LEVERAGING CONTEXT FOR FILE SEARCH AND ORGANIZATION

by

Samir Shah

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2009

Doctoral Committee:

Associate Professor Brian D. Noble, Chair
Professor Atul Prakash
Associate Professor Mark S. Ackerman
Assistant Professor Lada A. Adamic

ACKNOWLEDGMENTS

I can no other answer make, but,
thanks, and thanks.
William Shakespeare (1564–1616)

This dissertation would not have happened without the help of a number of people. I extend my thanks to:

First and foremost, my Ph.D. advisor Brian Noble. He taught me how to do research. Enough said.

The other members of my committee: Mark Ackerman, Lada Adamic, and Atul Prakash. They provided valuable feedback that undoubtedly enhanced this dissertation.

My co-authors and collaborators: Greg Ganger, T.J. Giuli, and Craig Soules. Each of them helped shape this research for the better.

My research group: Mark Corner, Landon Cox, Minkyong Kim, James Mickens, and Anthony Nicholson. A lot of nonsense, but many great ideas. Always fun.

The administrative staff: Kirsten Knecht, Steve Reger, and Bert Wachsman.

My family, particularly my parents.

Holly, who'll be my frog forever.

Friends and all the great people I met during my stay in Ann Arbor, who are too numerous to list. It's been wonderful.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

ABSTRACT

Ideally, a user's system would automatically organize, locate, and manage their data. Unfortunately, this ideal is currently elusive, as evidenced by the considerable effort users expend in organizing and finding their data, and the plethora of tools available helping them to do so.

Context—the set of facts and features surrounding data—is a promising aspect to aid in this information management problem. For instance, a user's system can observe his or her actions to determine which data is pertinent to other data, reordering and extending results during searches. This dissertation describes and evaluates such a scheme, which uses causally-related inputs to outputs to drive its contextual index. A field study with a prototype of this architecture indicates that the system increases user-perceived satisfaction in search.

Unfortunately, while this context proves useful, it is only maintained locally. A user's social neighborhood, however, can provide significant advantages for the collaborative sharing of context. Challenges arise, as naïvely adding context sharing to a system complicates it from the user's point of view: it becomes a new entity to manage. As well, any such mechanism must also respect personal boundaries on disclosure.

This work advocates leveraging acts of sharing to imbue context access rights. With this system, context is encapsulated in frames of reference, and these frames follow objects as they are shared. Frames render a separation of concerns: a recipient holding one particular frame cannot see the context residing in a frame not disclosed. Frames can be split and merged as contexts and circumstances change. To support the ad-hoc, spontaneous collaborations that occur naturally between individuals, the system eschews vestiges of centralized control and its eventual-consistency model aids mobile and disconnected users. A user study suggests this framing scheme is palatable to users and a prototype demonstrates contextual exchange induces negligible performance overheads.

This research aims to help ameliorate the information management problem and to facilitate the formation of ad-hoc, spontaneous collaborative groups.

CHAPTER 1

INTRODUCTION

> *If Edison had a needle to find in a haystack, he would proceed at once with the diligence of the bee to examine straw after straw until he found the object of his search...I was a sorry witness of such doings, knowing that a little theory and calculation would have saved him ninety per cent of his labor.*
>
> Nikola Tesla (1856–1943)

As storage capacity continues to increase, the number of files belonging to an individual user, whether a home or corporate desktop user, has increased accordingly [20, 62]. This information has become increasingly hard to manage, find, and retrieve as its scope has grown. The principal challenge is no longer efficiently storing this data, but rather organizing it. As data spans the boundaries of multiple users, machines, and administrative domains, the task becomes increasingly challenging still. This is especially true with spontaneous, collaborative sharing, which can include anything from a document for review to family photos. While distributed file systems provide global naming of persistent objects, allow universal access across clients, and ensure reasonable access control, users often turn to other mechanisms, such as email [98], to coordinate data sharing.

To address these challenges, researchers have turned to *context* [21]—features that describe the creation, use, and classification of data—as it holds significant promise for simplifying the organization, retrieval, and management of data. Indeed, a recent study showed that most users organize and search their repositories using context [90]. Such techniques include sticky notes [43], user tagging [31], location tagging [77], semantic links [3] and attributes [29], provenance identification [63], recent history [23], or—as explored in this research—context-enhanced personal desktop search [85].

The notion that individuals will organize their repositories is flawed, as most users cannot appreciate the long-term benefit of this organization [51]. Ad hoc sharing, coupled with users' laissez-faire attitude in organizing their files, often leads to disorganized data repositories as files become scattered across different directories, machines, and storage devices. The result? Even disciplined users have trouble finding old data [84].

To this end, several personal desktop search tools have emerged that build an index of the contents of a user's file system and provide rapid search over that index. These search tools are, however, incomplete solutions: they index content, but not context. They capture only static, syntactic relationships, not dynamic, semantic ones. One can instead use context to effectively aid these content-only search tools [85]. The first thrust of this research presents *Weft*, a context-enhanced search tool. Weft yields an increase in *user-perceived* search quality over traditional static indexing, as a field study demonstrates.

Perhaps most importantly, these information managing tools have access only to information stored on a user's local machine—other sources are not indexed and unsearchable. However, a user's social neighborhood provides enormous leverage for context sharing and collaborative work. Unfortunately, simply adding context to a system presents complications for the user: it becomes a new entity to manage. This is unpleasant for shared objects and especially painful when sharing without a common administrative domain of control. Ideally, when one party creates or captures context for an object, their friends with copies of that object should benefit without imposing undue burdens on any of them.

To illustrate this, consider someone returning from a vacation and sharing pictures with far-flung family and friends. Typically, this is done via email [98] or some other simple web service mechanism. Some file types permit inclusion of local metadata, which could be used to represent limited forms of context. Unfortunately, each type has a unique and often opaque structure. Worse, changes to context at one site require explicit redistribution of the enclosing file. Any non-local context—for example, inferring subjects of photos or distilling links to other relevant photos [80]—is lost entirely. For such a simple task, heavyweight mechanisms providing direct support for contextual sharing, authentication, and access control are unlikely to provide value.

In this vision, users effectively form communities based on the objects that they share. Challenges arise as these peers are frequently scattered geographically with varying levels of connectivity, computing power, and storage, are formed in an ad-hoc manner in different administrative domains, and must be weakly trusted.

Any solution to this data management problem has an important constraint: it must respect the trust boundaries and access limits that exist in the real world. Such trust boundaries arise naturally with the increasing importance of collaboration. For example, as scientists from different fields and institutions work on a problem, they must share data relevant to the project at hand without giving access to irrelevant data. Of course, researchers are not the only people with such conflicting demands. Consider a physician with a private practice. She interacts with family, friends, and colleagues, and uses machines at home and in both offices that are administered by different people. Family members viewing photographs of her children certainly cannot have access to any work-related materials. At the same time, the physician prefers to guard details of her home life from coworkers.

This dissertation, as its second thrust, presents *Ligature*, a system that acts as an attachment point for contextual data. Ligature supports a general interface for building a wide variety of distributed contextual applications. It provides a separation of concerns between an object and its context, allowing its context to grow and evolve over time. It also ensures a user's context is to do with as they please, rather than be tied to third-party providers. Ligature supports mobile and disconnected users, is scalable, and is self-organizing, requiring no centralized infrastructure. These features assist the ad hoc, spontaneous collaborations that manifest themselves regularly in everyday life. Ligature operates with minimal system overhead and, importantly, includes a new abstraction for specifying trust boundaries, which a user study demonstrates is reasonably palatable to users.

There are two utilitarian impacts from this work to society. First, finding the right information without undue effort has become a critical problem in almost every facet of society; this research aids in ameliorating this information management problem. Second, this work reduces the adversity in the formation and operation of ad hoc, spontaneous collaboration. Such collaborations arise often, particularly in the sciences. They require sharing of data, cannot afford centralized administration, and must respect personal boundaries on disclosure.

1.1 Thesis Statement

In response to this problem, this dissertation sets out to demonstrate the following thesis statement:

> *Context holds significant promise for simplifying the organization and retrieval of data and can be shared in a spontaneous, ad-hoc manner while respecting boundaries on disclosure.*

This thesis is validated with the following steps:

1. The design of Weft, an architecture for capturing context that reorders and extends traditional content-only personal desktop search.

2. The construction of a methodology for evaluating said system with a field study, which indicates users find Weft's search results superior to current desktop search tools.
3. An architecture, Ligature, for sharing this and other context among other users with primitives to set boundaries on disclosure.
4. The evaluation of the usability of Ligature—that is, whether users understand the privacy modalities present in the architecture—and the performance overhead of the system. The results indicate the system scales with user ability and bears minimal system overhead.

The remainder of this document explores these four thrusts.

1.2 Overview of the Dissertation

The dissertation is organized as follows.

Part I: *Employing Context* shows how context can improve the retrieval of data; specifically, in this case, to assist personal file search tools. Such traditional content-only search tools build a content index from a user's repository and provide search capabilities across it. Chapter I: *Employing Context* describes the architecture for a context-enhancing personal desktop search. Weft augments and reorders results from static search tools with additional hits extracted from a contextual index. Background tasks infer a user's context to build this index by interposing on application system-call behavior. This chapter introduces the notion of using strict causality to build the contextual index, provides detailed descriptions of the other necessary component algorithms, and describes a prototype system for capturing this context.

Evaluating such a system proves tricky, as it requires a user's corpus, search history, and a reasonably lengthy period of contextual state for model learning. Further, standard metrics, such as precision and recall, while useful, fail to judge the user-perceived difference in result set orderings. Chapter 3: *User Evaluation of Weft* explains the difficulties in devising such in-lab scenarios, and this work instead opts for a field study with non-expert users, which are likely representative of the population at large. The chapter presents the methodology and execution of an in-field evaluation. Users run Weft privately on their own machines for a month and the system tracks their search history. At the conclusion of the study, a selection of previously executed queries is presented to the user, ordered via content-only and context-enhancing methods, and they rate each ordering. The results of 27 participants proves that contextual search systems using causality as the dynamic indexing component do yield user-perceived improvements in search results. Also included as part of this chapter is an exploration of personal search behavior.

Given that context is useful in simplifying data retrieval, a nagging question is: can this context be effectively shared? Importantly, considerable data sharing occurs in an manner among those not belonging to a common administrative domain of control; any effective system must also support this ad-hoc collaboration without placing an undue burden on the user. This is the focus of Part II: *Sharing Context*. This research proposes to break this stalemate by leveraging acts of sharing to automatically exchange contextual information without requiring centralized repositories, strong identities, or other vestiges of a single domain of administrative control. In this vision, users spontaneously form communities based on the objects they share. This coupling/decoupling between objects and contexts allows context to grow and evolve over time, detaches context from third-party providers, and provides an intuitive and deterministic condition for contextual exchange by imbuing trust through sharing. To this end, Chapter II: *Sharing Context* introduces a framework for aiding contextual sharing that meets these precepts. Applications, such as Weft, use Ligature to share context among interested parties.

Of course, such proactive sharing of context brings with it risks, as any system must respect real-world trust boundaries and access limits. Chapter 5: *Contextual Frames* describes a mechanism and enclosing calculus that provides a simple and expressive explicit abstraction for specifying boundaries on disclosure. Basically, frames are enclosures around some context about an object, accruing context and following the objects to which they are bound. Context within a frame remains separate and independent from other frames. A user can bind more than one frame to an object, and fork, combine, or destroy them. The chapter concludes with several case studies.

Chapter 6: *Email Sharing Patterns* presents common sharing patterns—which guide Ligature's architectural design decisions—by the way of a large-scale study of email deliveries. Chapter 7: *Supporting Ligature's System Model* describes Ligature's architecture and prototype implementation. The architecture is self-organizing, requiring no additional infrastructure; eventually consistent, aiding users' mobile and ephemeral nature; and fair, as resources expended are proportion to the number of objects a user shares. The chapter also describes several applications, including a synergistic version of Weft, constructed atop this architecture.

Ligature is only beneficial if users can effectively employ its disclosure model. Chapter 8: *Evaluation of Ligature* presents a user study of contextual frames, where several tasks of varying difficulty were given to both novice and expert computer users to complete. The results of this study are encouraging, as the mechanisms scale with task difficulty: novice users were able to complete the easy and moderate tasks in almost all cases, while expert users were able to complete the difficult tasks as well. In addition, as context is metadata and, hence, subservient to primary data and uses, any context storage and exchange system must require minimal overhead. This chapter also profiles the performance overhead of exchanging context, the overhead of contextual frames, and the application performance penalty of using Ligature as a backing store for context. In all cases, Ligature's overhead is negligible.

Finally, Chapter 9: *Conclusions & Future Work* discusses this research's results as well as important avenues for future work.

PART I

EMPLOYING CONTEXT

CHAPTER 2

DESIGN OF *WEFT*—CONTEXT-ENHANCED PERSONAL FILE SEARCH

That which is static and repetitive is boring. That which is dynamic and random is confusing. In between lies art.

John Locke (1632–1704)

To reduce the friction users experience in finding their data, many personal search tools have emerged. These tools build a content index offline and allow keyword search across this index. Despite their growing prevalence, most of these tools are, however, incomplete solutions: they index content, not context. They capture only static, syntactic relationships, not dynamic, semantic ones. To see why this is important, consider the difference between compiler optimization and branch prediction. The compiler has access only to the code, while the processor can see how that code is commonly used. Just as run-time information leads to significant performance optimizations, users find contextual and semantic information useful in searching their own repositories [90].

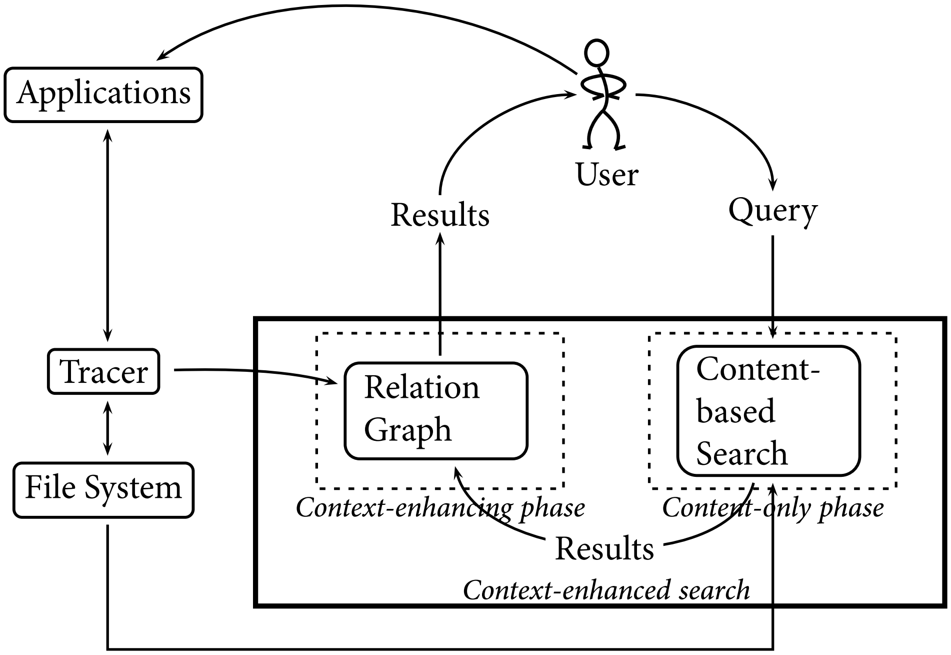


Figure 2.1: Architecture of context-enhanced search.

Context-enhanced search is beginning to receive attention, but it is unclear what dynamic information is most useful in assisting search. Soules and Ganger [85] developed a system, named Connections, that uses *temporal locality* to capture the provenance of data: for each new file written, the set of files read "recently" form a kinship or *relation graph*, which Connections uses to extend search results generated by traditional static, content-based indexing tools. This context attempts to capture the way in which many individuals recall events and items (e.g., "I know I was working on my report when I read that paper"). Temporal locality is likely to capture many true relationships, but may also capture spurious, coincidental ones. For example, a user who listens to music while authoring a document in her word processor may or may not consider the two "related" when searching for a specific document. These post-hoc errors can be even more precarious: that same word processor, minimized to the background and generating auto-save events, will tie together many unrelated files.

To capture the benefit of temporal locality while avoiding its pitfall, this research introduces *Weft*[1], a search tool using a different mechanism to deduce provenance: *causality*. That is, Weft uses data flow through and between applications to impart a more accurate relation graph. This context-enhancing search has been implemented for Windows platforms.

[1] In plain weaving, two parallel yarns, one horizontal and one vertical, form a fabric; a "weft" yarn is the complement to a "warp" yarn.

Weft's architecture matches that of Soules and Ganger [85] and is shown in Figure 2.1. Weft augments traditional content search using kinship relations between files. After the user enters keywords in the search tool, the tool runs traditional content-only search using those keywords—the **content-only phase**—and then uses the previously constructed relation graph, a data structure storing contextual relationships, to reorder these results and identify additional hits—the **context-enhancing phase**. These new results are then returned to the user. Background tasks on the user's machine periodically index a file's content for the content-only phase and monitor system events to build the relation graph for the context-enhancing phase.

There is a decoupling between the content-only and context-enhancing phases. This not only simplifies the design and allows plug-in replacements for the content-only phase, it also permits exclusive study of the search tool's context-enhancing effects without necessitating analysis of any interactive effects. This is particularly apropos when studying the space and time overheads of the context-enhancing phase (§3.3).

This chapter describes: (1) how Weft translates and encodes context from system-level actions into user-level beliefs; (2) how Weft uses these relationships to locate and re-rank related data during a search; (3) the advantages and pitfalls of this approach; (4) a Weft prototype; and (5) a survey of related work.

2.1 Inferring Kinship Relationships

A kinship **relation** $f \to f'$, where f and f' are files on a user's system, indicates that f is an ancestor of f', implying that f may have played a role in the origin of f'. These relationships are encoded in the relation graph, which is used to reorder and extend search results in the context-enhancing phase.

This work evaluates two methods of deducing these kinship relations: temporal locality and causality. Both methods classify the source file of a read as input and the destination file of a write as output by inferring user task behavior from observed actions.

2.1.1 Temporal Locality Algorithm

The **temporal locality** algorithm[2], as employed in Soules and Ganger [85], infers relations by maintaining a sliding **relation window** of files accessed within the previous t seconds system-wide. Any write operation within this window is tied to any previous read operation within the window.

[2] This is known as the read/write operational filter with directed links in Soules and Ganger [85], which was found the most effective of several considered.

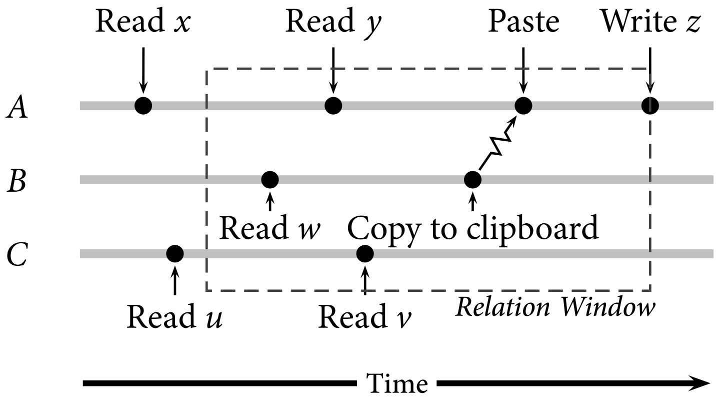


Figure 2.2: An example time diagram. This figure is used to illustrate the differences between the contextual algorithms.

Consider the sequence of system events shown in Figure 2.2. There are three processes, *A*, *B*, and *C*, running concurrently. *C* reads files *u* and *v*, *A* reads files *x* and *y*. *B* reads *w* and copies data to *A* through a clipboard IPC action initiated by the user. Following this, *A* then writes file *z*.

The relation window at *z*'s write contains reads of *y*, *w*, and *v*. The temporal locality algorithm is process agnostic and views reads and writes system-wide, distinguishing only between users. The algorithm thus returns the relations $\{y \rightarrow z, w \rightarrow z, v \rightarrow z\}$.

The relation window attempts to capture the transient nature of a user task. Too long a window will cause unrelated tasks to be grouped, but too short a window will cause relationships to be missed.

2.1.2 Causality Algorithm

Rather than using a sliding window to deduce user tasks, this research proposes viewing each process as a filter that mutates its input to produce some output. This **causality** algorithm tracks how input flows—at the granularity of processes—to construct kinship relations, determining what output is causally related to which inputs. This is known as where-provenance [11], as it is knowledge linking "where" the output data came from.

Specifically, whenever a write event occurs, the following relations are formed:

(a) Any previously read files within the same process are tied to the current file being written; and

(b) Further, the algorithm tracks IPC transmits and its corresponding receives, forming additional relationships by assessing the transitive closure of file system events across these IPC boundaries.

	Temporal Locality	Causality
Handles noise Disparate tasks running concurrently (e.g., music player)	✗	✓
Transition periods Handles temporary period when switching between disparate tasks.	✗	✓
Long-lived processes Lengthy operations (e.g., a long compilation build)	✗	✓
Monolithic processes Same process used for disparate tasks (e.g., emacs).	✓	✗
Hidden channels e.g., the user exercising her brain as the clipboard.	✓	✗

Table 2.1: Temporal locality and causality algorithm comparison.

That is, for each relation $f \to f'$, there is a directed left-to-right path in the time diagram starting at a read event of file f and ending at the write of file f'. There is no temporal bound within this algorithm.

Reconsidering Figure 2.2, A reads x and y to generate z; the causality algorithm produces the relations $\{x \to z, y \to z\}$ via condition (a). B produces no output files given its read of w, but the copy-and-paste operation represents an IPC transmit from B with a corresponding receive in A. By condition (b), this causes the relation $w \to z$ to be made. C's reads are dismissed as they do not influence the write of z or any other data.

2.1.3 Comparison

Table 2.1 outlines the differences between the temporal locality and causality algorithms.

Causality forms fewer relationships than temporal locality, attempting to avoid many false relationships. Unrelated tasks happening concurrently are more likely to be deemed related under temporal locality, while causality is more conservative.

During transition periods when a user switches between disparate tasks, there is a temporary period where incorrect relations may form under temporal locality, which is mitigated by the causality algorithm.

A user working on a spreadsheet with her music player in the background may form spurious relationships between her music files and her document under temporal locality, but not under causality; those tasks are distinct processes and no data is shared. Additionally, if she switches to her email client and saves an attachment, her spreadsheet may be an ancestor of that attachment under temporal locality if the file system events coincide within the relation window.

A user opening a document in a word processor, writing for the afternoon, then saving it under a new name would lose the association with the original document under temporal locality, but not causality. Causality more accurately captures the unit of work in these long-lived processes.

Monolithic processes used for many different tasks, such as the emacs text editor, will likely fare worse under causality. A user working with her text editor to author several unrelated documents within the same process would have spurious relations formed with causality, but perhaps not from temporal locality.

Causality can fare worse under situations where data transfer occurs through hidden channels due to loss of real context. This is most evident when a user exercises her brain as the "clipboard," such as when she reads a value off a spreadsheet and then keys it manually into her document. As future work, it may be possible to use window focus to demarcate user tasks [72] as a means to group related processes together and capture these hidden channels. Indeed, there is subsequent work in capturing and indexing the user interface layer, which is more directly coupled to user interaction, for contextual search [38].

2.1.4 Relation Graph

Relations formed are encoded in the **relation graph**: a directed graph whose vertices represent files on a user's system with edges constituting a kinship relation between files and the weight of that edge representing the strength of the bond. The edge's direction represents an input file to an output file. Figure 2.3 on the next page shows an example relation graph.

For each relation of the form $f \to f'$, the relation graph consists of an edge from vertex f to vertex f' with the edge weight equaling the count of $f \to f'$ relations seen. To prevent heavy weightings due to consecutive writes to a single file, successive write events are coalesced into a single event in both algorithms.

2.2 Reranking and Extending Results

After a query is issued, the tool first runs traditional content-only search using keywords given by the user, and then uses the relation graph to reorder results and identify additional hits. This basic architecture is identical to that of Soules and Ganger [85].

Each content-only result is assigned its relevance score as its initial rank value. The relation graph is then traversed breadth-first for each content-only result. The **path length**,

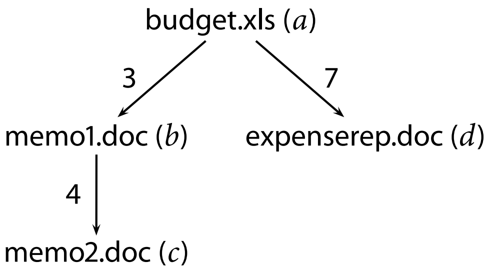


Figure 2.3: An example relation graph. This is used to illustrate the workings of the basic BFS algorithm.

P, is the maximum number of steps taken from any starting node in the graph during this traversal. Limiting the number of steps is necessary to avoid inclusion of weak, distant relationships and to allow the algorithm to complete in a reasonable amount of time.

Further, because incorrect lightly-weighted edges may form, an edge's weight must provide some minimum support threshold to be a viable candidate: it must make up a minimum fraction of the source's outgoing weight or the sink's incoming weight. Edges below this **weight cutoff** are pruned. As link weights are not normalized, the weight cutoff is expressed as a link weight ratio, which provides an online normalization method.

The tool runs the following algorithm, called **basic BFS**[3], for P iterations. Let E_m be the set of all incoming edges to node m, with $e_{nm} \in E_m$ being a given edge from n to m and $\gamma(e_{nm})$ being the fraction of the outgoing edge weight for that edge. w_{n_0} is the initial value, its content-only score, of node n. The α value dictates how much trust is placed in each specific weighting of an edge. At the i-th iteration of the algorithm:

$$w_{m_i} = \sum_{e_{nm} \in E_m} w_{n_{i-1}} \cdot \left[\gamma(e_{nm}) \cdot \alpha + (1 - \alpha) \right] \tag{2.1a}$$

After all P runs of the algorithm, the total weight of each node is:

$$w_m = \sum_{i=0}^{P} w_{m_i} \tag{2.1b}$$

In Equation (2.1a), heavily-weighted relationships and nodes with multiple paths push more of their weight to node m. This matches user activity as files frequently used together will receive a higher rank; infrequently seen sequences will receive a lower rank. The final result list sorts by Equation (2.1b) from highest to lowest value. Though straightforward, this breadth-first reordering and extension mechanism proves effective [85].

As an example, consider a search for "project budget requirements" that yields a content-only phase result of budget.xls with weight $w_{a_0} = 1.0$. Assume that during the context-enhancing phase, with parameters P=3, α=0.75 and no weight cutoff, the relation graph

[3] In Soules [83], this is now known as *basic breadth-first expansion* or *basic-BFE*.

shown in Figure 2.3 is loaded from disk. Take node expenserep.doc, abbreviated as d. The node's initial weight is $w_{d_0} = 0$ as it is absent from the content-only phase results. The algorithm proceeds as follows for P iterations:

$$\begin{aligned} w_{d_1} &= w_{a_0} \cdot \left[\gamma(e_{ad}) \cdot \alpha + (1-\alpha)\right] && \text{by (2.1a)} \\ &= 1.0 \cdot \left[(7/10) \cdot 0.75 + 0.25\right] = 0.775 \\ w_{d_2} &= 0 && \text{as } w_{a_1}{=}0 \\ w_{d_3} &= 0 && \text{as } w_{a_2}{=}0 \end{aligned}$$

Finally, the total weight of node d is:

$$w_d = 0 + 0.775 + 0 + 0 = 0.775 \qquad \text{by (2.1b)}$$

The final ordered result list, with terminal weights in parentheses, is: budget.xls (1.0), expenserep.doc (0.775), memo1.doc (0.475) and memo2.doc (0.475). In this example, both memo files have identical terminal weights; ties are broken arbitrarily.

2.3 Advantages and Limitations

Context-enhanced search provides three main advantages over purely content-only search. First, the system may increase recall by locating more relevant items for a search. Second, it may also locate opaque objects that cannot be indexed by a content-only mechanism (e.g., multimedia files). These two rewards allow users to find more of their relevant data. Last, Weft may re-rank existing content-only search results, particularly those that have close relevance scores, to more accurately reflect user work patterns.

The are, however, several limitations to this approach. There is an initial learning curve to build a reasonable relation graph. Without this state, Weft cannot provide any improved search capabilities. Second, the system requires initial seeding of content-only search results, which may be inadequate in environments with many opaque objects (e.g., a working set dominated by multimedia files). Third, a user's context undoubtedly changes over time, but the system as of yet does not perform any hysteresis; a procedure, as well as an analysis of, decaying relationships over time is necessary. Lastly, there is potential for poor relation inference and the consequential false positives. Particularly with the latter, Weft only captures application-level behavior, which may not directly correlate to user actions and true relationships. Both the temporal locality and causality algorithms are quite blunt, though they provide a useful starting point to glean relationships.

Interposed				
File System Operations				
Opening and closing files				
CreateFile†	CreateFileEx†	__lopen	CloseHandle	
Reading and writing files				
ReadFile	ReadFileEx	WriteFile	WriteFileEx	
Moving, copying, and unlinking files				
MoveFile†	MoveFileEx†	CopyFile†	CopyFileEx†	DeleteFile†
IPC Operations				
Clipboard (DDE)	GetClipboardData	SetClipboardData		
Mailslots	CreateMailSlot†			
Named pipes	CreateNamedPipe†			
Other				
Process creation and destruction	CreateProcess†	ExitProcess		
Not Interposed				
IPC Operations				
Sockets	WM_COPYDATA[a]	Shared Memory[b]		
Microsoft RPC	COM			

† Both ANSI and Unicode versions of these marked calls.

[a] Also known as “data copy.”

[b] This is known as “file mapping” in Windows parlance.

Table 2.2: The system calls the search tool interposes on.

2.4 Implementation

The implementation of Weft runs on Windows NT-based systems. Weft uses a binary rewriting technique [42] to trace all file system and interprocess communication calls. Such a user space solution was chosen as it allows tracking high-level calls in the Win32 API.

When a user first logs in, the prototype instruments all running processes, interposing on our candidate set of system calls as listed in Table 2.2. It also hooks the CreateProcess call, which will instrument any subsequently launched executables. Care was required to not falsely trip anti-spyware tools. Each instrumented process reports its system call behavior to a background collection daemon, which uses idle CPU seconds, via the mailslots IPC mechanism. For performance reasons, each process amortizes 32K or 30 seconds worth of events across a single message. The collection daemon contemporaneously creates two relation graphs: one using temporal locality (§2.1.1) and one using causality (§2.1.2).

If a file is deleted, its node in the relation graph becomes a zombie: it relinquishes its name but maintains its current weight. The basic BFS algorithm uses a zombie's weight in its calculations, but a zombie can never be returned in the search result list. Weft currently does not prune zombies from the relation graph.

Content indexing is done using Google Desktop Search (GDS)[4] with its exposed API. The actual internals of GDS are unfortunately opaque, but we expect it to use state-of-the-art information retrieval techniques to conduct its searches. GDS was chosen over other content indexing tools, such as Indri [1], because of its support for more file types. All queries enter through Weft's interface: only GDS's indexing component remains active, its search interface is turned off. GDS also indexes email and web pages, but Weft prunes these from the result set.

A complication arises, however. GDS allows sorting by relevance, but it does not expose the actual relevance scores. These are necessary as they form the initial values of the basic BFS algorithm (§2.2). Weft uses:

$$\psi(i) = \frac{2(n-i)}{n(n+1)} \tag{2.2}$$

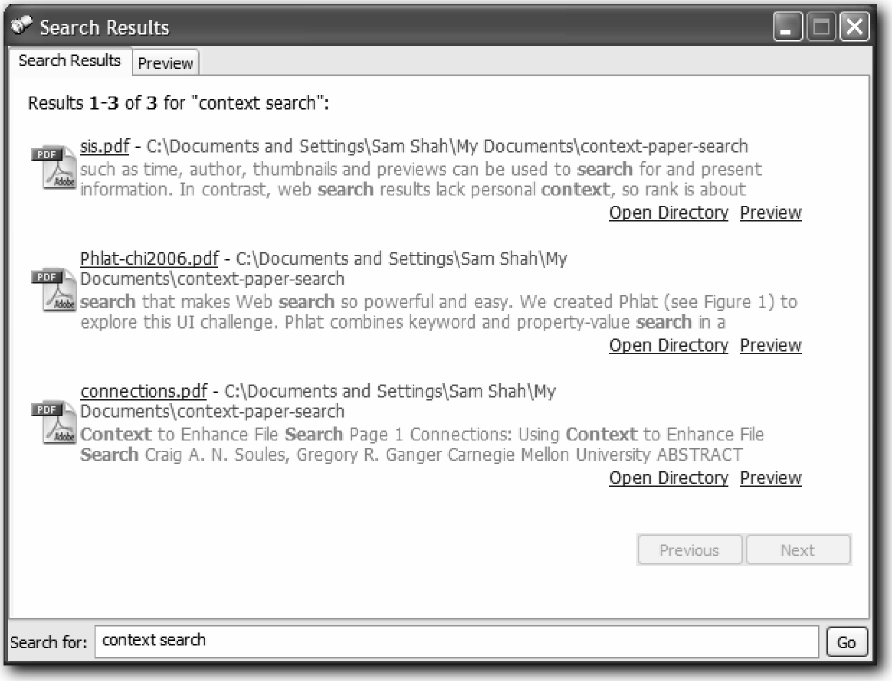
to seed the initial values of the algorithm. Here, n is the total number of results for a query, and i is the result's position in the result list. Equation (2.2) is a strict linear progression with relevance values constrained such that the sum of the values is unity, roughly matching the results one would expect from a TF/IDF-type system [4]. Soules [83] found that Equation (2.2) performs nearly as well as real relevance scores: it produces a 10% improvement across all recall levels in Soules's study, while real relevance scores produce a 15% improvement.

Users interact with the search system through an icon in the system tray. When conducting a search, a frame, shown in Figure 2.4, appears, allowing the user to specify her query keywords in a small text box. Search results in batches of ten appear in the upper part of the frame. A snippet of each search result, if available, is presented, along with options to preview the item before opening. Previewing is supported by accessing that file type's ActiveX control, as is done in web browsers.

Users may temporarily suspend tracing for up to 10 minutes by clicking a button found in the tool's system tray menu. The 10 minute limit prevents users from forgetting to resume tracing, which may significantly hamper contextual indexing.

In most desktop search applications, Weft included, the search system is available to users immediately after installation. Because the content indexer works during idle time and little to no activity state has been captured to build the relation graph when first installed,

[4] http://desktop.google.com



Results **1**-**3** of **3** for "context search":

sis.pdf - C:\Documents and Settings\Sam Shah\My Documents\context-paper-search
such as time, author, thumbnails and previews can be used to **search** for and present information. In contrast, web **search** results lack personal **context**, so rank is about

Phlat-chi2006.pdf - C:\Documents and Settings\Sam Shah\My Documents\context-paper-search
search that makes Web **search** so powerful and easy. We created Phlat (see Figure 1) to explore this UI challenge. Phlat combines keyword and property-value **search** in a

connections.pdf - C:\Documents and Settings\Sam Shah\My Documents\context-paper-search
Context to Enhance File **Search** Page 1 Connections: Using **Context** to Enhance File **Search** Craig A. N. Soules, Gregory R. Ganger Carnegie Mellon University ABSTRACT

Search for: context search

Figure 2.4: A screen shot of the search interface.

search results during this initial indexing period are usually quite hapless. Weft warns users that during this initial indexing period their search results are incomplete.

The Weft prototype uses a relation window of 30 seconds and basic BFS with a weight cutoff of 0.1% and parameters $P = 3$ and $\alpha = 0.75$. These parameters were validated by Soules and Ganger [85].

To prevent excessively long search times, Weft restricts the context-enhancing phase to 5 seconds and returns intermediate results from basic BFS if this threshold is reached. Although, as shown in our evaluation (c.f. §3.3.3), the system rarely hits this limit. Due to our unoptimized implementation, a commercial implementation is expected to perform slightly better than our results would suggest.

2.5 Related Work

There are various static indexing tools for one's filespace. Instead of strict hierarchal naming, the semantic file system [29] allows assignment of attributes to files, facilitating search over these attributes. These attributes comprise of category-keyword pairs; users can create virtual directories, potentially recursive, to refine their view by category. For example, this document could be assigned "University of Michigan" for the category "institution." Since most users are averse to ascribing keywords to their files, the semantic file system provides transducers to distill file contents into keywords. The semantic file system focuses on the mechanism to store attributes, not on content analysis to distill these attributes.

The HAC semantic file system [34] is similar to the semantic file system, but instead of providing different views of a user's files, it maps the actual query onto the file system: users can add or remove files to the query by simply manipulating the directory using standard UNIX tools. Nebula [9] is semantic file system where properties (key-value pairs) are assigned to files and views scoped to particular properties are possible. Basically, Nebula provides relational database access primitives to build semantic queries. Prospero [70] is another semantic file system that deals with providing views of a global name space. While other semantic file systems deal with an attribute-based mechanism, Prospero provides views based on pathnames. That is, one can create a journals directory that contains all of their journals that may be housed in different directories. This allows users to place related documents together, even though they may belong to different paths in their personal namespace.

The previously mentioned semantic file systems have failed to gain traction with users as these systems rely on either user input or content analysis to ascertain attributes. Assigning attributes for files is extremely tedious and caustic to users; naturally, most users fail to do so.

Instead, many content-based search tools have emerged. These include Google Desktop Search and Windows Desktop Search[5], among others. These systems extract a file's content into an index, permitting search across this index. While the details of such systems are opaque, it is likely they use forefront technologies from the information retrieval community. Several such advanced research systems exist, Indri [1] being a prime example. These tools are orthogonal to Weft in that they all analyze static data with well-defined types to generate an index, ignoring crucial contextual information that establishes semantic relationships between files.

The seminal work in using gathered context to aid in personal desktop file search is by Soules and Ganger [85] in the form of a file system search tool named Connections. Connections identifies temporal relationships between files and uses that information to expand and reorder traditional content-only search results, improving average precision and recall compared to Indri. Weft uses some component algorithms from Connections (§2.2) and our evaluation compares against its temporal locality approach (§2.1.1).

Weft's notion of provenance is a subset of that used by the provenance-aware storage system (PASS) [63]. PASS attempts to capture a complete lineage of a file, including the system

[5] http://www.microsoft.com/windows/products/winfamily/desktopsearch

environment and user- and application-specified annotations of provenance. A PASS filesystem, if available, would negate the need for Weft's relation graph. Indeed, the technique used by PASS to capture system-level provenance is very similar to the causality algorithm (§2.1.2).

Other systems interpose on file accesses and application use, in a similar vain to the causality algorithm, to capture and delineate user tasks [22, 44, 79]. These context-aware tasks are then used in an human-computer interactive way for activity display. It may be possible to augment these techniques to aid in reducing incorrect relations (false positives) and extracting new ones (false negatives) for the contextual index, though these algorithms suffer from their own post-hoc errors: background tasks and the multitasking nature of user work patterns can identify incorrect task groupings.

Several systems leverage other forms of context for file organization and search. Phlat [16] is a user interface for personal search, running on Windows Desktop Search, that also provides a mechanism for tagging or classifying of data. The user can search and filter by contextual cues such as date and person. Weft provides a simpler UI, permitting search by keywords only (§2.4), but could use Phlat's interface in the future. Another system, called "Stuff I've Seen" [23], remembers previously seen information, providing an interface that allows a user to search their historical information using contextual cues. The Haystack project [43] is a personal information manager that organizes data, and operations on data, in a context-sensitive manner. Lifestreams [25] provides an interface that uses time as its indexing and presentation mechanism, essentially ordering results by last access time. Our provenance techniques could enhance these systems through automated clustering of semantically-related items.

2.6 Summary

This chapter presents the architecture as well as a set of component algorithms for context-enhanced personal file search. Further, it provides a description of an implementation of this architecture. The following chapter outlines the difficulties in evaluating such a system, motivating the need for a user study with the prototype implementation.

CHAPTER 3

USER EVALUATION OF WEFT

Attempt the end, and never stand
to doubt;
Nothing's so hard but search will
find it out.
Robert Herrick (1591–1674)

As part of our evaluation, this research conducts a user study with Weft's prototype implementation to measure a user's perceived search quality directly. To accomplish this, two common techniques from the social sciences and human-computer interaction are adapted to the area of personal file search: first, a randomized, controlled trial to gauge the end-to-end effects of our indexing technique; and second, a repeated measures experiment, where users evaluate the different indexing techniques side-by-side, locally on their own machines. This style of experiment is methodologically superior as it measures quality directly while preserving privacy of user data and actions.

The results indicate that the causal provenance algorithm fares better than using temporal locality or pure content-only search, being rated 17% higher, on average over all queries, than the other algorithms by users with minimal space and time overheads. This chapter describes this user study in detail.

3.1 Experimental Approach

Traditional search tools use a corpus of data where queries are generated and oracle result sets are constructed by experts [4]. Two metrics, precision (minimizing false positives) and recall (minimizing false negatives), are then applied against this oracle set for evaluation.

Personal desktop search systems, however, are extremely difficult to study in the laboratory for a variety of reasons. First, as these systems exercise a user's own content, there is only one oracle: that particular user. All aspects of the experiment, including query generation and result set evaluation, must be completed by the user with their own files. Second, a user's search history and corpus are private. Since the experimenter lacks knowledge of each user's data, it is nearly impossible to create a generic set of tasks that each user could perform. Even if it were possible for a researcher to compose a set of generic tasks, as opposed to tasks motivated by the user, this has been shown to affect search performance [47]. Third, studying context-enhanced search is further complicated by the need to capture a user's activity state for a significant length of time, usually a month or more, to develop the dynamic indices—an impractical feat for an in-lab experiment.

In lieu of these difficulties with in-lab evaluation, Soules and Ganger [85] constructed a corpus of data by tracing six users for a period of six months. At the conclusion of their study, participants were asked to submit queries and to form an oracle set of results for those queries. Since each user must act as an oracle for their system, they are loathe to examine every file on their machine to build this oracle. Instead, results from different search techniques were combined to build a good set of candidates, a technique known as pooling [4]. Each search system can then be compared against each oracle set using precision and recall.

While an excellent initial evaluation, such a scheme may exhibit observational bias: users will likely alter their behavior knowing their work habits are being recorded. For instance, a user may be less inclined to use her system as she normally would, for she may wish to conceal the presence of some files. It is tough to find users who would be willing to compromise their privacy by sharing their activity and query history in such a manner.

Further, to generate an oracle set using pooling, we need a means to navigate the result space beyond that returned from content-only search. That is, we need to use results from contextual indexing tools to generate the additional pooled results. However, the lack of availability of alternative contextual indexing tools means that pooling may be biased toward the contextual search tool under evaluation, as that tool is the only one generating the extra pooled results.

We also care to evaluate the utility of the tool beyond the metrics of precision and recall, as they fail to gauge the differences in orderings between sets of results. That is, two identical sets of results presented in different order will likely be qualitatively perceived differently. Also, while large gains in mean average precision are detectable to the user, nominal improvements remain inconclusive [2]. We would like a more robust measurement that evaluates this perception of search quality.

For these reasons, we conduct a user study and deploy an actual tool participants can use. We run a *pre-post measures randomized controlled trial* [6, 49] to ascertain if users perceive end-to-end differences between content-only search and the causality algorithm with basic BFS. This study suggests that the tool is useful to users, but measuring end-to-end effects conclusively requires a probative degree of replication. In lieu of this, we conduct a *repeated measures experiment* to qualitatively measure search quality directly and conclusively: we ask users to rate search orderings of their previously executed queries constructed by content-only search and of results from the different dynamic techniques.

A pre-post measures randomized controlled trial is a study in which individuals are allocated at random to receive one of several interventions. One of these interventions is the standard of comparison, known as the "control," the other interventions are referred to as "treatments." Measurements are taken at the beginning of the study, the pre-measure, and at the end, the post-measure. Any change between the treatments, accounting for the control, can be inferred as a product of the treatment. In this setup, the control group handles threats to validity: that any exhibited change is caused by some other event than the treatment. For instance, administering a treatment can produce a psychological effect in the subject where the act of participation in the study results in the illusion that the treatment is better. This is known as the placebo effect.

As an example, consider that we have a new CPU scheduling algorithm that makes interactive applications feel more responsive and we wish to gauge any user-perceived difference in performance against the standard scheduler. To accomplish this, we segment the population randomly into two groups, one which uses the standard scheduler, the control group, and the other receives the improved scheduler, the treatment group. Neither group knows which one they belong to. At the beginning of the study, the pre-measure, we ask users to estimate the responsiveness of their applications with a questionnaire. It's traditional to use a Likert scale in which respondents specify their level of agreement to a given statement. The number of points on an n-point Likert scale corresponds to an integer level of measurement, where 1 to n represents the lowest to highest rating. At the end of the study, the post-measure, we repeat the same questionnaire. If the pre- and post-measures in the treatment group are statistically different than the pre- and post-measures in the control group, we can conclude the new scheduler algorithm is rated better by users.

Sometimes it is necessary or useful to take more than one observation on a subject, either over time or over many treatments if the treatments can be applied independently. This is known as a repeated measures experiment [6, 49]. In the scheduler example, we may wish to first survey the subject, randomly select an algorithm to use and have the subject's system run the algorithm for some time period. We can then survey the subject again and repeat. In this case, we have more than one observation on a subject, with each subject acting as its own control.

Traditionally, one uses ANOVA to test the statistical significance of hypotheses among two or more means without increasing the α (false positive) error rate that occurs with using multiple t-tests. With repeated measures data, care is required as the residuals aren't uniform across the subjects: some subjects will show more variation on some measurements than on others. Since we generally regard the subjects in the study as a random sample from the population at large and we wish to model the variation induced in the response by these different subjects, we make the subjects a random effect. An ANOVA model with both fixed and random effects is called a mixed-effects model [75].

3.1.1 Randomized Controlled Trial

In the study, we randomly segment the population into a control group, whose searches return content-only results, and a treatment group, whose searches return results reordered and extended by basic BFS using a relation graph made with the causality algorithm.

To reduce observational bias and protect privacy, the tool does not track a user's history, corpus, or queries, instead reporting aggregate data only. During recruitment, upon installation, and when performing queries, we specifically state to users that no personal data is shared during the experiment. We hope this frees participants to use their machines normally and issue queries without hindrance.

The interface of both systems is identical. To prevent the performance of the unoptimized context-enhancing implementation from unduly influencing the treatment group, both groups run the extended search, but the control group throws away those results and uses content-only results exclusively.

The experiment is double-blind: neither the participants nor the researchers knew who belonged to which group. This was necessary to minimize the observer-expectancy effect; that unconscious bias on the part of the researchers may appear during any potential support queries, questions, or follow ups. The blinding process was computer controlled.

Evaluation is based on pre- and post-measure questionnaires where participants are asked to report on their behavior using 5-point Likert scale questions. For example, "When I need to find a file, it is easy for me to do so quickly." Differences in the pre- and post-measures against the control group indicate the overall effect the causality algorithm has in helping users find their files. We also ask several additional questions during the pre-survey portion to understand the demographics of the population and during the post-survey to elicit user feedback on the tool.

We pre-test each survey instrument on a small sample of a half-dozen potential users who are then excluded from participating in the study. We encourage each pre-tester to ask questions and utilize “think-alouds,” where the participant narrates her thought process as she’s taking the survey. Pre-testing is extremely crucial as it weeds out poorly worded, ambiguous, or overly technical elements from surveys. For example, the first iteration of the survey contained the question, “I often spend a non-trivial amount of time looking for a file on my computer.” Here, the word “non-trivial” is not only equivocal, it is confusing. A more understandable question would be to set an exact time span: “I often spend 2 minutes or more a day looking for a file on my computer.”

We also conducted a pilot study with a small purposive sample of colleagues who have trouble finding their files. This allowed us to vet the tool and receive feedback on our study design. Naturally, we exclude these individuals and this data from the overall study.

3.1.2 Rating Task

We wish to evaluate the *n* different dynamic algorithms against each other. Segmenting the study population into *n* randomized groups can make finding and managing a large enough sample difficult. More importantly, as we will show, controlled experiments on broad measurements for personal search behavior are statistically indistinguishable between groups; we believe users have difficulty judging subtle differences in search systems after the fact.

To that end, we also perform a repeated measures experiment. As we can safely run each algorithm independently, we concurrently construct relation graphs using both the temporal locality and causality algorithms in both groups. At the conclusion of the study, we choose up to *k* queries at random that were previously *successfully* executed by the user and re-execute them. Different views, in random order, showing each different algorithm’s results are presented; the user rates each of them independently using a 5-point Likert scale. We use these ratings to determine user-perceived differences in each search algorithm.

We define “successfully executed” to be queries where the user selected at least one result after execution. To prevent users from rating identical, singular result lists—which would give us no information—we further limit the list of successful queries by only considering those where at least one pair of algorithms differs in their orderings. With this additional constraint, we exclude an additional 2 queries from being rated.

The rating task occurs at the end of the study and not immediately after a query as we eschew increasing the cognitive burden users experience when searching. If users knew they had to perform a task after each search, they might avoid searches because they anticipate that additional task. Worse, they might perfunctorily complete the task if they are busy. In a longer study, it would be beneficial to perform this rating task at periodic intervals to prevent a disconnect with the query’s previous intent in the user’s mind. Previous work has shown a precipitous drop in a user’s ability to recall computing events after one month [17].

		Control	Treatment	Overall	
	Participants	14	13	27	
	Gender	F=7, M=7	F=5, M=8	F=12, M=15	$\chi^2_1 = 0.0464, p<0.830$
Mean across group	Age	26.2	25.6	26.0	$t_{25} = 0.3713, p<0.714$
	Computing Hours/Day	7.21	6.76	7.00	$t_{25} = 0.4298, p<0.671$
	Frequency of Computer Use†	4.64	4.77	4.70	$t_{25} = -0.5982, p<0.555$
	Computer Skill†	3.57	3.61	3.59	$t_{25} = -0.1405, p<0.889$
	Organizational Effort†	3.78	3.69	3.74	$t_{25} = 0.2773, p<0.784$

†5-point Likert ratings

Table 3.1: Self-reported demographics of the study population.

Finally, we re-execute each query rather than present results using algorithm state from when the query was first executed. The user's contextual state may change between when the query was executed and at the time of the experiment; any previous results could be invalid and may potentially cause confusion.

In the experiment, we chose k=7 queries to be rated by the user. We anecdotally found this to provide a reasonable number of data points without incurring user fatigue. Four algorithms were evaluated: content-only, causality, temporal locality and a "**random**-ranking" algorithm, which consists of randomizing the top 20 results of the content-only method.

The rating task measures intra-user disagreement between the algorithms. That is, for a query, the study examines a user's attitude between each algorithm's results for only that specific query. Content-only indexing is the baseline for comparison, or control, in this experiment; the study tests how much better or worse each other algorithm is against it. Inter-user disagreement between all users' intra-user disagreements then similarly generalizes the algorithms to the population at large.

Since users are evaluating their own queries, this method correctly captures intents. For example, users presented with the query "triangle" and a set of results may very differently judge whether the search was successful: each user may respectively believe the search refers to either a polygon, the musical instrument, or a three-party social situation, all dependent on their perceived intent. As this study is evaluating a user's own queries, it can correctly capture his or her stated intent.

3.2 Experimental Results

The study ran during June and July 2006, starting with 75 participants, all undergraduate or graduate students at the University of Michigan, recruited from non-computer science fields.

Each participant was required to run the software for at least 30 days, a period allowing a reasonable amount of activity to be observed while still maintaining a low participant attrition rate. Of the initial 75 participants, 27 (36%), consisting of 15 men and 12 women, completed the full study—more than four times the number of Soules and Ganger [85]. The self-reported demographics of the sample population are shown in Table 3.1. Those who successfully completed the study received modest compensation.

To prevent cheating, the system tracks its installation, regularly reporting if it's operational. We are confident that we identify users who attempt to run the tool for shorter than the requisite 30 days. The average study length was 32.3 days (σ=2.2, min = 30, max = 38). Further, to prevent users from creating multiple identities, participants must supply their institutional identification number to be compensated. In all, we excluded 4 users from the initial 75 because of cheating.

3.2.1 Randomized Controlled Trial

Evaluating end-to-end effects as in the controlled trial yields suggestive, though not statistically significant, results. Achieving significance would require a probative number of participants given our resources. This portion of the study indicates that the repeated measures experiment is a superior alternative for studying personal desktop search tools.

Figure 3.1 shows box-and-whisker plots of 5-point Likert ratings for key survey questions delineated by control and treatment group. For those unfamiliar: on a box-and-whiskers plot, the median for each dataset is indicated by the center dot, the first and third quartiles, the 25th and 75th percentiles respectively—the middle of the data—are represented by the box. The lines extending from the box, known as the whisker, represent 1.5 times this interquartile range and any points beyond the whisker represent outliers. The box-and-whiskers plot is a convenient method to show not only the location of responses, but also their variability.

Figure 3.1a is the pre- and post-measures difference on a Likert rating on search behavior: "When I need to find a file, it is easy for me to do so quickly." Figures 3.1b and Figure 3.1c are post-survey questions on if the tool would change their behavior in organizing their files (i.e., "I would likely put less effort in organizing my files if I had this tool available") or whether this tool should be bundled as part of every machine (i.e., "this tool should be essential for any computer"). With all measures, the results are statistically insignificant between the control and treatment groups ($t_{25} = -0.2876, p<0.776$; $t_{25} = 0.0123, p<0.9903$; $t_{25} = -0.4995, p<0.621$, respectively).

We also consider search behavior between the groups. Figure 3.2 on page 28 shows the rank of the file selected after performing a query. Those in the treatment group select items

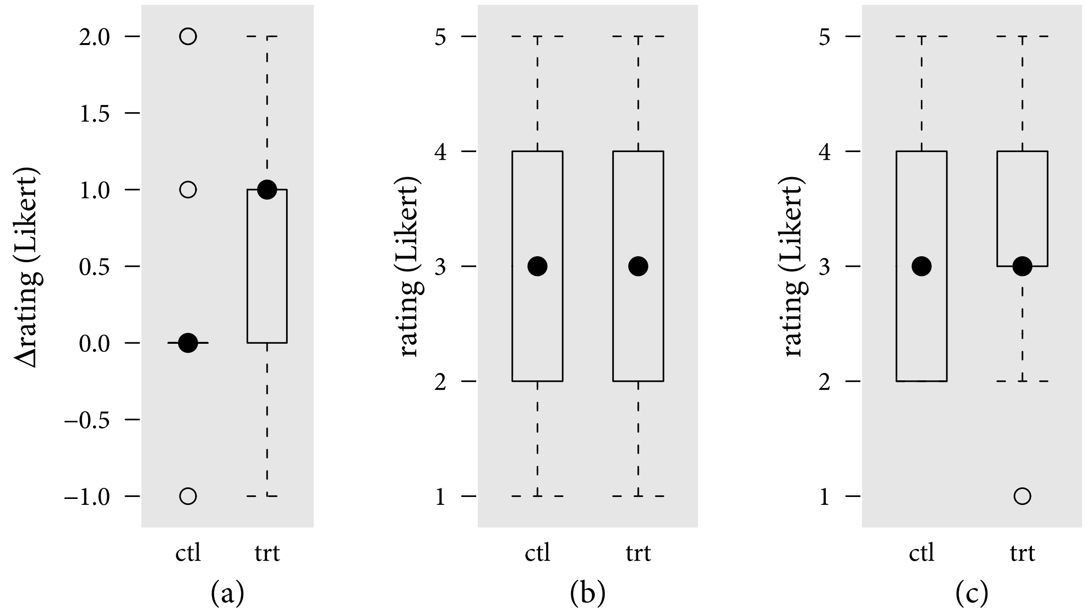


Figure 3.1: Pre- and post-measures differences between control and treatment groups: (a) *Difference* between pre- and post-measures 5-point Likert rating of "When I need to find a file, it is easy for me to do so quickly." While the treatment group has a slightly higher median difference, the results are statistically indistinguishable. (b) 5-point Likert rating of "I would likely put less effort in organizing my files if I had this tool available." (c) 5-point Likert rating of "This tool should be essential for any computer." (N = 27)

placed higher in the result list than those in the control group, although not significantly ($t_{51} = 1.759$, $p<0.0850$).

The treatment group returns more results on average (15.5 results per query) than the control group (11.1 results per query) as Figure 3.3 attests. While this may seem natural—the contextual reordering and extension mechanism is pooling additional items into the result set—the increase in the number of results is not statistically significant ($t_{180} = -1.443$, $p<0.1507$).

We divide query execution into sessions, where each session represents a series of semantically related queries. Following Cutrell et al. [16], we define a session to comprise queries with an inter-arrival rate of less than 5 minutes. The session length is the number of queries in a session, or, alternatively, the query retry rate. As Figure 3.4 on page 29 shows, the treatment group has a shorter average session length ($t_{97} = 2.136$, $p<0.042$), with geometric mean session lengths of 1.30 versus 1.66 queries per session, respectively. 13.5% and 19.0% of sessions in the control and treatment groups, respectively, ended with a user opening or previewing an item.

This data is, however, inconclusive. While at first blush it may appear that with the causality algorithm users are selecting higher ranked items and performing fewer queries for the

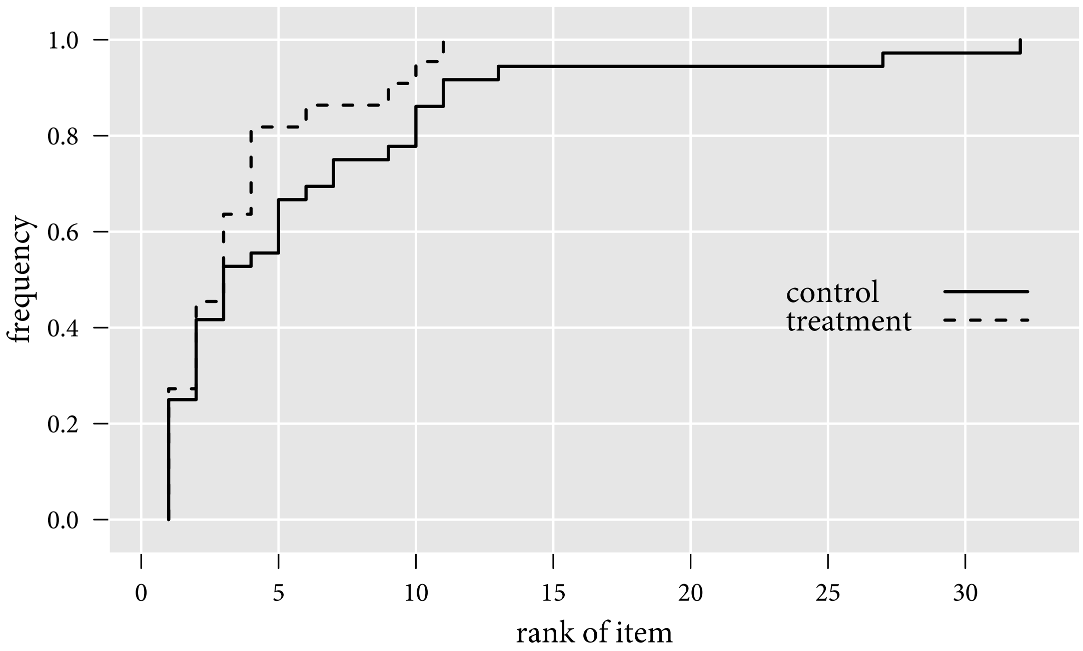


Figure 3.2: The rank of files opened by users after a search CDF.

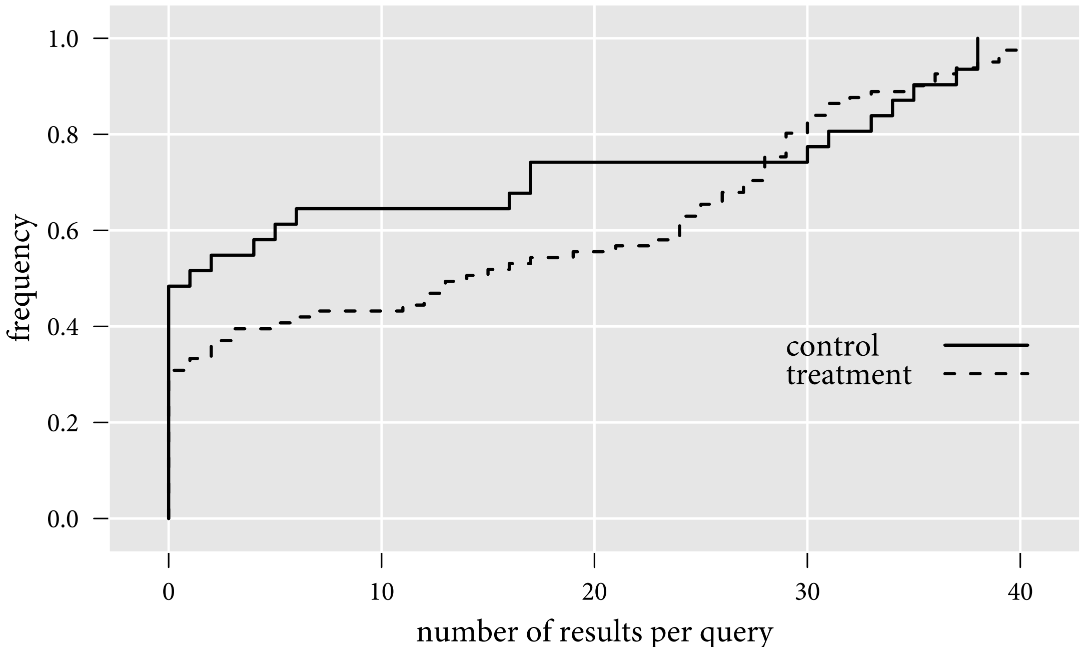


Figure 3.3: Number of results returned per query CDF.

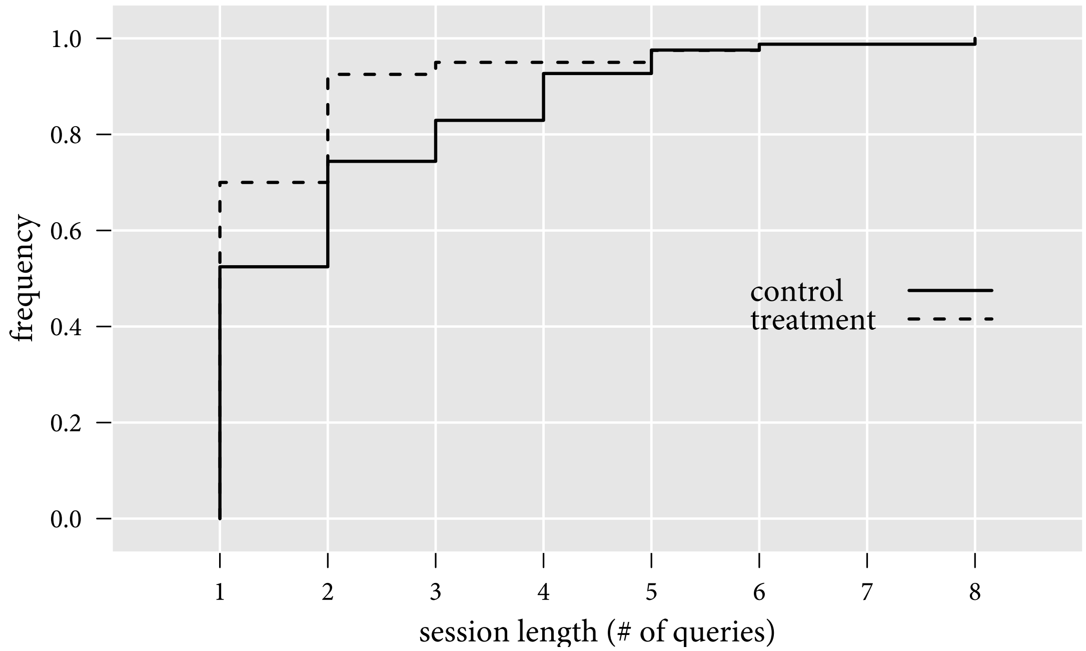


Figure 3.4: Session length CDF.

same informational need, it could be just as well that users give up earlier. That is, perhaps users fail to select lower ranked items in the treatment group because those items are irrelevant. Perhaps users in the treatment group fail to find what they are looking for and cease retrying, leading to a shorter session length. In hindsight, it would have been beneficial to ask users if their query was successful when the search window was closed. If we had such data available, we could ascertain whether shorter session lengths and opening higher ranked items were a product of finding your data faster or of giving up faster.

These results are suggestive of the benefits of context-enhanced search, but are not statistically significant. The lack of significant end-to-end effects stems from the relatively low sample size coupled with the heterogeneity of the participants. To achieve statistical significance, the study would require over 300 participants to afford the standard type II error of $\beta = 0.2$ (power t-test, $\Delta = 0.2$, $\sigma = 0.877$, $\alpha = 0.05$). Attaining such a high level of replication is prohibitively expensive given our resources. Instead, our evaluation focuses on the rating task.

3.2.2 Rating Task

The rating task yielded conclusive results. Sixteen out of the 27 participants rated an aggregate total of 34 queries, for an average of 2.13 queries per subject (σ=1.63). These 34 rated

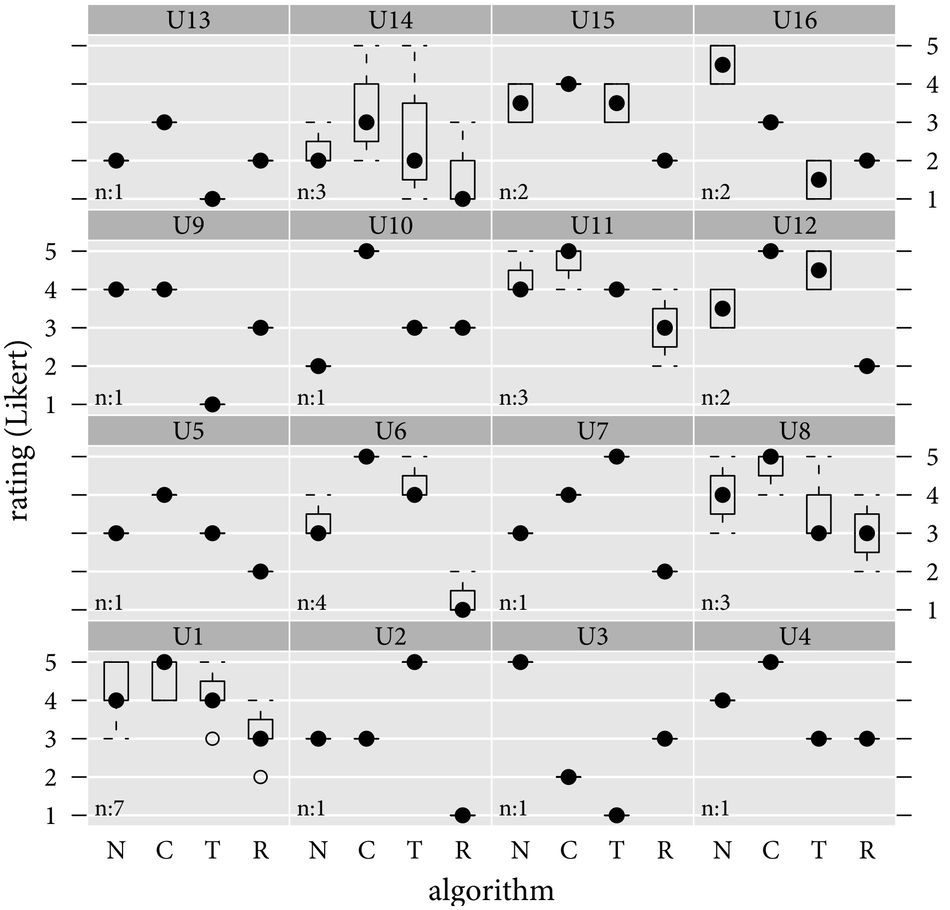


Figure 3.5: Box-and-whisker plots of each algorithm's ratings delineated by subject. The algorithms are: content-only ("N"), causality ("C"), temporal locality ("T"), and random ("R").

queries likely represent a better candidate selection of queries due to the "successfully executed" precondition (§3.1.2): the study only ask users to rate queries where they selected at least one item from the result set for that search. Eleven participants failed to rate any queries: 3 users failed to issue any, while the remaining 8 failed to select an item from one of their searches.

Those remaining 8 issued an average of 1.41 queries (σ=2.48), well below the sample average of 6.74 queries (σ=6.91). These likely represent failed searches, but it is possible that users employ search results in other ways. For example, the preview of the item might have been sufficient to solve the user's information need or the user's interest may have been in the file's path. Of those queries issued by the remaining 8, users previewed at least one item 17% of the time but never opened the file's containing directory through the interface. To confirm our suspicions about failed search behavior, it would have been beneficial to ask users as to whether their search was successful.

Figure 3.5 on the previous page shows a box-and-whiskers plot of each subject's ratings for each of the different algorithms. Subjects who rated no queries are omitted from the plot for brevity. Some cursory observations across all subjects are that the causality algorithm usually performs at or above content-only, with the exception of subjects U3 and U16. Temporal locality is on par or better than content-only for half of the subjects, but is rated exceptionally poorly, less than a 2, for a quarter of subjects (U3, U9, U13 and U16). Surprisingly, while the expectation is for random to be exceedingly poor, it is often only rated a notch below other algorithms.

Rigorous evaluation requires care as we have multiple observations on the same subject for different queries—a repeated measures experiment. Observations on different subjects can be treated as independent, but observations on the same subject cannot. Thus, we develop a mixed-effects ANOVA model [75] to test the statistical significance of our hypotheses.

Let y_{ijk} denote the rating of the i-th algorithm by the j-th subject for the k-th query. Our model includes three categorical predictors: the subject (16 levels), the algorithm (4 levels), and the queries (34 levels). For the subjects, there is no particular interest in these individuals; rather, the goal is to study the person-to-person variability in all persons' opinions. For each query evaluated by each subject, we wish to study the query-to-query variability within each subject's ratings. The algorithm is a fixed effect (β_i), each subject then is a random effect (ζ_j) with each query being a nested random effect (ζ_{jk}). Another way to reach the same conclusion is to note that if the experiment were repeated, the same four algorithms would be used, since they are part of the experimental design, but another random sample would yield a different set of individuals and a different set of queries executed by those individuals. Therefore, our model is as expressed in Equation (3.1).

$$y_{ijk} = \beta_i + \zeta_j + \zeta_{jk} + \epsilon_{ijk} \tag{3.1}$$
$$\zeta_j \sim \mathcal{N}(0, {\sigma_1}^2) \quad \zeta_{jk} \sim \mathcal{N}(0, {\sigma_2}^2) \quad \epsilon_{ijk} \sim \mathcal{N}(0, \sigma^2)$$

A maximum likelihood fit of Equation (3.1) is presented in Table 3.2. Each β_i represents the mean across the population for algorithm i. The temporal locality algorithm is statistically indistinguishable from content-only search ($t_{99} = -0.880$, p<0.3812), while the causality algorithm is rated, on average, about 17% higher ($t_{99} = 2.93$, p<0.0042). Random-ranking is rated about 36% worse on average ($t_{99} = -6.304$, p<0.0001).

Why is temporal locality statistically indistinguishable from content-only? Based on informal interviews, we purport the cause of these poor ratings is temporal locality's tendency to build relationships that exhibit post-hoc errors: the fallacy of believing that temporal succession implies a causal relation.

		95% Conf. Int.		
Algorithm	β_i	Lower	Upper	p-value†
Content only	3.55	3.16	3.93	
Causality	4.13	3.75	4.52	0.0042
Temporal locality	3.37	2.98	3.76	0.3812
Random	2.28	1.93	2.67	$<$0.0001
σ_1	0.383	0.176	0.831	
σ_2	0.486	0.294	0.805	
σ	0.815	0.710	0.935	

†In comparison to content-only.

Table 3.2: MLE estimate of the mixed-effects model given in Equation (3.1).

For example, U16 was a CAD user that only worked on a handful of files for most of the tracing period (a design she was working on). The temporal locality algorithm caused these files to form supernodes in the relation graph; every other file was related to them. Under results generated by the temporal locality algorithm, each of her queries included her CAD files bubbled to the top of the results list. U9 was mostly working on his dissertation and every file, as well as some of his music, was lightly related to each other. The temporal locality algorithm created a relation graph with 21,376 links with geometric mean weight of 1.48 (σ_l=0.688); the causality algorithm, an order of magnitude fewer, with 1,345 links and a geometric mean weight of 9.79 (σ_l=1.512). In his case, it appears that the temporal algorithm naïvely created many lightly-weighted superfluous relations compared with the causality algorithm.

A user's work habits will affect the utility of provenance analysis techniques. Temporal locality's tendency to generate large numbers of lightweight false-positive relationships can be detrimental in many cases, making more conservative techniques such as causality more broadly applicable.

The random reordering shares equivalent precision and recall values as content-only search, but is rated about 35.7% worse on average. We expect a random ordering to do phenomenally worse, but hypothesize that personal search tools are still in their infancy. That is, attention in the research community has been placed on web search; only recently has desktop search become a priority, and there is appreciable room for improvement. It may also be that users are simply content with having their desired result on the first page and are apathetic to each result's relative ordering within that page. More work is required to understand a user's perception of search orderings.

We further analyze any interactions between other covariates such as the demographics of participants or user features (e.g., size of disk, number of files, folder depth). We find these covariates either to be statistically insignificant or, because of the limited sample size, to over fit any model.

3.3 Performance

The results indicate that context-enhancing search with the causality algorithm building the contextual index increases user satisfaction with their file search results. Such a system, however, is only effective if minimum additional system resources are required for building, storing, and traversing the relation graph created by this algorithm. This section eschews discussion of content indexing overheads as these are already known [60].

3.3.1 Tracing Performance

We measure the impact building the relation graph has on foreground performance with the Postmark synthetic benchmark [45]. Postmark is designed to measure file system performance in small-file Internet server applications such as email servers. It creates a large set of continually changing files, measuring the transaction rates for a workload of many small reads, writes, and deletes. While not representative of real user activity in desktop systems, Postmark represents a particularly harsh setup for the collection daemon: many read and write events to a multitude of files inside a single process. Essentially, Postmark's workload creates a densely-connected relation graph.

We run 5 trials of Postmark, with and without tracing, with 50,000 transactions and 10,000 simultaneous files on an IBM Thinkpad X24 laptop with a 1.13 GHz Pentium III-M CPU and 640 MB of RAM, a meek machine by today's standards. The results are shown in Figure 3.6a. Under tracing, Postmark runs between 7.0% and 13.6% slower (95% conf. int.; t_8=7.211, p<0.001). Figure 3.6b shows a CDF of Postmark's transaction times with and without tracing across a single run. There is a relatively constant attenuation under tracing, which reflects the IPC overhead of the collection daemon and the additional disk utilization due to relation graph updates. This additional slowdown caused by relation graph construction is in line with other Win32 tracing and logging systems [55].

3.3.2 Space Requirements

We examine the additional space required by the relation graphs. During the user study, the tool logged the size of each relation graph every 15 minutes. Figure 3.7 on the following page

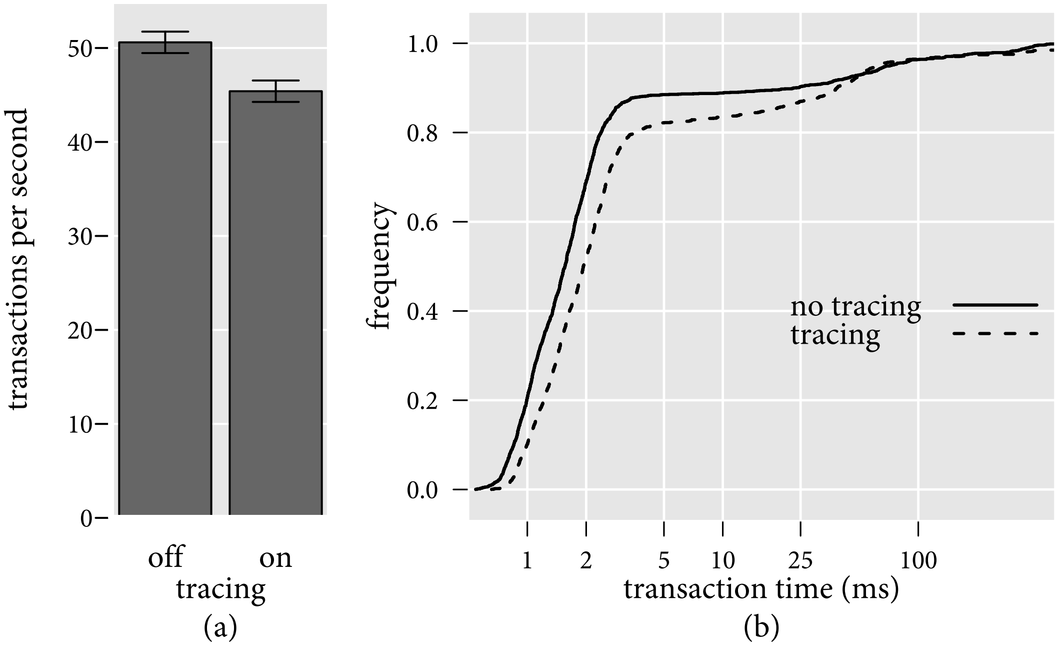


Figure 3.6: Comparison of running Postmark with and without tracing. (a) Comparison of running 5 trials of the Postmark synthetic benchmark with and without tracing on. The error bars represent 95% bootstrap confidence intervals. (b) CDF of transaction times for a single Postmark run when tracing is on or off.

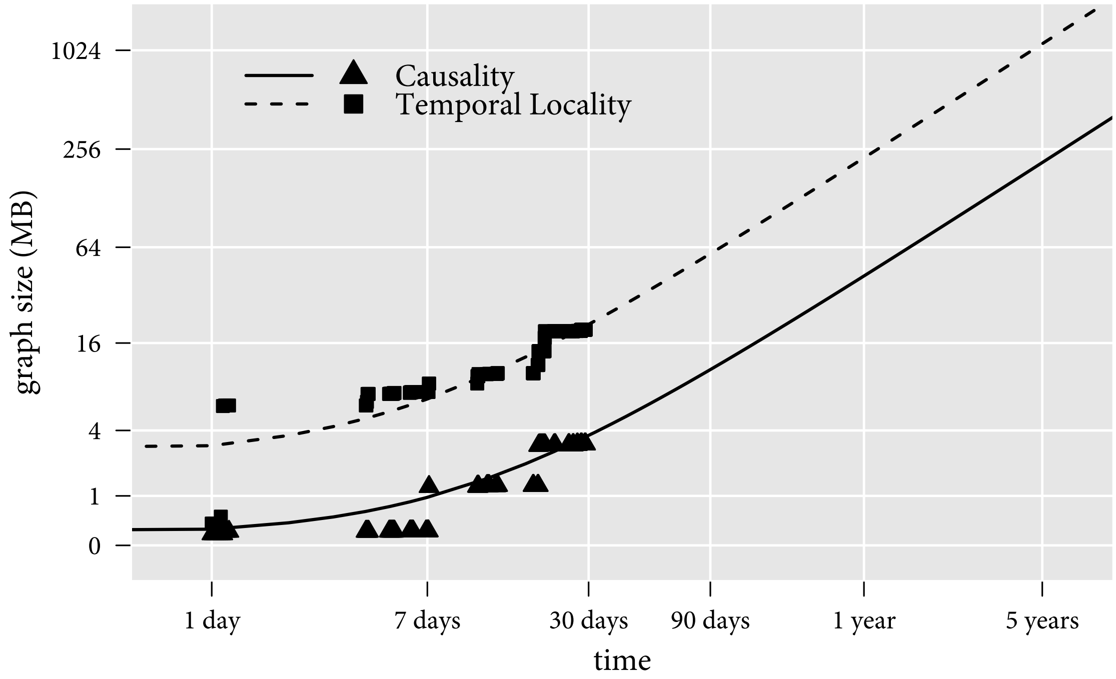


Figure 3.7: Relation graph growth curves for U3, the heaviest user in the study.

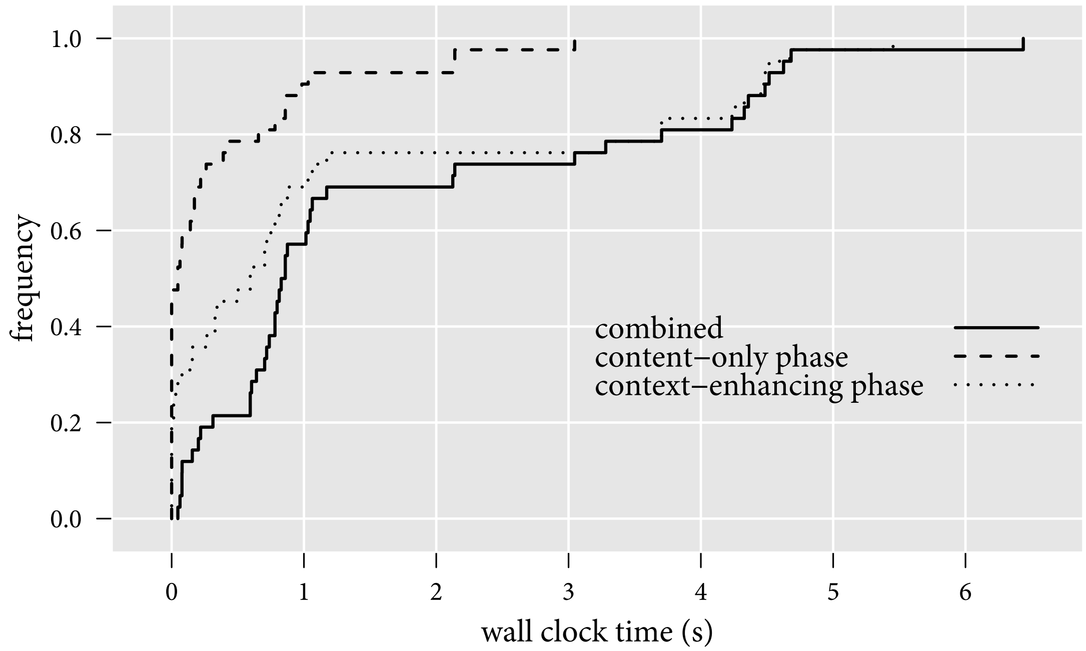


Figure 3.8: Search times CDFs for queries issued during the user study.

shows relation graph growth over time for the heaviest user in the sample, U3. Each relation graph grows linearly ($r^2 = 0.861$ and $r^2 = 0.881$ for causality and temporal locality, respectively). For those unfamiliar, the coefficient of determination, termed r^2, is a measure of correlation: it represents the fraction of variance explained by the proposed model. For example, an r^2 of 0.90 indicates that the model accounts for 90% of the variance in the data, signaling a very good fit.

While the worst case graph growth is $O(F^2)$, where F is the number of files on a user's system, these graphs are generally very spare: most files only have relationships to a handful of other files as a user's working set at any given time is very small. In one year, we expect the causality relation graph for U3 to grow to about 44 MB; in five years, 220 MB. This is paltry compared to the size of modern disks and represents an exceedingly small fraction of the user's working set. These results suggest that relation graph size is not an obstacle.

3.3.3 Search Performance

The time to answer a query must be within reasonable bounds for users to find the system usable. In our implementation (§2.4), we bound the context-enhancing phase to a maximum of 5 seconds.

For every query issued during the user study, we log the elapsed wall clock time in the content-only and context-enhancing phases. Figure 3.8 shows these results. Half of all

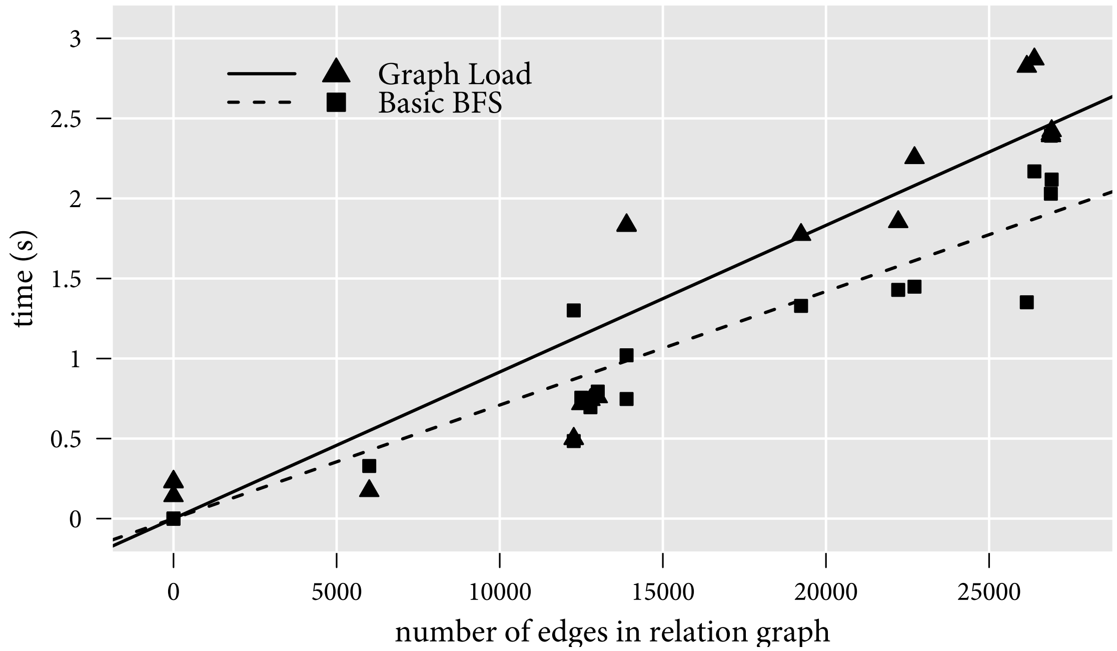


Figure 3.9: Performance profile of the context-enhancing phase. For queries issued during a 6-month trace of the author's system, the time spent loading the relation graph and executing the basic BFS algorithm against the number of edges in the relation graph. (5 trials per query; standard deviations were within 2% of the mean for each data point and are therefore omitted from the plot.)

queries are answered within 0.8 seconds; three-quarters within 2.8 seconds, but there is a heavy tail. The context-enhancing phase takes about 67% of the entire search process on average. We believe these current search times are within acceptable limits.

Recall that the context-enhancing phase consists of two distinct subphases: first, the loading of the relation graph, followed by execution of the basic BFS algorithm (§2.2). To understand the performance impact of these subphases, previous queries issued by the author were re-executed for 5 trials each under a cold cache with the relation graph from a 6-month trace. Figure 3.9 shows the time spent for each query based on the number of edges from the relation graph loaded for that query. For non-empty graphs, loading the relation graph took, on average, between 3.6% and 49.9% longer (95% conf. int.) than the basic BFS subphase (paired $t_{15} = -2.470$, $p<0.026$).

Both loading the relation graph and basic BFS execution support linear increase models ($r^2 = 0.948$ and $r^2 = 0.937$, respectively) with regard to the number of edges in the graph. This is apparent as each subphase requires both $\Omega(F^2)$ space and time, where F is the number of files on a user's system. As $E = O(F^2)$, where E is the number of edges, this naturally translates to the linear $O(E)$ growth observed. As these are lower bounds, the only way to save space and time would be to ignore some relationships. If we could predict *a priori*

Question	$\tilde{\mu}$	IQR
I would prefer an interface that shows more information.	4.0	1.0
I find it easy to think of the correct search keywords.	4.0	1.0
I would prefer if I could look over all my machines.	4.0	3.0
I like the interface.	3.0	2.0
This tool should be essential for any computer.	3.0	1.0
I would likely put less effort in organizing my files if I had this tool available.	3.0	2.0
I would prefer if my email and web pages are included in the search results.	2.0	3.0

Table 3.3: Specific user feedback at the end of the study. At the conclusion of the study, additional feedback, with 5-point Likert ratings, was solicited from treatment group users ($N = 13$).

which relationships were most relevant, we could calculate, at the expense of accuracy, Equation (2.1a) for those pairs. Further, we could cluster those relevant nodes together on disk, minimizing disk I/Os during graph reads.

3.4 User Feedback

During the post-survey phase of the study, the questionnaire contained additional 5-point Likert ratings. A tabulation of subject's responses for the treatment group are shown in Table 3.3. Likert ratings are treated as an ordinal level of measurement and are hence summarized by the median and, as a measure of variance, the interquartile range (IQR), which is the difference between the third and first quartiles or, alternatively, the middle 50% of the data.

An area for improvement is the user interface. Our results are presented in a list view (Figure 2.4), but using more advanced search interfaces, such as Phlat [16], that allow filtering through contextual cues may be more useful. Different presentations, particularly timeline visualizations, such as in Lifestreams [25], may better harness users' memory for their content. There is a relatively strong positive correlation ($\rho = 0.698$) between liking the interface and finding the tool essential; a better interface will likely make the tool more palatable for users.

Based on informal interviews, we found that participants used the search tool as an auxiliary method of finding content: they first look through their directory hierarchy for a particular file, switching to keyword search after a few moments of failed hunting. Participants neglect to use the search tool as a first-class mechanism for finding content. A system that

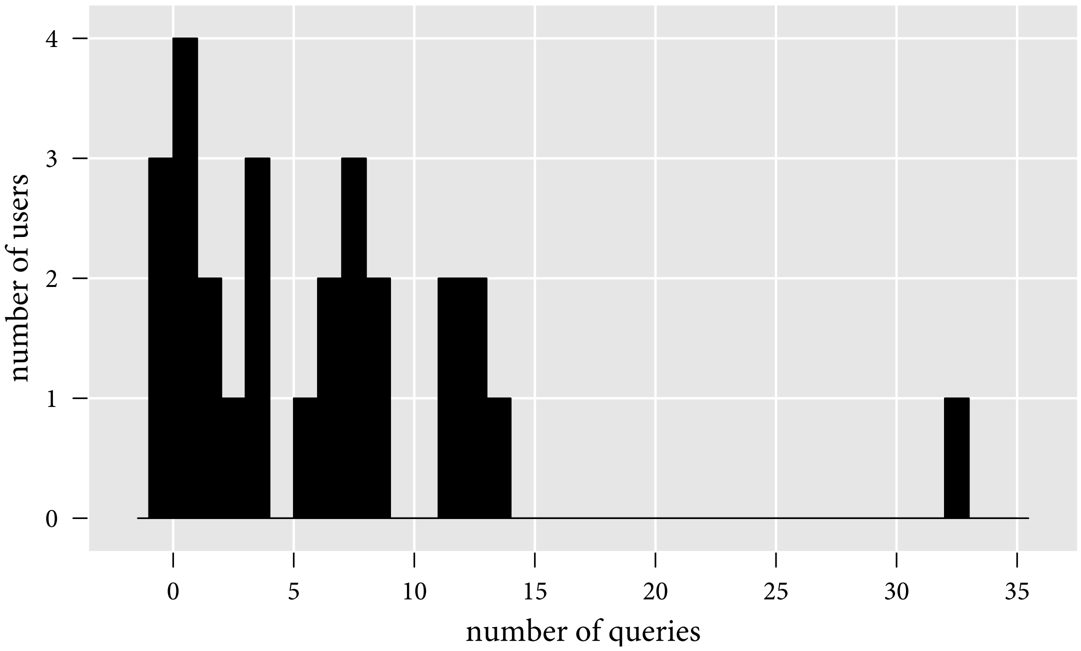


Figure 3.10: Distribution of the number of queries among users.

is integrated into the OS, including availability from within application "open" dialogs, may cause a shift in user's attitudes toward using search to find their files. Indeed, new operating systems available after this study was conducted now include this functionality.

We found it surprising that users wished to exclude email and web pages from their search results; two-thirds of users rate this question a three or below. Our consultations reveal that many of these users dislike a homogeneous list of dissimilar repositories and would rather prefer the ability to specify which repository their information need resides in. That is, a user knows if they're searching for a file, email or web page, let them easily specify which. We need not focus on mechanisms to aggregate heterogeneous forms of context spread across different repositories into a unifying search result list, but to simply provide an easy mechanism to refine the search to a specific repository.

3.5 Personal Search Behavior

Finally, as part of this work, we explore the search behavior of the sample population. To maintain participants' privacy, we are only able to provide aggregate data points. Also, recall that for privacy reasons we do not log any information about the content of users' indices or search results.

Our population issued 182 queries; the distribution per user is shown in Figure 3.10. The average number of queries issued per user is 6.74 (σ=6.91). Most queries, 91%, were fresh,

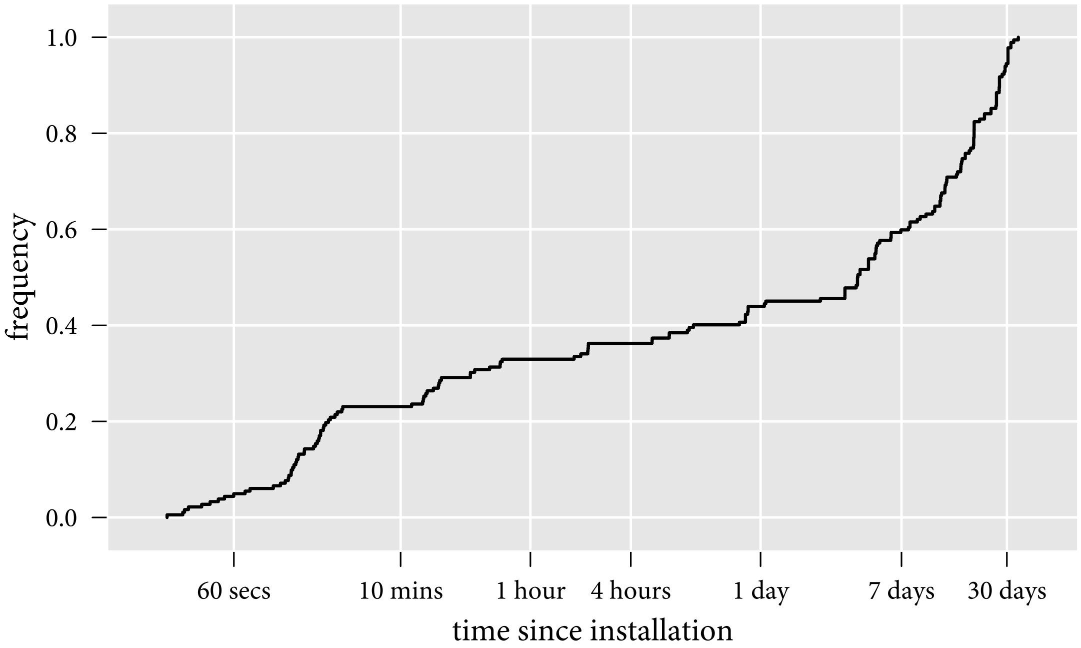


Figure 3.11: When queries are issued after installation CDF.

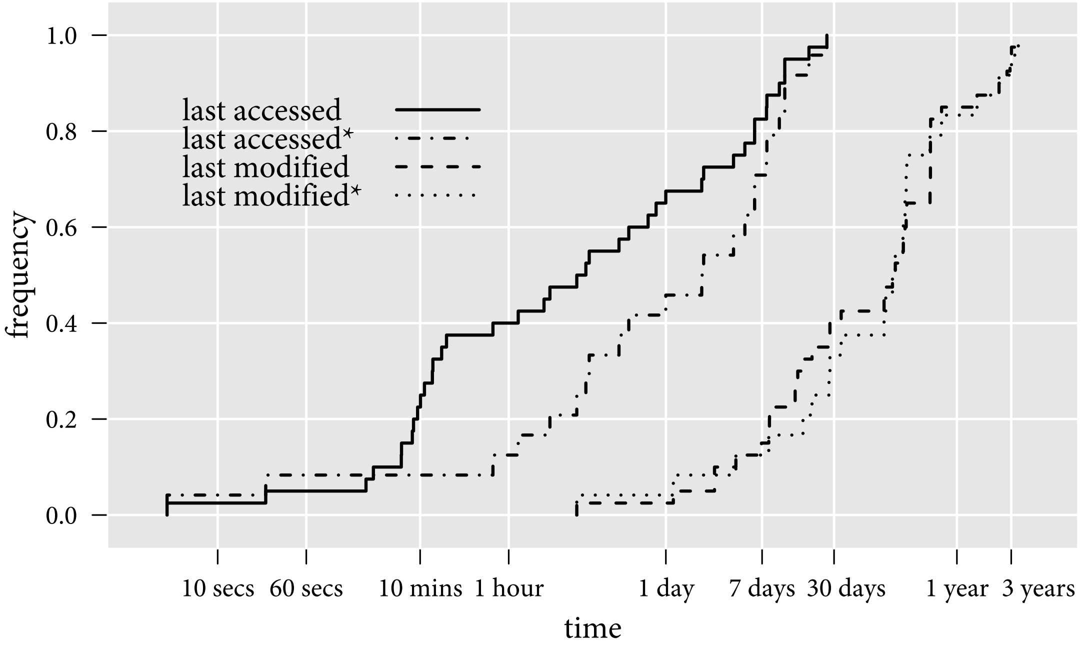


Figure 3.12: Access and modification times of items selected from the results list CDF. The starred versions exclude searches conducted during the first day after installation.

having never been issued before. About 9% of search terms were for filenames. Since Windows XP lacks a rapid search-by-filename tool similar to UNIX's slocate, users were employing Weft to find the location of files they already knew the name of. Most queries were very short, averaging 1.16 words (σ=0.462). This is slightly shorter than the 1.60 and 1.59 words reported for the desktop tools Phlat [16] and SIS [23], respectively, and comparatively very short to the average of 2.8 words for web searches [102].

Figure 3.11 on the preceding page shows when queries are issued after installation. A sizable portion of queries are issued relatively soon after installation as users are playing with the tool. Even though the tool warns users that search results are initially incomplete because the content indexer has not built enough state and the relation graph is sparse (§2.4), it may be prudent to disallow searching until a reasonable index has been built as not to create an unfavorable initial impression.

Figure 3.12 shows the last access time and last modification times of items opened after searching. The starred versions represent last access and modification times of queries issued at least a day after installation. During the first day, users might be testing the tool against recent work and, hence, recently accessed files. Anecdotal evidence of this effect can be observed by the shifted last accessed curve. After the warm-up period, half of all files selected were accessed within the past 2 days. It appears users are employing the tool to search for more than archival data.

3.6 Conclusion

By measuring users perception of search quality with the rating task (§3.2.2), this research shows that using causality (§2.1.2) as the dynamic re-indexing component increases user satisfaction in search, being rated 17% higher on average over all queries than content-only indexing or temporal locality. While the contextual search mechanism only suggested increases in end-to-end effects in the randomized controlled trial (§3.2.1), this stemmed from an insufficiently large sample size. It is prohibitively expensive to secure such high levels of replication, making the rating task a more appropriate methodology for evaluating personal search systems. These results validate that using the provenance of files to reorder and extend search results is an important complement to content-only indexing for personal file search.

PART II

SHARING CONTEXT

CHAPTER 4

LIGATURE—AN ARCHITECTURE FOR SHARING CONTEXT

> *There is no delight in owning anything unshared.*
>
> Seneca (c. 4 BC–AD 65)

Sharing objects, such as sending a paper out for review or sharing photos among friends and family, is an everyday occurrence. These objects are often associated with contextual information that describes and extends them. Unfortunately, most sharing mechanisms lose this context. Even when static context is preserved, it cannot be easily modified by the spontaneous collaborations formed by object sharing. This research presents *Ligature*[1], a service that supports shared contextual data across a user's social neighborhood, allowing different groups access to different contexts as appropriate. The system requires no common administrative domain of control and employs an eventually-consistent model, aiding scalability

[1] A ligature is something that bonds or connects. For instance, in music, a ligature is a symbol that connects multiple notes together in some meaningful way.

and user mobility. This research further describes a prototype of this architecture and shows that context can be shared with minimal overhead, while a user study demonstrates that the system's disclosure model scales with task difficulty.

4.1 System Features

Part I: *Employing Context* showed the usefulness of context, particularly in aiding the retrieval of data. Unfortunately, while this context proves useful, it is only maintained locally. However, when a party creates or captures context, they may want to share it with other interested parties. To illustrate this, consider a researcher presenting at a conference and after meeting a colleague from another institution, wishing to share some additional data with her. Typically, this is done via email [98], though any context—for example, provenance information [63] or links to other relevant data [3]—is lost entirely. This context could be embedded inside the objects themselves or sent via some context-aware archive, but that makes updates—especially with collaborative contextual applications—extremely painful.

Context sharing, as the preceding example exemplifies, often occurs impetuously among ad-hoc groups without a common administrative domain of control: heavyweight mechanisms providing direct support for contextual sharing along with authentication and access control are unlikely to provide value. In addition, any context sharing system should not excessively burden users by making context a new entity to manage. For instance, requiring users to explicitly delineate the context shared to each respective peer quickly becomes a management nightmare. However, any implicit sharing of context must respect a user's boundaries of disclosure. The researcher, for example, will likely share different context with her colleague than with her research group or her boss, even though the underlying object is the same.

Ligature is a system supporting flexible management and sharing of context among spontaneous groups of collaborators. Figure 4.1 shows how services interact with the Ligature service. In this scenario, there are two services, a note taking service, where users can attach notes to objects, and a tagging service, where free-form labels can be assigned to objects. The Photo Viewer uses a standard tags API to retrieve and set tags; this API performs calls to the tags service as needed. Each service, each with its own presentation layer, uses the architecture's primitives to respectively store, remove, and query an object's context. Ligature is responsible for transmitting context to other interested parties.

Ligature's model of context departs from the current state of affairs in three ways. First, it provides a separation of concerns between an object and its context. Rather than store context in the object itself, Ligature's model allows context to be structured, and to grow and

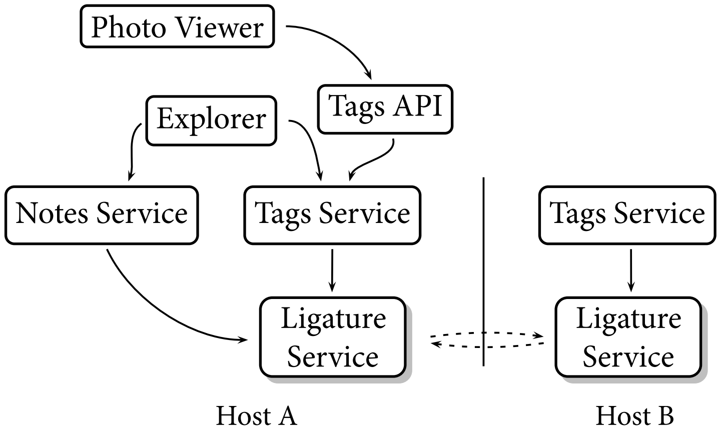


Figure 4.1: Illustration of how Ligature and its services interact. In this example, applications (e.g., a photo viewer) hook into respective services that run atop the Ligature service, which handles communication and contextual exchange among a user's social neighborhood.

evolve over time. Second, rather than tie context to third-party providers, Ligature ensures a user's context is her own to do with as she wishes. Third, and most important, it provides reasoned conditions for implicit context sharing by exploiting acts of data sharing to automatically exchange related context. The system does not require centralized repositories, strong identities, or a single domain of administrative control.

In Ligature, users spontaneously form communities based on the objects they share. Those holding a particular object form a *coterie*, also sharing context on that object. This model places boundaries on confidentiality and disclosure: the trust the sender imbues by sharing the object with the recipient carries over to its context (i.e., "I sent you the object, you get its context."). The resulting conditions for contextual exchange are simple and deterministic, and match user expectations about the meaning of sharing [95, 98].

This one-to-one mapping between objects and contexts is adequate for the common case, but breaks down when different contexts for the same object are necessary. To support these boundaries of disclosure, Ligature extends the model with the addition of explicit *contextual frames*—enclosures around some context about an object—along with primitives for manipulating them. Frames accrue context and follow the objects to which they are bound. A user can bind more than one frame to an object, and duplicate, combine, or destroy them. Distributed replicas of a frame are kept eventually-consistent, preserving scalability and user mobility.

Ligature is self-organizing, requiring no centralized infrastructure. For disseminating context, Ligature uses a publish/subscribe model [24] that prefetches an object's context, eliminating wide-area latencies from the critical path and coping with disconnected peers. The resulting system is lightweight, and usable even on modest platforms [97].

As an example of a solution with Ligature, consider a researcher preparing to give a talk at another lab overseas. Before leaving, she asks her co-author, who is located at another university and does not share a common administrative domain, for a copy of the "slides for the talk." Upon arriving, she discovers there are a few graphs she needs to be able to present the details behind the experiments. Unfortunately, due to time zone differences, her colleague is unavailable. Rather, she asks her system for the "source documents" for the talk; those documents that were most important in actually preparing this set of slides. It retrieves, via provenance records [63] stored as context and already fetched by Ligature, a set of candidate links from which she can choose. Importantly, work on a similar topic prepared by her colleague as a consultant to another company is not made available to her; that information is in a frame not disclosed. Among the candidate source documents is the original conference paper. Armed with this paper, she is now prepared for her presentation.

As part of this work, we constructed a prototype of Ligature and a common set of applications. This prototype indicates the additional mechanisms have a negligible performance impact. We also performed a user study, giving both novice and expert computer users several tasks of varying difficulty to complete using Ligature's frame mechanisms. The results of this study are encouraging as the mechanisms scale with task difficulty: novice users were able to complete the easy and moderate tasks in almost all cases, while expert users were able to complete the difficult tasks as well.

4.2 Related Work

The closest system to Ligature is Graffiti [58]. Graffiti uses a client-server model, which limits its usefulness among collaborators not belonging to a single administrative domain. Context is shared via object similarity in Graffiti, not based on sharing and disclosed through contextual frames, as is with Ligature. While Graffiti supports two separate contextual applications, tags and explicit relationships between files, Ligature is general; its model supports a large variety of applications, including those offered by Graffiti.

Several eventually consistent replicated systems exist. Coda's [48] disconnected reconciliation protocols allow servers and mobile clients to reconcile with each other. In addition, Ficus [37] and Ivy [65] both develop optimistic replication algorithms for distributed file systems. Bayou's [91] anti-entropy protocol is based on pair-wise communication between hosts and propagation of write operations; ordering constraints, as well as application-supplied conflict resolvers on write-write conflicts, maintain consistency.

Publish/subscribe systems [24] decouple senders and listeners of data by arbitrating messages through a broker. Subscribers express their interest in a set of events and are notified when a publisher produces such an event. Ligature is such a publish/subscribe system where consumers subscribe to the context of a particular object.

Contextual frames are based on information flow controls [66], which curb improper data dissemination through labeling and information flow analysis. There is a differing scope of these techniques: flow controls are a fine-grained mechanism to respect constraints on data sharing, while frames are merely a disclosure barrier.

There is substantial related work within both the semantic web and collaborative authoring communities. We use their work in semantic markup languages, particularly RDF [59], as the basis for Ligature's knowledge base. Additionally, web annotations [92] permit the semantic mark-up of web documents, but chiefly concerns itself with knowledge base representation (i.e., RDF) and automatic and manual tools to ease user burden in distilling these annotations. Actual dissemination is assumed to be a centralized store. Similarly, the purpose of collaborative authoring systems (e.g., MESSIE [81]) is to facilitate group editing and digital whiteboarding, not to act as a framework for storing and exchanging semantic metadata. They are therefore centralized, shunning ad-hoc group formation; strongly consistent; and usually only support a flat, unstructured data layout.

Considerable work exists in merging the semantic web with peer-to-peer technologies [87]. RDFPeers [12] is a peer-to-peer system where users subscribe to RDF content. It uses a DHT [88] to answer attribute and range queries over this RDF data. This system, and decentralized RDF storage and querying systems like it (i.e., Edutella [68] and its successor [69]), are orthogonal to Ligature. Ligature enables disconnected operation and engenders contextual exchange around shared objects. Particularly with the latter, it also permits sharing disparate context among the same or different peers through the use of contextual frames.

Several systems leverage forms of context for improved organization. Phlat [16] is a user interface for personal search that also provides a mechanism for tagging data. Phlat's implementation ties tags to the local file system by storing them in resource forks in the file system. The Semantic file system [29] permits applications to store file attributes, in the form of key-value pairs, within the file system. LiFS [3] extends this model to support inter-file links as well. Sedar [57] is a peer-to-peer semantic file system that uses the concept of semantic hashing to facilitate clustering of semantically-close objects. The Haystack project [43] is a personal information manager that organizes data, and operations on data, in a context-sensitive manner. These systems only deal with context at the local level. By employing Ligature, synergism among a user's social neighborhood could be exploited.

CHAPTER 5

CONTEXTUAL FRAMES

> *A set of ideas, a point of view, a frame of reference is in space only an intersection, the state of affairs at some given moment in the consciousness of one man or many men, but in time it has evolving form, virtually organic extension.*
>
> John Dos Passos (1896—1970)

Ligature is built around the notion of a *contextual frame*: an abstraction that encapsulates context for a particular perspective of an object. The system provides mechanisms for frame creation, modification, deletion, and combination. These mechanisms are integrated into file system operations, providing a natural, yet transparent set of extensions, both within a single user's namespace as well as across disjoint users' stores. To illustrate the nature of contextual frames, we provide example scenarios, and conclude with a discussion of these design points.

5.1 Frames: Encapsulating Context

The **contextual frame** is Ligature's central abstraction. It is the unit for encapsulating context, and the means by which context is associated to objects. The frame-object relationship is many-to-one: each frame must be **bound** to exactly one object, but an object may have zero or more frames bound to it. A frame consists of: a globally unique, opaque identifier; a pointer to the object it is describing; and the object's context.

Context acquired for an object is always associated with some particular frame—the frame of reference. Context is stored within the frame itself in a structured manner and can be any serializable data, including links to other objects. Context is represented as a log of idempotent notation operations, which exclusively consists of adding or removing contextual data. Applications can also inspect the context stored within frames or across the entire context repository through the use of a query language, as well as establish callbacks to be notified when some contextual key changes.

The hosts possessing a frame are said to be in its **coterie**. Such possession imbues both read and write access to all coterie members for all of a frame's context. It is unclear whether enabling policies other than full rights—which would complicate the system by requiring ownership semantics—is indeed useful, especially as almost all interpersonal sharing grants full rights [98] and there is no evidence to suggest context is any different. Judging how well this fits with Ligature's use cases is still ongoing research.

When an object is shared, its bound frames are also shared. Distributed updates are reconciled with an eventual consistency mechanism [91], aiding scalability and the ephemeral nature of peers. This meets most users' expectations as it mirrors the consistency semantics of other successful distributed systems, mainly the world wide web. Particularly, with web search, most engines are not particularly fresh—pages are re-indexed with a periodicity of days [54]—but the popularity of these sites is testament to their usefulness. The unified log is assumed to have a total ordering. When causal ordering is not sufficient to provide such a total order, one can be imposed arbitrarily by any unique client identifier. This scheme allows for semantically conflicting updates, but contextual systems typically already take on this burden. For example, tagging systems might use histograms to represent differing opinions amongst users [31].

5.2 Frame Manipulation

It is reasonable to expect that people will share the same object for a variety of purposes, and so the same object may well be bound to different frames at different times and circumstances. Ligature provides several mechanisms to manipulate the frames binding an object, supporting this notion of coteries. Frames may be controlled explicitly through the use of an API, or implicitly as a side-effect of object operations.

5.2.1 Frames API

Table 5.1 on the next page presents the frame operations. Frames can be created via either **new** or **fork**. The new operator creates a blank frame, while fork creates a new frame with a copy of the old frame's contents. After one frame is forked from another, the two remain independent; updates to one do not affect the other, and vice versa. Frames are unbound

$F = \textbf{new}(O)$	Creates a new frame F bound to object O.
$L = \textbf{catalog}(O)$	Returns the list of frames, L, bound to object O.
$\textbf{expire}(F)$	Unbinds frame F, marking it ready for garbage collection.
$F' = \textbf{fork}(F)$	Returns a duplicate of frame F bound to its same object.
$\textbf{attach}(F, G)$	Joins context from frame G to frame F forever (i.e., $F = F \cup G$).

Table 5.1: Summary of the frame operations.

```
once(F, G):
    $G' = \textbf{fork}(G)$
    $\textbf{attach}(F, G')$
    $\textbf{expire}(G')$
```

(a)

```
functional(F, G):
    $H = \textbf{fork}(F)$
    $\textbf{attach}(H, F)$
    $\textbf{attach}(H, G)$
    return $H$
```

(b)

```
symmetric(F, G):
    $\textbf{attach}(F, G)$
    $\textbf{attach}(G, F)$
```

(c)

Figure 5.1: *Common attach idioms.* (a) Join context in G's frame into F's frame *only once.* (b) A side effect-free version of attach: return a new frame with context linked between frames F and G. (c) Join context symmetrically between F and G.

from their objects by the **expire** operator, and are marked for later deletion, which occurs when any context has been safely reconciled. An application can obtain the **catalog** of all frames for an individual object.

The most interesting operator is **attach**, which takes two frames, F and G, and from that point on, context added to G is also propagated to F. In effect, attach confers trust from the source frame's coterie to that of the sink frame. Attach is perpetual, imperative, and asymmetric. Those who prefer one-shot semantics to perpetual propagation can achieve them by using the once idiom in Figure 5.1a. Likewise, one can produce a side effect-free union of two frames using the functional idiom in Figure 5.1b. Finally, reciprocal attach operations produce symmetric results with symmetric, as in Figure 5.1c.

5.2.2 Implicit Frame Changes

While users could explicitly perform all frame manipulations, doing so would be an unreasonable burden and will undoubtedly produce ambiguities. Instead, Ligature performs implicit manipulations as a side effect of object operations in an effort to handle most cases. Table 5.2 summarizes these events.

On creation, an object is given a new, blank frame. Reading an object does not change its frame state. An object that is renamed retains its current set of frames, and a copy of an existing object is bound to forked copies of the existing frames. File deletion removes

P **creates** O	Object O receives a new, blank frame.
P **inspects** O	—.
P **renames** O **to** O'	All frames follow the name change.
P **copies** O **to** O'	Object O' receives forked copies of object O's frames.
P **deletes** O	Frames are lazily garbage collected.
P **mutates** O **to** O'	Equivalent to creation of new object O'.
P **links** O **to** O'	Same as copy.

Table 5.2: The frames calculus. Here, O is an object and P is a principal acting on that object.

any associated frames with the expire operator; they can be reaped once any pending local updates have been reconciled.

The interesting case is object mutation, where there are several options:

1. **Symmetrically attach the antecedent's frames**. This effectively retains the current set of frames unchanged. This option, however, seems particularly unsatisfactory if the object is also shared by others. In such cases, two different objects have the same context and updates to either object's frames reflect in both objects. As these two objects are now different and are used in different circumstances and different coteries, this is awkward.

2. **Fork the antecedent's frames.** The system could instead fork copies of the antecedent's frames, but in many cases, acquired context for the original object will no longer be relevant. As a simple example, down-sampling an image will likely keep most of its context intact, except any resolution tags inserted as context. Requiring applications to provide resolvers is precarious, as they usually cannot discern the semantics of user actions [96].

3. **Attach the antecedent's frames.** This is particularly perilous, as any new context to the original object also applies to its descendant. With the exception of extremely narrow circumstances, there are no general use cases where this option is beneficial.

4. **Create a new frame for the descendant.** The system could create a new frame, completely divorced from the old, in the descendant on object mutations. This mitigates ambiguities present in the previous proposed options and is the solution currently employed by Ligature. Whether this fits well with all its expected use cases is ongoing research.

Finally, a hardlink effectively creates, from the user's perspective, what feels like a new object. This object has a new name and, consequently, a new set of circumstances surrounding it. Similarly, the referent object's frames are forked to the destination object. However,

```
disregard($O$):
  for all $F \in$ catalog($O$) do
    expire($F$)
  end for
```

(a)

```
clean($O$):
  disregard($O$)
  $F$ = new($O$)
```

(b)

```
subset($O, L, O'$):
  hardlink($O, O'$)
  disregard($O'$)
  for all $F \in L$ do
    $F'$ = new($O'$)
    symmetric($F', F$)
  end for
```

(c)

Figure 5.2: *Common frame composites.* (a) *Disregard:* Object O is frame-less and no further context is shared. (b) *Clean:* Remove all frames from object O and start anew. (c) *Subset:* The new object O' only has object O's frames in set L.

hardlinks pose problems in the face of referent mutation. If a link's referent is mutated, there is no clear mapping to the reference; modifying the reference's frames would be confusing to users. On the other hand, initially binding the same set of frames to both referent and reference dispels the notion that names inside a repository are significant to end users. Ligature's approach attempts the least aggrieving outcome. In any event, users seldom use links.

5.3 Sharing Model

The frame operations, coupled with their calculus in the presence of object operations, provide sufficient leverage for Ligature to support the automatic sharing of context across coteries. However, there are still some additional idioms that are needed to express common behaviors.

5.3.1 Common Composites

To prevent an object from containing and propagating context, the disregard composite, as shown in Figure 5.2a, is used. For instance, this could be used to share a personal picture on a user's web page free of context. Similarly, the clean composite in Figure 5.2b clears all frames and starts fresh. This is mostly used to divorce an object from its source. For example, if a user downloads a paper from a conference website, they may wish to disassociate all previous frames and make the object their "own."

By default, sharing an object shares all of its frames. One can expose a subset of an object's frames with the subset composite in Figure 5.2c. Here, hard links ensure cheap replication costs. The ensuant object can be shared and acted upon by the user or application developers; changes reflect between it and its parent object via the symmetric attach.

5.3.2 Revocation

Revoking membership within the coterie is handled by forking the frame and re-sharing that frame with all parties except those individuals being removed. This parallels the architecture's semantics and handles revocation ambiguities, such as if two individuals contemporaneously revoke each other's membership. In this case, there will be two distinct coteries that, if desired, must be reconciled out-of-band.

5.3.3 Log Truncation

Users, especially in a reviewing scenario, may wish to truncate a frame's log such that any intermediate context is completely discarded from view by others. Ligature provides an application that copies a frame's current context—essentially, the tail of its log—into a new frame.

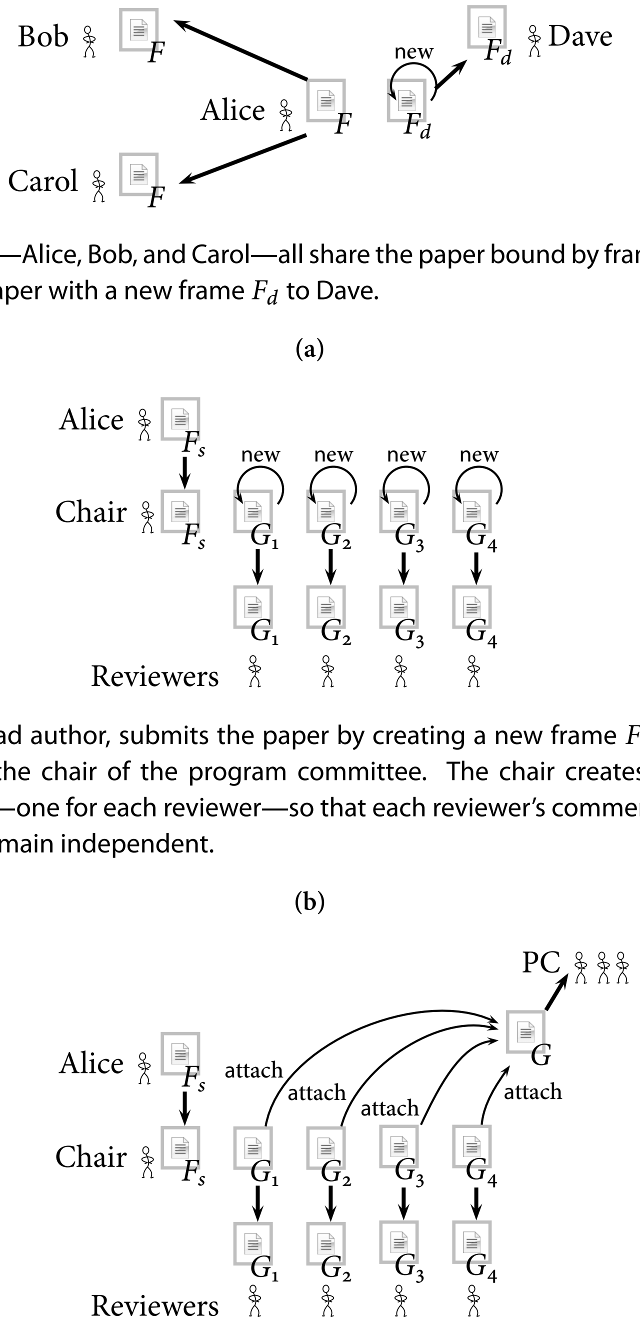
5.3.4 Monikers

Users, as a means of distinguishing frames within an object, may assign a **moniker** for the frame, which acts as another piece of context. In addition, arguments to context operations may be overloaded with this moniker so as to operate on a specific set of frames; the default is to operate on all frames. Within the file manager UI, frames, displayed via their monikers, form an overlay atop the object's own namespace. Users can navigate this "framespace" to observe and manipulate frames with the previously mentioned operations.

5.4 Putting It All Together: A Case Study

To see how these mechanisms might work in concert, consider the familiar scenario of submitting a paper to a conference. Suppose Alice, Bob, and Carol are authors, all sharing the manuscript. With just this simple set of actors, it would be sufficient to allow possession of the paper to imbue full rights on all acquired context.

However, such a simple model quickly breaks down. For example, suppose Alice believes there are potential research opportunities with Dave, and wishes to send him a copy of the paper for his perusal. Dave could include a note expressing his thoughts, but if Alice wishes to discreetly receive this feedback, this must be in a coterie separate from her co-authors Bob and Carol. This problem is exacerbated when the paper is submitted, for the program committee, as well as the reviewers, all hold a copy of the paper but have a separation of concerns.



The authors—Alice, Bob, and Carol—all share the paper bound by frame F. Alice sends the paper with a new frame F_d to Dave.

(a)

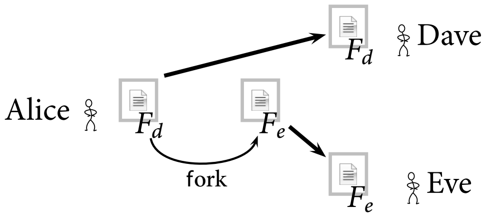
Alice, the lead author, submits the paper by creating a new frame F_s and sharing it with the chair of the program committee. The chair creates four new frames (G_i)—one for each reviewer—so that each reviewer's comments and annotations remain independent.

(b)

The chair attaches each of the reviewer's frames (G_i) into a frame G that is shared with the rest of the PC. Context flows in the direction of the arrows, which means the PC and the chair, but not the reviewers, exclusively see the full set of reviews.

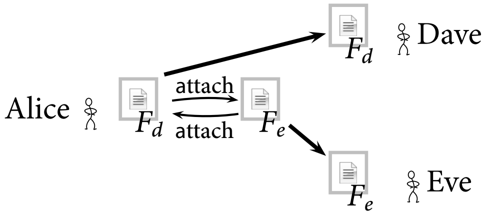
(c)

Figure 5.3: Submission to a conference illustration using contextual frames. The bold arrows indicate a sharing action and the file object is the paper being submitted.



Alice, believing collaboration with Eve may be possible, wishes for her to independently peruse Dave's comments and annotations. She forks frame F_d into frame F_e, which is then shared with Eve.

(d)



Later, Alice decides to bring Eve and Dave together, having received Eve's affirmation regarding future collaboration. She attaches frame F_d and frame F_e as shown; context flows in both directions between the frames.

(e)

Figure 5.3 (continued): Submission to a conference illustration. The bold arrows indicate a sharing action and the file object is the paper being submitted.

Simply using email or a web service is unacceptable, as while it may be adequate for one-time comments and annotations, it is insufficient for other ancillary forms of context, such as tags or provenance information, which would be lost entirely. Such mechanisms also separate context from its object, increasing its managerial burden and, importantly, updates become especially painful.

To begin, the authors all share the paper bound by a common frame, as shown in Figure 5.3a on the preceding page. Frame F's coterie is Alice, Bob, and Carol. To share the paper separately with Dave, a colleague given the paper for early review, Alice creates a new frame that is only disclosed to him. Since the frame is only shared between Alice and Dave, any context is separate and independent from that which is shared with Bob and Carol.

Likewise, when submitting the paper, Alice, the lead author, creates a new frame that is shared with the program chair, as shown in Figure 5.3b. To keep the submission process blind and to maintain independence of reviews, the chair creates a new copy for each reviewer, sharing the paper along with each respective frame. Each reviewer adds their evaluation, and with this setup, each reviewer's context is shared with the chair, but not with each other.

Naturally, the chair would like to collect all reviews and have them automatically shared with the program committee without his involvement. He would attach each reviewer's contexts into a collective whole, as shown in Figure 5.3c. Context flows only in the direction of the arrows to a new frame G, shared with the PC. The asymmetry of this frame attachment ensures that the complete set of reviews is hidden from the reviewers. As context is updated in each source frame, it automatically flows to its destination frame without the chair's involvement. After the committee meeting, the chair includes a note to frame F_s shared with Alice indicating acceptance of the paper, along with the anonymized set of reviews.

Later, suppose Alice, after having read Dave's comments, believes potential wider ranging collaboration is possible, perchance with Eve: Alice could create a new frame, copy Dave's thoughts into it, and relay the frame along with the paper to Alice. In essence, Alice forks the frame she shares with Dave into an independent, duplicate frame, as shown in Figure 5.3d.

After receiving affirmation from Eve about future collaboration, Alice may wish to bring all parties together. She can do this by attaching both frames, as shown in Figure 5.3e. Alice desires an escalation of trust such that Dave and Eve effectively belong to the same coterie; the persistent symmetric flow of context between both frames ensures this.

5.5 More Case Studies

Given this framework, we further envision scenarios like the following with the system.

Scenario #1 (Collaboration/Group control) Elizabeth, upon hearing a talk, believes there may be important avenues of joint work with the speaker, John. She sends the latest technical report of her work encased in a new frame to John in hopes of gauging his thoughts. Context, including notes, document annotations, and links to other work John adds to the report is thus only shared with Elizabeth. John believes there may be potential wider ranging collaboration and sends Elizabeth's report to a colleague, Kimberly, along with a fork of his and Elizabeth's frame. Kimberly receives all of John's context, but is isolated from Elizabeth. Later, having received Kimberly's affirmation regarding future collaboration, John symmetrically attaches all frames together: he, Kimberly, and Elizabeth now all share context.

Scenario #2 (Frames as a disclosure barrier) Sue, a geneticist, stores gene sequences using the standard file format necessary for her tools. Her system automatically determines and stores the provenance [63] of each sequence as context, effacing the need to keep a separate database or a lab notebook. This provenance information follows sequences as they are shared with her research group and colleagues. Also, Sue keeps two frames for each sequence object: a “private” frame, shared only among her research group, and a “public” frame, which is shared with her research group and outside colleagues. Provenance information is stored in this public frame, but experimental results (e.g., locating a matching gene in humans) are stored privately. After publication, Sue can appropriately sanitize results into the public frame.

Scenario #3 (Roles) Linda is a professor consulting for two different companies and must ensure that one company does not have access to proprietary information of the other. That is, a colleague at the first company who is accessing relevant state from her should not be able to obtain any context associated with work done for the second company. For instance, say, a research paper used as documentation at both companies. To support such separation, she uses two separate frames, one per role, for each shared object. Frame selection is made automatic by setting up appropriate association rules (e.g., Linda is VPN’d to the first company, therefore only use that pertinent frame.) The system’s strong enforcement of frame boundaries prevents information leaks.

Scenario #4 (Frames as discretionary access controls) The presence of a frame can signify access to context about the user. Consider a location publishing service. Most users trust their co-workers with their location only during business hours, but not otherwise [13]. The service publishes the user’s location as context to an “at work” frame during business hours, otherwise to an “at home” frame. The user discloses these frames depending on their privacy preferences. For example, the “at work” frame is shared with co-workers, the “at home” frame is shared with friends and family, and both frames are shared with the user’s spouse.

Scenario #5 (Small devices/Ubiquitous computing) A wealth of contextual information can be automatically collected from embedded sensors [97] and stored to frames at the time of their creation or modification. For example, embedded sensors can be used to tag photos with location information [73]. If people are present at the time the photo is taken and they have any device that connotes personal presence, their presence can be recorded as part of the frame. Users would be able to search for their media by where the media was created and who was present. This information would automatically flow as the underlying media is stored on the user’s PC or shared with others.

5.6 Discussion

The notion of genres of disclosure, as illustrated by our examples, is based in the work of Goffman [30], who explored the “fronts” people employ while playing varying social roles.

Our notion of sharing as the means to imbue access seems to capture this concept. For example, consider Alice's escalation, bringing Eve into an existing coterie. One may mistakenly believe this breaks trust, but most interpersonal sharing not only implies read access, but redistribute access [95]. There is an implied social contract between parties who share data and context preventing unauthorized redistribution [98]; the architecture merely provides a useful abstraction to the common operation of branching and merging context.

Fundamentally, people disclose differing versions of personal information to different parties depending on differing, multivariate conditions. Access control lists (ACLs) are not a sufficient solution as they are often cumbersome to employ [98], require a centralized authority, suffer from granularity problems in selecting which particular subset of context is enforced by which particular ACL, and are a weak metaphor for separating and merging of contexts.

Instead, contextual frames form a user-visible concept encapsulating this multivariate nature of personal disclosure. In addition to being consistent with risk models of disclosure [40], contextual frames can be transparent to oblivious users and applications, but support context-aware tasks. To explore the degree to which frames capture this notion, we conducted a pilot user evaluation (c.f. §8.1).

CHAPTER 6

EMAIL SHARING PATTERNS

> *Societies have always been shaped more by the nature of the media by which men communicate than by the content of the communication.*
>
> Marshall McLuhan (1911–1980)

To aid in architectural design decisions, an analysis of how individuals share data is necessary. Unfortunately, studies of sharing behaviors are lacking. There are high-level studies of sharing behavior [95, 98], but they do not provide quantitative data of object exchange or sharing topologies. Email studies [7, 33] concentrate on delivery characteristics (e.g., message sizes and interarrival times), not on message payloads. Similarly, studies of distributed file systems [5, 86], instant messaging [100], and file sharing [36] also do not provide this data.

This dissertation makes use of sharing patterns to aid in design decisions for context sharing and for use in evaluation. Particularly, the most compelling questions are:

- How do individuals share objects? Is there locality in their sharing preferences (i.e., do users usually share objects with those they have with before)?
- What types of objects are shared? Do these types support embedding of auxiliary data? This is eminently useful in a system for tracking the path of sharing.
- What is a typical sharing topology and at what rate are the objects shared? This data is necessary for evaluation.

We study email, as it has become a *de facto* file-transfer mechanism: users often send vacation photos to their friends and families via email; colleagues often make revisions to a document in a collaborative manner using email as a version-control system. The simplicity of email makes it the preferred vehicle for transport. We expect the sharing patterns of email extrapolate to other vehicles of transport and object sharing in general.

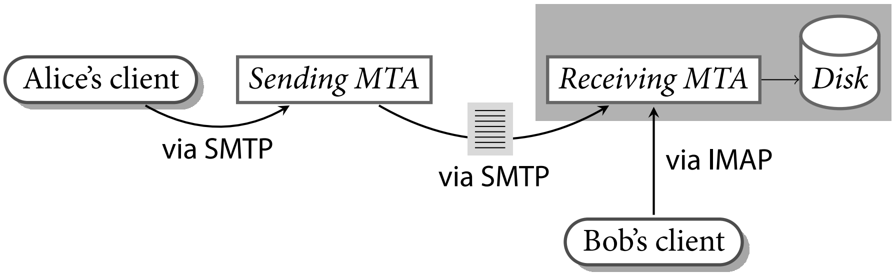


Figure 6.1: Interactions between mail subsystems in a typical mail transaction.

This chapter attempts to answer these raised questions by tracing the University of Michigan EECS department's mail server, examining about 2.85 million messages over 7 months. The focus is on these raised questions; a more complete analysis of this email tracing can be found in Shah and Noble [82].

6.1 Collecting the Data

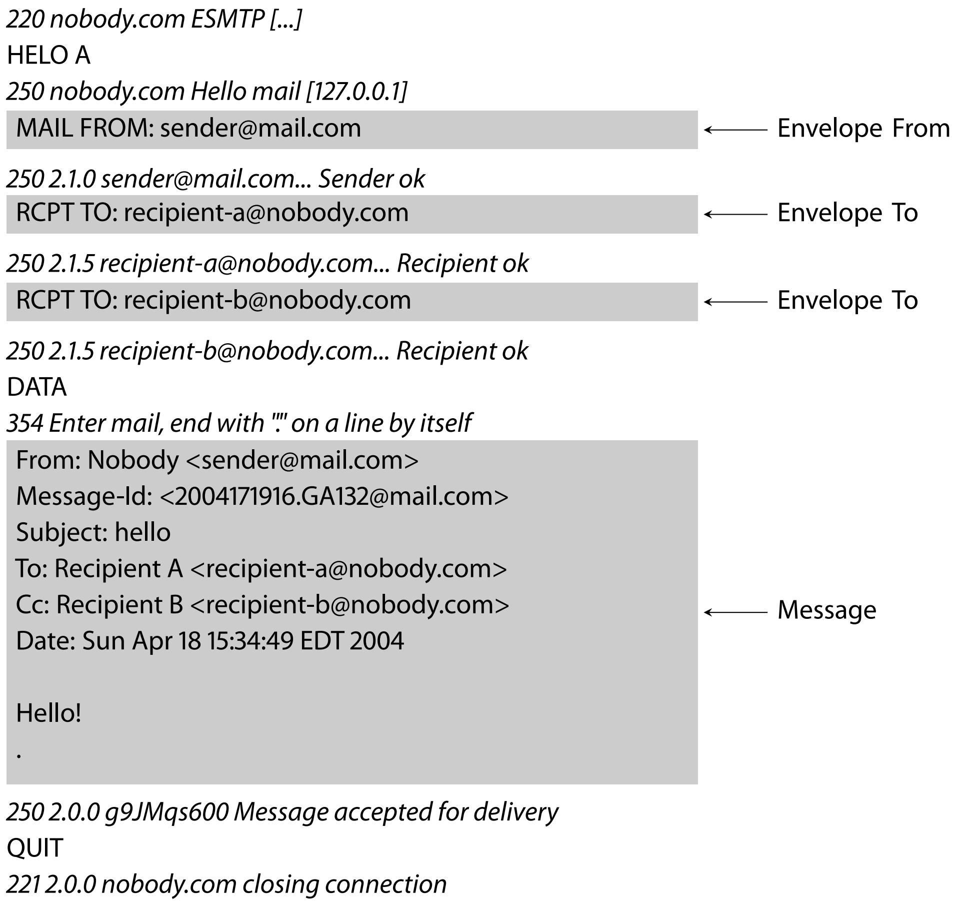
To understand how the data was collected, a cursory knowledge of email protocols is necessary.

6.1.1 Basics of Mail Transfer

Figure 6.1 shows the interactions between mail components in a typical mail transaction when Alice sends a message to Bob. Alice's client contacts its *Message Transfer Agent* (MTA), typically sendmail, which relays her message over the *Simple Mail Transfer Protocol* (SMTP) [76] to the destination MTA, which stores it to disk. Bob's client reads his mailbox via the *Internet Message Access Protocol* (IMAP) [15] and presents the message to Bob.

SMTP is a plain-text, unauthenticated protocol that is used to send mail. A sample SMTP session is shown in Figure 6.2 on the next page. The protocol is rather self-evident. Multiple recipients of a message can be specified as multiple *envelope to* commands or can be separate mail transactions. It is important to note that the message's header is completely unrelated to and unverified against the envelope's contents. In other words, the message header could contain a different sender and different recipients than the actual envelope's contents.

SMTP is closely related with the *Multi-purpose Internet Mail Extensions* (MIME) specification [8]. Virtually all Internet email is transmitted in MIME format. Each MIME message can contain one or several *parts*. These parts could specify alternate versions of the message's textual body (plain text or HTML, for example) or specify the inclusion of attachments. Each part has an associated content type (e.g., "text/plain" for simple text). Further, since SMTP



```
220 nobody.com ESMTP [...]
HELO A
250 nobody.com Hello mail [127.0.0.1]
MAIL FROM: sender@mail.com
250 2.1.0 sender@mail.com... Sender ok
RCPT TO: recipient-a@nobody.com
250 2.1.5 recipient-a@nobody.com... Recipient ok
RCPT TO: recipient-b@nobody.com
250 2.1.5 recipient-b@nobody.com... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
From: Nobody <sender@mail.com>
Message-Id: <2004171916.GA132@mail.com>
Subject: hello
To: Recipient A <recipient-a@nobody.com>
Cc: Recipient B <recipient-b@nobody.com>
Date: Sun Apr 18 15:34:49 EDT 2004

Hello!
.
250 2.0.0 g9JMqs600 Message accepted for delivery
QUIT
221 2.0.0 nobody.com closing connection
```

Figure 6.2: A sample SMTP session. The italics type indicates server responses and the labeled contents denote typical names for parts of the transaction.

is a 7-bit protocol, each MIME part specifies an encoding rule for 8-bit binary content; the most common of these is *base64*.

6.1.2 Mail instrumentation

To instrument mail, we capture messages received on our department's mail server by observing SMTP traffic. At the time, our departmental mail server supported about 300 users, virtually all faculty and graduate students in the department.

We place a dedicated collection machine on the same Ethernet segment and sniff messages, as opposed to directly capturing the messages on the server, so not to increase system load and disturb email delivery. To capture packets and reconstruct mail sessions, a tracer daemon, built on libpcap, captures mail traffic on port 25 (SMTP).

The message is parsed as follows: the SMTP envelope MAIL FROM (also called envelope from) and RCPT TO addresses (envelope recipient) are anonymized to maintain privacy; the message body is MIME-parsed; and the resulting MIME types, as well as the anonymized from and to addresses and a timestamp, are stored in a database. The anonymizing procedure consists of taking the SHA-1 hash [67] of the data.

The tracer uses the envelope from and envelope recipient instead of parsing the message body's from and to headers as these are the most accurate: they also capture blind carbon-copies. Since mail is unauthenticated, we cannot discern senders, but for our purposes we do not care.

Since the departmental mail server allows TLS/SSL connections, the tracer cannot sniff those connections. Secure connections account for 12.9% of all deliveries. Furthermore, many local machines are configured not to use the departmental SMTP server, but rather a locally installed MTA. It is not practical to quantify how many machines do so.

6.2 Data Analysis

The trace data collection ran for more than seven months from 27 November 2002 until 9 July 2003. In that time, the tracer instrumented 2.85 million messages totaling about 50.5 GB.

Furthermore, since messages, particularly attachments, are encoded, for simplicity we use term *size* hereinafter to mean the encoded size: the actual bytes stored on disk and transmitted over the wire. All attachments seen were base64-encoded. As base64 uses a simple translation table, the 8-bit data size is approximately three-fourths of its encoded size.

To provide simple, analytical models for some properties of email, this work uses quasi-Newton search [71], a method for minimizing mean square differences, to find parameters fitting the empirical data. These are approximations of the empirical data, not necessarily governing distributions.

6.2.1 Sharing Locality

Sharing locality is the phenomenon that users share most of their objects with the same group of individuals. Figure 6.3 on the following page shows the rate of new sharers—those individuals a given user has not shared with before—as a function of time. Almost immediately, these individuals represent a modest amount of new data, usually around 20–30% of newly shared objects, and the distribution remains stable. This means that a significant amount of sharing occurs among individuals a user already knows, indicating locality of sharing.

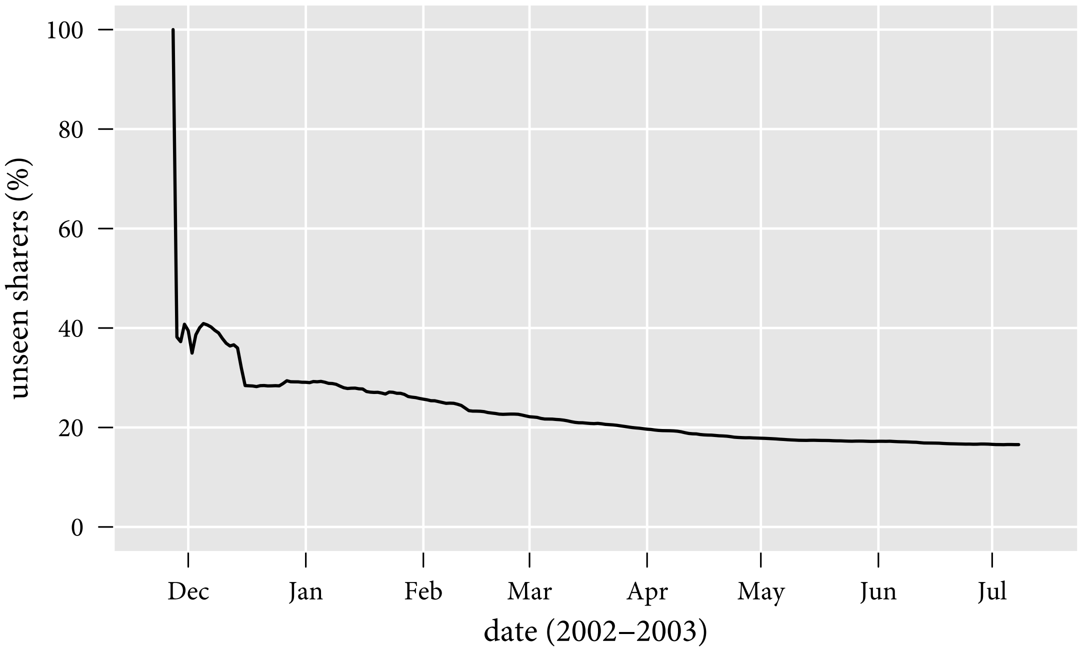


Figure 6.3: Evidence of sharing locality. The plot shows the percentage of new object sharers as a function of time, which indicates that there is sharing locality: most objects shared are to those shared with before.

6.2.2 Content Types

Each message can be composed of several MIME components or parts. Most messages, 77%, consist of only the textual message body, 20% of messages contain only one attachment, 2% contain two attachments, and less than 1% contain three or more attachments. This is unsurprising as we expect most messages to be correspondence.

Figure 6.4 on the next page shows the frequency of content types seen of shared objects. The figure is further delineated by whether these types support fields for embedding data. For readability, classifiable, but obscure file types—those that accounted for less than 0.5% of attachments in terms of frequency—are omitted. Surprisingly, there are a sizable portion of attachments, marked with a ? in the figure, that are unclassifiable by their MIME type or file name. We generally believe these consist of executables exchanged between users and mis-tagged by a user's mailer.

The data indicates that most shared objects, around 70%, support embedding. Since the trace is of an academic computer science department, many text objects shared are program source; we expect real world sharing patterns to include more objects of an embeddable nature.

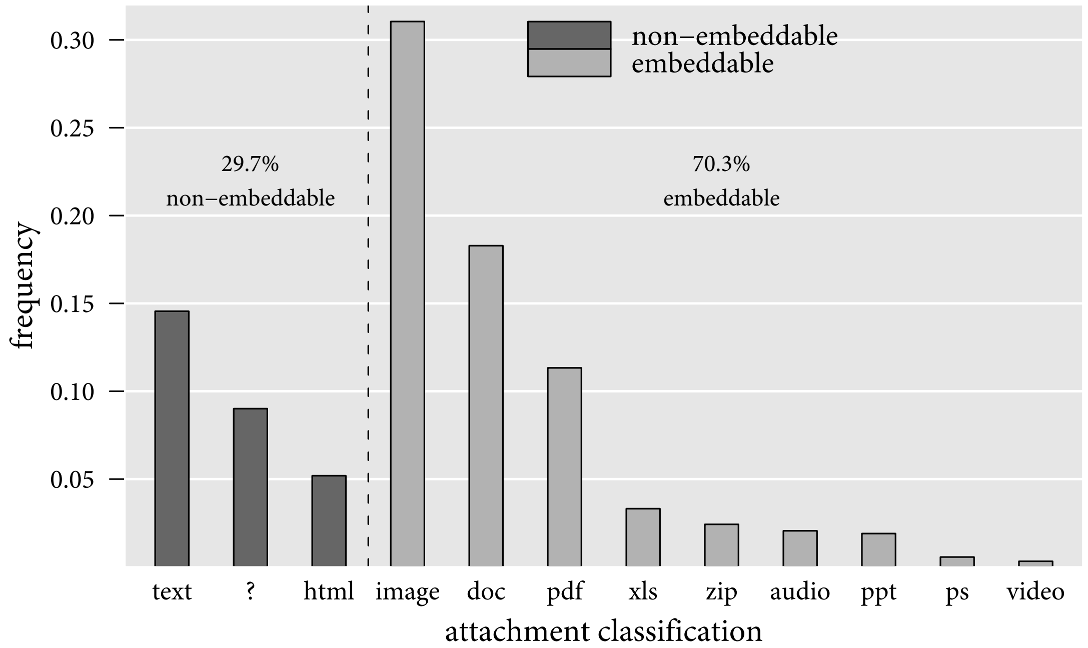


Figure 6.4: Distribution of objects shared delineated by their content type. The figure is further subdivided into whether that type supports fields for embedding data.

6.2.3 Sharing Characteristics

For reproducibility, this section describes the characteristics of the sharing trace.

Recipients Figure 6.5 shows the rank of the number of objects received per user. Notice that most objects are received by a few users, following a Zipf distribution with $\theta = 0.540, b \approx 46.4$ ($r^2 = 0.975$).

Time-Varying Nature Figure 6.6 shows the amount of mail transfer over the tracing period. Since mail traffic was observed at an educational institution, most messages are sent during the weekdays with the weekends experiencing a precipitous drop in traffic (shown with the original curve). The dip in the trace during the months of December 2002 and January 2003 represents the decreased volume of mail processed during the winter holidays.

Figure 6.7 shows a typical day of sharing objects over email in terms of both objects received and bytes received. Object sharing follows diurnal patterns, where most sharing occurs during the workday, especially late in the evening.

Estimating a stationary distribution of object sharing interarrival times requires care. As our dataset exhibits strong non-stationary trends—daily and weekly variations as depicted in Figure 6.6—we need to estimate an interarrival distribution based on a segment in which

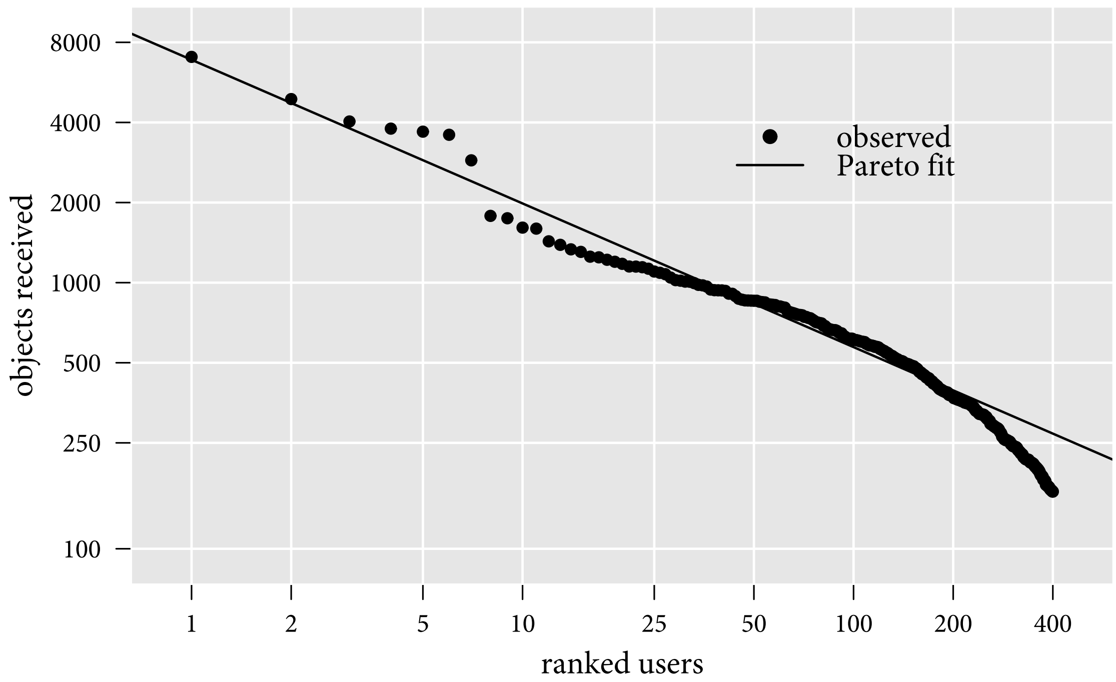


Figure 6.5: The number of objects received by each user.

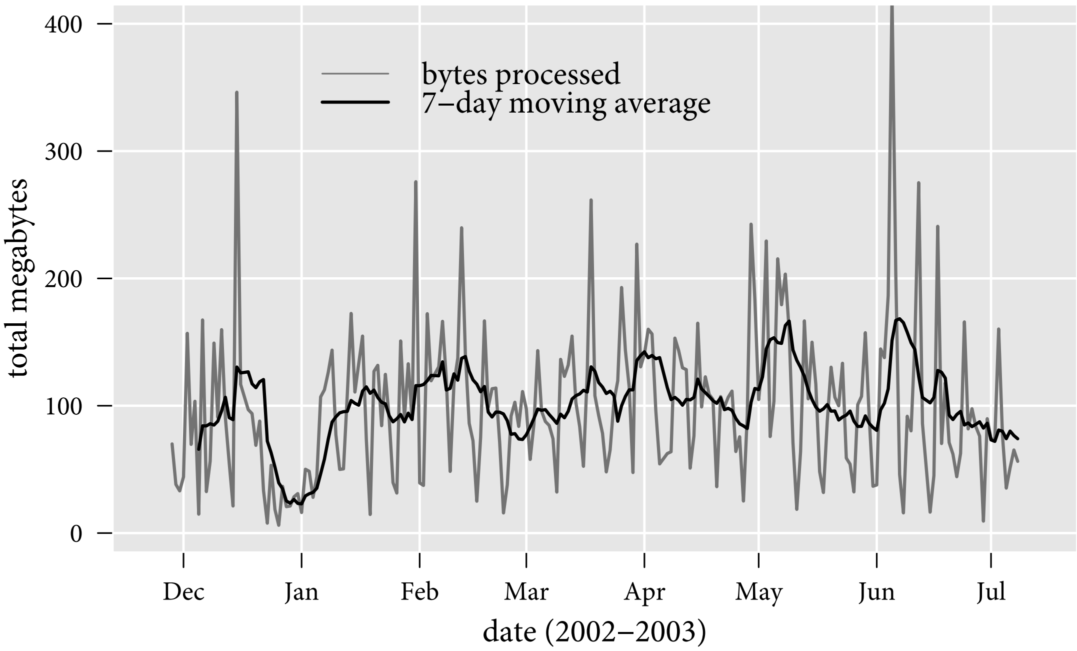


Figure 6.6: The amount of mail received, per day, during the tracing period.

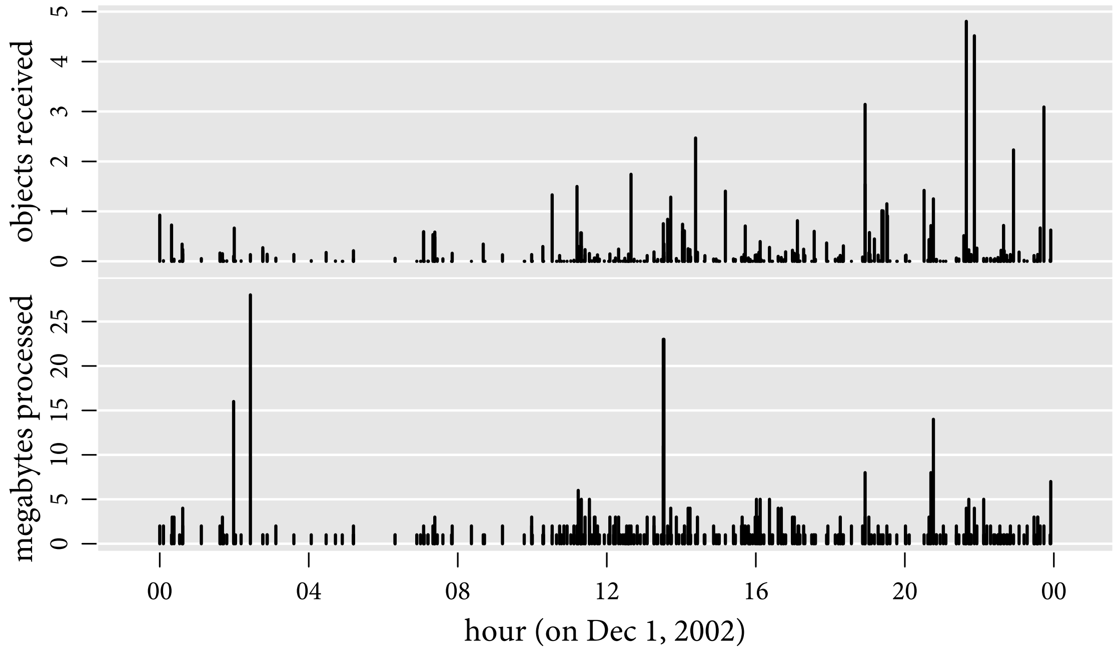


Figure 6.7: Object sharing over a typical day. The top figure shows the number of objects received; the bottom figure shows the number of bytes received.

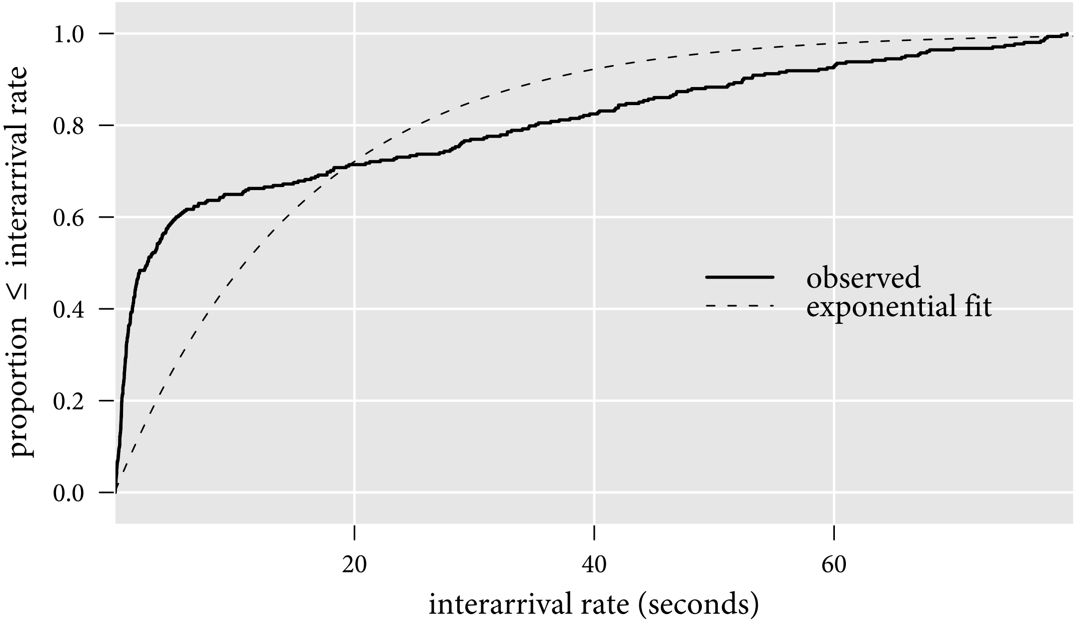


Figure 6.8: Object sharing interarrival times. Due to non-stationarity concerns, we use an 8-hour segment on December 1, 2002, which represents a typical transfer window.

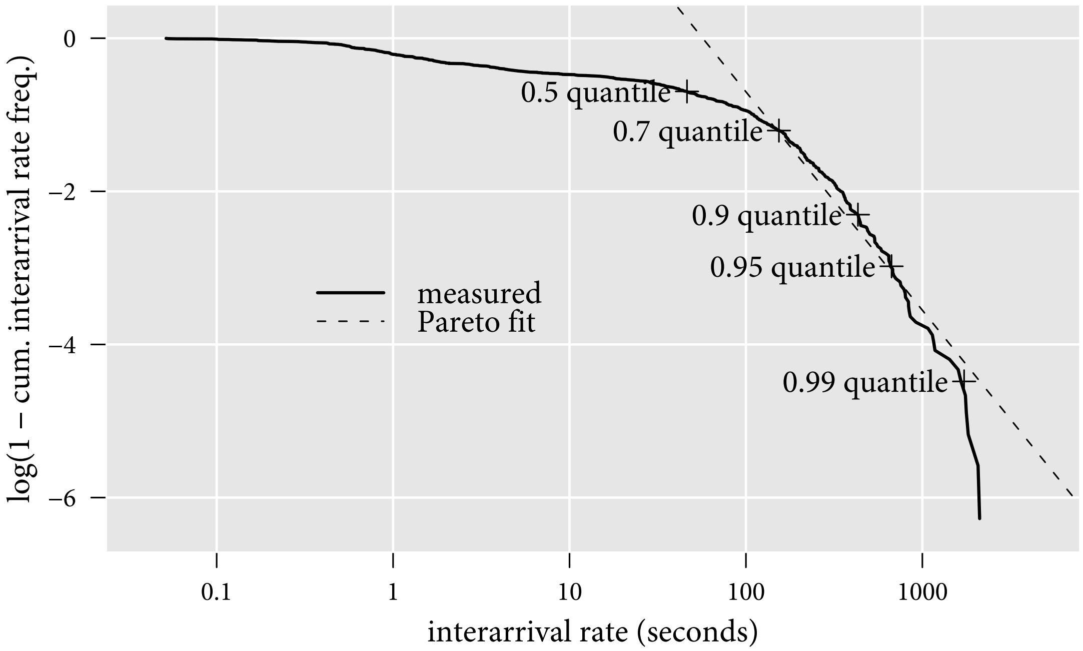


Figure 6.9: Object interarrival time LLCD. Due to non-stationarity concerns, we use an 8-hour segment on December 1, 2002, which represents a typical transfer window.

message arrivals are well-modeled as stationary. We chose an 8-hour segment on December 1, 2002 for this purpose.

Figure 6.8 shows that object sharing interarrival times, in seconds, closely follow an exponential distribution with parameter $\lambda = 0.064$ ($r^2 = 0.910$) in the body. Interarrival times also show evidence of a Pareto power-law tail after 80 seconds (60th quantile) as shown in Figure 6.9, which can can be estimated with $\alpha = 2.24$ ($r^2 = 0.939$). That is, to correctly model object exchange traffic, a piece-wise model with a exponential body ($\lambda = 0.064, r^2 = 0.910$) censored after 80 seconds (60th quantile) with a Pareto tail ($\alpha = 2.24, r^2 = 0.939$) is adequate.

Figure 6.10 shows the number of messages received across time scales of six different orders of magnitude. The abscissa represents time, in seconds, and the ordinate shows the number of messages processed during the time scale. Starting in the lower-right with 1 second, each subsequent plot's time resolution increases by a factor of 8. The most obvious feature of these plots is that they appear bursty at all time scales. Further, there is no characteristic size of a burst: at every timescale there are highly bursty periods separated by smaller bursty periods. This bursty nature manifests itself in transient congestion, requiring care during modeling.

A notion of burstiness is characterized by self-similarity [14, 35, 53]: any section of the data has the same statistical properties as any other with such a time series exhibiting bursts,

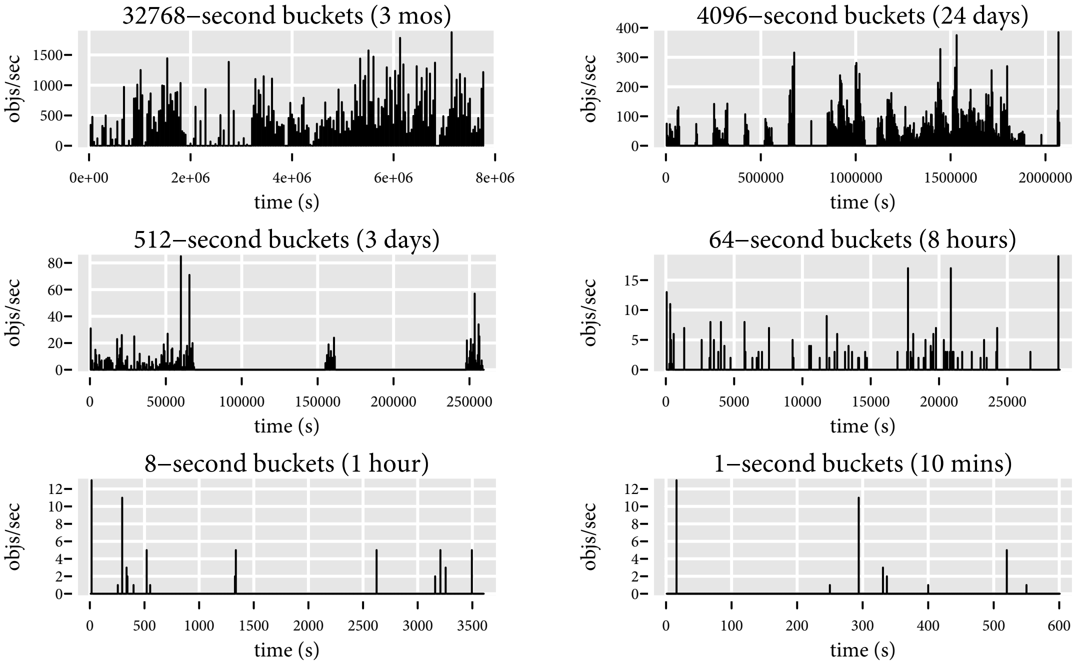


Figure 6.10: Object processing rates shown at different time scales. The abscissa represents represents time, in seconds, and the ordinate shows the number of objects processed during the time scale.

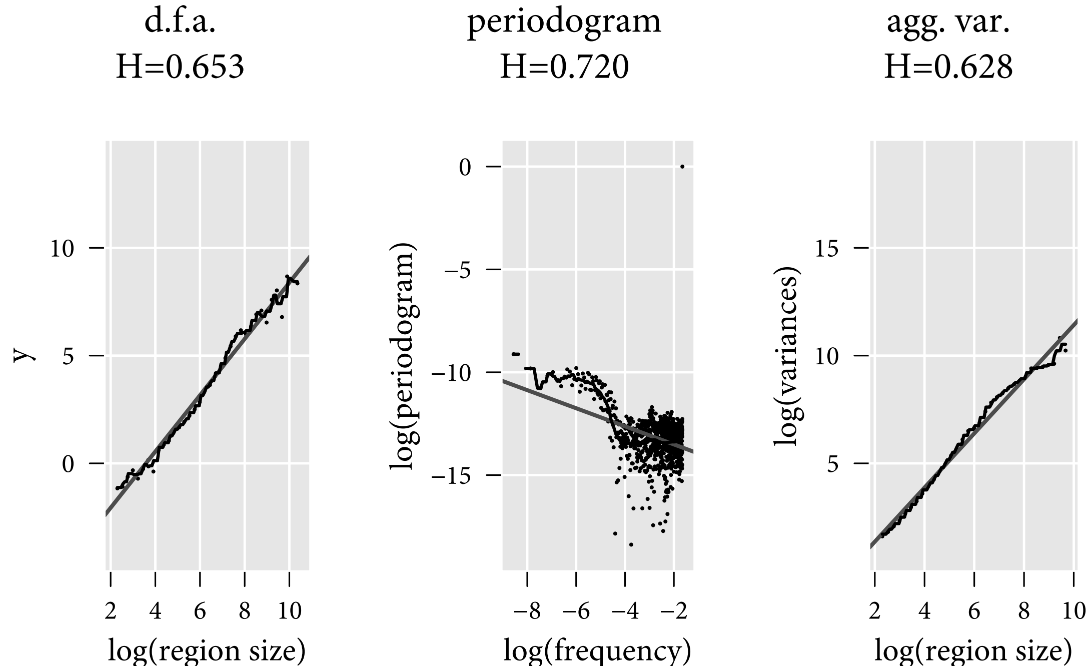


Figure 6.11: Self-similarity analysis of arrival times: detrended fluctuation analysis (d.f.a.), periodogram and variance-time plots of the message processing rate.

extended periods greater than average, at a wide range of timescales. While Figure 6.10 provides pictorial evidence of the self-similar nature of message processing, a robust quantitatively measure is required. The degree of self-similarity is defined via the *Hurst exponent*, H. A data source is self-similar if $0.5 < H \leq 1$, with increasing H indicating a greater extent of burstiness.

Graphical methods of determining the Hurst exponent—detrended fluctuation analysis (d.f.a.) [74], a periodogram [14, 53, 89], and a variance-time plot [14, 35, 53, 89]—are shown in Figure 6.11 for a subset of 8 hours of the trace. These techniques produce estimated Hurst exponents of 0.653, 0.720, and 0.628, respectively. We thus estimate the Hurst exponent of the message processing rate to be $H \approx 0.67$, indicating that email traffic is indeed self-similar.

A strictly exponential arrival time distribution implies Poisson process modeling of object arrivals is adequate. Self-similarity, however, relies crucially on the heavy-tailed property of interarrival times [53]. To correctly model email sharing traffic, one can superimpose heavy-tailed ON/OFF processes, where ON times correspond to the sharing of an object by a client and the OFF times correspond to periods when that client is idle.

6.3 Summary

This chapter presents a large-scale study of email sharing patterns. We collected data over a 7 month period, instrumenting about 2.85 million messages. This work supplies analytical distributions and possible explanations for several sharing parameters of interest: the presence of sharing locality, the content types of data shared, and the characteristics of the sharing trace.

CHAPTER 7

SUPPORTING LIGATURE'S SYSTEM MODEL

Context and memory play powerful roles in all the truly great meals in one's life.

Anthony Bourdain (1956–)

Ligature must:

1. Embrace users from different administrative domains;
2. Be scalable and support user mobility;
3. Be agnostic to the underlying data store and method of object transport (i.e., objects may be stored in heterogeneous data repositories scattered across different disks, file systems, and machines);
4. Be lightweight enough to be usable on mobile devices, such as cell phones and PDAs, which have modest levels of computing power, storage, and network connectivity, as well in ubiquitous computing scenarios [97], where a multitude of small devices share context on their surroundings.

This chapter outlines Ligature's architecture and describes a prototype system satisfying these precepts.

7.1 Architecture

This section outlines Ligature's architecture, which consists of the different components shown in Figure 7.1. The store component holds the context knowledge base and provides query capabilities across it. This knowledge base is synchronized to peers within the coterie through the conch module. Each host eventually receives context from all its peers; the architecture stitches this context together for applications and services to query. The frames

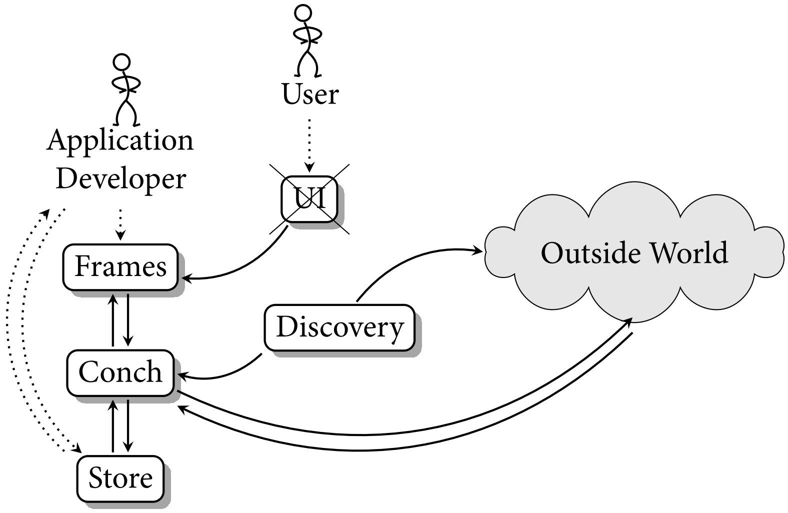


Figure 7.1: Ligature's components. Lines represent communication flows between various components. This work is not concerned with the user interface (UI) implementation.

module implements the primitives for manipulating contextual frames and the discovery module ascertains a frame's coterie. Each module is discussed in turn.

7.1.1 Contextual Store

The store is represented as RDF [59], the *lingua franca* of context, where semantic markup is stored as triples. An SQL-like language, called SPARQL [78], provides query capabilities across this RDF data. Ligature augments the RDF triple into a quintuple with two new pieces of data: the source, who is the user attesting the tuple, and the frame the tuple belongs to. If needed, applications may query this additional state with special SPARQL predicates (e.g., to build contextual services where the tuple's author is necessary). Each frame consists of a *set* of these tuples, not a collection, as tuples are never duplicated.

7.1.2 Contextual Exchange

Given a frame and its coterie, as received from the discovery module, the conch module exchanges tuples with each peer for that frame through periodic anti-entropy exchanges [32].

Each node has an **identity**, which is the hash of the public key of a generated public/private key pair. A standard challenge-response protocol ensures nodes are communicating with same peer in the future. In this scheme, identities are weak, but nodes may publish pertinent user-specified contact information (e.g., name, email address) as context.

Each frame is effectively a log ordered by version-id, which acts as a Lamport clock [50]. Operations are idempotent due to the set nature of frames. Ligature resolves write-write conflicts by assuming a well-known total ordering scheme. These conflicts are rare as most applications only update tuples where the local user is the tuple's publisher. For efficient querying, the system stores a rolling checkpoint of the log's tail.

Frame initialization, where initial contextual state is transferred upon reception of a new frame, involves transferring the frame's log to a new peer. Each peer is at least responsible for sending its own events, but high-powered altruistic nodes may transfer the entire log for faster frame reception.

It is important to note the eventual consistency semantics of the architecture mean contextual exchange could occur using idle network bandwidth [94].

Further, there may be objects a user wishes to keep private, irrespective of whether they are shared with friends. Ligature allows a user to specify a set of directories always to be ignored.

7.1.3 Contextual Frames

A frame consists of a **frame-id**, its frame identifier; its **object-id**, a pointer to the object it is describing; and its context. The frame-id is a globally-unique identifier (GUID), of which there are several well-established schemes to ensure uniqueness [52]. Possessing the frame-id is the credential to inspect and mutate a frame's context. The secure hash [67] of the frame-id is the credential limiting access to inspection only and is known as the **immutable frame-id**. The object-id is a cryptographically secure hash of the object's contents.

The new, fork, catalog and expire operations are non-distributed operations. The creation of a new frame involves the frame receiving a fresh GUID and no context. The fork operation is a composite of existing primitives: the creation of a new frame and the copying of all the source frame's context to it. The catalog primitive returns the list of frame-ids bound to an object. For the expire primitive, the system marks the frame as deleted, positioning it for garbage collection. Since the system cannot know the extent with which a frame is shared (especially in the case of transfer by removable media), this context may need to be reconciled later. Deleted frames only participate in outgoing contextual exchange and are reclaimed when storage space demands it. Reclamation biases toward those frames infrequently accessed and those with no attached children.

An attach operation combines one context set into another. It does this by inserting a special piece of system-level context inside the destination frame indicating the source's immutable frame-id—necessary to prevent write access to the source frame—which is propagated to members of the destination frame's coterie. Peers may need to fetch context for this source frame if it is not currently held. The source frame is merged through timestamps vis-á-vis version-ids, recursing on any of the source frames' incoming attaches. In the end, the context of the destination frame is the union of itself and the transitive closure of its source frames.

In their queries, application developers may need to specifically treat write-write ambiguities in attached frames. For example, in a service where users can rate their objects, an application developer may erroneously assume a single rating tuple per user. However, the linking of two frames with different ratings may result in two distinct rating tuples by the same user and a consequently ambiguous rating in the attached frame. In such a case, the application developer would need to structure their query not assuming a single rating tuple; for instance, by calculating the mean rating. These ambiguities can also often be solved if applications deem their context immutable. For example, if a user forks a frame with a note, modifies the note in the descendant frame, then attaches the descendant back with its parent, an ambiguity will arise: there will be two different notes, both with the same note-id. The notes service could use application-defined resolvers to, say, display the note with the latest modification time, but a general solution is to force note immutability such that notes form an append-only log.

A frame's moniker is context within the frame itself under a special namespace; the architecture's normal context functions are used to manipulate it. The default resolution order is to search locally-assigned monikers first, followed by global ones.

7.1.4 Discovering Peers

The discovery module builds a frame's coterie: it determines with whom to share context for the frame and how to find that peer. This information is given to conch, which handles context propagation.

There are several requirements we necessitate. First, the system should not interfere in the way users share objects. As a corollary, Ligature should be agnostic to the underlying storage substrate or vehicle of transport. Second, users are mobile and may often be disconnected; they must be able to work autonomously.

The frame-id is the credential to access the frame's context. Given the frame-ids encompassing an object's frames, the system needs to locate any peers also holding those same frames.

A centralized directory service could exist, arranging peer and object discovery, but the commercial viability of such a centralized authority is uncertain. This directory service could be decentralized over several administrative domains as part of an organization's normal computing infrastructure, but it is unclear any would be inclined to do so. Instead, we focus on a fully decentralized or peer-to-peer solution, as it is the most interesting and most applicable to the problem domain.

This leads to several preconditions. We do not assume the existence of a PKI infrastructure. Also, to ensure fairness, any resources contributed to the system by a user should be proportional to the number of objects shared. Structured P2P networks (i.e., DHTs [88]) are unsuitable for mapping frame-ids to coterie memberships. Besides allocation fairness, there is no control over where data resides, meaning low resource devices may end up servicing a large coterie. Further, the number of objects is only increasing [62], entailing commensurately increasing DHT repair costs on joins and departures. At the other end of the spectrum, unstructured overlays are also problematic, as flooding the entire network looking for peers holding the same frame-id will not scale.

Instead, recipients of the object contact a coordinator, presenting the frame-id. The coordinator publishes to the recipient any additional peers holding the object and peers thereafter gossip [19] membership changes. The coordinator is defined to be the node who created (or forked) the object's frame. Small devices may optionally delegate a more powerful machine (e.g., the user's desktop computer) to be the coordinator. The identity of the coordinator is passed along with the object's frame-id.

In the event the coordinator is unavailable, the host floods previously shared-with peers searching for members of the frame's coterie. Analysis of email sharing patterns (c.f. §6.2.1) indicates there is *sharing locality*: users generally have a small clique of individuals with whom they share the majority of their objects with. As peers become available through this flooding, they gossip membership changes as before. The flooding mechanism uses a challenge-response protocol with secure hashesto locate peers without leaking credentials or enabling reflection attacks. In Ligature, hosts perform this protocol first with the immutable frame-id and those hosts with read-only access are then queried for full credentials. It may be feasible, in terms of bandwidth and successful peer location, to flood further than directly accessible neighbors, but our data on sharing locality is limited in scope. With this scheme, the coordinator's purpose is to merely perform introductions. We evaluate the feasibility of this approach in our evaluation (§8.2).

Firewalls, network address translators (NATs), and dynamic IP assignment complicate the ability for a pair of nodes in the world to communicate. For this reason, Ligature builds an overlay such that each node has a static nodeid, of which there are several proposals on how to do so [26]. This nodeid is simply the host's identity.

A failed or unwilling coordinator not performing introductions could, at worst, prevent others from sharing context on frames they themselves have created. Though, if prior sharing relationships exist, these peers will eventually find themselves.

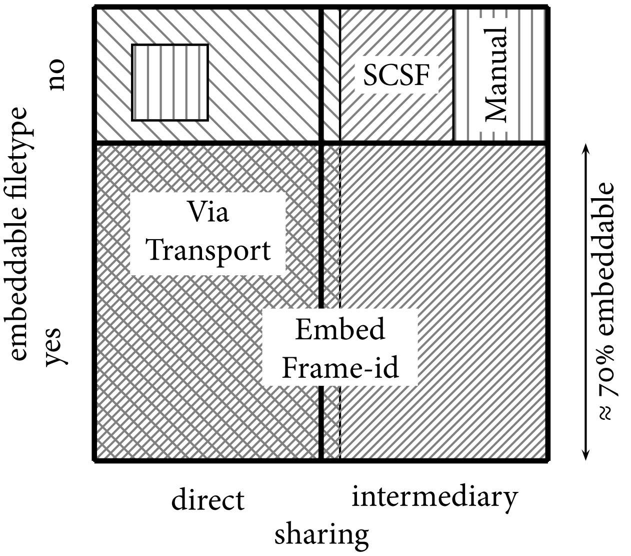


Figure 7.2: Methods by which Ligature exchanges frame-ids.

Surreptitiously passing frame-ids and coordinator node-ids during sharing is problematic, as any mechanism must support the diverse vehicles of transport, including transfer via intermediaries (e.g., websites), and be backward compatible with users not using Ligature. The system cannot accept frame leaks, so any kind of frame-id inference procedure is not applicable. To satisfy these constraints, Ligature uses a multitude of methods, which are shown pictorially in Figure 7.2 and are as follows.

Ligature compacts the set of coordinator node-id and frame-id pairs belonging to an object into a textual string applications view as opaque. As objects are shared, these pairs are shared along with the object by way of the transport mechanism. For example, email and web protocols use header extensions; the file system uses resource forks. To support transfer via intermediaries as well as sharing protocols that may not support these header extensions, Ligature also embeds these pairs inside objects supporting such comment fields. Analysis of email sharing indicates 70.3% of attachments are embeddable file types (c.f. §6.2.2). Unclaimed objects—those that are non-embeddable objects and shared via nonsupporting transport protocols—are handled through object registration, a one-time operation that maps an object to its coterie, and manual reconciliation, an out-of-band operation sharing the object-id with potential peers.

Object registration with a DHT involves mapping the secure hash of the object to a list of frame coordinators and the number of frames created by that coordinator for that object. Objects with a single coordinator and containing a single frame (SCSF) have only one possible coterie for that object hash and represent a common case where a user themselves

Technique	Works for all file types	Compatible with existing apps	No changes to sharing apps	Compatible w/ users not running Ligature	Supports sharing via intermediaries	Aesthetically pleasing to user	No filesystem ambiguities	Lightweight
Filename Use part of the filename for the frame-ids.	✓	✗[i]	✓	✓	✓[ii]	✗[iii]	✓	✓
Envelope Wrap the object in an envelope that includes its frame-ids.	✓	✓	✗	✗[iv]	✗	✓	✓	✓
Protocol Interposition Capture frame-ids by interposing on sharing traffic (i.e., by parsing protocol traffic).	✓	✓	✗[v]	✓	✗	✓	✓	✗
Embed Embed the frame-ids inside comment fields/as watermarks within the file.	✗[vi]	✓[vii]	✓	✓	✓	✓[viii]	✗[ix]	✓
Via Transport Use extensions to sharing protocols to send the frame-ids.	✓	✓	✗	✓[x]	✗	✓	✓	✓

✓ is considered good

[i] Certain applications are dependent on the exact naming of files (e.g., the make utility with Makefiles).

[ii] Must preserve the frame-id portion of the filename, a potentially long string.

[iii] Although this could be masked by the OS to some extent (for those running Ligature anyway), at some point the frame-id would be visible to end-user applications.

[iv] Everyone must be running Ligature otherwise their system will balk at the envelope.

[v] Incompatible with encrypted transports (e.g., TLS/SSL), which is in wide-spread use in SMTP (electronic mail) and HTTP (web browsing).

[vi] Though, the class of filetypes that do not support embedded comment fields is small, around 30% of all shared objects (c.f. §6.2.2).

[vii] The comment field is sometimes used by applications (e.g., for photo descriptions), though it is likely they can co-exist.

[viii] Some users could be dismayed that parts of their files are storing "strange" data (e.g., comment field of MP3s), although this is a marginal concern.

[ix] Changes the file itself, which could lead to strange effects with certain applications (e.g., rsync). For example, syncing a file changes the destinations contents (the copy causes a fork of the source file's frames and new frame-ids are stored).

[x] Sharing protocols have header extensions that by default are automatically ignored by applications that do not understand them.

Table 7.1: Matrix of strengths and weaknesses of various frame-id exchange approaches. Ligature uses a combination of transport exchange and object embedding, along with techniques for handling any remaining unclaimed objects.

creates an object they share. Upon reception of an object without a frame-id, the host checks whether it satisfies this SCSF property by querying the DHT, joining its coterie if it does.

Finally, with manual reconciliation, the user selects with whom to fetch frame-ids for an object based on previous sharing history. Her system interposes on that peer, inquiring which frames, if any, should be exchanged. As a last recourse, users may exchange identifiers manually through out-of-band channels.

Ligature attempts the most thorough solution to the frame-id exchange problem without resorting to manual exchange. A multitude of other options were considered, and the most worthy, delineated along with their pitfalls, are shown in Table 7.1 on the preceding page. Overloading the filename with frame-id and coordinator pairs poses several intractable problems, such as incompatibilities with existing applications that require exact filenames and requiring the frame-id portion of the filename to remain intact. Importantly, it is aesthetically unpleasing for users, as these pairs, even if masked by the OS to some extent, will eventually be visible to end-user applications—and always be visible to users not running Ligature. This option is, at best, a hack, and not general purpose. Wrapping the object in an envelope that contains these pairs as a header is also not a viable solution as it is incompatible with existing end-user applications and those systems not running Ligature. Interposing on protocols to send frame-id pairs is a heavyweight solution and is ill-sorted to sharing over encrypted transports. Ligature's solution of extending transport mechanisms where applicable and simultaneous object embedding capture most forms of direct and indirect sharing while being lightweight and compatible with existing end-user applications, sharing transports, and systems not running Ligature.

7.2 Prototype

We have built a prototype, feature-complete Ligature implementation running on Windows. This prototype substantiates our design principles and is used in our evaluation. The prototype runs as a Windows Service, which is a user-space daemon. Applications link to a client library that communicates with the Ligature service over named pipes. Currently, a client library exists for Python and C++, but practically any language could be supported.

A frame's context and its frame identifier with its corresponding path in the filesystem is stored in a database. An alternative is to store a frame's context using resource forks in the file system, but this information needs to be always available for contextual exchange and the corresponding objects may be stored on removable media or a network drive. Further, a centralized checkpoint greatly improves query speed by minimizing disk seeks. The prototype monitors file system events via the ReadDirectoryChanges API to adhere to the frames

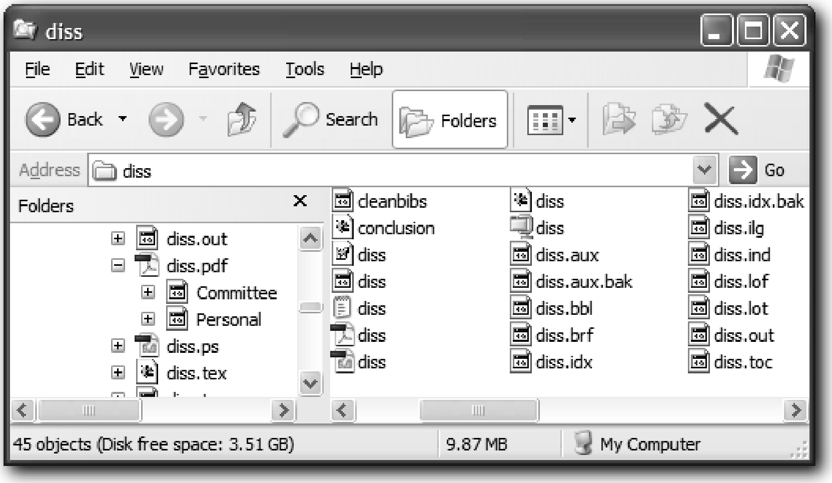


Figure 7.3: A screen shot of Ligature's "framespace." Here, the dissertation document has two distinct frames with moniker's "committee" and "personal."

calculus (§5.2.2). The framespace (§5.3.4), particularly the frame view, is implemented as a namespace extension to Windows Explorer and a sample screenshot is shown in Figure 7.3.

The Ligature prototype uses a modified version of the RDFLib library[1] for its RDF store and SPARQL query processor implementations. BerkeleyDB acts as the backend store for all of the system's data structures. The current prototype directly connects to peers over SSL, but could support routing messages over an overlay in the future.

Also, as a proof of concept, we have modified the Mozilla Thunderbird[2] email client to support exchanging identifiers. When sending email, the client uses Ligature's getid function to retrieve the object's identifier and inserts a special MIME extension header [8] as part of each attachment's envelope. When the receiving user saves an attachment, the client parses this header and stores the object's identifier with the system's setid function.

7.3 Applications

Several applications, as outlined, have been constructed using this architecture. The line count of the context specification and query portion for each service is shown in parenthesis.

Tags Users may assign free-form labels to objects and accrue up to one vote for any tag assigned by another user. These histograms of tags are useful in search and to aid in finding semantically-related items. (≈ 20 lines)

[1] http://www.rdflib.net [2] http://www.mozilla.org

Notes Users may attach a note to an object, creating a conversation log. Notes may be public or private. (≈ 25 lines)

Context-enhanced search The synergistic version of Weft exchanges and updates the relation graph (§2.1.4) across an object's coterie, which may improve search results as the model learns from many different users. (≈ 50 lines)

Mark as deprecated A user is able to mark a document as deprecated; any other user opening that document is presented with a deprecation warning. (≈ 15 lines)

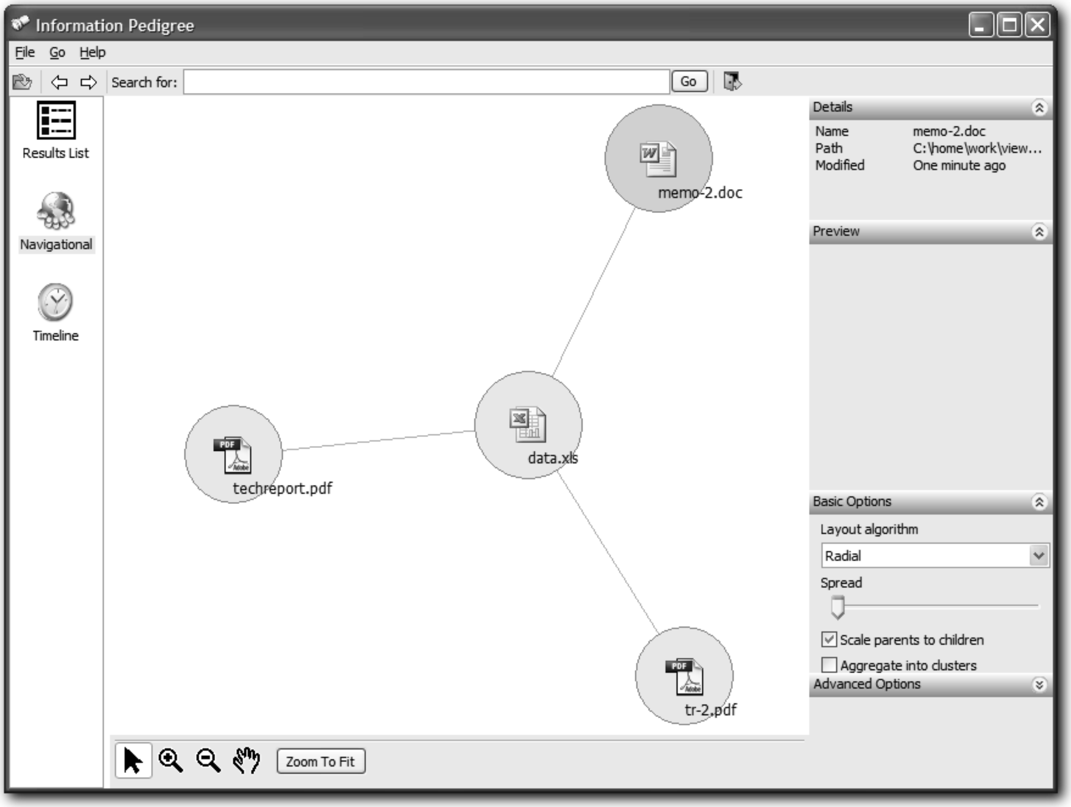
To-dos To dos for tasks can be assigned to the objects they belong to. (≈ 20 lines)

Ratings Users are able to rate objects on a scale of 1 to 5 with the mean score across all peers shown for that object. This is particularly useful for media objects such as personal images and videos. (≈ 15 lines)

Provenance Maps This tool allows the user to explore a map of the provenance of their data [63], as shown in Figure 7.4 on the following page. The tool permits navigation of a user's repository by relationship: the user can see the source data for a report (e.g., in Figure 7.4, the data.xls spreadsheet for the technical report) or the descendant objects for any source piece of data. (≈ 50 lines)

Internally, this pedigree information is the relation graph (§2.1.4) gleaned via the causality algorithm (§2.1.2) by a background task. This provenance is stored as object-id references and as objects are shared, this context is also shared: a user possessing both source and sink of a document can see the relationships between them. The provenance map can be shown in a basic radial layout with fisheye distortion [27] or as a flattened hierarchical layout [28], but more sophisticated layout algorithms, scaling to thousands and even tens of thousands of nodes [64], could also be plugged in. The user may also view the provenance trail as a timeline, similar to the Lifestreams system [25].

Many scientific and business communities employ systems that manage workflows [18, 56], which are graphs representing discrete computational components with edges representing paths along which data and results can flow between these components. Besides results and raw input data, these workflows contain information about provenance, data quality, attribution, audit trails, and other curated data. These workflow systems could use Ligature in a similar vain to this pedigree tool to automatically exchange necessary workflows as a byproduct of sharing.



Name memo-2.doc
Path C:\home\work\view...
Modified One minute ago

Layout algorithm: Radial
Spread
☑ Scale parents to children
☐ Aggregate into clusters

Figure 7.4: A screen shot of the provenance tool. Here, the spreadsheet (data.xls) is the source for two technical reports and a memo.

7.4 Barriers to Adoption

An important pragmatic consideration for any new architecture is what barriers it is likely to encounter in any attempt at large-scale deployment.

7.4.1 Technical Challenges

There are some technical challenges. The biggest is to migrate existing sharing applications to exchange frame-ids, which has already been discussed (c.f. §7.1.4). Encouraging signs are that to support frame-id exchange via protocol extensions, there are only a few vehicles of transport (i.e., email, instant messaging, and the world wide web) and a lack of software diversity in each of these areas. Most of these sharing applications support easy extension via plug-ins, which could be shipped with Ligature. In addition, most shared objects support embedding (c.f. §6.2.2), and there is also a lack of file type diversity [20]; frame-id embedding must only support a few file types to capture most shared objects. These reasons allow incremental deployment one host at a time without breaking interoperability with existing, legacy systems.

Another important technical challenge is the user interface (UI). A UI for presenting and manipulating frames as well as for applications to hook into for displaying their context is necessary. There is a homogeneity of operating systems and file manager interfaces, lessing this engineering burden. Small mobile devices, such as smart phones and Internet tablets, will be an engineering challenge, as they all lack common interfaces.

7.4.2 Economic Challenges

There are some economic concerns with wide-scale deployment of Ligature. On the positive side, Ligature does not require any additional infrastructure; it is therefore unlikely to be thwarted by budgetary concerns of individuals and organizations. In addition, if Ligature is widely deployed, market forces will tend to favor applications sharing their data as context and sharing transports supporting frame-id exchange. There are also incentives for major players to include Ligature as part of their commodity operating systems—to aid collaboration among a user's friends and family, a major focus of new software—which will ensure wide-deployment.

On the negative side, Ligature relies on network efforts: users only gain utility from the system if they share data with others also running Ligature, posing a barrier to entry. Ligature's fully decentralized architecture also poses threats to adoption, as any vendor would undoubtedly like to keep control—or at least retain access—over a user's context, which is often leveraged for targeted advertising and other purposes. This latter point may create secrecy and privacy issues from the users' perspective, as they may not trust vendors providing contextual services.

7.5 Summary

This chapter presented a detailed description of Ligature's architecture, including how context is stored and exchanged, frames are supported, and peers are discovered. The architecture affirms the goals presented at the beginning of the chapter and of this dissertation.

As well, this chapter described the implementation of Ligature, along with several useful sample applications constructed atop this architecture. These applications validate the utility in exchanging context and showcase several functional tools for the organization and management of a user's repository.

Lastly, this chapter outlined deployment issues Ligature may face.

CHAPTER 8

EVALUATION OF LIGATURE

To know an object is to lead it through a context which the world provides.

William James (1842–1910)

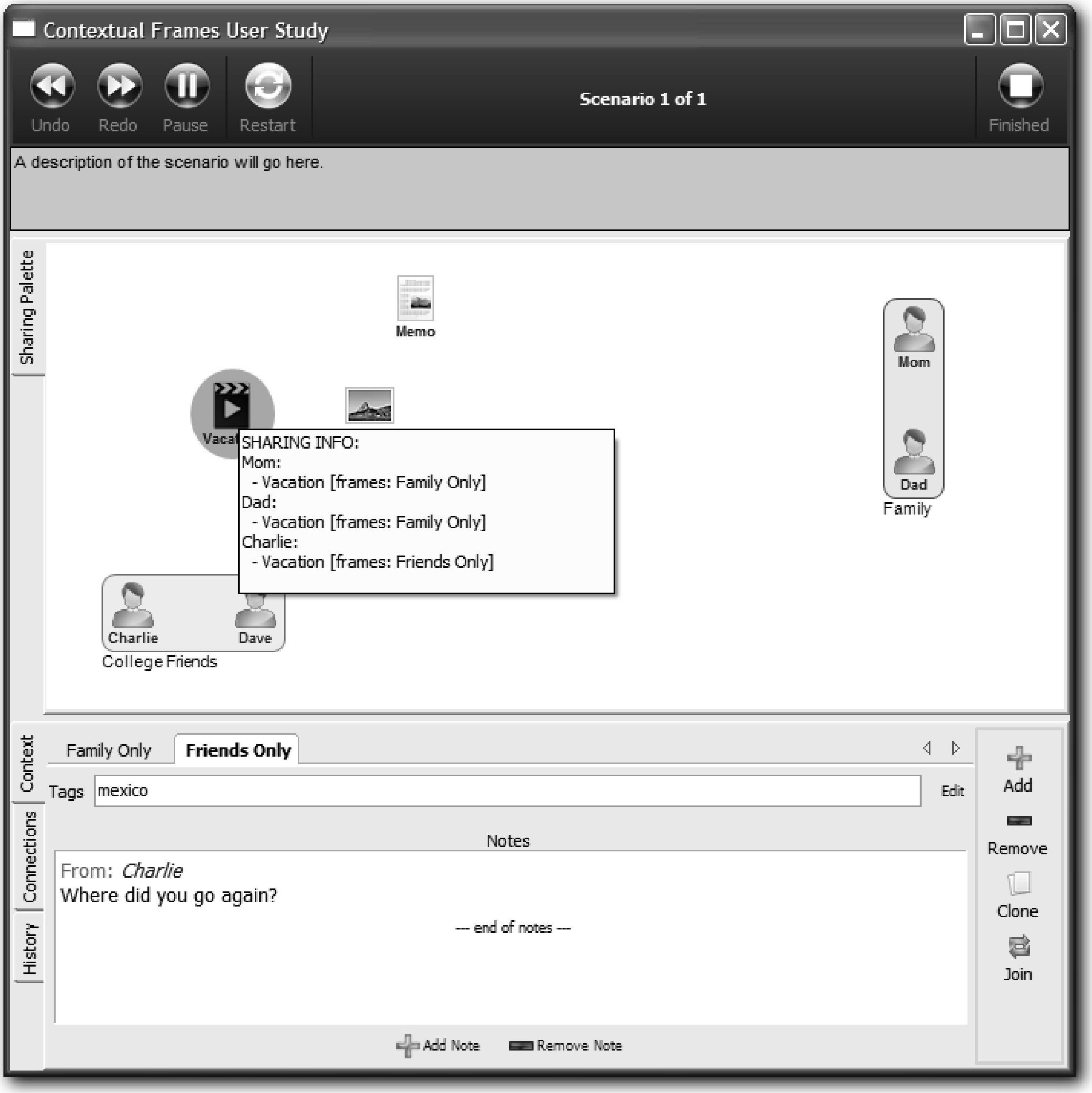
The salient goal of Ligature is to facilitate convenient and easy contextual exchange with minimal system overhead while respecting boundaries on disclosure. Thus, our evaluation answers the following questions:

1. Can users compose appropriate frame relationships as needed?
2. Is Ligature's peer discovery mechanism (c.f. §7.1.4) resilient to host unavailability?
3. Does Ligature inhibit foreground user work?
4. What is the application performance penalty of using Ligature as a backend storage mechanism?

8.1 User Evaluation

A longitudinal study, where users are repeatedly observed using Ligature over a long period of time, is ideal, but burdensome: it requires a multi-year effort, and more onerously, a participant population that shares objects amongst themselves. Rather, we conduct an in-lab, scenario-based evaluation to determine if users can correctly understand and manipulate frames.

The study presents certain scenarios to the user requiring the arrangement of frames and their corresponding relationships that they must then solve. To produce generalizable results, some care is required with these scenarios, as rarely is preventing unintended disclosure a user's primary goal [99]. Instead, the scenarios are hidden under the guise of acting as



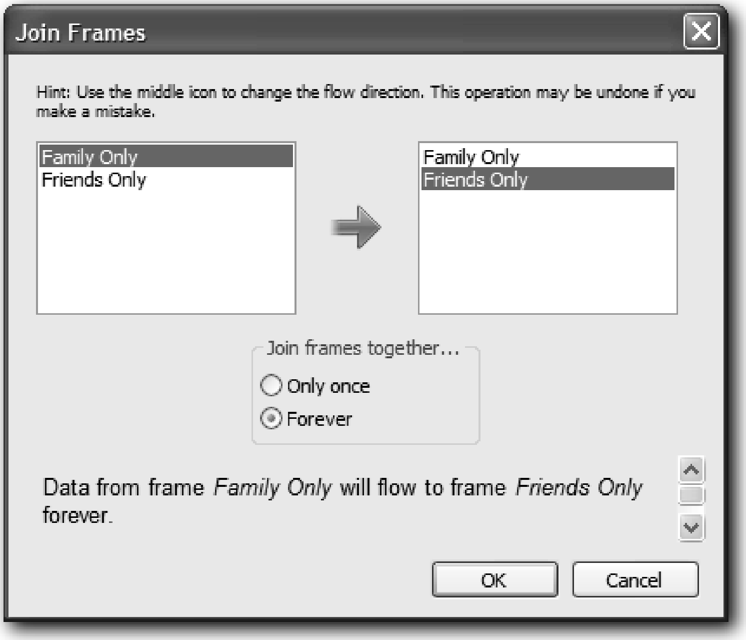
A description of the scenario will go here.

SHARING INFO:
Mom:
- Vacation [frames: Family Only]
Dad:
- Vacation [frames: Family Only]
Charlie:
- Vacation [frames: Friends Only]

Tags: mexico

From: *Charlie*
Where did you go again?
--- end of notes ---

Figure 8.1: The sharing palette interface. The uppermost frame lists the scenario's description, the middle pane permits the user to share objects and frames (and keeps a visually-persistent record of sharing), while the bottom frame shows the object's frames and allows the user to manipulate those frames.



Hint: Use the middle icon to change the flow direction. This operation may be undone if you make a mistake.

Family Only
Friends Only

Family Only
Friends Only

Join frames together...

○ Only once

● Forever

Data from frame *Family Only* will flow to frame *Friends Only* forever.

Figure 8.2: The attach frame dialog. The user selects frames from the left and right list boxes and the direction of the attachment by changing the middle arrow (the choices are: left-to-right, right-to-left, or both). Frames may be optionally joined once or forever. The bottom text reflects user choices and describes the proposed frame attachment in plain English.

a fictitious individual's personal assistant, performing tasks with Ligature. Completing the task is the primary goal; maintaining the boss's privacy is a secondary one, mirroring real life.

Ideally, a useful metric is to compare against other disclosure model approaches, notably access control lists (ACLs), which would act as a control. However, it is unclear how ACL-based solutions would support the separation and merging of contexts—the main use case for frames—in any meaningful way. Thus, any such comparison would be, at best, baroque.

The UI mimics a "sharing palette", a proven user interface for sharing files [95] and is shown in Figure 8.1 on the previous page. The palette provides a visually persistent record of what objects and frames have been shared to whom, permitting the study to focus on frame mechanisms, not on any particular vehicle of object exchange. With the palette, contextual services are limited to notes and tags for simplicity's sake, as most users are familiar with those constructs. The UI makes use of our prototype Ligature implementation as its repository for context. Low-level frame operations, such as attach, are hidden behind wizards (as shown in Figure 8.2) to mask complexity and elucidate their operation. Undo and redo operations are supported (except sharing is irreversible), though only one user ever exercised this functionality. Additionally, as part of the interface, users may optionally view

	#	Scenario
Easy	1	Please tag my vacation pictures as appropriate and share them with my college friends.
	2	Please send my vacation video to my work associates, but set it up so that their comments are kept separate from everybody else's.
	3	For my reports, I only have to keep 3 years worth of information. Get rid of all the old stuff.
Medium	4	On the technical reports, bring Alice into the consulting team with Bill and I. I want everybody to share comments with each other.
	5	I'm not so sure about Section 3 in report TR-1; I think Alice might be mistaken. Pass along the report to Brian asking him to review this part of the report. Keep his feedback discreet, but let him see all existing comments.

Table 8.1: Scenarios for the frames user study. The table continues on the following page.

the frame attachment graph visually, although no user chose to do so. The UI also contains context-sensitive help.

The sample population consisted of two distinct groups: 3 participants who self-identified themselves as computer experts and 8 novice users, who were mostly secretarial and administrative staff not affiliated with any of our departments. All had at least some college education and used a computer daily. Before the study, participants were required to watch a short tutorial video and were encouraged to spend as much time as needed on each scenario.

The scenarios were anecdotal, tiered into 3 difficulty groups (easy, medium, and hard) and presented in increasing order of perceived difficulty. The scenarios were not cumulative. Careful consideration was made with evaluation on a small pilot study to ensure the scenarios did not contain excessive domain-specific knowledge that may hamper a participant.

After completing each scenario, the system queried the participant to gauge, using 5-point Likert ratings (i.e., on a scale of 1–5), whether they understood what was asked of them in the scenario, their perceived difficulty of the scenario, and their confidence that they answered correctly. Likert ratings are treated as an ordinal level of measurement and are hence summarized by the median and, as a measure of variance, the interquartile range (IQR), which is the difference between the third and first quartiles or, alternatively, the middle 50% of the data. Also, as part of the study, we conduct concluding short, semi-structured interviews.

	#	Scenario
Medium	6	TechReport-2 supersedes TechReport-1, so I need you to move all of TechReport-1's notes and tags over, but still keep their data separate.
	7	I'd like the consultants to read Carol's comments on the memos, but Carol's not authorized to see their's. Set this up.
Hard	8	I've been assigned to edit a publication, which happens to have blind reviews (one reviewer cannot see another's review). I need you to share the manuscript with each reviewer such that they are able to add their separate comments.
	9	Give the completed set of reviews to Eve, allowing us to discuss the paper further. The completed set of reviews must stay hidden from the reviewers.

Table 8.1 (continued): Scenarios for the frames user study.

Table 8.2 on the following page shows the outcome of the study with Table 8.1 listing the corresponding scenarios. The results indicate the system scales well with user ability: whereas some systems are so complex that only expert users gain utility from them, it appears that novice users are able to leverage considerable functionality while expert users can take full advantage. Expert users were almost invariably able to correctly compose the correct frame relationships for each scenario. Their errors were due to participants perfunctorily completing tasks: in the 6th scenario, a participant failed to select the attach-once check box. Novice users were able to successfully complete most of the easy and intermediate difficulty scenarios. With the 5th scenario, some users were confused as to the nature of a duplicate frame. In the 6th scenario, a few users did not realize that attach-once semantics were needed. With the 8th and 9th scenarios, users seemed to have trouble understanding the scenario's requirements (5-point Likert ratings: $\tilde{\mu}$=2.5, IQR=1.0; $\tilde{\mu}$=2.0, IQR=1.0, respectively), which contributed to the marginal results.

There is a strong correlation between understanding the scenario's requirements and correctly answering it (ρ=0.88, p<0.01), as well as being confident in one's answer and correctly completing the scenario (ρ=0.81, p<0.01). This may suggest that once users understand a scenario, they are able to map the correct frame operations to satisfy it. Qualitatively, from the interviews, we found that users understood that a frame represented a separation of concerns, but were confused about frame attachment. Apropos, a more lucid interface, perhaps presenting common attachment patterns or some kind of feedback mechanism—for example, seeing connections through the eyes of another user—may help. This improved in-

	#	**Description**	**Behaviors Tested**	**Novice Users** (N=8)			**Expert Users** (N=3)		
				Accuracy	Time (m:s)	Difficulty	Accuracy	Time (m:s)	Difficulty
Easy	1	≈ Fig. 5.3a	new	100%	1:21 (0:13)	2.0 (1.2)	100%	0:48 (0:26)	1.0 (0.0)
	2	Share # of diff. frames	new	100%	1:18 (0:18)	2.0 (1.2)	100%	0:45 (0:23)	1.0 (0.0)
	3	Remove some frames	expire	100%	1:59 (0:24)	2.0 (1.0)	100%	0:51 (0:25)	1.0 (1.0)
Medium	4	≈ Fig. 5.3e	symmetric†	100%	2:10 (0:20)	2.5 (1.0)	100%	1:00 (0:07)	1.0 (0.5)
	5	≈ Fig. 5.3d	fork	75%	2:38 (0:29)	3.0 (1.0)	100%	1:20 (0:22)	2.0 (1.5)
	6	Join context only once	once†	75%	1:59 (0:24)	3.0 (0.5)	67%	1:13 (0:05)	2.0 (1.0)
	7	Join context one-way	attach	88%	2:02 (0:20)	3.5 (1.0)	100%	1:08 (0:11)	2.0 (1.0)
Hard	8	≈ PC ex. Fig. 5.3b	new	62%	3:06 (0:36)	4.0 (1.0)	100%	2:14 (1:02)	2.0 (1.0)
	9	≈ PC ex. Fig. 5.3c	attach	50%	1:56 (0:38)	5.0 (1.0)	100%	1:30 (0:11)	3.0 (1.0)

† These are attach idioms (c.f. Figure 5.1 on page 48).

Table 8.2: User evaluation results. The scenarios are listed in Table 8.1 on page 83. Some are isomorphic (≈) to those already presented in this paper. Accuracy is the percentage of users completing the scenario correctly. Time represents the mean time spent per scenario, in minutes and seconds, with the standard deviation shown in parenthesis. Difficulty is the median of a reported 5-point Likert rating with the interquartile range shown in parenthesis.

terface may aid users, as the UI in this study was rated only marginally acceptable (5-point Likert rating: $\tilde{\mu}$=3.0, IQR=0.5).

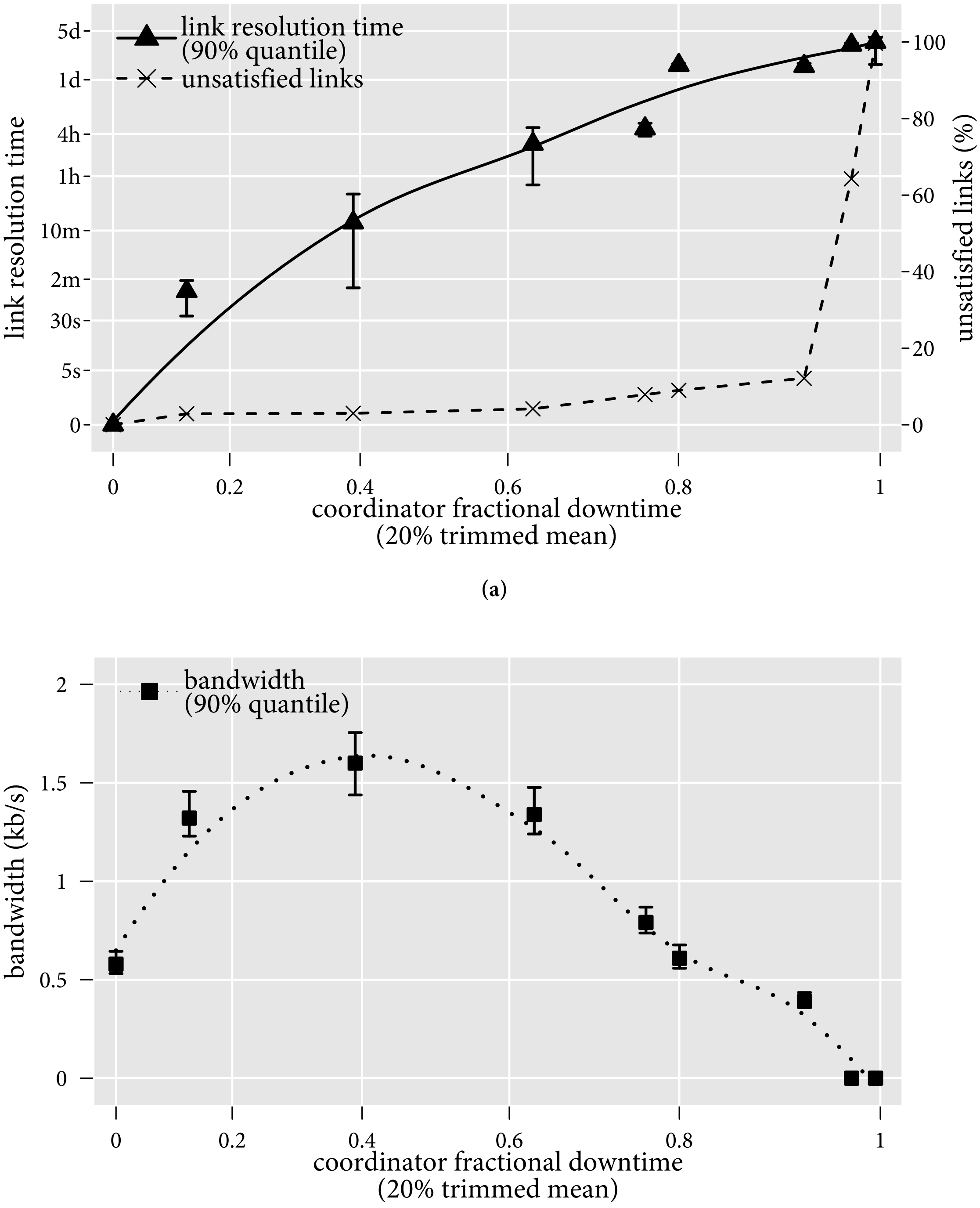
8.2 Discovering Peers

Using a coordinator for coterie introductions is naturally sensitive to the coordinator's availability. To examine the robustness of Ligature's coterie formation in the face of coordinator unavailability, we use a trace of email attachments *sent* through the EECS departmental mail server at the University of Michigan for a period of 9 months and described in detail in Chapter 6. Only messages sent are considered as the observational frame of reference is solely at one server; therefore, only the recipient list and sharing frequencies of attachments local users send are complete. In this trace, approximately 350 entities sent about 201,000 objects. We assume every object sent forms a coterie with the sender, who acts as the coordinator, and the message's recipients. While other forms of sharing undoubtedly exist, we suspect their sharing patterns mimic those found in the email trace.

To understand how peer discovery scales in the presence of coordinator failures, an availability model is applied to a simulation of Ligature's peer discovery protocol for a replay of the email trace. We are only concerned with coordinator failures as they impede coterie formation: the unavailability of non-coordinator nodes is irrelevant, as those offline are unavailable to exchange context anyway.

We cannot apply work done to characterize the diurnal availability patterns of hosts through segmentation into various uptime classes (e.g., always-on, work-week periodic, etc.) [61] for two reasons. First, there lacks a clear mapping between a coordinator and their particular uptime class. Second, there lacks a single parameter by which a coordinator's uptime may be tuned. This is necessary as these studies observed corporate or academic networks where a considerable number of hosts are always-on. Understanding failure mechanics with multiple, varying uptime classes is untractable. Instead, we use a hypothetical, but reasoned availability model, which is as follows.

In the trace, the sender/coordinator must have been online at every message send through the mail server. As a consequence, intuitively, a user's uptime is proportional to the frequency of email they send. This lends itself naturally to a simplistic and easily-reasoned availability model: after every message sent, the coordinator may experience downtime with some probability p until its next send event. That is, assuming a coordinator sends objects with periodicity T, its expected downtime periodicity is T/p. From this model, we can extract, *a posteriori*, the mean fractional downtime of all coordinators in our simulation. We use a 20% trimmed mean as this downtime distribution has significant skew due to outliers.



(a)

(b)

Figure 8.3: Ligature's elasticity to coordinator failure. For varying mean coordinator fractional downtimes in the email trace (shown on a log scale): (a) link resolution time (90% quantile) and the percentage of unsatisfied links; and (b) bandwidth consumed (90% quantile). The bars represent 95% bootstrap confidence intervals.

An important caveat is that as a coordinator's fractional downtime approaches unity, there will be links that cannot be satisfied; the coordinator may have such an infrequent object exchange rate that it is unavailable for the entire simulation.

The simulation results suggest Ligature's peer discovery mechanism is resilient to coordinator unavailability. Figure 8.3a on the previous page shows the 90% quantile of link resolution time as well as the percentage of unsatisfied links given the 20% trimmed mean coordinator fractional downtime. Only as coordinators become virtually uniformly unavailable does service degrade to an unusable level. Otherwise, in a highly unavailable environment, most links can be resolved in at most a few days; in a moderately available environment, at most a few hours. The results further suggest, as shown in Figure 8.3b, bandwidth usage is not a concern.

8.3 Performance Evaluation

The Ligature service runs silently in the background exchanging context and it therefore must not interfere with the user's foreground work. Thus, there are two salient performance measures: the system utilization when exchanging context and the efficiency of implicit frame changes in the presence of object operations (c.f. §5.2.2).

8.3.1 Contextual Exchange

For the former, the main performance measure of the service is system utilization: the fraction of total service time, including CPU and I/O times, over total elapsed time. This measure must be modest enough as to be imperceivable to the user. Since we are unable to ascertain how context is created and assigned without having the system deployed in the field, we evaluate this key performance characteristic with a synthetic workload and our prototype Ligature implementation.

Some simplifying assumptions are as follows. First, we assume each peer is identical in its computing resources. Second, content creation at each of these peers is modeled as a Poisson process with an expected tuple addition rate of $1/\alpha$ arrivals per second *per object shared*. This implies an exponential waiting time distribution with an expected value of α seconds. Each peer amortizes its updates over a window of 30 seconds, which means updates could be deferred for at most 30 seconds. This is the tunable parameter of freshness.

We use as our sharing topology the relationships that exist at the endpoint of the email trace (c.f. Chapter 6). The topology includes objects received from any recipient, but we show results exclusively for local users as only their object exchange histories are complete.

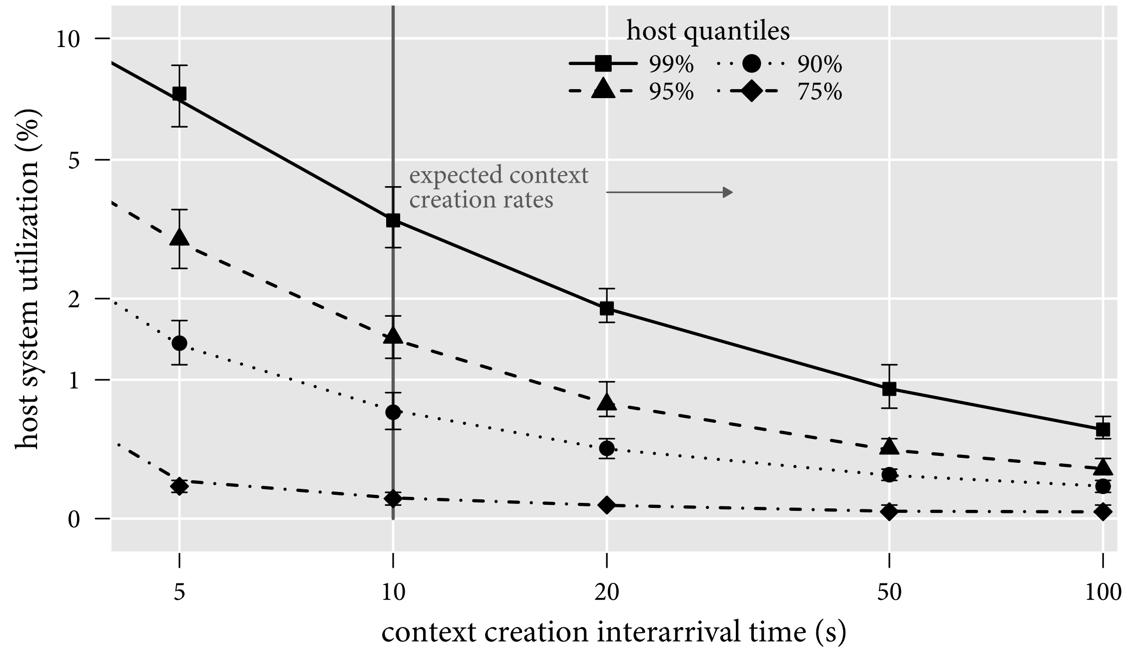


Figure 8.4: System utilization of hosts running Ligature. The figure shows host system utilization in 99%, 95%, 90%, and 75% quantiles of hosts, respectively, for varying context creation interarrival times per shared object (α). That is, at an interarrival time of 10 s, 99% of hosts in the trace exhibit less than 3.5% utilization; 95% of hosts, less than 1.5%. Only local users within the email trace are included. The bars represent 95% bootstrap confidence intervals.

For each peer, we generate traces of requests and use several load generators to drive our prototype implementation. For these experiments, the host under evaluation is a Pentium 4 2.26 GHz machine with a 7200 RPM disk and 512 MB of RAM. Based on data from the manufacturer, the disk has an average seek time of 8.9 ms for reads and 10.9 ms for writes.

Context creation is mostly a human-centric activity, and as a result, will exhibit infrequent interarrival times on the order of *tens of seconds*, or more likely *minutes*. For example, the rate at which users could possibly tag objects is limited by the time to locate the object, the think time to conceive of an appropriate label, and the user's typing ability. Most contextual applications—including machine-generated context—fall into this class of long, human scale interarrival times. For example, with the context-enhanced search service, the underlying data structure is almost exclusively updated as a consequence of user actions. Or, if in some hypothetical application where mobile users are continually publishing their location, most users are stationary [39]. Further, it is important to note that improvements in machine performance occur at a much faster rate than individuals share objects.

Figure 8.4 shows system utilization times for local users within the email trace. For expected context creation rates, most nodes in the system could effectively amortize contextual

	Phase	**Elapsed Time** (s)		Δ
		Native	Ligature	
1	Unpack	5.67 (0.2)	5.97 (0.3)	-5.3%
2	Copy Tree	4.21 (0.1)	4.52 (0.1)	-7.4%
3	Compile	259.8 (3.3)	261.1 (3.9)	—†
4	Unlink	0.16 (0.0)	0.38 (0.0)	-138%

† Within experimental error.

Table 8.3: Modified Andrew benchmark results. The benchmark uses an OpenSSL source tree with ≈1,875 objects. The parenthesis represent standard deviations based on 5 trials each.

exchange costs, which are usually less than 2% utilization. Almost all nodes exhibit less than 5% utilization. Nodes sharing many objects will naturally need to be over-provisioned to handle the additional load.

An important consideration is large coteries, especially in the presence of flash crowds. In these cases, pair-wise exchange among peers may prove to be unsatisfactory; a multi-cast solution [46] may be preferred.

8.3.2 Implicit Frame Changes

To evaluate the performance of implicit frame changes (c.f. §5.2.2) during file system events, we use a modified Andrew benchmark [41] with the OpenSSL 0.9.8i source tree, which contains approximately 1,875 file objects. The benchmark has four phases important to our analysis: (1) unpacking the archive; (2) copying the source tree; (3) compiling and linking the source; and (4) deleting the source tree. While not indicative of true user workloads, the benchmark does stress the side effects of object operations due to the relatively large number of objects in its working set. Under Ligature, the calculus demands: for phase (1), the creation of a new frame for each unpacked object; for phase (2), the forking of existing frames to the copied objects; phase (3), the creation of a new frame for each newly compiled object; and phase (4), removing frames with expire. The results of these phases run on a cold cache natively without Ligature and with the Ligature prototype implementation is shown in Table 8.3.

For most phases, the performance penalty is less than 10%. For the copy phase, the assumption is that each object has a single frame with 16K of context. During the compile phase, all context operations are amortized away ($t_8{=}-1.074$, $p{<}0.157$). The unlink phase is particularly expensive due to communication costs to the daemon. We expect real user workloads to perform better than our results would suggest, as they are unlikely to be laden with as many tightly interspersed file operations.

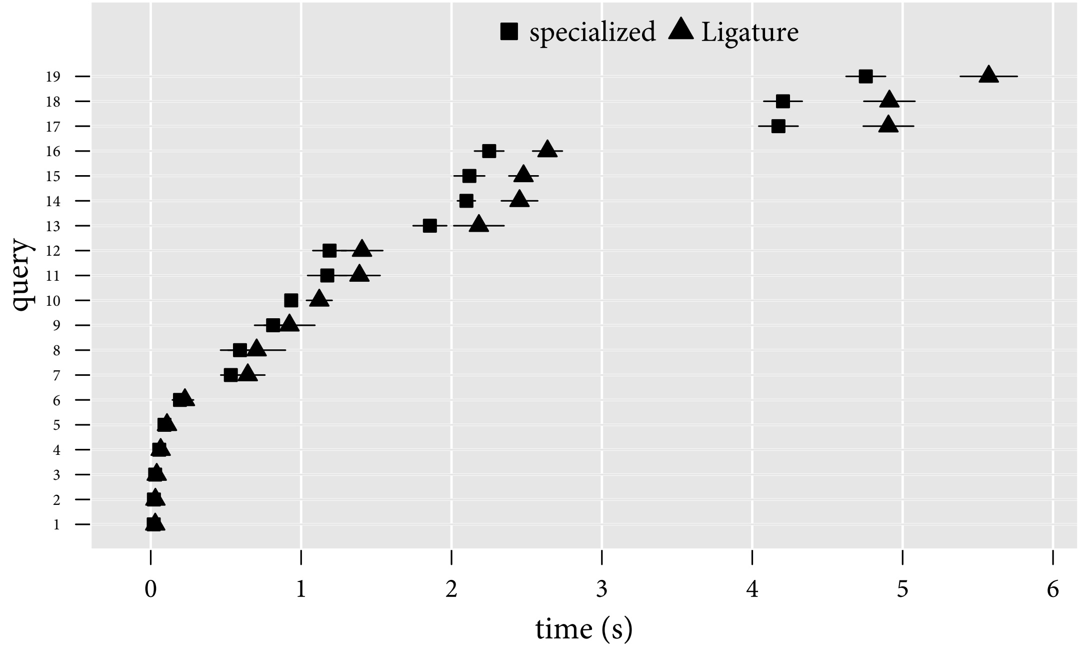


Figure 8.5: The application performance penalty with Ligature. Search times for queries from an 8-month trace of the author's system with a specialized data store and with Ligature as the backend store. The bands represent 95% confidence intervals based on 5 trials each. With Ligature, queries take, on average, 0.25 seconds or 17.4% longer (95% conf. int.: 0.12–0.37 seconds, 8.67–26.16%).

8.4 Case Study: Context-Enhanced Search

Storing context in Ligature's generalized data store, as opposed to a specialized data structure, comes with extra latency costs in accessing data. Particularly, with context-enhanced search, the time to answer a query must be within reasonable bounds for users to find the system palatable.

Weft's specialized data structure stores relations in a BerkeleyDB database as triples of the form: parent, child, and weight. An index on child entries allows efficient reverse lookups of that child's parents. Weft under Ligature serializes each relation as two tuples: one holding the child, another holding the link's weight. For efficient reverse lookups, the service also serializes reverse-relations at each child object to its parent.

Figure 8.5 shows search result latencies for 19 queries issued by the author over an 8-month tracing period. Each query is run for 5 trials each using either the specialized or Ligature backends. With Ligature, queries take an average of 0.25 seconds or 17.4% longer (95% conf. int.: 0.12–0.37 seconds, 8.67%–26.16%). A quarter second increase, on average, in search time latency is unnoticeable to the user. Result sets using both backends were randomly presented to seven users who attempted to judge which displayed faster. Over the course of 49 randomly selected queries (7 queries per user), 27 (55%) were correctly identified while 22 were not, indicating users were mostly unable to discern any difference ($t_{42} = -0.705$, $p<0.484$ given a binomial linear mixed model [93]).

8.5 Conclusion

The principal impact of Ligature is to leverage user collaboration by stitching together into a collective the context of individual users in a social neighborhood. In this manner, users can enjoy cooperative benefits from their effort.

As this chapter proves, Ligature can do so with negligible performance overheads: peers holding the same frame can be discovered relatively quickly, while contextual exchange costs with those peers can be amortized away for most hosts. As well, the additional mechanisms to support contextual frames have a minimal effect on file system performance and the application performance penalty of using Ligature as a backend is minimal.

Importantly, this chapter describes a small user study of contextual frames that was conducted. Both novice and expert computer users were given several tasks of varying difficulty to be completed using Ligature's mechanisms. The results of this study are encouraging, for the mechanisms scale with task difficulty: novice users were able to complete the easy and moderate tasks in almost all cases, while expert users were able to complete the difficult tasks as well.

CHAPTER 9

CONCLUSIONS & FUTURE WORK

For me, context is the key—from that comes the understanding of everything.

Kenneth Noland (1924–)

Context is an important and emerging instrument to assist users in the organization, search, and management of their data. This dissertation has presented two important architectures that engage context to aid in this information management problem.

Part I focused on employing context. Chapter I introduced Weft, a system that captures context from the user to reorder and extend search results. Chapter 3 demonstrated that users find the system beneficial, as their search results are improved.

Part II dove into sharing this and other kinds of context. Chapter II described Ligature, a system for sharing context. Chapter 5 introduced a new abstraction, the contextual frame, for sharing disparate context and maintaining personal boundaries on disclosure. Common email sharing patterns, which guide Ligature's architectural decisions, are presented in Chapter 6. The architecture and implementation of Ligature, as well as sample applications built with the system, were described in Chapter 7. Chapter 8 showed that the performance overhead of the system is negligible and that frames are palatable to users.

This chapter offers conclusions about this research, revisits its main contributions, and presents several items of future work.

9.1 Conclusion

At any given time, a user generates a wealth of contextual information, most of which is lost entirely. Even if this contextual information was preserved, it is unclear what kinds of context are the most useful.

In the realm of personal desktop search, this dissertation demonstrates that relatively straightforward provenance, acquired via strict causality, can be used to build a contextual index that later reorders and extend search results from traditional static tools. This work contributes a prototype system, named Weft, for capturing this context and a methodology for studying these personal desktop search tools that evaluates the user-perceived differences in search results.

Such a field study with non-expert users found that searches using Weft's causality algorithm to build the relation graph were of higher quality, on average, than both static content-only search and the current state-of-the-art in contextual indexing, temporal locality. These results validate that automatically gathered context increases the effectiveness of file indexing and search tools, despite the simplicity of the approach. The mechanism requires minimal space and time overheads, substantiating the usefulness of context.

Given that context is indeed a useful mechanism as indicated by Weft, it may be beneficial to share it across users in a cooperate manner. Naïvely storing context is problematic, however. For instance, if context was in some global store, there would be issues in resource allocation (i.e., who provides and services the infrastructure for this data?) and ownership. Importantly, this solution neglects the trust boundaries that exist in the real world. Thus, any such solution must not burden users by making context a new entity to manage.

This dissertation provides a context sharing system, named Ligature. Ligature leverages acts of sharing to form coteries of individuals who share context on an object: the acting of sharing the object imbues access rights to its context. While this basic model is adequate for the common case, Ligature permits context to be encapsulated inside a novel new abstraction, the contextual frame, which can be disparately shared among users, and even split and later merged. This yields easy, deterministic conditions for contextual exchange and facilitates the genres of disclosure common in human activity [30]. This work conducts a user study of these frames, which presented various scenarios to both novice and expert users to be completed using Ligature's frame mechanisms. The study found that novice users can extract most of their functionality, while expert users can employ all their functionality.

Avoiding unnecessary reliance on central services or authorities is a basic principle applied throughout Ligature's design. This preserves the user's autonomy to manage her own context without requiring registration with some organization, and preserves her privacy by avoiding an architectural requirement that a central service know a user's objects or context. Most importantly, it allows individuals without a common administrative domain of control to share context without any additional infrastructure. In addition, Ligature supports mobile and disconnected users by employing an eventually-consistent model, a common technique for human-scale events.

Ligature's use of a coordinator for coterie formation ensures allocation fairness. The peer discovery mechanism exploits sharing locality to find peers if the coordinator is down and can do so relatively quickly, as our results indicate. As well, this research captured a real-world sharing topology, and found that contextual exchange costs can be amortized away for most hosts given the sharing patterns in the trace. Ligature's frame operations have a minimal effect on file system performance and any application performance penalty is likely negligible. These results suggest that Ligature can be widely-deployed without imposing undue overhead.

Weft's contextual index can be synergistically shared among those the user has shared the object with, likely further improving search results. Other applications of context sharing include, but are not limited to, tags, notes, audit trails, or provenance records. This kind of fluid, ad-hoc and spontaneous collaboration is common in one's personal life, in business, and in scientific environments. As such, Ligature permits users to share context without undue managerial burden.

9.2 Contributions

In review, the main contributions of this dissertation are:

1. The identification of causality as a useful mechanism to inform contextual indexing tools and a description of a prototype system for capturing it.
2. A user study, including a methodology, for evaluating personal search systems that evaluates user-perceived differences in search results. The results demonstrate that causality-based indexing provides higher quality search results than those based on the, as of writing, current state of the art.
3. A system for distributing this and other context across a user's social neighborhood, which leverages acts of sharing to delineate access. The system's model supports the ad-hoc collaborative sharing that naturally occurs, all with negligible performance overheads.
4. A new abstraction, the contextual frame, that encapsulates context. It allows separation and later combination of context and facilitates genres of disclosure. A user evaluation of this abstraction suggests the mechanism scales with user ability.

9.3 Future Work

There are many avenues for important future work, both in employing context for personal desktop file search and in disseminating it to others.

9.3.1 Context-Enhanced Personal File Search

Although effective, the causality indexing mechanism used by Weft is coarse, for it can easily fail to capture real relationships, or alternatively, capture extraneous ones. Adding temporal bounds to causality-based creation of relation graphs may help reduce these false positives, as might instrumenting the user interface. For example, using window switching [72] to delineate units of work may ascertain extra relations and may lead to other kinds of relationships (e.g., read-read, as opposed to the system's current read-write). Indeed, there is work being done in instrumenting the UI layer for these relationships [38]. Application-assistance could also help, but it is likely developers would be loathe to do so.

Other repositories of context, such as a user's email and web activity, could provide additional—perhaps even sub-document—relationships. Hints from the environment and locally-available devices, such as a user's PDA or mobile phone, are another interesting avenue [21].

Weft's reordering and extension mechanism could be improved. Algorithms such as Pagerank [10] or those that are Pagerank-like [101] may be more applicable with a causally-based contextual index. As well, machine learning techniques may yield more accurate relevance scores.

A user's context will likely change over time. Since the relation graph does not store links as a function of time, it is difficult to perform any kind of hysteresis. The issue of what hysteresis methods are optimal for preventing dilution of relationships, stabilizing relation graph growth, and handling major task and work habit switching will eventually need to be addressed.

9.3.2 Context Sharing

There is considerable work in context sharing. As users begin to share more contextual data, questions arise regarding inter-user context sharing. Consider sharing the relation graph with Ligature. If Alice accesses data also used by Bob, how far should Alice's actions be used to assist Bob in organizing and searching the data? Can one obscure Alice's actions for privacy reasons and will any such obfuscation still be effective in search? The preceding supports the addition of a policy component to Ligature such that users and applications can provide rich annotations concerning the boundaries under which context should be shared. For example, the Weft service may wish to indicate that a relation should be only exchanged if the sender has shared both source and sink objects of the link with the recipient. Currently, this kind of policy is impossible with Ligature. A significant challenge is in devising a language supporting easy expression of these kinds of semantic constraints all the while retaining enough expressive power for common application use cases.

Each user plays a number of different roles over the course of a day, each with different obligations, with roles possibly overlapping. For example, a user might be an instructor, a researcher, a consultant, and a member of a family. In the context of each role, there is a group of other people with whom they interact, and these groups govern context access. Ligature provides a solution through the use of contextual frames, but this requires explicit manual intervention; an ideal solution is to adaptively set up access rights automatically. Since humans are habitual, it may be possible to construct a classifier to demarcate a user's current role and select apropos frames in an automatic, unsupervised manner.

Ligature currently permits sharing of inherently semi-private data in which users delineate their social neighborhood through acts of sharing. However, there are many cases in which it is useful to form communities implicitly, such that a system is sharing inherently semi-public data. For instance, fans at a football game may wish to share photos and their associated context with other fans at the game. Forming these groups without excessively compromising the user's privacy is a difficult challenge.

In a world in which context freely flows with objects, there may be context "overload" in that the user may have to wade through undesired context to reach his or her desired contextual need. Appropriate context-aware filters, especially useful on devices with small fixed-size screens, are another interesting avenue of research.

As contextual information rises to become more pertinent, inquiries into its use, its applicability, and its dissemination will undoubtedly grow as well. This dissertation establishes how context improves the utility of personal file search and how it can be shared with others without imposing unreasonable burdens, facilitating an exploration of these kinds of questions.

BIBLIOGRAPHY

[1] Nasreen Abdul-Jaleel, James Allan, W. Bruce Croft, Fernando Diaz, Leah Larkey, Xiaoyan Li, Donald Metzler, Mark D. Smucker, Trevor Strohman, Howard Turtle, and Courtney Wade. UMass at TREC 2004: Notebook. In *TREC 2004: Proceedings of the Thirteenth Text Retrieval Conference*, pages 657–670, 2004. 16, 18

[2] James Allan, Ben Carterette, and Joshua Lewis. When will information retrieval be "good enough?". In *SIGIR 2005: Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 433–440, Salvador, Brazil, 2005. 21

[3] Alexander Ames, Carlos Maltzahn, Nikhil Bobb, Ethan L. Miller, Scott A. Brandt, Alisa Neeman, Adam Hiatt, and Deepa Tuteja. Richer file system metadata using links and attributes. In *MSST '05: Proceedings of the 22nd IEEE/13th NASA Goddard Conference on Mass Storage Systems and Technologies*, pages 49–60, Monterey, CA, USA, 2005. 1, 42, 45

[4] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. ACM Press, New York, NY, USA, 1999. 16, 20, 21

[5] Mary G. Baker, John H. Hartman, Michael D. Kupfer, Ken W. Shirriff, and John K. Ousterhout. Measurements of a distributed file system. In *SOSP 1991: Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles*, pages 198–212, Pacific Grove, California, United States, 1991. 57

[6] H. Russell Bernard. *Social Research Methods*. Sage Publications Inc., 2000. 21, 22

[7] Laura Bertolotti and Maria Carla Calzarossa. Models of mail server workloads. *Performance Evaluation*, 46(2-3):65–76, October 2001. 57

[8] Nathaniel Borenstein and Ned Freed. MIME (Multipurpose Internet Mail Extensions) part one: Mechanisms for specifying and describing the format of Internet message bodies. RFC 1521 (Proposed Standard), September 1993. 58, 76

[9] C. Mic Bowman, Chanda Dharap, Mrinal Baruah, Bill Camargo, and Sunil Potti. A file system for information management. In *Proceedings of the Conference on Intelligent Information Management Systems*, Washington, DC, June 1994. 18

[10] 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. 96

[11] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Why and where: A characterization of data provenance. In *ICDT 2001: Proceedings of the 8th International*

Conference on Database Theory, pages 316–330, London, UK, 2001. Springer-Verlag. 10

[12] Min Cai and Martin Frank. RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network. In *WWW 2004: Proceedings of the 13th International Conference on World Wide Web*, pages 650–657, New York, New York, 2004. 45

[13] Sunny Consolvo, Ian E. Smith, Tara Matthews, Anthony Lamarca, Jason Tabert, and Pauline Powledge. Location disclosure to social relations: why, when, & what people want to share. In *CHI 2005: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 81–90, Porland, OR, USA, 2005. 55

[14] Mark Creorovella and Azer Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. In *Proceedings of the 1996 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer systems*, pages 160–169, Philadelphia, PA, USA, May 1996. 65, 67

[15] Mark R. Crispin. Internet Message Access Protocol – version 4rev1. RFC 2060 (Proposed Standard), December 1996. 58

[16] Edward Cutrell, Daniel C. Robbins, Susan T. Dumais, and Raman Sarin. Fast, flexible filtering with *Phlat*—personal search and organization made easy. In *CHI 2006: Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 261–270, Montréal, Québec, Canada, 2006. 19, 27, 37, 40, 45

[17] Mary Czerwinski and Eric Horvitz. An investigation of memory for daily computing events. In *HCI 2002: Proceedings of the Sixteeth British HCI Group Annual Conference*, pages 230–245, London, England, 2002. 24

[18] Susan B. Davidson and Juliana Freire. Provenance and scientific workflows: challenges and opportunities. In *SIGMOD 2008: Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1345–1350, Vancouver, British Columbua, Canada, 2008. 77

[19] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *PODC 1987: Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, pages 1–12, Vancouver, British Columbia, Canada, 1987. 72

[20] John R. Douceur and William J. Bolosky. A large-scale study of file-system contents. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems*, pages 59–70, 1999. 1, 78

[21] Paul Dourish. What we talk about when we talk about context. *Personal and Ubiquitous Computing*, 8(1):19–30, 2004. 1, 96

[22] Anton N. Dragunov, Thomas G. Dietterich, Kevin Johnsrude, Matthew McLaughlin, Lida Li, and Jonathan L. Herlocker. Tasktracer: a desktop environment to support multi-tasking knowledge workers. In *IUI 2005: Proceedings of the 10th International Conference on Intelligent User Interfaces*, pages 75–82, San Diego, CA, USA, 2005. 19

[23] Susan T. Dumais, Edward Cutrell, J. J. Cadiz, Gavin Jancke, Raman Sarin, and Daniel C. Robbins. Stuff I've Seen: A system for personal information retrieval and re-use. In *SIGIR 2003: Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 72–79, Toronto, Ontario, Canada, 2003. 1, 19, 40

[24] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003. 43, 44

[25] Scott Fertig, Eric Freeman, and David Gelernter. Lifestreams: An alternative to the desktop metaphor. In *CHI 1996: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 410–411, Vancouver, British Columbia, Canada, April 1996. 19, 37, 77

[26] Bryan Ford. Scalable Internet routing on topology-independent node identities. Technical Report MIT-LCS-TR-926, Massachusetts Institute of Technology, October 2003. 72

[27] George W. Furnas. Generalized fisheye views. *SIGCHI Bulletin*, 17(4):16–23, 1986. 77

[28] Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *Software: Practice & Experience*, 30(11):1203–1233, 2000. 77

[29] David K. Gifford, Pierre Jouvelot, Mark A. Sheldon, and James W. O'Toole. Semantic file systems. In *SOSP 1991: Proceedings of the 13th ACM Symposium on Operating Systems Principles*, pages 16–25, Pacific Grove, CA, USA, October 1991. 1, 17, 45

[30] Erving Goffman. *The Presentation of Self in Everyday Life*. Doubleday, New York, NY, USA, 1959. 55, 94

[31] Scott A. Golder and Bernardo A. Huberman. Usage patterns of collaborative tagging systems. *Journal of Information Science*, 32(2):198–208, 2006. 1, 47

[32] Richard A. Golding. *Weak-consistency group communication and membership*. PhD thesis, University of California at Santa Cruz, December 1992. 69

[33] Luiz Henrique Gomes, Cristiano Cazita, Jussara M. Almeida, Virgílio Almeida, and Wagner Meira, Jr. Characterizing a SPAM traffic. In *IMC 2004: Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, pages 356–369, Taormina, Sicily, Italy, 2004. 57

[34] Burra Gopal and Udi Manber. Integrating content-based access mechanisms with hierarchical file systems. In *OSDI 1999: Proceedings of the Third Symposium on Operating Systems Design and Implementation*, pages 265–278, New Orleans, Louisiana, 1999. 18

[35] Steven D. Gribble, Gurmeet Singh Manku, Drew Roselli, Eric A. Brewer, Timothy J. Gibson, and Ethan L. Miller. Self-similarity in file systems. In *Proceedings of the 1998 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, pages 141–150, 1998. 65, 67

[36] Krishna P. Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy, and John Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *SOSP 2003: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pages 314–329, Bolton Landing, NY, USA, 2003. 57

[37] Richard G. Guy, John S. Heidemann, Wai Mak, Thomas W. Page, Jr., Gerald J. Popek, and Dieter Rothmeier. Implementation of the Ficus replicated file system. In *USENIX Summer Conference*, pages 63–71, Anaheim, CA, June 1990. 44

[38] Karl Gyllstrom and Craig Soules. Seeing is retrieving: building information context from what the user sees. In *IUI 2008: Proceedings of the 13th International Conference on Intelligent User Interfaces*, pages 189–198, Gran Canaria, Spain, 2008. 12, 96

[39] Tristan Henderson, David Kotz, and Ilya Abyzov. The changing usage of a mature campus-wide wireless network. In *MobiCom 2004: Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*, pages 187–201, Philadelphia, PA, USA, 2004. 89

[40] Jason I. Hong, Jennifer D. Ng, Scott Lederer, and James A. Landay. Privacy risk models for designing privacy-sensitive ubiquitous computing systems. In *DIS 2004: Proceedings of the 5th Conference on Designing Interactive Systems*, pages 91–100, Cambridge, MA, USA, 2004. 56

[41] John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West. Scale and performance in a distributed file system. *ACM Transactions on Computing Systems*, 6(1):51–81, 1988. 90

[42] Galen Hunt and Doug Brubacher. Detours: Binary interception of Win32 functions. In *3rd USENIX Windows NT Symposium*, pages 135–143, Seattle, WA, USA, 1999. 15

[43] David Huynh, David R. Karger, and Dennis Quan. Haystack: A platform for creating, organizing and visualing information using RDF. In *Semantic Web Workshop*, Honolulu, HI, USA, May 2002. 1, 19, 45

[44] Victor Kaptelinin. UMEA: translating interaction histories into project contexts. In *CHI 2003: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 353–360, Ft. Lauderdale, FL, USA, 2003. 19

[45] Jeffrey Katcher. Postmark: A new filesystem benchmark. Technical Report 3022, Network Appliance, October 1997. 33

[46] Idit Keidar, Roie Melamed, and Ariel Orda. Equicast: scalable multicast with selfish users. In *PODC 2006: Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing*, pages 63–71, Denver, CO, USA, 2006. 90

[47] Kyung-Sun Kim and Bryce Allen. Cognitive and task influences on web searching behavior. *Journal of the American Society for Information Science and Technology*, 53 (2):109–119, 2002. 21

[48] James J. Kistler and Mahadev Satyanarayanan. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems*, 10(1):3–25, 1992. 44

[49] David R. Krathwohl. *Methods of Educational and Social Science Research: An Integrated Approach*. Waveland Inc., 2nd edition, 2004. 21, 22

[50] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978. 69

[51] Mark W. Lansdale. The psychology of personal information management. *Applied Ergonomics*, 19(1):55–66, March 1988. 2

[52] Paul J. Leach, Michael Mealling, and Rich Salz. A Universally Unique IDentifier (UUID) URN Namespace. RFC 4122 (Proposed Standard), July 2005. 70

[53] Will E. Leland, Walter Willinger, Murad S. Taqqu, and Daniel V. Wilson. On the self-similar nature of Ethernet traffic. *ACM SIGCOMM Computer Communication Review*, 25(1):202–213, 1995. 65, 67

[54] Dirk Lewandowski, Henry Wahlig, and Gunnar Meyer-Bautor. The freshness of web search engine databases. *Journal of Information Science*, 32(2):131–148, 2006. 47

[55] Jacob R. Lorch and Alan Jay Smith. The VTrace tool: building a system tracer for Windows NT and Windows 2000. *MSDN Magazine*, 15(10):86–102, October 2000. 33

[56] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice & Experience*, 18(10): 1039–1065, 2006. 77

[57] Mallik Mahalingam, Chunqiang Tang, and Zhichen Xu. Towards a semantic, deep archival file system. In *FTDCS 2003: Proceedings of the The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems*, page 115, Washington, DC, USA, 2003. IEEE Computer Society. 45

[58] Carlos Maltzahn, Nikhil Bobb, Mark W. Storer, Damian Eads, Scott A. Brandt, and Ethan L. Miller. Graffiti: A framework for testing collaborative distributed file system metadata. In *Proceedings in Informatics 21*, pages 97–111, March 2007. 44

[59] Frank Manola and Eric Miller, editors. RDF primer. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/, February 2004. 45, 69

[60] Donald Metzler, T. Strohman, Howard Turtle, and W. Bruce Croft. Indri at TREC 2004: terabyte track. In *TREC 2004: Proceedings of the Thirteenth Text Retrieval Conference*, 2004. 33

[61] James W. Mickens and Brian D. Noble. Exploiting availability prediction in distributed systems. In *NSDI 2006: Proceedings of the 3rd Symposium on Networked Systems Design & Implementation*, pages 73–86, San Jose, CA, USA, 2006. 86

[62] Robert J. T. Morris and Brian J. Truskowski. The evolution of storage systems. *IBM Systems Journal*, 42(2):205–217, 2003. 1, 72

[63] Kiran-Kumar Muniswamy-Reddy, David A. Holland, Uri Braun, and Margo Seltzer. Provenance-aware storage systems. In *USENIX 2006: Proceedings of the USENIX Annual Technical Conference*, pages 43–56, Boston, MA, USA, 2006. 1, 18, 42, 44, 54, 77

[64] Tamara Macushla Munzner. *Interactive visualization of large graphs and networks*. PhD thesis, Stanford University, 2000. 77

[65] Athicha Muthitacharoen, Robert Morris, Thomer M. Gil, and Benjie Chen. Ivy: a read/write peer-to-peer file system. *SIGOPS Operating Systems Review*, 36(SI):31–44, 2002. 44

[66] Andrew C. Myers and Barbara Liskov. A decentralized model for information flow control. In *SOSP 1997: Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, pages 129–142, Saint Malo, France, 1997. 45

[67] *FIPS 180-2: Secure Hash Standard*. National Institute of Standards, U.S. Department of Commerce, August 2002. 60, 70

[68] Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek, Ambjörn Naeve, Mikael Nilsson, Matthias Palmér, and Tore Risch. EDUTELLA: a P2P networking infrastructure based on RDF. In *WWW 2002: Proceedings of the 11th International Conference on World Wide Web*, pages 604–615, Honolulu, Hawaii, 2002. 45

[69] Wolfgang Nejdl, Martin Wolpers, Wolf Siberski, Christoph Schmitz, Mario Schlosser, Ingo Brunkhorst, and Alexander Löser. Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks. In *WWW 2003: Proceedings of the 12th International Conference on World Wide Web*, pages 536–543, Budapest, Hungary, 2003. 45

[70] B. Clifford Neumann. The Prospero file system: A global file system based on the virtual system model. In *FAST 1992: Proceedings of the USENIX Workshop on File Systems*, pages 13–27, May 1992. 18

[71] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 1999. 60

[72] Nuria Oliver, Greg Smith, Chintan Thakkar, and Arun C. Surendran. SWISH: Semantic analysis of window titles and switching history. In *IUI 2006: Proceedings of the 11th International Conference on Intelligent User Interfaces*, pages 194–201, Sydney, Australia, 2006. 12, 96

[73] Shwetak N. Patel and Gregory D. Abowd. The ContextCam: Automated point of capture video annotation. In *UbiComp 2004: Proceedings of the 6th International Conference on Ubiquitous Computing*, pages 301–318, Nottingham, UK, 2004. 55

[74] Chung-Kang Peng, Sergey V. Buldyrev, Shlomo Havlin, M. Simons, H. Eugene Stanley, and Ary L. Goldberger. Mosaic organization of DNA nucleotides. *Physical Review E*, 49:1685–1689, 1994. 67

[75] José C. Pinheiro and Douglas M. Bates. *Mixed-Effects Models in S and S-Plus*. Springer, New York, NY, USA, 2000. 23, 31

[76] Jonathan B. Postel. Simple Mail Transfer Protocol. RFC 821 (Standard), August 1982. 58

[77] Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan. The cricket location-support system. In *MobiCom 2000: Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, pages 32–43, Boston, Massachusetts, United States, 2000. 1

[78] Eric Prud'hommeaux and Andy Seaborne, editors. SPARQL, the query language for RDF. http://www.w3.org/TR/2007/CR-rdf-sparql-query-20070614/, June 2007. 69

[79] Tye Rattenbury and John Canny. CAAD: an automatic task support system. In *CHI 2007: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 687–696, San Jose, CA, USA, 2007. 19

[80] Risto Sarvas, Erick Herrarte, Anita Wilhelm, and Marc Davis. Metadata creation system for mobile images. In *MobiSys 2004: Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services*, pages 36–48, Boston, MA, USA, 2004. 2

[81] Martina Angela Sasse, Mark James Handley, and Shaw Cheng Chuang. Support for collaborative authoring via email: the MESSIE environment. In *ECSCW 1993: Proceedings of the Third Conference on European Conference on Computer-Supported Cooperative Work*, pages 249–264, Milan, Italy, 1993. 45

[82] Sam Shah and Brian D. Noble. A study of e-mail patterns. *Software: Practice & Experience*, 37(14):1515–1538, November 2007. 58

[83] Craig A. N. Soules. *Using context to assist in personal file retrieval.* PhD thesis, Carnegie Mellon University, 2006. 13, 16

[84] Craig A. N. Soules and Gregory R. Ganger. Why can't I find my files? New methods for automating attribute assignment. In *HOTOS 2003: Proceedings of the 9th Conference on Hot Topics in Operating Systems*, pages 20–24, Lihue, Hawaii, USA, 2003. 2

[85] Craig A. N. Soules and Gregory R. Ganger. Connections: using context to enhance file search. In *SOSP 2005: Proceedings of the Twentieth ACM Symposium on Operating Systems Principles*, pages 119–132, Brighton, UK, 2005. 1, 2, 8, 9, 12, 13, 17, 18, 21, 26

[86] Mirjana Spasojevic and Mahadev Satyanarayanan. An empirical study of a wide-area distributed file system. *ACM Transactions on Computing Systems*, 14(2):200–222, 1996. 57

[87] Steffen Staab and Heiner Stuckenschmidt, editors. *Semantic Web and Peer-to-Peer*. Springer, Berlin, Germany, 2006. 45

[88] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *SIGCOMM 2001: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 149–160, San Diego, California, 2001. 45, 72

[89] Murad S. Taqqu and Vadim Teverovsky. On estimating the intensity of long-range dependence in finite and infinite variance time series. In R. J. Adler, R. E. Feldman, and M. S. Taqqu, editors, *A practical guide to heavy tails: statistical techniques and applications*, pages 177–217. Birkhauser Boston Inc., 1998. 67

[90] Jaime Teevan, Christine Alvarado, Mark S. Ackerman, and David R. Karger. The perfect search engine is not enough: a study of orienteering behavior in directed search. In *CHI 2004: Proceedings of the SIGCHI Conference on Human Factors In Computing Systems*, pages 415–422, 2004. 1, 7

[91] Douglas B. Terry, Marvin M. Theimer, Karin Petersen, Alan J. Demers, Mike J. Spreitzer, and Carl H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *SOSP 1995: Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 172–182, Copper Mountain, Colorado, 1995. 44, 47

[92] Victoria Uren, Philipp Cimiano, Jose Iria, Siegfried Handschuh, Maria Vargas-Vera, Enrico Motta, and Fabio Ciravegna. Semantic annotation for knowledge management: requirements and a survey of the state of the art. *Web Semantics: Science, Services and Agents on the World Wide Web*, 4(1):14–28, 2006. 45

[93] William N. Venables and Brian D. Ripley. *Modern Applied Statistics with S*. Springer, New York, NY, USA, 4th edition, 2002. 92

[94] Arun Venkataramani, Ravi Kokku, and Mike Dahlin. TCP Nice: a mechanism for background transfers. In *OSDI 2002: Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, pages 329–343, Boston, Massachusetts, 2002. 70

[95] Stephen Voida, W. Keith Edwards, Mark W. Newman, Rebecca E. Grinter, and Nicolas Ducheneaut. Share and share alike: exploring the user interface affordances of file sharing. In *CHI 2006: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 221–230, Montréal, Québec, Canada, 2006. 43, 56, 57, 82

[96] Geoffrey I. Webb, Michael J. Pazzani, and Daniel Billsus. Machine learning for user modelling. *User Modeling and User-Adapted Interaction*, 11(1):19–29, 2001. 49

[97] Mark Weiser. Some computer science issues in ubiquitous computing. *Communications of the ACM*, 36(7):75–84, 1993. 43, 55, 68

[98] Tara Whalen, Diana Smetters, and Elizabeth F. Churchill. User experiences with sharing and access control. In *CHI 2006: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1517–1522, Montréal, Québec, Canada, 2006. 1, 2, 42, 43, 47, 56, 57

[99] Alma Whitten and J. D. Tygar. Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In *SSYM 1999: Proceedings of the 8th Conference on USENIX Security Symposium*, pages 169–184, Washington, D.C., USA, 1999. 80

[100] Zhen Xiao, Lei Guo, and John Tracey. Understanding instant messaging traffic characteristics. In *ICDCS 2007: Proceedings of the 27th International Conference on Distributed Computing Systems*, pages 51–59, Toronto, Ontario, Canada, 2007. IEEE Computer Society. 57

[101] Gui-Rong Xue, Hua-Jun Zeng, Zheng Chen, Wei-Ying Ma, Hong-Jiang Zhang, and Chao-Jun Lu. Implicit link analysis for small web search. In *SIGIR 2003: Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, pages 56–63, Toronto, Ontario, Canada, 2003. 96

[102] Gui-Rong Xue, Hua-Jun Zeng, Zheng Chen, Yong Yu, Wei-Ying Ma, WenSi Xi, and WeiGuo Fan. Optimizing web search using web click-through data. In *CIKM 2004: Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management*, pages 118–126, Washington, D.C., USA, 2004. 40

INDEX

Bold page numbers are used to indicate the defining reference for an entry, while *italic* numbers indicate the entry's main reference. For many entries, these delineations are often one in the same, as is indicated.