Operating System Support for Full-Text Search in File Systems

Stefan Büttcher Charles L. A. Clarke

School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

{sbuettch,claclark}@plg.uwaterloo.ca

Abstract

Full-text file system search tools have experienced an enormous boom during the last year. While they constitute a great improvement over conventional disk scanning methods, they are still far from being perfect. We present a list of basic requirements, including indexing efficiency and disk space consumption, analyze four different search systems, and show that none of them fulfills these requirements. We then argue that the existing shortcomings cannot be eliminated inside the search tools, but are at least in parts caused by lack of support by the operating system. Finally, we propose a new framework that is part of the operating system and offers support for efficient, full-text file system search.

1 Introduction

The ever-increasing storage space of hard disk drives makes finding a particular piece of information more and more difficult. Hard disk size has increased by a factor of 10,000 over the last 30 years, while disk bandwidth has increased by a factor of less than 100, and – even worse – disk access time has only decreased by a factor of 10 [7]. This gap will continue to widen in the foreseeable future. Therefore, it is infeasible to scan the entire hard disk whenever a user is searching for a file that contains a certain piece of information. Instead, an index structure that can be used to obtain search results more quickly has to be created ahead of time.

10 years after Manber and Wu [5] expressed the need for efficient, full-text file system search and presented their *Glimpse* search tool, major software companies are finally working on such systems. Full-text file system search, also known as "desktop search", has become a very active area during the last year. The recent advent of desktop search tools by Google, Microsoft, Yahoo, and others suggests that full-text file system search will be an integral component of future operating systems.

In this paper, we list a few essential requirements of desktop search systems and analyze existing solutions with respect to these requirements. We show that current search tools only meet some of the requirements and in fact cannot meet all of them – due to the lack of operating system support. We then describe which OS extensions are necessary to support real-time full-text file system search.

2 Related Work

Hardy and Schwartz [3] were among the first that articulated the need for special file system search tools. In 1993, they presented their *Essence* system that summarized file contents and built a key word index. In 1994, Manber and Wu [5] presented their *Glimpse* full-text search system, a combination of an inverted index and sequential search techniques.

A few years later, Microsoft integrated the Tripoli full-text indexing system into their Internet Information Server. Peltonen [6] reported about the Tripoli engine and the major design decisions, which were based on some basic requirements: "The search engine must be a nearly invisible, natural extension, [...] must scale with the operating system, [...] must never compromise [...] file system security." [6, p. 386]

In the following years, file system search almost became forgotten wisdom from an earlier age. The new era began with Dumais et al. [2] and their presentation of the *Stuff I've Seen* search system. After that, major software companies collectively understood the real importance of full-text search, and Google (Oct 2004), Microsoft (Dec 2004), Yahoo (Jan 2005), and many others released their new desktop search tools.

3 Some Basic Requirements

Before we examine the capabilities of current file system search programs, we list a few basic requirements that a search system should fulfill. Some of them are in line with Peltonen's point of view [6], while others go beyond his requirements. Although we see a search system as an OS service that can be uniformly used by all applications, the following requirements apply to desktop search tools as well:

Effectiveness

The search engine has to offer *full-text search*, support many different file formats and be able to index even very large files. Search results have to refer to the exact position inside a file that matches the query, rather than to the entire file. Nobody needs to be told that the e-mail he is looking for is inside his mailbox file!

Efficiency

The search engine has to offer *high search performance* (i.e. processing a query should take less than a second) and require *little disk space*, preferably less than 10% of the file system. In particular, every file has to appear in one index – not n indexes, where n is the number of users that may read the file.

Furthermore, we demand *high indexing performance*. This includes instantaneous updates (if a file is created or deleted, the index has to reflect this change immediately), as well as the ability to deal with *content-preserving changes*: If a file is moved to another directory, it should not be re-indexed; if data is appended to an existing file (happens frequently for `mbox` files used by e-mail clients), it should not be necessary to re-index the entire file, which can be very large.

Index locality

A file system index has to be tied to the file system that it refers to. It is pointless to create an index of the files found on a USB stick and keep the index after the memory stick has been removed. Moreover, if the index is part of the file system, it can be re-used when the USB stick is plugged into another computer.

Data security and privacy

Our co-worker is not supposed to know or infer that we are secretly sending love e-mails to his girlfriend, even if we share the same computer.

4 The Current Situation

We installed four different desktop search programs that run under Microsoft Windows (similar tools exist for other operating systems, but Windows currently shows the most activity):

- Google Desktop Search Beta;
- MSN Desktop Search Beta;
- Yahoo Desktop Search Beta;
- Copernic Desktop Search 1.2.

The test system was a small Windows XP installation with a few thousand indexable files and a few hundred e-mails. Each program was analyzed using the requirements from section 3.

Effectiveness

Three out of the four programs tested support full-text indexing on the sub-file level, i.e. it is possible to search for phrases such as "file system search". Google claims to offer full-text indexing but in fact indexes only the first few kilobytes when it processes a large text file.

All programs differ drastically in their ability to index various file types and to return sub-file positional information along with the search result. While we had no problems with Copernic and Yahoo, MSN Desktop Search did not index any PDFs, and Google even refused to index our Word documents. Furthermore, while both Yahoo and Copernic let the user browse through a matching file (highlighting matching terms and displaying page numbers), MSN in many cases only returned a "preview" containing the first few sentences of the file – regardless of which part of the file actually matched the query.

Efficiency

All programs offer satisfying *search performance*: When we ran our tests, the results usually showed up in less than a second after the query had been submitted to the search system, with some systems even supporting incomplete, prefix searches ("search as you type").

Disk space consumption was around 100 MB or less in all cases, which seems acceptable. Unfortunately, this number is misleading because none of the programs tested actually uses a single, system-wide index. MSN, Yahoo, and Copernic all create one index per user. When there are many user accounts on a computer, this leads to multiply indexed files and potentially much higher *disk space consumption* than in the single-user scenario we simulated. Google has one single index for the entire system. However, this index can only be queried by users with administrator privileges – which makes its usefulness somewhat questionable.

Instantaneous updates are supported by three of the four programs: When a new file is created or an existing file is deleted, all but Yahoo's search tool are able to update their index immediately to reflect this change (might take a few seconds with MSN). Yahoo only detects the change during its next scheduled full file system scan.

We tested two kinds of *content-preserving changes*: Moving a file to a different directory and appending new data to an existing file. None of the search programs tested was able to detect that all (or most) of the data inside the file had not changed and consequently take advantage of this knowledge. Instead, the entire file (100 MB) had to be re-indexed.

Index locality

All search programs create per-user indexes instead of per-file-system indexes. When we plugged a USB stick into the test computer, created a file, and then removed the USB stick from the system, the file still showed up as a search result in both MSN and Copernic (Google and Yahoo refuse to index anything but the primary file system). Even worse, Windows did not let us remove the USB stick until we terminated MSN's search tool. This shows that none of the programs is able to deal with the existence of multiple file systems in an appropriate way.

Data security and privacy

Since all programs create per-user indexes, read permissions cannot be violated by the search system. However, if a user by mistake creates a file that is readable by everybody and later corrects this mistake, the file will still be inside other users' indexes and thus searchable.

Summary

It is clear by now that none of the search systems examined by us fulfill the requirements we have listed in section 3. However, this is not entirely the fault of the search system developers. While the disk space and privacy problems may (and have to) be solved inside the indexing system, support by the operating system is necessary in order to deal with content-preserving file changes and removable storage devices in an appropriate way. So far, this support does not exist, as we show in the next section.

5 Current Operating System Support

We examined the possibilities to register for file system change notification under both GNU/Linux and Microsoft Windows and found that only Linux offers at least partial support for content-preserving file changes.

Microsoft Windows

All recent Windows versions support the `FindFirstChangeNotification` system call, which can be used to register and wait for notifications on file changes by the operating system. Notifications by the operating system include information about file name or attribute changes and file write operations.

The crucial shortcoming of this mechanism is that it does not inform about the exact changes that have taken place. For example, when data is written to a file, the OS does not provide information on what part of the file has been changed. Similarly, when a file is moved from one directory to another, this results in two events (file removed from one directory; file added to another), and it is difficult or impossible to say which pair of events belongs together.

Another shortcoming is that, in order to be notified of file changes, an application has to be running at the time the change takes place. This is why some applications use information from the NTFS journal file to keep track of file system changes. Unfortunately, using journal data leads to limitations that are similar to those of the first method. Besides, it is only possible if (i) the file system is journalled and (ii) the OS only supports a small number of different file system formats (2 in this case).

GNU/Linux

Linux supports file change notification through the `dnotify` interface. However, it can only monitor individual directories and needs a file handle for each – a severe limitation. Its anticipated successor, `inotify`, does not have this limitations, but has the same shortcomings regarding content-preserving changes as Windows. The same holds for third-party solutions, such as `fam`, `filemon`, and `dazuko`. A new `dnotify` kernel patch [4], developed independently, provides additional information about the exact part of a file that has been changed by a write operation. However, changes caused by `mmap` are not detected, and the patch is not actively maintained.

`fschange` is a patch for the Linux kernel that is compatible with the framework presented in the next section. It can be downloaded from the author's website [1]. `mmap` is dealt with in a conservative way by assuming that the entire region that is mapped has been modified (except for read-only and copy-on-write maps).

6 A Framework for Full-Text File System Search

In this section, we propose a framework (shown in Figure 1) that includes both kernel and user space and that fulfills the requirements listed in section 3.

The kernel part

We first describe the kernel part of our framework: In most computer systems, we have more than one file system. Every file system can either be *indexed* or *not indexed*, as specified when it is created. Each indexed file system sends details about every change to the OS kernel, which forwards the information to the system logging service, a permanently running process with administrator privileges. Having the file system implementation inform about changes makes it possible to take advantage of implementation-specific knowledge, like the effects of memory-mapped write operations, or hard links.

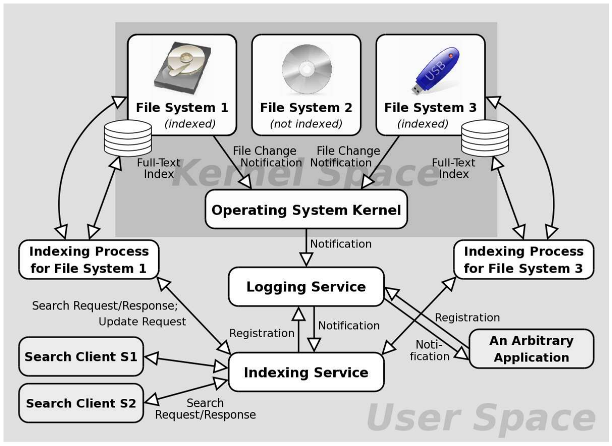

Figure 1: A framework for full-text file system search. Each indexed file system has its own local index data and an indexing process maintaining the index. Processes running with administrator rights are shown in white, ordinary user processes in light grey.

The messages that are sent contain specific information about file and directory changes. We give them in the form "`MESSAGE_TYPE` *attribute*":

- `CREATED/DELETED` *file path*
- `MOVED` *old file path*, *new file path*
- `TRUNCATED` *file path*, *new size*
- `PERMISSION_CHANGED` *file path*, *new perm.*
- `OWNER_CHANGED` *file path*, *new owner*
- `CONTENT_CHANGED` *file path*, *start*, *end*

The message types are self-explanatory. The important detail here is that the `MOVED` message lets us detect content-preserving changes. The *start* and *end* attributes of a `CONTENT_CHANGED` message can be used to determine which exact part of a file has been changed and to avoid re-indexing of already indexed portions of a large file. In order to reduce the overhead introduced by `CONTENT_CHANGED` messages, they are only sent after a file has been closed. This also lowers the risk of having to index the same file again in the near future, as open files tend to be changed more frequently than closed ones.

In addition to these messages, the kernel sends a message to the logging service every time a file system is activated ("mounted") or deactivated ("unmounted").

The logging service

The logging service stores messages from the kernel, not necessarily restricted to the ones described above. Arbitrary applications may register with the logging service and ask for messages of certain types. Depending on the privilege level of the application, the logging service either forwards all messages, or a subset, such as information on file changes for all files that can be read by the user running the application.

An important feature of the logging service is that it can store messages persistently so that the indexing service can even be notified of file changes that take place when it is not running, such as system startup and shutdown.

The indexing service and per-file-system indexing processes

The indexing service registers with the logging service to receive all messages dealing with file system changes, such as file creation and file system insertion. When an indexed file system is mounted, the indexing service is informed and creates a new indexing process responsible for indexing this file system. When the file system is unmounted later on, the process associated with that file system is terminated.

There are two different types of messages sent from the indexing service to one of the indexing processes:

- When notified of a file system change, the indexing service sends an index update request to the respective indexing process.
- When an application submits a search request, the indexing service contacts all indexing processes and gathers the information necessary to produce the search result.

Having one index per file system allows us to react to the activation or deactivation of file systems immediately. It also has the nice property that an existing index on a removable storage device can be re-used. Furthermore, by having the indexing service request data from a remote indexing process instead of a local one, the framework can easily be modified to support searching in network file systems.

7 Conclusion and Future Work

We looked at the problem of integrating full-text file system search into an operating system and presented a few fundamental requirements, including indexing efficiency, retrieval effectiveness, and file system security. We analyzed existing file system search tools and came to the conclusion that they are unable to meet these requirements – due to the lack of adequate operating system support. Among their shortcomings were the unability to deal with content-preserving file changes (move, append) and dynamically activated file systems (e.g. removable storage devices).

We then presented a framework, consisting of a system logging service, a central indexing service, and several per-file-system indexing processes, that allows to deal with content-preserving file changes and the dynamic activation and deactivation of file systems. In contrast to the existing search systems, our framework does not need any per-user index data, which leads to much lower disk space consumption on multi-user systems. In addition, it seamlessly supports searching in remote file systems (like NFS or SMB). We believe that the framework is both general enough as well as relatively easy to implement so that it fits into most currently available operating systems.

We have not presented a solution to the security problems that arise in the absence of per-user index data. However, we believe that this problem can be solved within the indexing service and are currently investigating efficient techniques to adjust the search results collected from the individual indexing processes in such a way that no security constraints (file permissions) are violated.

References

[1] `fschange` – Linux Filesystem Change Notification. http://stefan.buettcher.org/cs/fschange/, 2005.

[2] S. Dumais, E. Cutrell, J. Cadiz, G. Jancke, R. Sarin, and D. Robbins. Stuff I've Seen: A system for personal information retrieval and re-use. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2003.

[3] D. R. Hardy and M. F. Schwartz. Essence: A resource discovery system based on semantic file indexing. In *USENIX Winter*, pages 361–374, 1993.

[4] Lambda Computing. `dnotify` – Enhanced file change notification for Linux. http://www.lambda-computing.com/projects/dnotify/, 2005.

[5] U. Manber and S. Wu. Glimpse: A tool to search through entire file systems. In *USENIX Winter*, pages 23–32, 1994.

[6] K. Peltonen. Adding full text indexing to the operating system. In *Proceedings of the Thirteenth International Conference on Data Engineering (ICDE 1997)*, pages 386–390. IEEE Computer Society, 1997.

[7] Wikipedia, the free encyclopedia: "Hard disk". http://en.wikipedia.org/wiki/Hard_disk, 2005.