Confluence: Enhancing Contextual Desktop Search

Karl Gyllstrom
University of North Carolina
Chapel Hill, NC USA
27599-3175
karl@cs.unc.edu

Craig Soules
Hewlett Packard Laboratories
1501 Page Mill Road, MS 1134
Palo Alto, CA 94304-1126
craig.soules@hp.com

Alistair Veitch
Hewlett Packard Laboratories
1501 Page Mill Road, MS 1134
Palo Alto, CA 94304-1126
alistair.veitch@hp.com

ABSTRACT

We present *Confluence*, an enhancement to a desktop file search tool called *Connections* which extracts conceptual relationships between files by their temporal access patterns in the file system. A limitation of a purely file-based approach is that as file operations are increasingly abstracted by applications, their correlation to a user's activity weakens and thereby reduces the applicability of their temporal patterns. To deal with this problem, we augment the file event stream with a stream of window focus events from the UI layer. We present 3 algorithms that analyze this new stream, extracting the user's task information which informs the existing *Connections* algorithms. We present results and conclusions from a preliminary user study on Confluence.

Categories and Subject Descriptors: H.3.3 [Information Search and Retrieval]: Clustering

General Terms: Human Factors, Experimentation

Keywords: Contextual search, User modeling

1. INTRODUCTION

Unlike web search, desktop search lacks local hyperlinks which provide the foundation for structural search algorithms like PageRank. Consequently, desktop search is typically limited to text-based methods, placing the onus on the user to provide more descriptive queries, and often reducing the quality of results. Temporal context provides desktop search an alternative method with which to understand the relationships between files; namely that files which exhibit similar access patterns are likely to share a task commonality – even when those files bear no content similarities. *Connections*[3] is a local file search tool that departs from the traditional desktop search paradigm to incorporate these contextual relationships in search results.

Connections is composed of two main parts: context building and search. Contextual relationships are captured by a *relation graph*, where nodes represent files, and the links between them reflect the strength of their contextual relationships. To build the relation graph, a kernel-layer file system monitor records file operations such as *read* and *write* as a user goes about their work. While these events occur, Connections maintains a *relation window* (*RW*), which is a log of all file events occurring in the last n-seconds. When a new write event enters the *RW*, the relation graph link to the newly written file from each file read in the *RW* is incremented. Similarly, a new entering read event has its file's links to files written in the *RW* incremented.

Upon a user query, a pool of results is created using a traditional text based method (e.g. *tf-idf*). For each file in this pool, Connections identifies a subgraph of contextually related files by selecting all files on the relation graph connected to that file within n hops and of at least s link strength. A ranking algorithm such as PageRank or HITS is then used to transform this subgraph into an ordered list of files that is augmented to the original pool.

Despite positive results in [3], we perceived some limitations inherent in a purely file-based approach. As applications grow increasingly sophisticated, they tend to further insulate users from low level file operations, forging a divide between a user's conceptual document interaction and its file layer manifestation. We have developed *Confluence*, a set of modifications to the existing Connections algorithms that incorporate *application window focus* events, which are generated by the OS whenever a user changes the active window (typically through a mouse click or closing of the previously active window).

One problem with a file system based approach is the difficulty in differentiating background noise (e.g. *reads* from a virus checker) from user events (e.g. *writes* from a text editor). While much of the background noise is generated by system-owned processes, it can also be generated by passivated user processes (e.g. a text editor which automatically saves open files even when the application window has been minimized and not recently used). Our first algorithm, the *Focused Window Filtering (FWF)* algorithm, helps deal with this problem by applying information gleaned from the focused application window to inform the interpretation of that application's file operations. *FWF* assumes that the currently focused window dictates the active user task, and applies a filter to the file event stream which removes all file operations except those whose *process identifier (pid)* (or some parent *pid*) matches that of the currently focused window. The reduction of noise enables us to expand the duration and scope of the *RW*, as the original duration (30 seconds) and read-to-write increment restriction existed to manage the prohibitive volume of file operations. *FWF* allows a *focus-based relation window* that is sized to reflect a user's task, starting a new *RW* when an application window gains focus and ending it when the window loses focus. In addition, it allows for reads to be correlated with other reads within a common *RW*. Furthermore, the reduction of files helps maintains a leaner relation graph.

While a useful starting point, *FWF*'s inability to connect file operations which occur across different focus events limits its ability to effectively capture the typical task model, where users switch between multiple windows or applications as they go about their work. The *Focused Task Filtering (FTF)* algorithm broadens the definition of user task to the set of recently focused windows among which the user has switched focus as part of their work over a longer time interval, overcoming *FWF*'s inability to consider relationships between files which are accessed while different windows are focused. *FTF* maintains a log of focus-based *RW*s that occur over the last n seconds (e.g. 300 to 600). For each file event, *FTF* increments the links to each of the files in previous *RW*s in addition to the active *RW*, broadening the time period within which file relationships can be built while maintaining the advantages of filtering.

Another problem faced by Connections is the tendency of some applications to obviate file system activity through caching. For example, a user who opens a PDF file may refer to it many times through an application window while working; however, because the application caches the PDF, this activity is not reflected in the file system. The *Weight Carrying (WC)* algorithm extends *FTF*, maintaining a record of the set of file events that occurred while that widget last had focus.[1]. If that widget is focused again without witnessing a new file event matching the widget's *pid*, *WC* retrieves the last set of file events which occurred while that widget had focus, and inserts "fake" copies of those events into the file stream. Connections then has more information about how a file might be being used in concert with other files within the active task. Distinguishing widget from window allows for a more fine-grained application of this technique.

2. EVALUATION

We conducted a user study involving 4 volunteers over a 3-6 week period to evaluate the effectiveness of Confluence compared to Connections. Users installed the Confluence software, which included a kernel-layer file system event monitor and a UI event monitor, both recording to a common, secure file on their system. At the end of the period, we collected these traces and generated relation graphs using the various Confluence algorithms (parameterizing the FTF and WC algorithms for 5 and 10 minute intervals), as well as the original Connections algorithm.

From the file system logs, we generated and presented users a list of files accessed during the trace period and asked them to select a set of 5-10 disjoint tasks with which they were engaged during the period. For each task, they selected one or two files which were used as part of it. We used these files to seed searches on each of the algorithms' relation graphs, producing lists of files which were considered by the algorithms to be strongly contextually related to the seeds. For each seed file, we merged the set of the various algorithms' file lists into a single pool. To increase the pool's coverage of potentially related files, we also merged results from a directory search algorithm which produced a list of all files that existed at some point within the same directory as the seed file, under the premise that user's directory organization of their files at least partially reflects the commonality of those files. From this pool, users rated each listed file on a 0-3 scale, where 3 indicated the file was highly related to the seed file and 0 indicated no relationship. Using these ranked pools, we evaluated each algorithm by recall, or the percentage of the algorithm's results which had a rank of 3.

[1]Widgets, such as text panes, are distinct from windows in that they are components of other windows, and different widgets within a common window can operate on different files (e.g. tabbed text editors)

Method	5	10	15	20	25	30
FTF (300)	0.13	0.20	0.28	0.36	0.41	0.45
FTF (600)	0.13	0.19	0.27	0.31	0.38	0.41
WC (300)	0.10	0.20	0.28	0.33	0.36	0.40
WC (600)	0.10	0.18	0.29	0.33	0.35	0.39
Standard	0.13	0.15	0.18	0.19	0.20	0.20

Figure 1: Average Recall over Result Size.

Method	5	10	15	20	25	30
FTF (300)	0.03	0.66	1.16	2.05	2.57	3.10
FTF (600)	0.01	0.55	1.04	1.46	2.21	2.60
WC (300)	0.52	0.70	1.20	1.61	1.90	2.47
WC (600)	0.43	0.41	1.27	1.64	1.87	2.32

Figure 2: t-values ($df = 61$) of mean recall differences between Confluence algorithms and Connections. Dark gray is significant with $P < 0.01$, medium gray is $P < 0.05$, and light gray is $P < 0.1$.

3. RESULTS AND CONCLUSIONS

We evaluated Confluence and Connections for 31 file pools spanning the 4 users. Figure 1 depicts the algorithms' average recall values for results containing a score of 3 (*FWF* performed predictably poorly and its results are omitted from the graph). For a result size of 5, the different algorithms performed similarly. As result size increases, the improvement of the Confluence algorithms over the pure file-based approach grows substantial. Figure 2 depicts the Student's t scores comparing Confluence to Connections.

We did not observe a large performance difference between the *FTF* and *WC* methods, nor between task durations. However, while not strongly evident in the collective numbers, *WC* did find unique, accurate file relationships, and performs better at higher result sizes (50-100). *WC* presents a trade-off; its "fake" file insertion has the potential to find relationships that would be lost by other methods at the risk of enhancing false relationships which can push valid results further down the list.

In light of the strong performance of the *FTF* and *WC* methods, the poor performance of *FWF* indicates that it is not purely the filtering – but rather the increased RW duration and file operation flexibility enabled by the filtering – which constitutes the primary advantage of Confluence.

4. REFERENCES

[1] S. Dumais, E. Cutrell, J. Cadiz, G. Jancke, R. Sarin, and D. C. Robbins. Stuff I've seen: a system for personal information retrieval and re-use. In *Proc. of SIGIR '03*, pages 72–79, New York, NY, USA, 2003. ACM Press.

[2] D. Karger, K. Bakshi, D. Huynh, D. Quan, and V. Sinha. Haystack: A General Purpose Information Management Tool for End Users of Semistructured Data. In *CIDR '05*, pages 13–26, 2005.

[3] C. A. N. Soules and G. R. Ganger. Connections: using context to enhance file search. In *Proc. of SOSP '05*, pages 119–132, New York, NY, USA, 2005. ACM Press.