Proceedings of the Ninth Workshop on Hot Topics in Operating systems, USENIX Association, May 2003.

Why can't I find my files? New methods for automating attribute assignment

Craig A. N. Soules, Gregory R. Ganger
Carnegie Mellon University

Abstract

Attribute-based naming enables powerful search and organization tools for ever-increasing user data sets. However, such tools are only useful in combination with accurate attribute assignment. Existing systems rely on user input and content analysis, but they have enjoyed minimal success. This paper discusses new approaches to automatically assigning attributes to files, including several forms of context analysis, which has been highly successful in the Google web search engine. With extensions like application hints (e.g., web links for downloaded files) and inter-file relationships, it should be possible to infer useful attributes for many files, making attribute-based search tools more effective.

1 Introduction

As storage capacity increases, the amount of data belonging to an individual user increases accordingly. Soon, storage capacity will reach a point where there will be no reason for a user to ever delete old content – in fact, the time required to do so would be wasted. The challenge has shifted from deciding what to keep to finding particular information when it is desired. To meet this challenge, we need better approaches to personal data organization.

Today, most systems provide a tree-like directory hierarchy to organize files. Although this is easy for most users to reason about, it does not provide the flexibility required to scale to large numbers of files. In particular, a strict hierarchy provides only a single categorization with no cross-referencing information.

To deal with these limitations, several groups have proposed alternatives to the standard directory hierarchy [5, 9, 11]. These systems generally assign attributes to files, providing the ability to cluster and search for files by their attributes. An attribute can be any metadata that describes the file, although most systems use keywords or ⟨category, value⟩ pairs. The key challenge is assigning useful, meaningful attributes to files.

To assign attributes, these systems have suggested two largely unsuccessful methods: user input and content analysis. Although users often have a good understanding of the files they create, it can be time-consuming and unpleasant to distill that information into the right set of keywords. As a result, users are understandably reluctant to do so. On the other hand, content analysis takes none of the user's time, and it can be performed entirely in the background to eliminate any potential performance penalty. Unfortunately, the complexity of language parsing, combined with the large number of proprietary file formats and non-textual data types, restrict the effectiveness of content analysis.

A complementary alternative to these methods is context analysis. Context analysis gathers information about the user's system state while creating and accessing files, and uses it to assign attributes to those files. This can be useful in two ways. First, such context is often related to the content of a file. For example, a user may read an email about a friend's dog and then look at a picture of that same dog. Second, the context may be what a user remembers best when searching for some files. For example, the user may remember what they were working on when they downloaded a file, but not what they named the file.

This paper discusses two categories of context analysis: access-based context analysis and inter-file context analysis. The first gathers information about the state of the system when a user accesses a file. The second propagates attributes among related files. Combining these methods with existing content analysis and user input will increase the information available for attribute assignment.

The remainder of this paper is organized as follows. Section 2 discusses background and related work. Section 3 describes access-based context analysis. Section 4 discusses recognition and use of inter-file relationships. Section 5 presents some initial findings. Section 6 discusses some challenges facing this work, and ideas on how to approach them.

2 Background

Users already have difficulty locating their files. There exist a variety of tools for locating files by searching

through directory hierarchies, but they don't solve the problem. Several groups have proposed attribute-based naming systems that rely on user input and content analysis to gather attributes, but they remain largely unused. Web search engines, however, have found greater success obtaining attributes by combining content analysis with context analysis. This section discusses common approaches to file organization, proposed systems, and relevant web search-engine approaches.

2.1 Directory Hierarchies

There are three key factors that limit the scalability of existing directory hierarchies. First, files within the hierarchy only have a single categorization. As the categories grow finer, choosing a single category for each file becomes more and more difficult. Although linking (giving multiple names to a file) provides a mechanism to mitigate this problem, there exists no convenient way to locate and update a file's links to reflect re-categorization (since they are unidirectional). Second, much information describing a file is lost without a well-defined and detailed naming scheme. For example, the name of a family picture would likely not contain the names of every family member. Third, unless related files are placed within a common sub-tree, their relationship is lost.

One way to try and overcome these limitations is to provide tools to search through these hierarchies. Today, on UNIX systems, many users locate files via tools such as *find* and *grep*. These tools provide the ability to search throughout a hierarchy for given text within a file, providing rudimentary content analysis. *Glimpse* [14] is a system that provides similar functionality, but utilizes an index to improve the performance of queries. Microsoft Windows' search utility provides a similar indexing service using filters to gather text from well-known file formats (e.g., Word documents). Going a step further, systems such as *LXR* and *CScope* [22], perform content analysis on well-known file formats to provide some attribute-based searching features within a hierarchy (e.g., locating function definitions within source code).

2.2 Proposed Systems

To go beyond the limitations of directory hierarchies, several groups have proposed extending file systems to provide attribute-based indexing. For example, BeFS extends the directory hierarchy by adding a new organizational structure for indexing files by attribute [8]. The system takes a set of ⟨file, keyword⟩ pairings and creates an index allowing fast lookup of an attribute value to return the associated file. This structure is useful for files that have a set of well-known attributes on which to index (e.g., ⟨email message, sender⟩).

The semantic file system [9] provides a way to assign generic ⟨category, value⟩ pairings to files, increasing the scope of their namespace. These attributes are assigned either by user input or by file content analysis. Content analysis is done by a set of *transducers* that each understand a single well-known file format. Once attributes are assigned, the user can create virtual directories that contain links to all files with a particular attribute. The search can be narrowed by creating further virtual subdirectories.

Several groups have explored other ways of merging hierarchical and attribute-based naming schemes. Sechrest and McClennen [21] detail a set of rules for constructing various mergings of hierarchical and flat namespaces using Venn diagrams. Gopal [10] defines five goals for merging hierarchical name spaces with attribute-based naming and evaluates a system that meets those goals.

Other groups have looked at the problem of providing an attribute-based naming scheme across a network of computers. Harvest [3] and the Scatter/Gather system [5] provide a way to gather and merge attributes from a number of different sites. The Semantic Web [1] proposes a framework for annotating web documents with XML tags, providing applications with attribute information that is currently not available.

These systems provide a number of interesting variations on attribute-based naming. But they all rely upon user input and content analysis to provide useful attributes, with limited success.

2.3 Context Analysis

Early web search-engines, such as Lycos [15], relied upon user input (user submitted web pages) and content analysis (word counts, word proximity, etc.). Although valuable, the success of these systems has been eclipsed by the success of Google [4].

To provide better search results, Google utilizes two forms of context analysis. First, it uses the text associated with a link to decide on attributes for the linked site. This text provides the context of both the creator of the linking site and the user who clicks on the link at that site. The more times that a particular word links to a site, the higher that word is ranked for that site. Second, Google uses the actions of a user after a search to decide what the user wanted from that search. For example, if a user clicks on the first four links of a given search, and then does not return, it is likely that the fourth link was the best match. This provides the user's context for those search terms; the user believes that those terms relate to

that particular site.

Unfortunately, Google's approach to indexing does not translate directly into the realm of file systems. Much of the information that Google relies on, such as links between pages, do not exist within a file system. Also, Google's query feedback mechanism relies on two properties: users are normally looking for the most popular sites when they perform a query, and they have a large user base that will repeat the same query many times. Unfortunately, neither of these properties are true in file systems: (1) users usually search for files that have not been accessed in a long time, because they usually remember where recently accessed files reside and access them directly, and (2) there is generally only a single user for each set of files; thus, it is unlikely that frequent queries will be generated for any given file.

3 Access-based Context Analysis

This section outlines two approaches to automatically gathering attributes when a file is created or accessed. These approaches use the context of the user's session at the time a file is accessed to assign attributes. The first uses application assistance, and the second uses existing user inputs.

Application assistance: Although most computers can provide a vast array of functionality, most people use their computer for a limited set of tasks. Most of these tasks are performed by a small set of applications, which in turn access and create most of the user's files. Modifying these applications to provide hints about the user's context could provide invaluable attribute information.

For example, if a user executes a web search for "asparagus" and downloads several pictures, it is likely that these are pictures of "asparagus." Similarly, if a user saves an email attachment and the subject of the email is "Re: Marketing report" then it is likely that the attachment is related to both "marketing" and "report."

Existing user input: Although most users are not willing to input additional information, they already are willing to choose a directory and name for the file. Each of the sub-directories along the path and the file name itself probably contain context information that can be used to assign attributes. For example, if the user stores a file in "∼/papers/FS/Attribute-based/Semantic91.ps," then it is likely that they believe the file is a "paper" having to do with "FS," "attribute-based," and "semantic."

Like Google, an attribute-based file system can obtain information from user queries. If a user initially queries the system for "semantic file system" and chooses a file that only contains the attribute "semantic," then the additional terms "file" and "system" could be applied to that file. Also, if the possible matches are presented in the order that the system believes them to be most relevant, having the user choose files further into the list may be an indicator of success or failure. Also, as is done in some web search engines, a system could elicit feedback from the user after a query has completed, allowing them to indicate the success of the query using some sort of scale. Unfortunately, as mentioned above, individual files are likely to have few queries, reducing the amount of information available through this method.

4 Inter-file Relationships

Once relationships are established, attributes can be shared between related files. This helps to propagate attributes among individually hard-to-classify files. In conjunction with approaches that generate attributes (such as application assistance or content analysis), such propagation should categorize a much broader set of files. This section outlines two approaches to automatically gather inter-file relationships. The first approach leverages user access patterns, and the second approach examines content similarities between potentially related files.

User access patterns: As users access their files, the pattern of their accesses provides a set of temporal relationships between files. These relationships have previously been used to guide a variety of performance enhancements [12, 16, 23]. Another possible use of this information is to help propagate information between related files. For example, accessing "SemanticFS.ps" and "Gopal.ps" followed by updating "related.tex" may indicate a relationship between the three files. Subsequently, accessing "related.tex" and creating "WhyCantIFindMyFiles.ps" may indicate a transitive relationship.

Inter-file content analysis: Content analysis will continue to be an important part of automatically assigning attributes. In addition to existing per-file analysis techniques, our focus on creating context-based connections between files suggests another source of attributes: content-based relationships. For example, some current file systems use hashing to eliminate duplicate blocks within a file system [2, 18], or even locate similarities on non-block aligned boundaries [13, 17]. Such content overlap could also be used to identify related files, by treating files with large matching data sets as related.

Often, users (or the system [19]) will keep several slightly different versions of a file. Although these files generally contain differences, often the inherent information contained within does not change (e.g., a user may keep three instances of their resume, each focused for a different type of job application). This gives the sys-

tem two opportunities for content analysis. First, content comparison can identify related files. Second, by performing content analysis solely on the differences between versions, it may be possible to determine version-specific attributes, making it easier for users to locate individual version instances.

5 Initial Findings

This section discusses insights drawn from trace analysis of user activity.

5.1 Exploring Creation-time Attributes

Figure 1 shows two charts indicating the percentage of files created by different programs within a single user's home directory. This data was gathered from a trace of a single graduate student's "home" directory tree over a one month period. The first chart shows a breakdown of every file created within the directory tree. The second chart shows a breakdown of files explicitly organized by the user (rather than created and named by a program for itself) and believed to have some permanence (rather than being temporary or scratch files). This excludes things such as caches, logs, program configuration files, compiler output, and CVS source repositories, which are all organized by an external entity (generally the programs that create them).

Although a large number of files were created within this user's home directory, most of the files were organized by user-invoked programs rather than by the user themselves. Most of the user-organized files in the trace were created by three applications: a text editor/email program (emacs), a web-browser (mozilla), and document creation tools (latex). The others were created by various manual FS tools (e.g., "cp," "cat," etc.).

Examining these results suggests how a combination of the automated attribute assignment techniques described above can provide useful context information:

- The web-browser can generate hints for the files that it creates. For example, in this trace, the file "∼/docs/online.pdf" was downloaded after doing a search for "SML Robert Harper" and clicking through Robert Harper's home page until the SML programming guide was located.

- Files created by text editors are generally accessed in conjunction with various other files, creating inter-file relationships. In this trace, several source code files were accessed in conjunction with a file named "∼/class/814/homework2.tex," indicating that the files probably all related to "class," "814," and "homework."

- Document creation tools like LaTeX take input from several different files and output a single postscript file (such as "homework2.ps"). This many-to-one relationship can be used to distill all of the input attributes into a smaller set of shared attributes that can be assigned to the output file. Also, these shared attributes could be passed back to any input files that do not have them.

- Illustrator (an image manipulation program) was used to create a poster outlining this work, importing text and images from a variety of related sources, resulting in a similar many-to-one relationship.

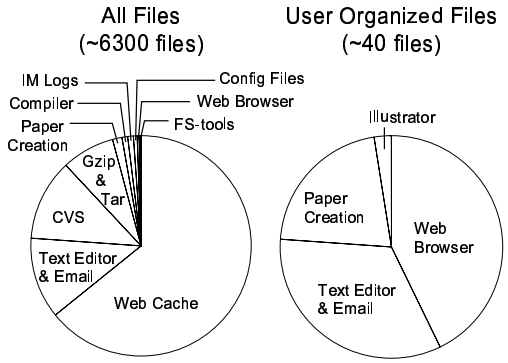

Figure 1: **Programs that create files.** Shows the programs that created files within a single graduate student home directory. The chart on the left shows a breakdown of every created file. Most of the files in this category are caches of either web pages or email, although archived source code (Tar & Gzip) and CVS repositories also figure in heavily. The chart on the right shows only those files explicitly organized by the student. These include those files downloaded from the web, hand edited files, files created by paper creation tools, and an image of a technical poster (Illustrator).

5.2 Exploring Inter-file Relationships

To further examine inter-file relationships, we created a simple tool to extract inter-file relationships from the trace. This tool tracks the last file access made by a program, and relates that file to the next file accessed. These relationships form groupings of related files.

Using this method, the tool successfully groups many files correctly (based on manual inspection by the owner). For example, a source tree was grouped with its resulting program output and backup tarballs, while a variety of unrelated source files were separated. Unfortunately, also grouped with the source tree were a variety of unrelated files (false positives). An examination of

the false positives showed that many were created by occasional use of *find* and *grep*. The graduate student in question uses *find* and *grep* to search by content for particular files. In a attribute-based naming system, *find* and *grep* would be replaced by an integrated searching system. This both removes the false positives, and could potentially improve accuracy using the feedback from user queries as described in Section 3.

6 Ongoing Challenges

Although our initial results are encouraging, there are still a large number of challenges beyond what has already been described. This section outlines some of these challenges, and initial ideas on how to approach them.

6.1 System Evaluation

One of the toughest research challenges faced when exploring automated attribute assignment is evaluating its accuracy. Although several groups have done automated file content analysis, little evaluation of the accuracy of these mechanisms has been reported. This is probably due to the difficulty of such an evaluation: what is "accurate?" More importantly, the true value of this kind of system is in helping users locate lost files, which is difficult to demonstrate without long-term deployment. Unfortunately, getting users to use such a system without first proving its value is difficult, resulting in a classic "Catch-22."

One possible approach is to feed a trace of user activity and application hints into the attribute assignment system and then compare its results to attribute assignment done by that same user. Unfortunately, this approach fails to account for user behavior. Although the user may initially categorize a file one way, they may later use it or look for it in another way. For example, the search terms they use a year after file creation may end up differing from their initial categorization.

6.2 Mechanisms

Although successfully assigning file attributes is one step in creating an attribute-based naming system, there are two other important aspects: the mechanism for storing attribute mappings and the user interface to the system. As mentioned in Section 2, several groups have looked at methods for storing attribute mappings. Until now, these methods have generally worked with a small number of attributes. By automatically identifying large numbers of attributes, two challenges arise. First, the existing methods may need to be extended to handle large numbers of attributes. Second, the system must identify the most relevant attributes for a file from the large set of associated attributes (i.e., weighting and false positive removal).

Several groups have also looked at the problem of user interfaces for attribute-based naming system. MyLifeBits [7] stores ⟨file, attribute⟩ pairings within a database, and provides a variety of file visualizations that help a user locate their files. Lifestreams [6, 20] provides a time-ordered stream of incoming information to the user, as well as a simple interface for filtering and sorting this information using a variety of attributes. Our work complements these and may provide useful insight into these two aspects of attribute-based naming.

6.3 User Context Switches

Context information has the potential to provide a large number of useful attributes. When a user switches context, however, the relationships created may be invalid. It would be helpful if the system could notice user context switches. One solution is user input, where the user indicates to the system what they are currently working on. If the user is not diligent, however, then the system may create more false positives than before. Another possibility is to infer user context switches from their actions. For example, switching to or from a particular application (e.g., the email browser) may consistently indicate a context switch.

7 Conclusions

As the data set associated with a user grows, organizing that information becomes more difficult. Although hierarchies have several useful aspects, they do not scale. A more flexible, attribute-based naming scheme is needed to effectively manage large personal data sets. This paper proposes automating attribute assignment using at-file-access context analysis and inter-file relationships. By obtaining many new attributes, these schemes should greatly increase the utility of attribute-based naming.

Acknowledgments

We thank the members and companies of the PDL Consortium (including EMC, Hewlett-Packard, Hitachi, IBM, Intel, Microsoft, Network Appliance, Oracle, Panasas, Seagate, Sun, and Veritas) for their interest, insights, feedback, and support.

References

[1] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, **284**(5):34–43, 2001.

[2] W. J. Bolosky, S. Corbin, D. Goebel, and J. R. Douceur. Single instance storage in Windows 2000. USENIX Windows Systems Symposium, pages 13–24. USENIX Association, 2000.

[3] C. M. Bowman, P. B. Danzig, U. Manber, and M. F. Schwartz. Scalable internet resource discovery: research problems and approaches. *Communications of the ACM*, **37**(8):98–114, 1994.

[4] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, **30**(1–7):107–117, 1998.

[5] D. R. Cutting, D. R. Karger, J. O. Pedersen, and J. W. Tukey. Scatter/Gather: a cluster-based approach to browsing large document collections. ACM SIGIR International Conference on Research and Development in Information Retrieval, pages 318–329. ACM, 1992.

[6] S. Fertig, E. Freeman, and D. Gelernter. Lifestreams: an alternative to the desktop metaphor. ACM SIGCHI Conference, pages 410–411, 1996.

[7] J. Gemmell, G. Bell, R. Lueder, S. Drucker, and C. Wong. MyLifeBits: fulfilling the Memex vision. ACM Multimedia, pages 235–238. ACM, 2002.

[8] D. Giampaolo. *Practical file system design with the Be file system*. Morgan Kaufmann, 1998.

[9] D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. W. O'Toole Jr. Semantic file systems. ACM Symposium on Operating System Principles. Published as *Operating Systems Review*, **25**(5):16–25, 13–16 October 1991.

[10] B. Gopal and U. Manber. Integrating content-based access mechanisms with hierarchical file systems. Symposium on Operating Systems Design and Implementation, pages 265–278. ACM, 1999.

[11] D. R. Hardy and M. F. Schwartz. Essence: a resource discovery system based on semantic file indexing. Winter USENIX Technical Conference, pages 361–373, 1993.

[12] G. H. Kuenning and G. J. Popek. Automated hoarding for mobile computers. ACM Symposium on Operating System Principles. Published as *Operating Systems Review*, **31**(5):264–275. ACM, 1997.

[13] J. MacDonald. *File system support for delta compression*. Masters thesis. Department of Electrical Engineering and Computer Science, University of California at Berkeley, 2000.

[14] U. Manber and S. Wu. GLIMPSE: a tool to search through entire file systems. Winter USENIX Technical Conference, pages 23–32. USENIX Association, 1994.

[15] M. L. Mauldin. Retrieval performance in Ferret a conceptual information retrieval system. ACM SIGIR Conference on Research and Development in Information Retrieval, pages 347–355. ACM Press, 1991.

[16] G. Memik, M. Kandemir, and A. Choudhary. Exploiting inter-file access patterns using multi-collective I/O. Conference on File and Storage Technologies, pages 245–258. USENIX Association, 2002.

[17] A. Muthitacharoen, B. Chen, and D. Mazieres. A low-bandwidth network file system. ACM Symposium on Operating System Principles. Published as *Operating System Review*, **35**(5):174–187. ACM, 2001.

[18] S. Quinlan and S. Dorward. Venti: a new approach to archival storage. Conference on File and Storage Technologies, pages 89–101. USENIX Association, 2002.

[19] D. S. Santry, M. J. Feeley, N. C. Hutchinson, R. W. Carton, J. Ofir, and A. C. Veitch. Deciding when to forget in the Elephant file system. ACM Symposium on Operating System Principles. Published as *Operating Systems Review*, **33**(5):110–123. ACM, 1999.

[20] Scopeware, http://www.scopeware.com/.

[21] S. Sechrest and M. McClennen. Blending hierarchical and attribute-based file naming. International Conference on Distributed Computing Systems, pages 572–580, 1992.

[22] J. L. Steffen. Interactive examination of a C program with Cscope. Winter USENIX Technical Conference, pages 170–175. USENIX Association, 1985.

[23] S. Strange. *Analysis of long-term UNIX file access patterns for applications to automatic file migration strategies*. UCB/CSD–92–700. University of California Berkeley, Computer Science Department, August 1992.