Toward automatic context-based attribute assignment for semantic file systems

Craig A. N. Soules, Gregory R. Ganger

CMU-PDL-04-105
June 2004

Parallel Data Laboratory
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract

Semantic file systems enable users to search for files based on attributes rather than just pre-assigned names. This paper develops and evaluates several new approaches to automatically generating file attributes based on context, complementing existing approaches based on content analysis. Context captures broader system state that can be used to provide new attributes for files, and to propagate attributes among related files; context is also how humans often remember previous items [2], and so should fit the primary role of semantic file systems well. Based on our study of ten systems over four months, the addition of context-based mechanisms, on average, reduces the number of files with zero attributes by 73%. This increases the total number of classifiable files by over 25% in most cases, as is shown in Figure 1. Also, on average, 71% of the content-analyzable files also gain additional valuable attributes.

Acknowledgements: We thank the members and companies of the PDL Consortium (including EMC, Hewlett-Packard, HGST, Hitachi, IBM, Intel, LSI Logic, Microsoft, Network Appliance, Panasas, Seagate, Sun, and Veritas) for their interest, insights, feedback, and support. We thank IBM and Intel for hardware grants supporting our research efforts.

Keywords: semantic filesystems, context, attributes, classfication

1 Introduction

As storage capacity continues to increase, the number of files belonging to an individual user has increased accordingly. Already, storage capacity has reached the point where there is little reason for a user to delete old content—in fact, the time required to do so would be wasted. A recent filesystem study [10] found that both the number of files and file lifetime have increased significantly over studies conducted during the previous 20 years. The challenge has shifted from deciding what to keep to finding particular information when it is desired. To meet this challenge, users need better tools for personal file organization and search.

Most file systems today use a hierarchical directory structure. Although easy to reason about at small and medium scales, hierarchies often do not scale to large collections and the fine-grained classifications that result. In particular, as the hierarchy grows, each piece of data is increasingly likely to have several different classifications, but can only have one location within the hierarchy. To find the file later, the user must remember the one classification that was originally chosen.

To help alleviate this problem, several different groups have proposed semantic file systems [13, 14, 30]. These systems assign attributes (i.e., keywords) to files, providing the ability to cluster and search for files by their attributes. The key challenge is assigning useful, meaningful attributes to files.

Most semantic file systems rely upon two sources of file attribute assignment: 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, many users are understandably reluctant to do so. On the other hand, content analysis automatically distills keywords from the file contents, taking none of the user's time. Unfortunately, the complexity of language parsing, combined with the large number of proprietary file formats and non-textual data types, restricts the effectiveness of content analysis. And, more importantly, it fails to capture an important way that users think about their data: context.

Context is defined by the Merriam-Webster dictionary [27] as "the interrelated conditions in which something exists or occurs." Thus, the context of a file refers to any information external to the contents of the file that may be present while the file is being created or accessed. Examples of this might include active applications, other open files, the current physical location of the user, etc. Context is often the way that humans remember things, such as the location of a previously stored file; a recent study of user search behavior found exactly this tendency: users usually use context to locate data [2].

This paper makes the case for context-based attribute assignment, introducing approaches from two categories: application assistance and user access patterns. Application assistance uses application knowledge of user workloads to provide information about the user's current state. For example, a web browser can usually provide information about downloaded files. User access patterns indicate inter-file relationships that can then be used to propagate attributes among strongly related files. This paper also introduces a new content-based approach, *content similarity*, that propagates attributes between files with similar content.

Figure 1 summarizes the benefits of combining existing content analysis with the three new categories of file attribute assignment examined in this paper. The multi-part bar on the left shows that adding these schemes allows the system to automatically classify an additional 33% of the user's files, taking the total from 65% to 98%. The four single-category bars on the right show that many of the classification schemes overlap, resulting in more attributes for each file. By gathering more attributes for more files, we believe that semantic file systems will provide more useful and accurate results to users.

The remainder of this paper is organized as follows. Section 2 discusses existing search and organizational tools. Section 3 outlines the space of automated attribute assignment. Sections 4, 5, and 6 describe our three new attribute generation tools and evaluate their merits. Section 7 discusses how existing systems will need to change given these new tools. Section 8 concludes.

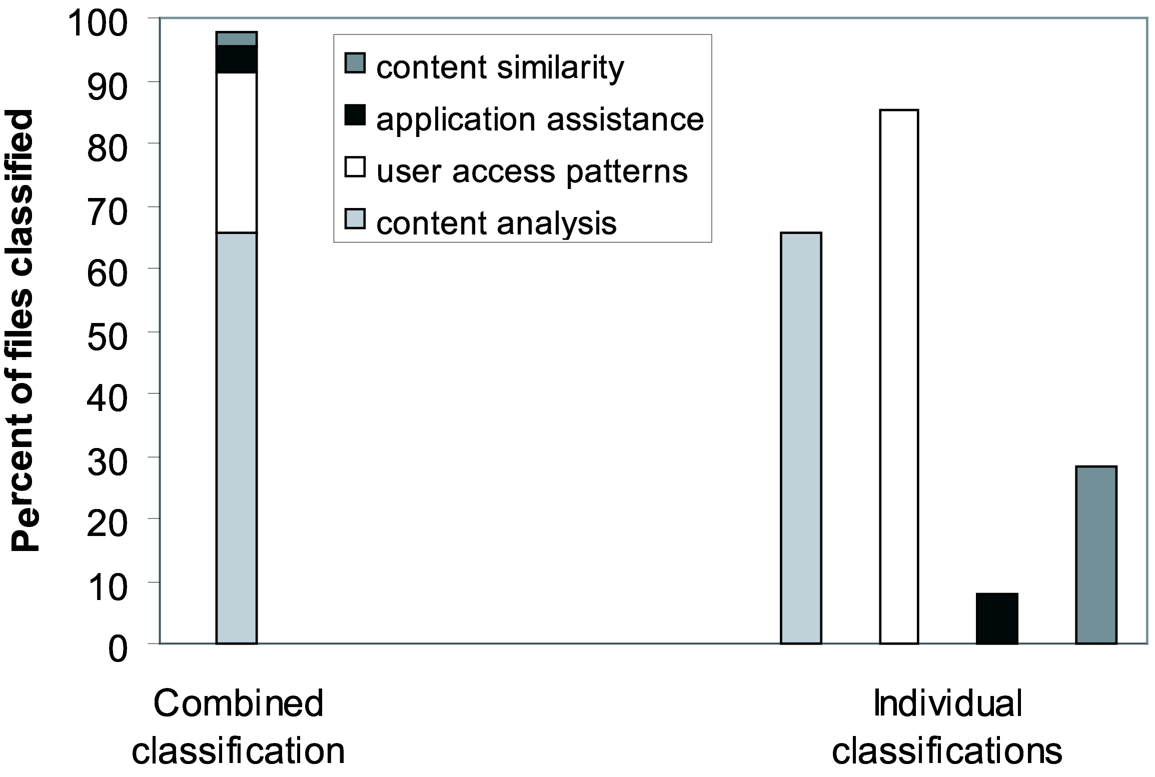


Figure 1: **Breakdown of classifications.** This figure shows the benefits of combining context- and propagation-based attribute generation schemes with existing content analysis schemes. We ran the schemes over the four month "donut" trace described in Section 4. By combining schemes, it is possible to classify more files than any individual scheme. The overlap of classifications by different schemes also provides more attributes to each of the classifiable files than any individual scheme.

2 File organization and search

When searching for recently accessed files, or within a small collection, users can often find what they are looking for. However, when searching for older files, or within a large collection, users often forget the name or location of particular files, making them difficult to find. Most people have stories of wasting 10 minutes or more searching through their files to find a specific, important document; as a user's file collection grows, this problem becomes worse. This section describes existing tools for file organization and search, as well as additional related work from outside the realm of file systems.

2.1 Directories

By far, the most common organizational structure for files is the directory hierarchy. The directory hierarchy relies entirely upon user input to provide useful classifications to files, in the form of a pathname. The advantage of user input is that any provided classification should closely match how the user thinks about the file, and often combines both content and context attributes. Unfortunately, these attributes become far less effective as the number of files grows.

There are four factors that limit the scalability of traditional 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 links (giving multiple names to a file) provide 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 subtree, their relationship is lost. Fourth, because the entire pathname of the file is required to retrieve it, the user must remember the exact categorization for a desired file.

Users are now faced with a dilemma: either organize files into detailed categories and remember the exact set and order of attributes for each file, or do not organize but face the difficulty of locating a single file within a sea of filenames. Clearly, users require better organization and search mechanisms.

2.2 Indexing and search tools

Today, many users rely upon indexing and search tools to help find files lost in their directory hierarchy. Tools such as *find*/*grep* and *Glimpse* [24] use text analysis and file pathnames to provide searchable attributes for files. Others, such as Microsoft Windows' search utility, use filters to gather attributes from well-known file formats (e.g., Word documents). Recent systems, such as Grokker [18] and Fractal:PC [12], layer new visualizations over existing indexing technologies in an attempt to help users locate and organize files.

The advantage of these tools is that they require no additional user action to provide benefit. Instead, they use existing directory information and combine it with automated content analysis. Unfortunately, these tools work best for files with well-understood data formats that can be reduced to text, leaving a large class of files with only the original hierarchical classification.

2.3 Semantic file systems

To go beyond the limited hierarchical namespace of directories, several groups have developed systems that extend the namespace with attribute-based indexing. BeFS [13] provides an additional organizational structure for indexing files by attribute. The system takes a set of ⟨keyword, file⟩ pairings and creates an index allowing fast lookup of a keyword 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., ⟨sender, email message⟩).

The Semantic File System [14] provides a searchable mapping of generic ⟨category, value⟩ pairings to files. These attributes are assigned either by user input or by an external indexing tool. These indexing tools, referred to as transducers, traditionally perform content analysis, but could provide any attribute that can be extracted automatically (such as context information). 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 sub-directories.

Several groups have explored other ways of merging hierarchical and attribute-based naming schemes. Sechrest and McClennen [30] detail a set of rules for constructing various mergings of hierarchical and flat namespaces using Venn diagrams. Gopal [16] defines five goals for merging hierarchical name spaces with attribute-based naming and evaluates a system that meets those goals.

By assigning multiple attributes to a single file, the information available to search tools is increased dramatically. Unfortunately, although these systems generally make no limitations on the methods for gathering attributes, previous examples of such systems use only content analysis to provide additional attributes, making them no more effective than existing search tools.

2.4 Additional related work

This section discusses additional work from outside file systems research that is related to gathering and using context for file organization and search.

Web search engines: Over the last 10 years, a large number of web search engines have come into existence [1, 15, 23, 34]. Early web search-engines, such as Lycos [25], 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 [6].

To provide better search results, Google utilized 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.

Recently, Google announced their intent to develop a local file search; however, their approach for ranking web pages does not translate directly into the realm of file systems. The links between pages that are inherent in the web do not exist between files. As such, it is likely that their initial tools will instead leverage their extensive knowledge of text analysis and indexing.

Improved interfaces: The Haystack project [28] aims to improve the organization of user data by providing improved interfaces that manipulate data using open, well-annotated file formats. By easing the methods for users to input information into the system, Haystack can gather more useful attributes than most existing systems. Combining a user-interface such as Haystack with automated methods of attribute assignment and propagation could provide the user with an even greater ability to locate information than either system alone.

Mobile cache hoarding: As distributed and decentralized systems have become more prevalent, many groups have examined the idea of automatically hoarding networked files locally on mobile nodes [19, 20, 33]. The Aura project has examined schemes for allowing users to specify hoarded files by specifying tasks [31], a set of applications and files that make up a particular context. Assuming that users are willing to provide this information, it could also provide context hints to the indexing system.

Networked attributes: Researchers have explored the problem of providing an attribute-based namespace across a network of computers. Harvest [5] and the Scatter/Gather system [8] provide a way to gather and merge attributes from a number of different sites. The Semantic Web [4] proposes a framework for annotating web documents with XML tags, providing applications with self-describing attribute information for each file, and its data format is the basis for much of the Haystack work described above.

3 Sources of attributes

We believe that there is a broad array of automated attribute assignment approaches that can be exploited for semantic file systems. We separate them into two groups—direct and indirect—based on how the attributes are assigned. Direct assignment approaches provide attributes to files at creation or access time. Indirect assignment approaches propagate attributes from a file to other closely related files. This section describes content- and context-based approaches that fall into these two groups.

3.1 Direct attribute assignment

Although significant potential exists for a variety of attribute generation schemes, existing systems have only used three particular direct assignment schemes: user input, content analysis, and file system attributes. This section outlines these existing schemes, as well as three new schemes that use context information.

3.1.1 Existing schemes

User input: The most common form of attribute assignment is user input. Although users are often unwilling to provide detailed attributes for each file, the high accuracy of these attributes makes them an important part of attribute assignment. Attributes from user input include the filename, the pathname, any additional hard and soft links, and any keywords (in systems that accept them).

Content analysis: Content analysis uses an understanding of a file's format to extract keywords to use as attributes for the file. The most common form of this is text analysis, used by most of the systems described in Section 2. Some systems, such as Microsoft Window's search utility, use their knowledge of certain file formats (e.g., Word or PowerPoint) to reduce file contents into raw text, allowing text analysis of the contents. Researchers continue to work on content analysis of music, images, and video files [7, 9], but this remains a complex problem that is far from solved.

File system attributes: Most filesystems store a small set of context attributes within each file's metadata, such as the owner of the file and the creation time. This information is one way that a user may search for a file, and is often used to enrich the set of searchable attributes in existing search tools. Going further, the Lifestreams system uses this context information as a first-class entity, providing a stream-like view of all documents based on access time [11]. The utility of these few context attributes suggests that additional context information could be quite helpful to the user.

3.1.2 Context-based schemes

Application assistance: Although personal computers can provide a vast array of functionality, most people use their computer for a small set of routine tasks. Most of these tasks are performed by a small set of applications, that in turn access and create most of the user's files. These applications could explicitly provide attributes for these files based on their knowledge of the user's context. For example, if a user executes a web search for "chess openings" and then downloads a file, the browser can note that this file probably relates to "chess openings." It could also extract keywords from the web address, the hyperlink text, other information on the linking page, and other pages accessed recently. The evaluation in Section 5 explores the specific case of context-based attributes that can be provided by web browsers.

User feedback: Many search tools gather information about their results from either direct or inferred user feedback. Similarly, an attribute-based search tool 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 a pre-defined scale.

Although this particular form of context is effective for web search engines, it is unlikely to be as effective for user file systems for two reasons. First, user file collections are rarely shared among users, reducing the amount of feedback provided for each collection. Second, because users can generally find recently accessed files, they generally only search for old files, making file searches far less frequent than web searches.

External context: There also exists a significant amount of context that the user may have from events occurring outside of the system. For example, the physical location of the user (e.g., at the local coffee shop, in a scheduled meeting), the weather, significant news events, recent phone calls, etc., may all be clues that the user would like to use to find a particular file. Although this information is not directly available to the system, it may be possible to gather some of this information from external sources, such as calendar programs, web searches, wireless network roaming services, phone records, etc. We view such external context sources as an interesting area for future exploration.

3.2 Indirect attribute assignment

Until now, the area of indirect attribute assignment has been left entirely unexplored. This section outlines two schemes for propagating attributes among files found to be related: one that uses context and one that uses content. Sections 4 and 6 explore these in more detail.

User access patterns: As users access their files, the pattern of their accesses records a set of temporal relationships between files. These relationships have previously been used to guide a variety of performance enhancements (e.g., prefetching, cache hoarding [21, 26, 32]). Another possible use of this information is to help propagate information between contextually related files. For example, accessing "SemanticFS.ps" and "Gopal98.ps" followed by updating "related-work.tex" may indicate a relationship between the three files. Subsequently, accessing "related-work.tex" and creating "osdi-submit.ps" may indicate a transitive relationship.

Content similarity: Even within a single user's file space, there may be multiple copies or versions of a single file. For example, being unable to find a particular file, a user may re-download the file from the web to a new location. Each time the user re-downloads the file, he is likely to provide a different context, and thus a different set of attributes. By relating these copies of the file, the attributes from each copy can be shared, hopefully making the file easier to locate the next time the user searches for it.

4 Propagating attributes using access patterns

As users work, their pattern of file accesses represent temporal relationships between their files. These relationships form a picture of the user's context, capturing concurrently accessed files as a single contextual unit. Using these connections, the system can share attributes among related files, allowing classification of previously unclassifiable files.

This section evaluates algorithms for context-based attribute propagation. There are three goals of this evaluation. The first is to quantify the potential of these schemes to classify files, by measuring the increase in classifications over direct assignment using ideal content analysis. The second is to measure the effect of varying each aspect of the algorithm. The third is to get a feel for the accuracy of these relationships through a by-hand examination of the generated relationships.

4.1 Algorithm design

A number of existing projects have extracted file inter-relationships from access patterns to provide file prefetching and hoarding [3, 20, 22]. These algorithms convert the file system into a graph, with files at the nodes and each relationship specified as a weighted link between nodes. The weight of a link is the number of times that the relationship is seen, where "relationship" is traditionally defined as an access to the two files in sequence.

To better explore the space of options, our tool generalizes existing algorithms into one algorithm with five component parts: a graph style, a set of operation filters, a relation window, a stream extractor, and a weight cutoff. A graph-generation algorithm is formed by choosing a point along each of these five axes.

Graph style: The "graph style" specifies whether relationship links in the graph are directed or undirected. The direction of a link indicates the flow of attributes during indirect assignment, and is specified by the operation filters. Undirected links allow attributes to flow in both directions.

Operation filters: This component specifies the set of file system operations considered when generating the relationships. The filter specifies operations as input, output, or both. Input operations indicate the files that are the source of a relationship, while output operations indicate the files that are the sink of a relationship.

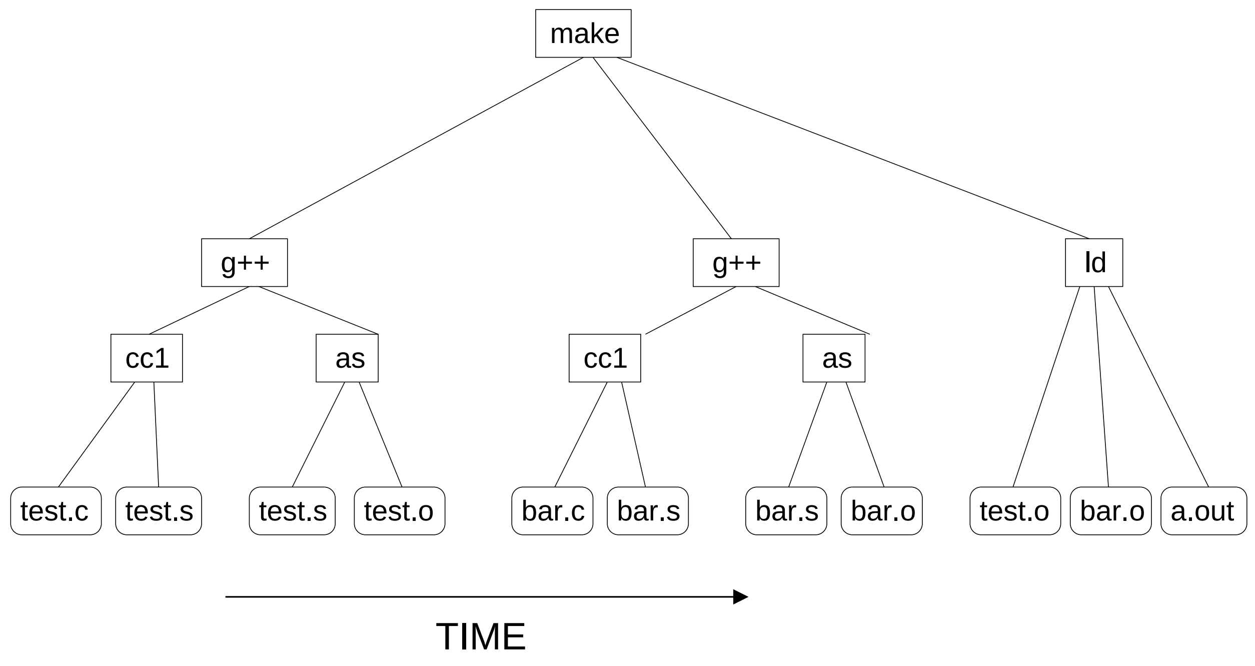


Figure 2: **Access tree.** This graphic illustrates the parent/child process tree and (along the bottom) the associated files accessed. In this example, a user runs *make* from the command line. This process then spawns a number of sub-processes that then access files. The advantage of this scheme over a simple PID-based scheme is that relationships across related programs are now captured. For example, "test.s" is accessed by both "cc1" and "as" and so the relationship between "test.c" and "test.o" is captured.

The tool provides three main filters and a layerable filter that is used in concert with one of the main filters. The *open* filter treats `open` calls as both inputs and outputs. The *read/write* filter treats `read` operations as inputs and `write` operations as outputs. The *complex* filter treats all access operations as inputs, including `mmap`, `open`, `read`, `stat`, and the source file from a `dup`, `link`, or `rename`. It treats all output operations as outputs, including `write`, `mknod`, and the destination file from a `dup`, `link`, or `rename`. The layerable *user* filter discards all operations that do not act on a file owned by the user performing the operation.

Relation window: The relation window tracks the current set of input files. When an input file is passed from the operation filter, it is placed into the window. When an output file is passed from the operation filter, it becomes related to the files in the window. The window size is defined as either a number of files (managed in LRU order) and/or a set period of time.

Stream extractors: This component separates independent streams of requests from the trace. Each stream maintains a separate relation window, meaning that operations from one stream will not affect the relationships generated by another stream.

The tool currently supports three stream extractors. The *Simple* extractor treats the trace as a single stream of requests. The *UID* extractor creates a separate stream of requests for each user ID in the trace. The *Access Tree* extractor [22] creates a stream of requests based on the file operations performed by a parent process and all of its sub-processes (as illustrated in Figure 2). The root of an access tree is a process spawned directly by the user or the system, and is currently defined as any process spawned by a user-interface (such as X-windows or bash) or init.

Weight cutoff: Once the graph is generated, the link weights determine which relationships provide indirect assignment. A simple scheme, such as following every relationship, is unreasonable, because the result is a large number of false positives (i.e., relationships with little contextual value). Unfortunately, cutting the links using a fixed weight is also problematic, because important relationships to less frequently accessed files are lost. For example, if a file is downloaded, printed, and then never accessed again, the

Machine Name	**Unique Files Written**	**Content-Classifiable**
bliss	11963	58%
cloudnine	2147	74%
donut	1618	65%
garfield	1035	78%
glee	6959	71%
merriment	2424	49%
nirvana	1180	98%
odie	1928	54%
rapture	1498	51%
sylvester	6984	98%

Table 1: **File system traces.** This table lists characteristics of each of the ten file system traces.

single weight links from that file would likely be cut, making it impossible to locate that file later.

Our solution is to only consider links whose weight makes up more than a certain percentage of a file's total outgoing or incoming weight. In this manner, lightly weighted links coming from or to files with few total accesses still receive indirect assignments, but only the most heavily weighted links are considered for frequently accessed files. This percentage is referred to as the *weight cutoff*.

4.1.1 Summary of graph-generation algorithms

Using these five components, we can capture the behavior of various existing cache hoarding and prefetching tools. For example, the model examined by Griffioen and Appleton [17] uses a directed graph style, the open filter, a fixed window size of 2, the simple stream extractor, and a 0% cutoff. The model examined by Lei and Duchamp [22] uses a directed graph style, the open filter, a fixed window size of 1, the access tree extractor, and a 0% cutoff.

The goal of this work is not to capture the exact next access given a particular file, but rather find several relationships that may be of use. As such, models that use very strict cutoff rules, such as last-successor, first-successor, and Noah [3], cannot be captured by our algorithms, although many of the underlying techniques are the same.

4.2 Experimental setup

As a starting point for evaluation, we assume that there exists a perfect content analysis tool that can provide useful attributes for all ASCII text (such as source code, html, raw text, etc.), Postscript, and PDF files. Our traces and experiments are on Linux systems. On MS Windows systems one would assume the tool could also classify file formats like Word and Powerpoint. Based on this, we partition files into *content-classifiable* and *content-unclassifiable*.

4.2.1 Traces

To capture user access patterns, ten researchers ran a specialized Linux kernel module on their desktop machines for a period of four months. On startup, the module dumps the state of all processes and open files, and during operation it tracks all file system and process management calls made in the system. Each entry in the trace keeps enough information to maintain the state of the entire system throughout the lifetime of the trace. Table 1 shows a summary of the ten traces.

4.2.2 Assumptions

Unless otherwise noted, all figures show only data for files owned by users. This is because we believe that the average user is only interested in finding files created by himself or other users, and have little interest in files created by the system (such as system logs, hardware configuration files, etc.). An examination of the results for all files, as opposed to only the user files, showed slightly lower increases in the percentage of classifiable files. This is because the system generated files make up most of these files, and less often have useful context.

Also, except in the cases of undirected links, only those files that are marked as output files by the operation filter are considered when calculating the number of classifiable and unclassifiable files. At some point during the lifetime of the system, each file must have been used as an output, and thus if the traces were available for that time period, they could have potentially been classified. Considering these files in the calculation without this information would be unfairly biased against these algorithms.

Once a graph is generated, attributes are propagated along the links to related files. Often, the connection between two files in a sub-graph is through other files (i.e., "foo.c" to "foo.o" to the "foo" executable). To capture this, attributes may be propagated more than one step from a classifiable file. By taking extra steps, it is possible to classify more and more of the files within a sub-graph, until the entire sub-graph is classified. Our figures either plot the steps taken along the x-axis, or use a specified number of steps.

4.2.3 Metrics

We consider three metrics when evaluating a graph-generation algorithm. First is the number of content-unclassifiable files that now receive attributes. Second is the number of content-classifiable files that now receive additional attributes. Third is the accuracy of the generated relationships.

To get a feel for the accuracy of the generated relationships, we examined the "donut" trace in detail[1]. Although not an exact science, this subjective evaluation provides critical insight into the success of the different algorithms.

4.3 Overall effectiveness

After exploring a large cross-product of different algorithm component values, we found a default scheme that consistently performed well based on our three metrics. The default scheme uses the UID extractor, directed graph style, read/write+user filter, two minute window, and a 1% cutoff.

Figure 3 shows the percentage of content-unclassifiable files that can be classified using the default scheme and taking up to two steps (in the file relationship graph) from content-classifiable files for each of the ten traces. The benefit is clear: more than 40% of all content-unclassifiable files can be classified in all cases, and more than 80% in half the cases.

Although the percentage increase in classifiable files is high, in two of the traces ("nirvana" and "sylvester") a large percentage of the files were already classifiable using content analysis. In these cases, attribute propagation still provides benefit, by adding additional attributes to files. Figure 4 shows the increase in attributes for content-classifiable files for each of the ten traces using the default algorithm. With the exception of one trace ("glee"), more than 70% of content-classifiable files receive extra attributes.

Examining the generated relationships shows most of them to be valid and useful. Although source code is generally well-structured, compiled source code was related to the generated ".o" and program files. For example, the code for the trace analysis tools was related to the tool itself. More importantly, the tool became related to its generated output (such as log files), which were usually stored in a less structured

[1]This machine was used by one of the authors, allowing a detailed understanding of both the files and the accuracy of the generated relationships.

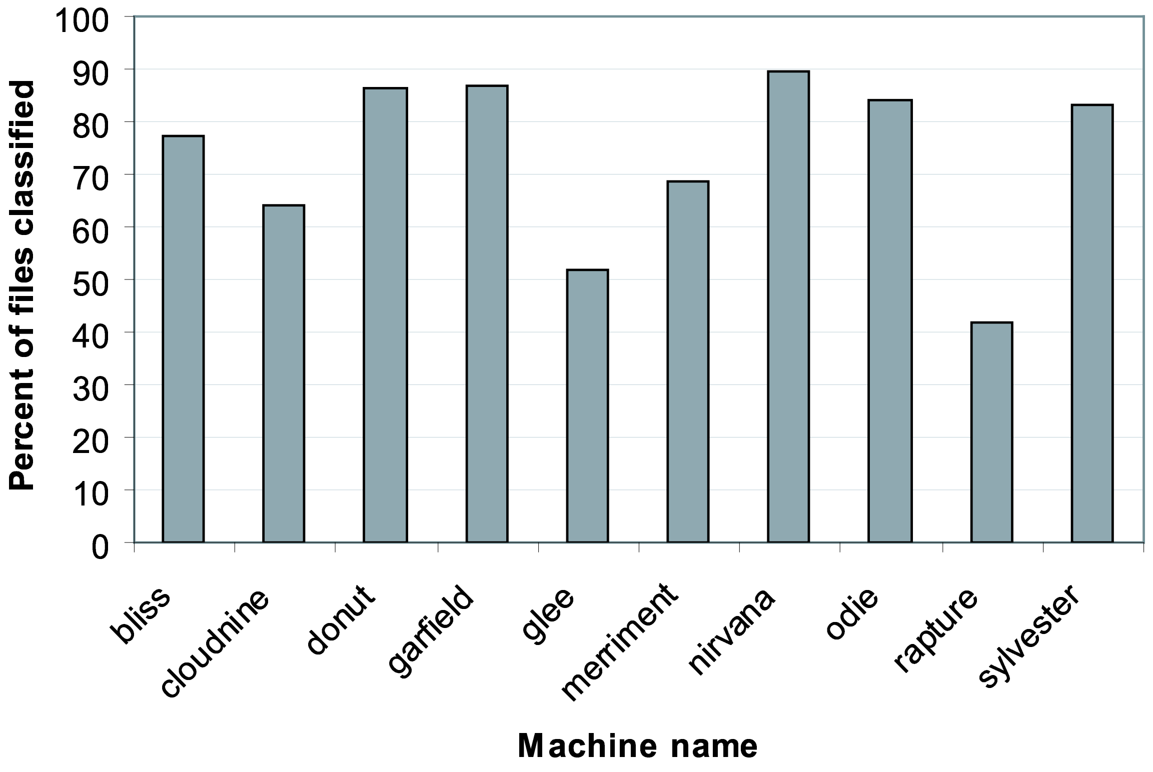


Figure 3: **Newly classified files.** This figure shows the percentage of content-unclassifiable files that can be classified by context-based propagation for each of the ten traces. The "rapture" and "glee" traces observed the smallest increase in classifiable files, because their users regularly downloaded pictures from their digital cameras (an action with no connection to content-classifiable files).

fashion. Also, additional copies of the tool created for separate simulation runs were also related to the source code through the original copy of the tool. Other examples of useful relationships we saw included instant message logs to files downloaded from links sent by friends, PDFs of related work to the files making up a paper being edited (and vice versa), and various files to old versions and backups.

4.4 Exploring the algorithms

To explore the trade-offs involved with each of the components of the graph-generation algorithms, we ran experiments varying each component of the default scheme. We present the results of the "donut" trace, as we have the best understanding of the accuracy of the relationships generated by this trace, although the presented results are similar for each of the other nine traces.

4.4.1 Weight cutoff

Figure 5 shows the effect of varying the cutoff percentage on the "donut" trace. As expected, increasing the cutoff percentage decreases the number of classifiable files, however, all cutoffs smaller than 2% perform extremely well, classifying over 30% of the content-unclassifiable files.

In addition to cutting some files off from classification altogether, increasing the cutoff percentage also makes the remaining sub-graphs more sparse. As a result, some files within large sub-graphs grow further

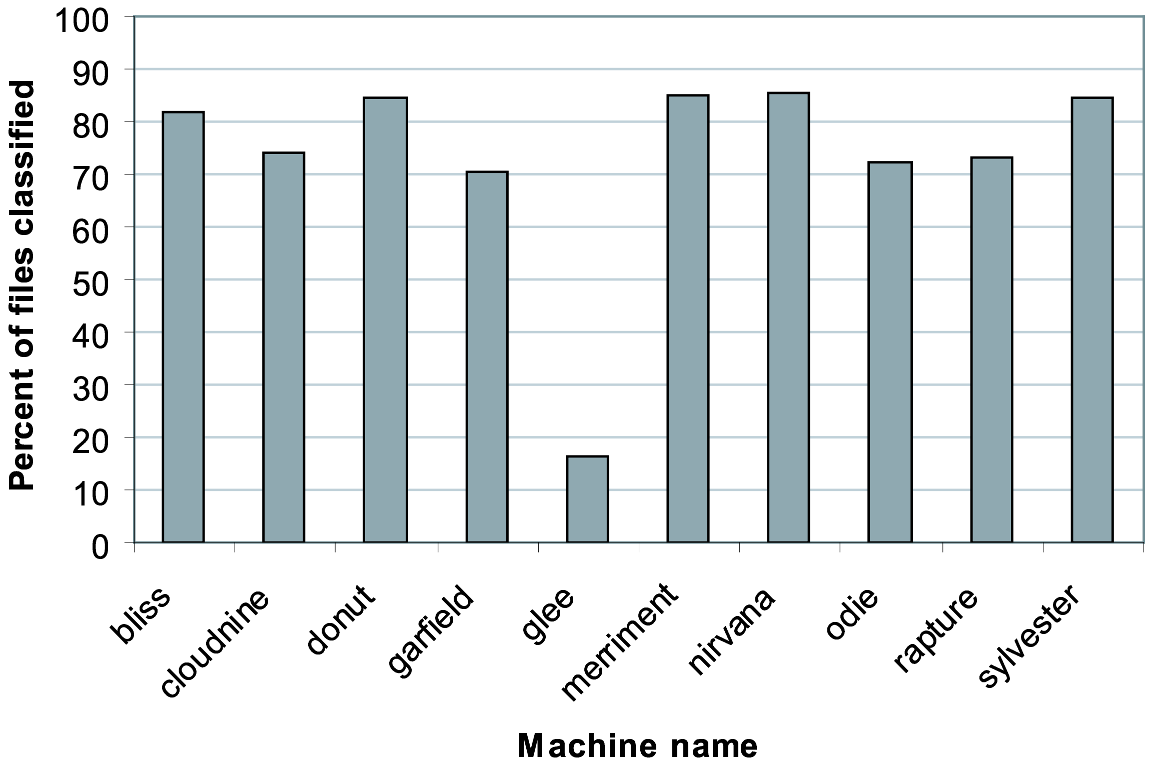


Figure 4: **Additional attributes.** This figure shows the percentage of content-classifiable files that receive additional attributes from attribute propagation for each of the ten traces. The "glee" trace had the fewest additional classifications because it had the smallest amount of trace data; that user performed a significant amount of his work on an untraced laptop machine while travelling during the four months of tracing.

away from the content-classifiable files, and thus it takes more steps for the attributes to be propagated to them. This can be seen by the difference in the maximum number of steps taken between the 2% and 5% cases: 3 and 6 respectively.

Even at a 0% cutoff there are still some unclassifiable files. These files are created by programs that generate data without using existing files as input. Examples of this kind of data are web log files, files transfered from another machine, or images retrieved from a digital camera. Although it would be possible to generate relationships for these files using a larger relation window, these relationships would likely be invalid. For example, if a user hasn't used his computer for more than an hour, and then downloads images from a digital camera, it is quite unlikely that the work done an hour ago is related to the images. Using an undirected graph style may help to classify some of these files, however, most of these files are either never accessed (such as the web logs), or only accessed as a group (examining the images from a trip) and thus have no outgoing links. The most likely way to provide attributes to these files is through other assignment schemes, such as application assistance.

For each cutoff percentage, we examined the set of unclassifiable files and compared it to the unclassifiable files from the next smallest cutoff (i.e., comparing the 2% cutoff to the 1% cutoff). This allowed us to see exactly which relationships were cut between different values of the cutoff. In our by-hand analysis, we found that that using a cutoff higher than 1% starts to give a large number of false negatives (removing file connections that are valid). We found that most of the connections between files at this cutoff appeared

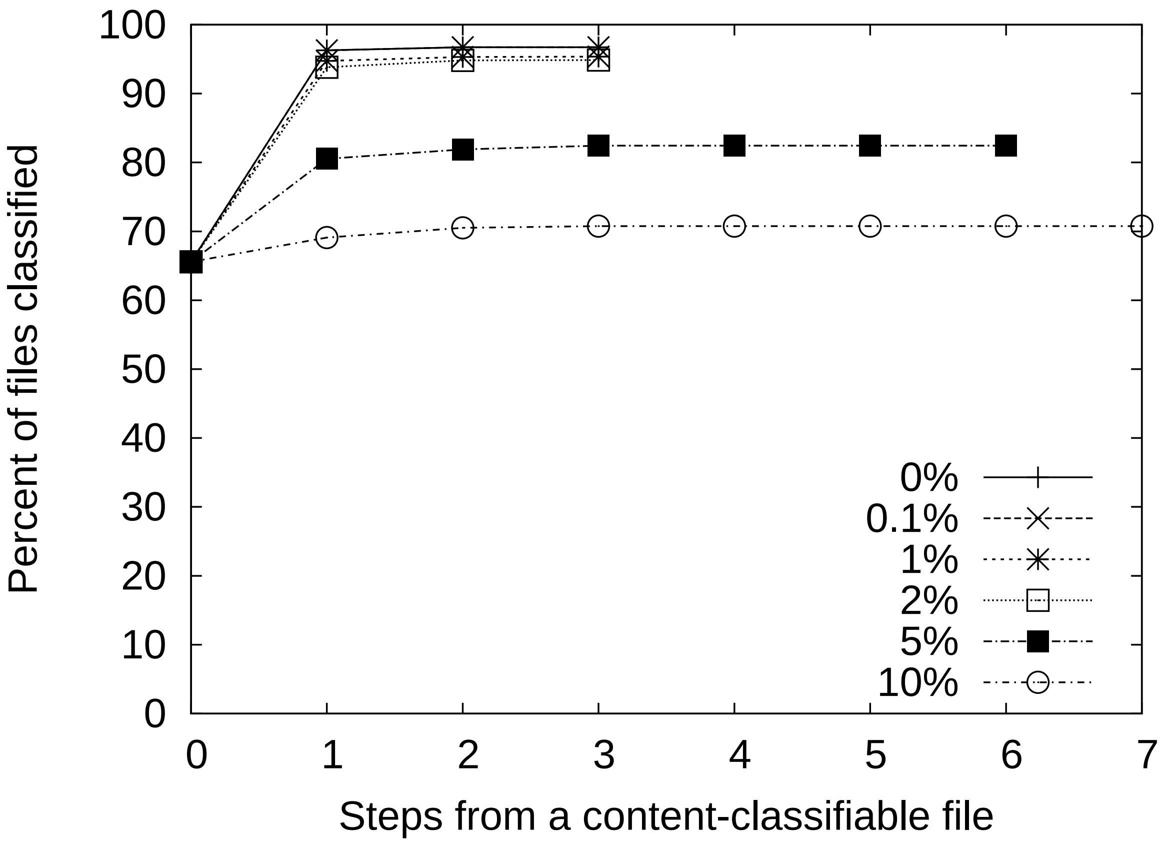


Figure 5: **Trade-off of the cutoff percentage.** This figure shows the percentage of classified files using a variety of cutoff percentages. The x-axis shows the number of steps from a content-classifiable file, where 0 indicates files that are content-classifiable. This graph was generated from the trace of the "donut" desktop machine.

valid, but that very low cutoff values resulted in many false positives. We also found that invalid connections become weaker as more data is gathered (i.e., as trace length grows) because by their nature they will occur less frequently.

4.4.2 Operation filters

Figure 6 shows the effect of the different operation filters on the "donut" trace. Although there are six possible operation filters, Figure 6 only reports the results for four of them. This is because the choice of filter has a large affect on the number of links created, and thus the amount of memory required to store and analyze the results. Using the *open* and *complex* filters alone, each resulted in close to 100 million links between files over the four month trace, requiring more than the 3 GB of memory available to processes in Linux. We are currently examining schemes to either reduce the memory required to examine the traces or store portions of the data out-of-core.

The four schemes shown in Figure 6 all perform well in terms of quantitative classification, increasing the number of classifiable files by 25–30%, with the read/write filters performing the best. A by-hand examination of the classifications generated by the read/write filters also show them to be the most accurate. The open filter makes no distinction between files opened for writing, and files opened for reading. Because files opened for writing had far fewer connections, many of the connections outgoing from these files were inaccurate. The complex filter provided very similar results to the read/write filter, however, files accessed using

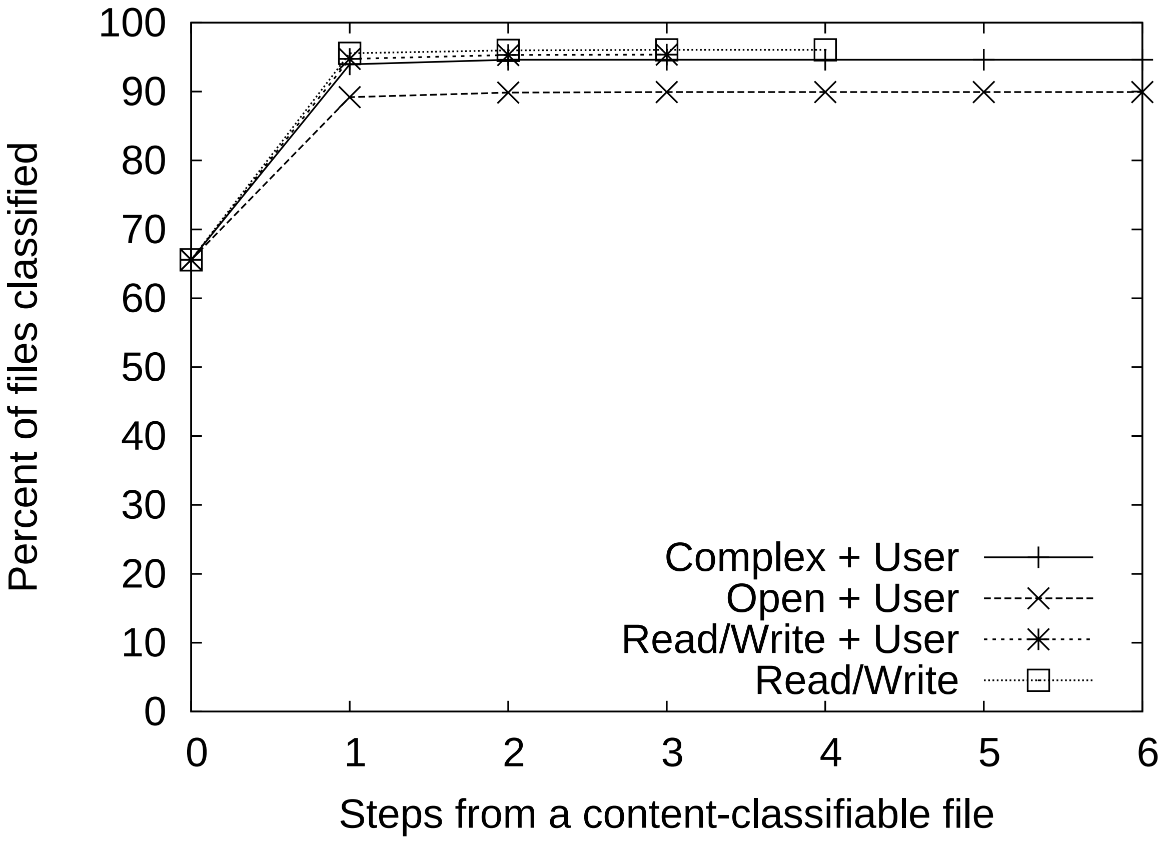


Figure 6: **Trade-off of the filter scheme.** This figure shows the percentage of classified files using each of the six filtering schemes. The x-axis shows the number of steps from a content-classifiable file, where 0 indicates that files that are content-classifiable. This graph was generated from the trace of the "donut" desktop machine.

the `stat` syscall were often unrelated to files written later in the window, resulting in invalid connections.

An examination of the read/write and read/write+user filters shows that they produce nearly identical classifications. In a desktop's single-user environment, it is rare that a user accesses another user's files (generally only the "system" user during background tasks). The result is that these infrequent events have little effect on classification.

4.4.3 Graph style

An examination of the ten traces found that there were no cases of directed links from unclassifiable to classifiable files. The result is that changing links from directed to undirected has no affect on the number of classifiable files. Using undirected links, it is possible to pass additional attributes among already classifiable files, although in all ten traces less than 25% of the classifiable files received extra attributes.

4.4.4 Stream extractor

Figure 7 shows the effect of the different stream extractors on the "donut" trace. The access tree extractor classifies fewer files than the others, because useful connections can also be formed among files from different applications. The simple and user extractors perform identically. This is because a desktop's single-user workload results in no extra relationships being formed between files of differing users. Background tasks

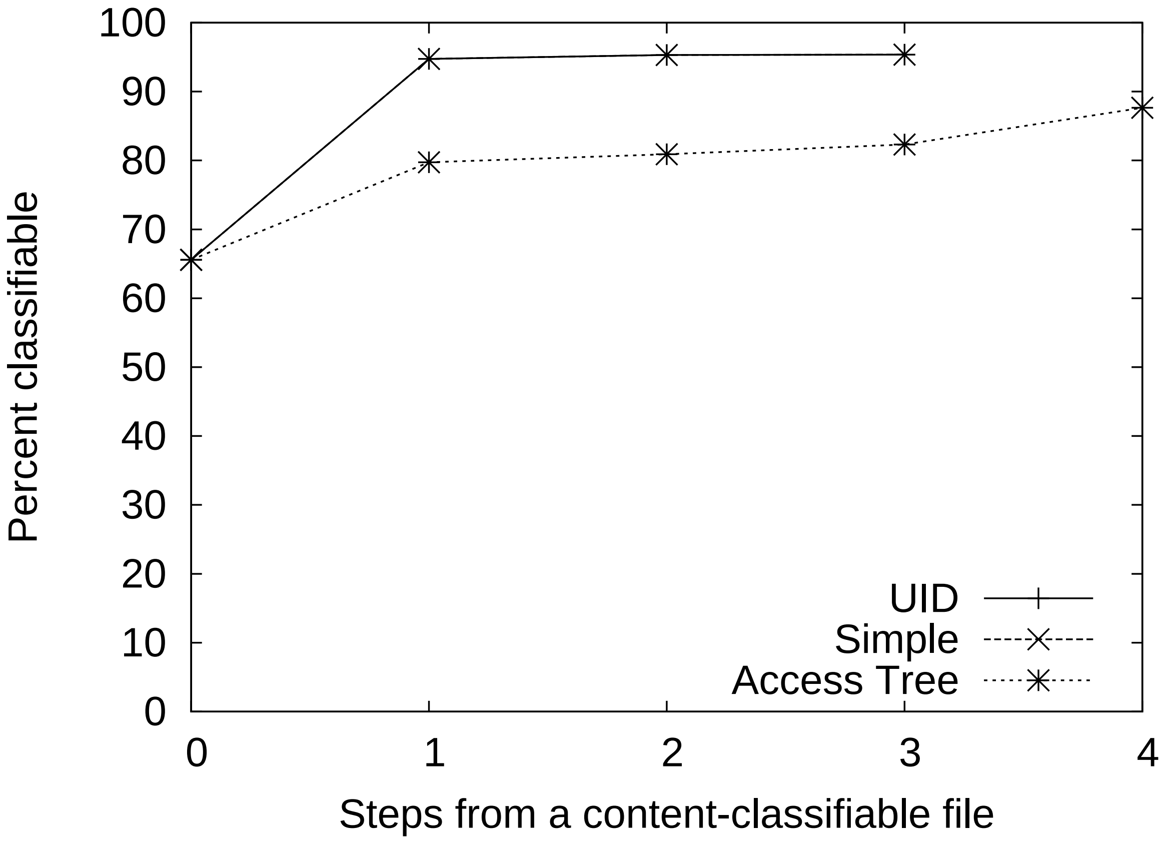


Figure 7: **Trade-off of the chosen stream extractor.** This figure shows the percentage of classified files using each of the three stream extractors. The x-axis shows the number of steps from a content-classifiable file, where 0 indicates files that are content-classifiable. This graph was generated from the trace of the "donut" desktop machine.

accessing files concurrently with the user occur infrequently enough that such links are ignored (due to weight cutoff).

4.4.5 Relation window

Figure 8 shows the effect of varying window sizes on the "donut" trace. The interesting aspects of this graph are at the two extremes of the window sizes. The largest window sizes (5 and 10 minutes) classify the fewest files. This is because the increased number of links are eventually cut as invalid by the 1% weight cutoff. At the smallest window size, 10 seconds, the first step of classification provides fewer attributes than the 1, 2, and 3 minute windows. This is because the window provides so few connections as to limit even potentially valid direct connections between files. Taking additional steps can renew some of these connections by stepping through an intermediary file, however, a comparison of the remaining unclassified files for the 10 second and 2 minute windows shows that taking additional steps does not always create the same classifications. Instead, because the 10 second window does not have enough information, it creates and follows links between less strongly connected files. As a result, a window size between 1 and 3 minutes appears to most accurately capture the set of files that make up the user's context at that time.

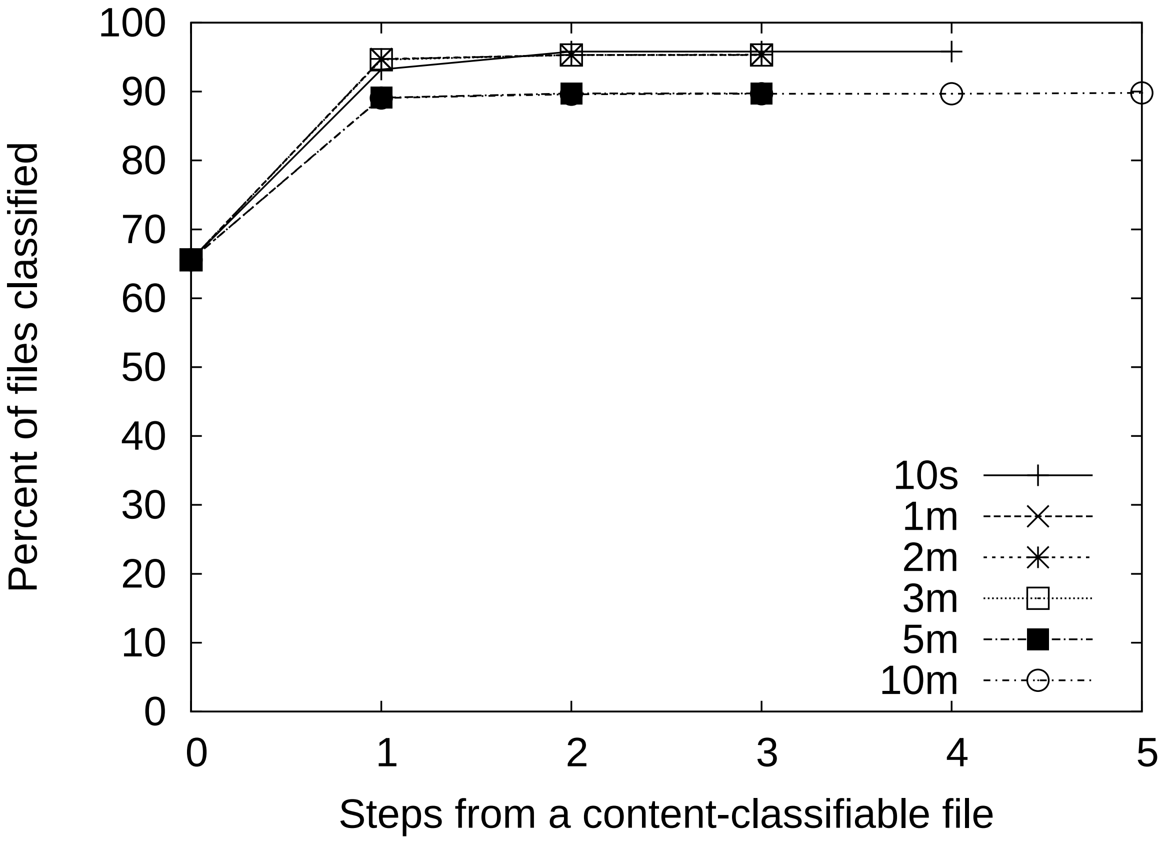


Figure 8: **Trade-off of the window size.** This figure shows the percentage of classified files using a variety of window sizes. The x-axis shows the number of steps from a content-classifiable file, where 0 indicates files that are content-classifiable. This graph was generated from the trace of the "donut" desktop machine.

5 Application assistance using web browsing

Because most applications are designed for specific tasks, they are uniquely suited to provide attributes describing what their user is currently doing. An examination of the traces indicates that web-browsing is one of the most frequently performed user activities. As such, it has potential to gather a significant amount of user context. We examine two methods of gathering context information from a web browser: gathering the context of downloaded files from recent user activity, and marking periods of time with the context of the user using recently browsed pages.

5.1 Algorithm design

As a user browses the web, the pages he visits form trees of activity, with attributes added along each link. Pages that are accessed directly (e.g., bookmarks, direct entry, external programs) act as root nodes. Each page request generated from a given page (either through a hyperlink or form) acts as a child to that page. The link between a parent and its child is marked with the attributes of the request: either the text of the hyperlink or the submitted form data. The attributes for a given page are determined by tracing the links from the page to the root and gathering all of the attributes along the path. Downloaded files are treated as leaves in the tree, and as such, are assigned attributes by following the path to the root.

Periods of browsing activity are marked using a windowing scheme similar to the relation window described in Section 4. When a page is accessed, its attributes are placed into the window. Output files (as specified by the system's operation filter) accessed within this window are assigned all attributes in the window.

5.2 Experimental setup

To gather the information about user web activity, we instrumented a module for the Mozilla web browser to keep a trace of web activity. One of the ten researchers (the user of the "donut" system) ran the tracing tool for two of the four months of file system tracing.

To make use of more of the available data, we compared the set of downloaded files from the web trace against the set of files created by Mozilla in the file system trace for those two months (not including those files created in the browser-specific ".mozilla" directory), and found an exact match. We also confirmed that no files in the web trace were downloaded without the user having first clicked on at least one hyperlink. As such, we consider it safe to assume that any such files created in any of the file system traces could be considered files downloaded by Mozilla, and would receive additional attributes from application assistance.

5.3 Overall effectiveness

To evaluate the effectiveness of the downloading scheme we ran the tool over four of the ten file system traces[2]. To evaluate marking periods of web activity, we combined the existing web trace with the matching two months of file system tracing.

5.3.1 File downloads

Figure 9 provides a summary of the downloaded files for four of the traced machines. It shows the total number of files downloaded over the four month period, as well as a breakdown of the file type of the downloaded files. This breakdown is based on a file's likelihood of being content-classifiable; text, Postscipt, and PDF files are considered content-classifiable, while "unclassifiable" includes all remaining files, mostly consisting of images, video, music, and compressed archives.

One disappointing aspect of this data is that few of the files created by the users were downloaded from the web. However, the files downloaded off the web are often some of the least well organized in the user's system. An examination of the pathnames of downloaded files found that between 40% and 95% of the files downloaded were placed into a single directory, often either the user's home directory or a personal directory called "downloads" or "tmp." This indicates that the additional attributes generated by application assistance could be quite valuable to the user later when performing a search.

Although these files make up a small fraction of the created files, their application-applied attributes could be propagated to many additional files via the inter-file relationships generated by user access patterns. Figure 10 shows the percentage of files classified by combining these schemes. Although combining these extra attributes with content-based attributes has little effect on the total number of classifiable files, it can increase the number of useful attributes per file. This represents an added benefit of application assisted attributes.

Finally, by-hand examination of these assignments shows them to be quite relevant. For example, one file saved to a user's home directory named "KA2.mpg" was given the attributes "Honda car commercial." Another file saved to the home directory named "gw.zip" appeared to be related to both "guild wars" and "demo." Although unable to remember the contents of either file initially, when presented with the keywords, the user was able to indicate that they were quite accurate in both cases.

[2]The users of the other six traced machines downloaded no files using the Mozilla web-browser. Some used other browsers, and some performed no downloading.

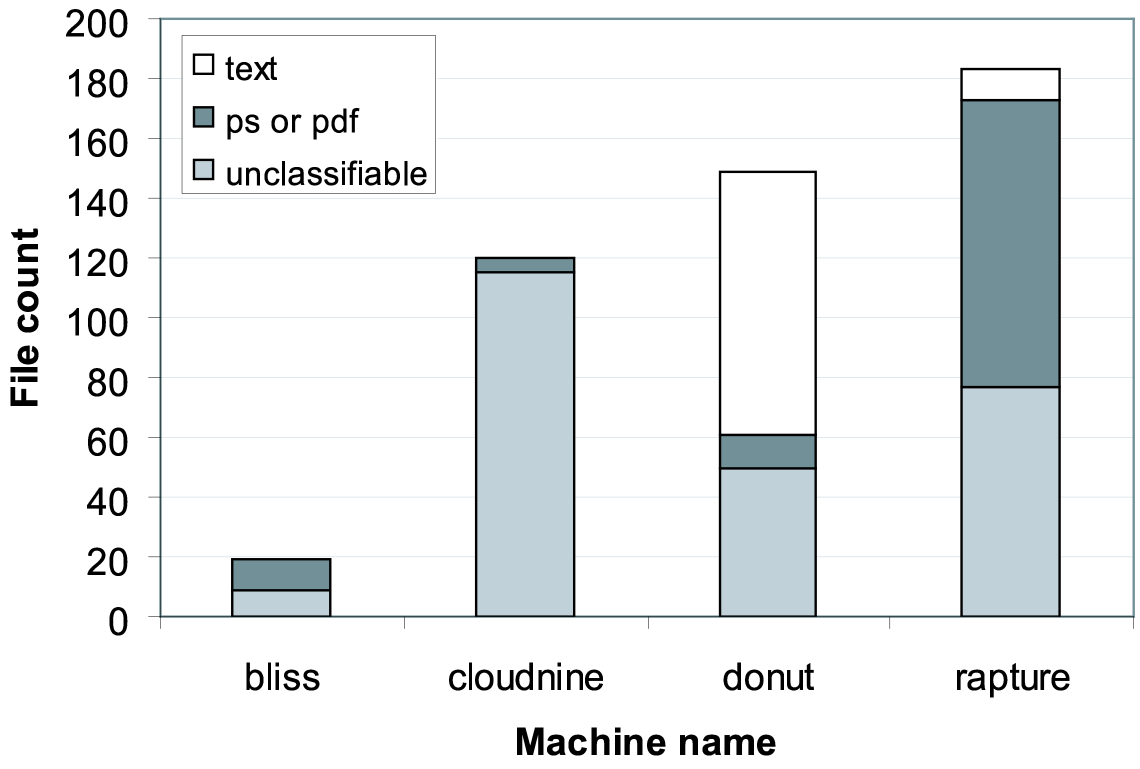


Figure 9: **Web downloads.** This figure shows a count of files downloaded on four different desktop machines over a period of four months. The "text" and "ps or pdf" bars for each machine shows the number of downloaded files that can be classified using traditional content analysis schemes. The "unclassifiable" bars are the remaining, unclassified files that were downloaded.

5.3.2 Web activity

By marking periods of web activity with context attributes, it was possible to classify 53 files, including 36 previously unclassifiable files. Figure 11 shows the result of combining this scheme with user access pattern propagation over the two months of available traces. Similarly to the propagation of downloaded file attributes, combining this small set of files with indirect attribute assignment provides a large set of classifiable files.

Unfortunately, unlike previous schemes, many of the classifications made by this scheme seem inaccurate. This appears to stem from the user's behavior of browsing the web during program compilations. The result is that many source and program files become associated strongly with a variety of news websites. One way to try and prevent this is to classify the user's web sites into categories based on usage patterns. If a user frequents a particular website during a variety of activities, then it is probably not tied to any particular one of these. However, if a user performs a directed search or visits a website only in correlation with a particular task, then they could be strongly connected. Experiments with this heuristic (and others) will be a useful direction for future work.

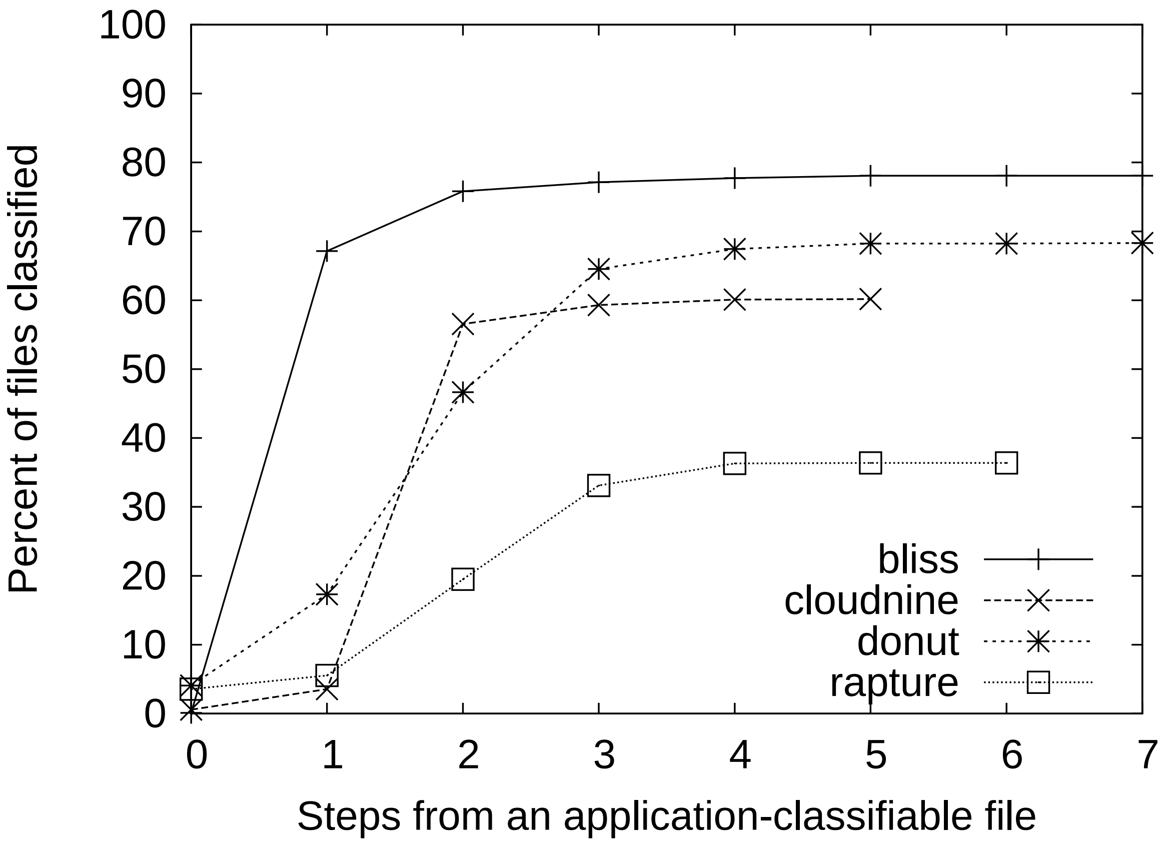


Figure 10: **Attribute propagation from web downloads.** This figure shows the effect of combining the attributes generated for downloaded files with user access pattern propagation for the four machines shown in Figure 9. The y-axis shows the percentage of files classified and the x-axis shows the number of steps taken from a downloaded file.

6 Propagating attributes using content similarity

To get a feel for the potential of relationships formed by content similarity, we ran an experiment to find all duplicate copies of files owned by the user of the "donut" desktop system. To do so, we generated an md5 checksum [29] of all files owned by the user and then found duplicate checksums that also had identical file sizes. Although possible, the likelihood of a collision between two different files is extremely low.

The experiment located 2,644 sets of identical files, with an average set size of 4.54 files. Using these additional connections, it was possible to classify an additional 2% of files beyond those unclassifiable by content, application assistance, or user access patterns, advancing the total classification to 97% (from 95%).

The largest sources of duplicate files came from duplicate or versioned source trees, CVS generated duplicates, and apparently re-downloaded or hand copied files. This last set of files is of particular interest, as these files are "organized" by the user. An example of this is two identical files we found in the user's home directory, "submit.ps" and "usenix.final.ps," that were created approximately two hours apart. Interestingly, when asked about the files, the user was able to remember the context in which the files were created, but was unable to recall the exact title of the paper in question.

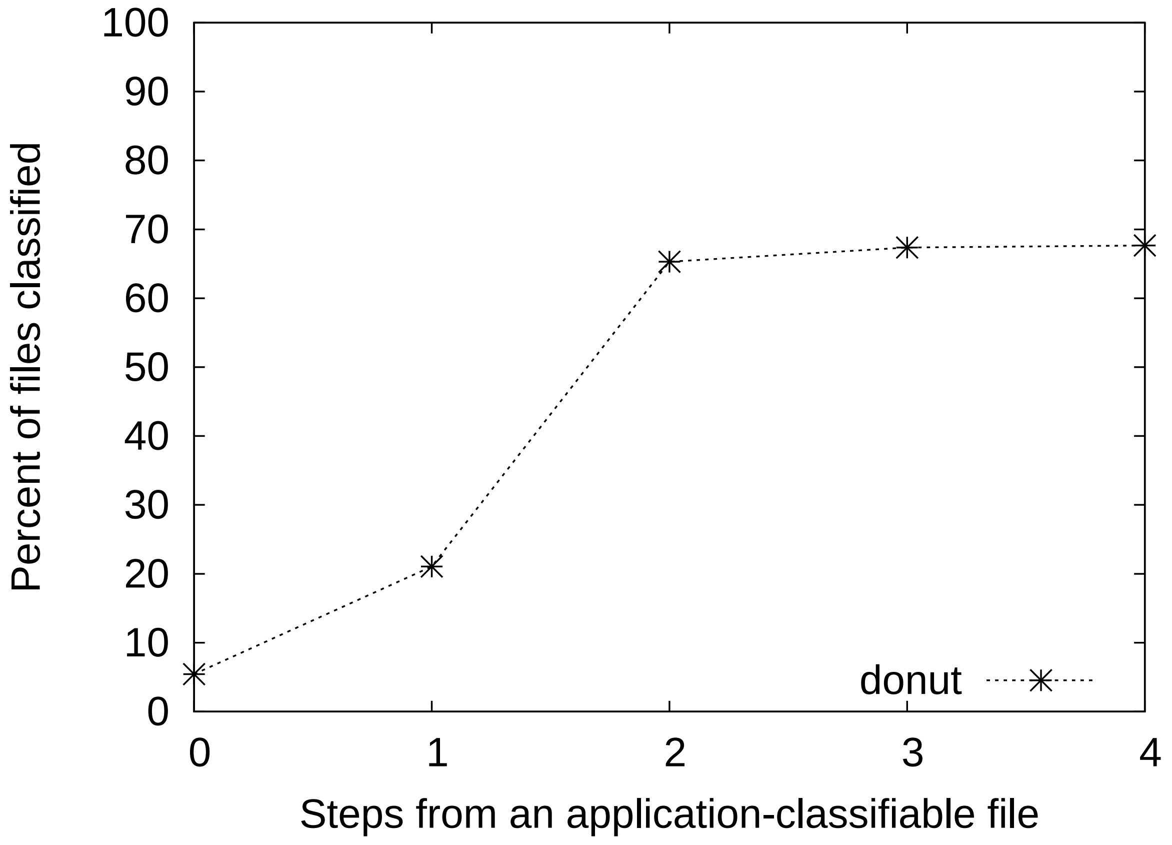


Figure 11: **Direct attribute assignment based on concurrent web browsing.** This figure shows the effect of combining the attributes generated from concurrent web browsing with user access pattern propagation using the "donut" trace. The y-axis shows the percentage of files classified and the x-axis shows the number of steps taken from a downloaded file.

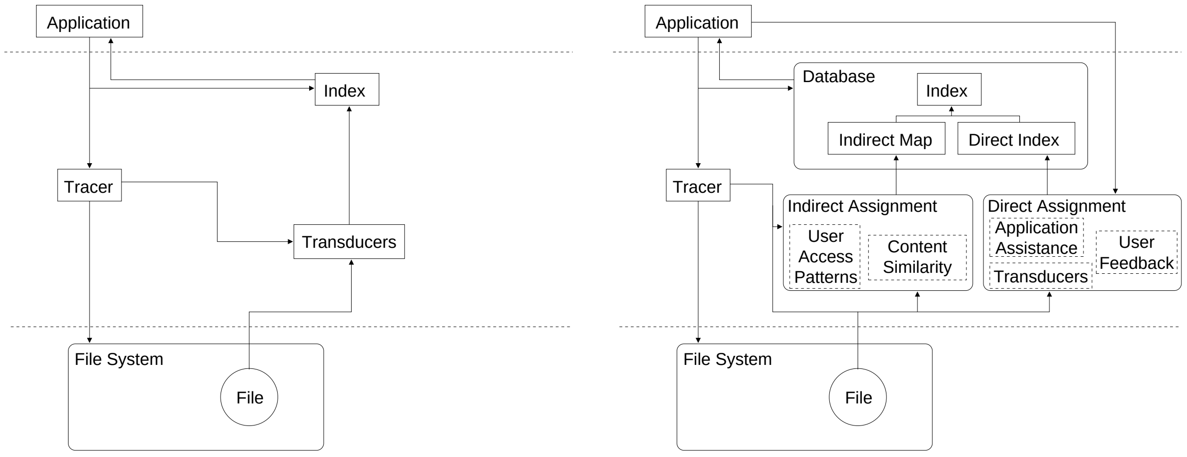
7 System architecture

This paper takes a critical first step towards the use of automated context-based attribute assignment in semantic file systems, by developing and evaluating algorithms for doing so. The next step is to deploy, and gain real user experiences with, such attributes in a real semantic file system. This section describes the architecture of such a system and how the tools described in Sections 4, 5, and 6 fit into it.

7.1 Extensions to traditional semantic file systems

Figure 12a shows the basic design of most semantic file systems (simplified from [14]). The system traces application requests to watch for file updates, and potentially to overlay its indexing system on the existing file system interface by extending the namespace [14, 16]. When a file is updated, the tracer notifies its transducer, which reads the file and updates its attributes in the index. Applications query the index using either a separate interface or a namespace extension.

Figure 12b shows a diagram of a prototype system for indexing and searching attribute information that incorporates the various tools for automatic attribute assignment described Sections 4, 5, and 6. As before, application requests are routed through the tracer, but are now passed both to the direct assignment tools (to notate file updates) and to the indirect assignment tools (to trace user access patterns and notify the content similarity tool of updates). Applications may also provide attributes through application assistance or user



(a) Traditional semantic file systems.

(b) Our prototype system.

Figure 12: **System design.** This figure shows both the traditional design of semantic file systems, and the design of our new prototype system. The key difference in our prototype system is the introduction of the new attribute assignment tools described in Sections 4, 5, and 6. These tools fall into the two categories of indirect and direct assignment, and their different properties also affect the design of the indexing database.

feedback. These parts of the system map directly onto the tools described in this paper, and make up the bulk of the prototype system.

The database maintains three separate indices. The direct index stores direct attribute assignments, while the indirect map stores the inter-file relationships. The main index stores the combined attribute assignments after applying inter-file relationships to the direct attribute assignments.

Although not discussed in our evaluation, there are both computational and memory overheads for running these tools, most significantly for the tools that generate inter-file relationships from user access patterns. A single 4-month simulation takes approximately 2 hours and ranges between using 1 and 3 GB of memory. However, this is not representative of how a system would use these tools. More likely, indexing would be performed each night, and relationships would be stored out-of-core in the database. Running the tool over a single day of traces requires only a few minutes, and takes very little memory (dependent upon the amount of activity performed during that day, but generally less than 100MB). The result is that the performance overheads of these tools are likely to be as little or less than current content-based indexing.

7.2 Challenges and trade-offs

The addition of so many and varied attribute generation schemes introduces challenges to the design of a complete system. Most significant are the increase in available attributes for all files, and the introduction of attribute propagation schemes. This section discusses how these changes affect the unbuilt portions of the system: the search interface and the attribute database.

7.2.1 Search interfaces

In most existing file search systems, when a user searches for a particular keyword, he is presented with a list of all matching files. With a dramatic increase in the number of attributes assigned to each file, file search tools will be presented with a problem similar to that of web search tools: too many results. To deal with this problem, web search tools assign weights to the keywords for each page. File search tools will require a similar solution, allowing the user interface to order search results by weight.

The addition of weights introduces two secondary problems. First, the weights generated by an attribute assignment tool must be normalized against the other tools in the system, to prevent a single tool from overriding the attributes generated by other tools. This normalization could be agreed upon in advance by all tools, or it could be managed by the system using the average weighting of each tool. Second, the database must maintain a separate weight by tool for each ⟨attribute,file⟩ pairing, simplifying normalization in the case of more than one tool generating the same attribute for a file.

7.2.2 Managing indirect attributes

The key challenge of indirect attribute assignment is to avoid *re-propagation*. For example, if an attribute is passed along two steps of a sub-graph from file “A” to file “B” during the indexing phase, it must not be re-propagated two steps from file “B” during a later indexing phase (and thus 4 steps in total). To avoid this, the database should maintain a separate mapping for indirect attribute assignment. This can then be re-combined with any directly assigned attributes during an indexing phase.

This separate indirect map has two potential implementations: either the map stores the attributes propagated at the time the relationship is generated, or the map stores the relationship between the files and re-gathers the propagated attributes during each indexing phase. Each approach has potential merit. By storing the attributes, the map captures the context relationship between the files at the exact time the relationship was formed. By storing the relationship, the map may capture new attributes that are valid to both files with additional direct attribute assignments. Comparing these two schemes is an area of ongoing work.

8 Conclusion

The addition of context-based attributes can make semantic file systems more effective, by tackling their primary shortcoming: the unavailability of quality attributes. Context attributes capture the state of users while they accessing their files, a key way that users recall and search for data [2]. Combining context and content attributes with attribute propagation increases both the number of classifiable files, and the number of attributes assigned to each file. The result is a more informed system that has a far better chance of successfully helping users locate specific files when desired.

Acknowledgments

We thank Andy Edmonds for his help with the design and implementation of the Mozilla tracing tool used in this work.

References

[1] AltaVista, http://www.altavista.com/.

[2] Christine Alvarado, Jaime Teevan, Mark S. Ackerman, and David Karger. *Surviving the information explosion: how people find their electronic information*. Technical report AIM-2003-006. Massachusetts Institute of Technology, April 2003.

[3] Ahmed Amer, Darell Long, Jehan-Francios Paris, and Randal Burns. File access prediction with adjustable accuracy. *International Performance Conference on Computers and Communication* (Phoenix, AZ, April 2002). IEEE, 2002.

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

[5] 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.

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

[7] Joachim M. Buhmann, Jitendra Malik, and Pietro Perona. Image recognition: Visual grouping, recognition, and learning. *Proceedings of the National Academy of Sciences*, **96**(25):14203–14204. National Academy of Sciences, December 1999.

[8] Douglass R. Cutting, David R. Karger, Jan O. Pedersen, and John 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.

[9] Roger B. Dannenberg. Listening to Naima: an automated structural analysis of music from recorded audio. *International Computer Music Conference* (San Francisco, CA, 2002), pages 28–34. International Computer Music Association, 2002.

[10] John R. Douceur and William J. Bolosky. A large-scale study of file-system contents. *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (Atlanta, GA, 1–4 May 1999). Published as *ACM SIGMETRICS Performance Evaluation Review*, **27**(1):59–70. ACM Press, 1999.

[11] Scott Fertig, Eric Freeman, and David Gelernter. Lifestreams: an alternative to the desktop metaphor. *ACM SIGCHI Conference* (Vancouver, British Columbia, Canada, 13–18 April 1996), pages 410–411, 1996.

[12] Fractal:PC, http://www.fractaledge.com/.

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

[14] David K. Gifford, Pierre Jouvelot, Mark A. Sheldon, and James W. O'Toole Jr. Semantic file systems. *ACM Symposium on Operating System Principles* (Asilomar, Pacific Grove, CA). Published as *Operating Systems Review*, **25**(5):16–25, 13–16 October 1991.

[15] Google, http://www.google.com/.

[16] Burra Gopal and Udi Manber. Integrating content-based access mechanisms with hierarchical file systems. *Symposium on Operating Systems Design and Implementation* (New Orleans, LA, 22–25 February 1999), pages 265–278. ACM, 1999.

[17] James Griffioen and Randy Appleton. *The design, implementation, and evaluation of a predictive caching file system*. Technical Report CS–264-96. University of Kentucky, June 1996.

[18] Grokker, http://www.grokker.com/.

[19] James J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems*, **10**(1):3–25. ACM Press, February 1992.

[20] Geoffrey H. Kuenning. *Seer: predictive file hoarding for disconnected mobile operation*. Technical Report UCLA–CSD–970015. University of California, Los Angeles, May 1997.

[21] Geoffrey H. Kuenning and Gerald J. Popek. Automated hoarding for mobile computers. *ACM Symposium on Operating System Principles* (Saint-Malo, France, 5–8 October 1997). Published as *Operating Systems Review*, **31**(5):264–275. ACM, 1997.

[22] H. Lei and D. Duchamp. An analytical approach to file prefetching. *USENIX Annual Technical Conference* (Anaheim, CA, January 1997). USENIX Association,, 1997.

[23] Lycos, http://www.lycos.com/.

[24] Udi Manber and Sun Wu. GLIMPSE: a tool to search through entire file systems. *Winter USENIX Technical Conference* (San Francisco, CA, January 1994), pages 23–32. USENIX Association, 1994.

[25] Michael L. Mauldin. Retrieval performance in Ferret a conceptual information retrieval system. *ACM SIGIR Conference on Research and Development in Information Retrieval* (Chicago, IL, 1991), pages 347–355. ACM Press, 1991.

[26] Gokhan Memik, Mahmut Kandemir, and Alok Choudhary. Exploiting inter-file access patterns using multi-collective I/O. *Conference on File and Storage Technologies* (Monterey, CA, 28–30 January 2002), pages 245–258. USENIX Association, 2002.

[27] Merriam-Webster OnLine, http://www.m-w.com/.

[28] Dennis Quan, David Huynh, and David R. Karger. Haystack: a platform for authoring end user semantic web applications. *International Semantic Web Conference* (Sanibel Island, FL, 20–23 October 2003), 2003.

[29] Ronald L. Rivest. The MD5 Message-Digest Algorithm. RFC 1321.

[30] Stuart Sechrest and Michael McClennen. Blending hierarchical and attribute-based file naming. *International Conference on Distributed Computing Systems* (Yokohama, Japan, 9–12, June 1992), pages 572–580, 1992.

[31] Joao Sousa and David Garlan. Aura: an architectural framework for user mobility in ubiquitous computing environments. Pages 29–43.

[32] Stephen 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.

[33] Carl Tait, Hui Lei, Swarup Acharya, and Henry Chang. Intelligent file hoarding for mobile computers. *International Conference on Mobile Computing and Networking* (Berkeley, CA, 13–15 November 1995), pages 119–125. ACM, 1995.

[34] Yahoo!, http://www.yahoo.com/.