 (April 2005).

[17] LANSDALE, M. W. The psychology of personal information management. *Applied Ergonomics 19*, 1 (1988), 55–66.

[18] MIKA, P. Ontologies are us: A unified model of social networks and semantics. *Lecture Notes in Computer Science*, 3729 (2005), 522–536.

[19] NEUMAN, B. C. The prospero file system: A global file system based on the virtual system model. *Computing Systems 5*, 4 (1992), 407–432.

[20] PACHET, F. Knowledge management and musical metadata. In *Encyclopedia of Knowledge Management*, D. Schwartz, Ed. Idea Group, 2005.

[21] RAINIE, L. 28% of online americans have used the internet to tag content. http://www.pewinternet.org/PPF/r/201/report_display.asp, January 31 2007.

[22] SCHWIDDER, J., TALBOTT, T., AND MYERS, J. D. Bootstrapping to a semantic grid. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005)* (May 2005), IEEE, pp. 175–181.

[23] SINGH, A. A file system change logger. http://www.osxbook.com/software/fslogger/, May 2005.

[24] WHITEHEAD, J. Webdav: versatile collaboration multiprotocol. *Internet Computing 9*, 1 (2005), 66–74.

Carlos Maltzahn is with the Department of Computer Science at the University of California, Santa Cruz, CA, U.S.A. E-mail: carlosm@soe.ucsc.edu

Nikhil Bobb is with the Department of Computer Science at the University of California, Santa Cruz, CA, U.S.A. E-mail: nikhil@soe.ucsc.edu

Damian Eads is with the Department of Computer Science at the University of California, Santa Cruz, CA, U.S.A. E-mail: eads@soe.ucsc.edu

Mark W. Storer is with the Department of Computer Science at the University of California, Santa Cruz, CA, U.S.A. E-mail: mstorer@soe.ucsc.edu

Scott A. Brandt is with the Department of Computer Science at the University of California, Santa Cruz, CA, U.S.A. E-mail: scott@soe.ucsc.edu

Ethan L. Miller is with the Department of Computer Science at the University of California, Santa Cruz, CA, U.S.A. E-mail: elm@soe.ucsc.edu