Dessy: Towards Flexible Mobile Desktop Search

Eemil Lagerspetz
Helsinki Institute of Information Technology
P.O. Box 9800
FIN-02015 HUT, Finland
lagerspe@cs.helsinki.fi

Tancred Lindholm
Helsinki Institute of Information Technology
P.O. Box 9800
FIN-02015 HUT, Finland
tancred.lindholm@hiit.fi

Sasu Tarkoma
Helsinki Institute of Information Technology
P.O. Box 9800
FIN-02015 HUT, Finland
starkoma@hiit.fi

ABSTRACT

In the near future, mobile devices are expected to have a storage capacity comparable to today's desktop machines. As the amount of infomation grows, conventional search tools become less effective. To meet these challenges, we turn to desktop search. This paper presents Dessy, a **DES**ktop **S**earch director**Y** system for mobile and desktop computers alike. It allows a user to find files by their content, metadata, and context information, provides an interface for locating files for both users and applications, and allows finding files with just their metadata available. Dessy supports separate synchronization of file metadata and data. Dessy is unique among desktop search software in that it clearly separates extraction, exchange, and querying of search metadata. In particular, metadata extraction may be performed on one device and querying on another, with only search metadata being exchanged between the devices. Dessy emphasizes extensibility: adding new indexed file types, metadata fields, and index storage methods is easy. Finding files is done with virtual directories, which are views into the user's files, browseable by regular file managers.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: Content Analysis and Indexing, Information Storage, Information Search and Retrieval

General Terms

Mobile computing

Keywords

desktop search, synchronization, mobile computing, metadata

1. INTRODUCTION

In the near future, mobile devices are expected to have a storage capacity comparable to today's desktop machines. As the amount of information grows, conventional search tools become less effective. To meet these challenges, we turn to desktop search.

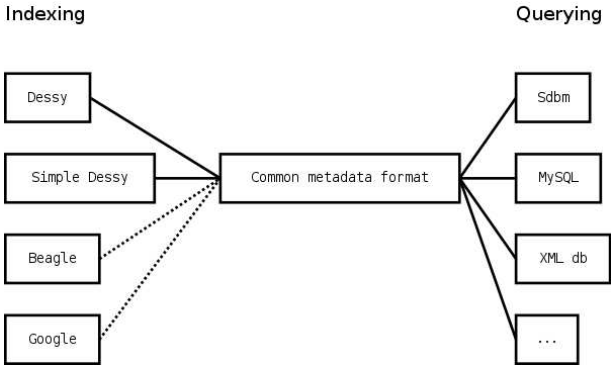

Figure 1: Dessy separates metadata extraction, exchange and use.

This paper presents Dessy, a **DES**ktop **S**earch director**Y** system. Dessy allows a user to find files by their content, metadata, and context information. It provides an interface for locating files for both users and applications, and allows finding files with just the files' metadata available. Dessy is built with data synchronization in mind. Finding files is performed by virtual directories [4]. These can be used from applications and by the user by directly browsing them with a file manager. The Dessy GUI provides easy access to the features of the virtual directory system. The virtual directories are provided through an NFS [12] interface, which allows applications unaware of Dessy to browse them.

Dessy clearly separates extraction, exchange, and querying of search metadata. In particular, metadata extraction may be performed on one device and querying on another, with only search metadata being exchanged between the devices.

The exchanged metadata is in a simple plaintext format that is agnostic to query engine implementations, and well suited for network protocols. Thus, one device may implement queries on search metadata using an SQL database, and another may use a simple and query-wise more limited key-value database suited for limited devices. Still, these devices exchange the search metadata in a common format over the network.

The same reasoning applies to search metadata extraction, i.e., indexing. On a powerful machine, advanced algorithms may be used to extract high-quality search metadata, whereas on a limited device, a lighter approach may be used to extract some simple metadata. Again, regardless of its origin, the metadata is exchanged in the same format. Figure 1 illustrates this separation of metadata extraction and querying. The simple metadata exchange format could be used to exchange metadata with existing systems, such as Google Desktop[1] or Beagle[2].

When combining the standard metadata exchange format with a synchronization system that supports separate synchronization of data and metadata, we get a flexible search architecture that allows decoupling of search metadata producers from consumers, and metadata from actual data. Since query indexes are built from search metadata, rather than the actual data, we may query files for which we have only synchronized the metadata. The actual content need only be synchronized when the required file has been found and is opened for viewing or editing. In this paper, we use the Syxaw file synchronizer, which supports separate synchronization of data and metadata.

Dessy is extensible in many ways. Users can add tags similar to the ones used in other desktop search software to represent context associations of files. Tags can also be used by context-aware applications to add context information to files. Developers can easily add *Indexing helpers* that handle new file types and add new properties used to find files. Changing the indexing is possible by adding a new *Indexer*, while storage in on-disk indexes can be changed by writing a new *Index*, and handling of user queries is modifiable with new *QueryPlugin*s.

The features that distinguish Dessy from other systems are the following. The main feature is the support for both mobile and desktop systems. Dessy has integrated support for file data and metadata synchronization between desktop systems and mobile terminals. It separates metadata extraction, exchange and use. Dessy integrates seamlessly in the current file system scheme and can be used through any file manager on desktop machines. Finally, the system emphasizes extensibility through pluggable file indexers.

Dessy is implemented on top of the file system layer, between the file system and applications. It runs on Java Personal Profile [17], Linux, and Windows desktops. The Dessy prototype is implemented in Java 1.5. The Dessy GUI is written in Java AWT.

The structure of the rest of this paper is as follows: Section 2 discusses related work and introduces terms and concepts used in the rest of the paper. Section 3 explains the synchronization model used in Dessy. Section 4 contains an example use case. Section 5 presents the architecture of Dessy. Section 6 shows scalability and performance experiments and discusses central implementation concepts. Section 7 concludes the paper and outlines future plans.

[1] `http://desktop.google.com/features.html`
[2] `http://beagle-project.org/Main_Page`

2. RELATED WORK

Due to recent improvements in desktop computer disk space availability, more files are stored on home and work computers. The inefficiency in finding files on one's own disk compared to the speed of Internet searches has prompted a rise in desktop search software development.

Recent years have seen a multitude of desktop search applications being developed. Some of the most known examples of these are Apple's Spotlight[3], Novell's open-source project Beagle[4], Copernic Desktop Search[5], Google Desktop[6], Microsoft's SiS [2], and Windows Desktop Search[7].

Beside the usual user interfaces for queries, finding files has also been approached in different manners, such as using virtual directories [4] and directory namespaces [6]. A *virtual directory* (also called a *virtual folder*) is a directory that does not exist in the file system hierarchy, but instead shows found documents according to search criteria specified for the directory. In [4], two types of virtual directories were used: *field virtual directories*, that denoted a field or property name, and *value virtual directories*, that indicated the value of the property that the parent field virtual directory identified. For example, a file having the value `bob` for the field `owner:` would be found in the virtual directory `owner:/bob`. A virtual directory-like approach is also present in later versions of Apple's Spotlight. There it is called *Smart Folders*. In Windows Vista saved desktop searches are called *Search Folders*. In [4], the virtual directories were accessed through one server, and indexes were not transferrable to other devices. Dessy adopts the virtual directories for desktop searching, using the local device for index storage instead of a server. This allows offline browsing. In the rest of this paper, field virtual directories will be called *property virtual directories* to better match their use in Dessy.

In many desktop search applications, such as Beagle and [4], indexing of different file types is done by *indexing helpers*. Indexing helpers are specialized property and text extractors designed to handle one or more file types. Dessy uses indexing helpers also.

Many of today's desktop search applications look for files according to their content or fields specific to file types. In Connections [16], document creation and modification times were used to create links between documents to help the user find related files. This technique was used to improve a regular desktop search system. The use of context information and user-friendly metadata was explored in [3] and [6] by using application- and user-defined *tags*. A tag is a keyword or a (property, value) pair assigned to an object. In Dessy, modification dates and other metadata properties can be used to search for a file. Adding custom metadata and context tags is possible.

In search engine research, two common indexing techniques

[3] `http://www.apple.com/macosx/features/spotlight/`
[4] `http://beagle-project.org/Main_Page`
[5] `http://www.copernic.com/en/company/index.html`
[6] `http://desktop.google.com/features.html`
[7] `http://www.microsoft.com/windows/desktopsearch/search/default.mspx`

exist. In *inverted file* [5] indexing, a list of file identifiers is stored for each keyword. When the keyword appears in a query, the list is added to the results or intersected with them, depending on whether the query was a boolean **or** or **and**. In *signature file* indexing, a hash function is used on the words in a file to determine the *signature* of the file. Queries are then hashed and matched against this signature to determine if files match. These methods of indexing are compared in [19]. According to the results of the comparison, inverted file indexing is superior. The index is smaller since compression can be used to reduce index size without sacrificing too much query performance, unlike when using signature file indexing. Inverted file query performance is generally better, since checking for false positives is not necessary. Inverted file indexing is used in Dessy.

In [10], *n-grams*, which are word fragments of length n, were used instead of words in an inverted index. As the number of alphabetical *2-grams* and *3-grams* is relatively small (676 and 17 576, respectively), using these instead of words limits the size of an index. However, this technique causes false positives, since a phrase can include the n-grams of another, even though it does not contain the words of the latter.

To better match queries to words occurring in indexed files, it is common to *stem* query terms and indexed words. The process of *stemming* reduces a word to a form near its basic dictionary form. For example, `stemming`, `stemmer` and `stemmed` could be reduced to `stem`. One popular stemming algorithm is the Porter Stemming algorithm [14]. This algorithm is used in Dessy.

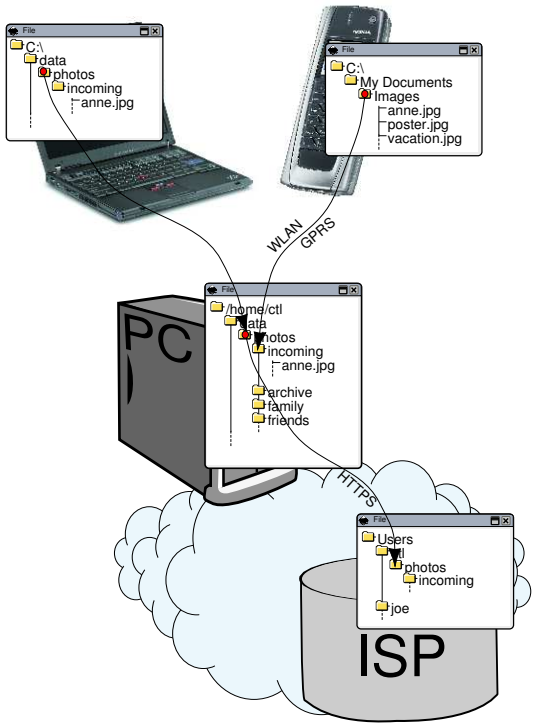

Figure 2: Syxaw synchronization links.

3. SYNCHRONIZATION MODEL

For file synchronization, we use the Syxaw file synchronizer [18]. The model for sharing data in Syxaw is based on establishing *synchronization links* between local and remote *objects*, i.e., files and directory trees. The link is persistently stored along with other object metadata. When synchronizing an object a linked to an object b, we propagate changes made to these objects since the last point of synchronization. In the case that the objects are directory trees, the full contents, including contained files and directories, are synchronized. Links are unidirectional, meaning that only the device from which the link originates knows about the link. We use the term *client* for the link originator, and *server* for the link target.

From a synchronization point of view, we model data objects as follows. Each object is identified by a unique identifier (UID), which is an opaque string of bits. UIDs are used in the synchronization protocol, rather than file paths. Objects have both a *metadata* and a *data* part. These may be synchronized separately. In particular, for some collection of objects, we may retrieve the metadata before fetching the actual data. Finally, there are objects that enumerate other objects. The directory tree object, which provides a mapping between hierarchical file names and UIDs, is the most prominent example of such an object.

Syxaw synchronization links are illustrated in Figure 2 in the form of a photo archive setup. Let us call the fictive user of the setup Bob. Initially, Bob stored his pictures in a directory subtree (`photos/`) on his PC. He put unsorted images in the `incoming/` directory. When Bob got a camera phone, he created a synchronization link from its `Images` directory, in which the camera phone stores pictures, to the `incoming/` directory. This lets him synchronize new pictures to the PC over the network.

Later Bob got a laptop, and also decided he wanted to store his photos in a more durable location. For this purpose, he rented properly backed up storage space from an Internet Service Provider (ISP), created a synchronization link to the space, and regularly synchronizes the photos to this space, thus ensuring they are backed up. To be able to view and modify the photos on the laptop, he also creates a synchronization link from its `photos/` folder to the PC.

Our concurrency model is optimistic [15], which is well suited for the mobile environment [8]. Optimistic concurrency does, however, introduce the need for data reconciliation. In addressing this need, we looked beyond the intrinsic requirements of the file synchronizer, towards a more general goal of providing a generic reconciliation framework for applications aware of the ongoing file sharing. The chosen design provides reconciliation services for XML data by utilizing the XML three-way merging algorithm developed by us in [9].

We built the synchronization protocol using HTTP requests initiated by the mobile client. To minimize the impact of the high latency of current cellular data networks, we designed the protocol to make as few requests as possible. In particular, we batch object requests, so that a complete file

system is typically synchronized in only two HTTP requests. To support varying pricing and hot-spots, data synchronization consists of two stages of network-intensive discrete runs, where the first stage synchronizes file metadata (i.e., the layout of the directory tree) and the second stage synchronizes file content. We contrast this to systems that impose a continuous, typically lighter, load on the network [13, 7, 11].

4. A SEARCH USE CASE

Researcher Ralph works at HIIT. During the writing of a paper, he often gathers a lot of documents in one place. Considering himself an efficient worker, he sometimes reads papers while commuting. Ralph stays on the leading edge of current technology, and prefers not to carry extra papers, so he reads them on his Nokia[8] 9500, a mobile device with a display well suited for reading. Let us say Ralph wanted to read a paper that mentions his name. There are about twenty PDF files in his documents directory. Downloading all of them to his 9500 over the cellular network would be expensive and take a lot of time. Luckily, his 9500 has a synchronization link to the directory on the desktop machine, which is indexed by Dessy. Ralph synchronizes the metadata of the PDF files in a fraction of the time required for downloading them all. He then queries for his name with Dessy, and gets the name of the one document containing it. He then presses the "synchronize results" button, and his 9500 retrieves the file. He can then start reading it, knowing it is the correct paper.

5. DESSY ARCHITECTURE

In the example use case, file metadata plays a major part. Metadata used in Dessy consists of (property, value) pairs. A property is a lowercase text string that ends with a colon. Adding a colon to the end of properties simplifies implementation and serves as a clear indicator whether the virtual directory is a property virtual directory or a value virtual directory. For example, property virtual directories `author:` and `text:` are used. A value of a property, and is a lowercase text string, such as `bob` or `disambiguation`.

Figure 3 shows an example file with its Dessy metadata. The metadata is obtained when Dessy indexes the file. For example, the Email indexing helper finds the `subject:`, `from:` and `to:` fields from an email file, and collects and stems its text words to obtain the values of the `text:` property. Similarly, the modification date helper records the week, date, and full date and time when the file was last modified.

Dessy metadata is stored in an index. Dessy implementations on different devices can use different index storage methods. The synchronization of metadata is independent of the index storage method.

Figure 4 shows synchronization of Dessy metadata. When synchronizing changed metadata, the client reads the metadata of files from the index and sends it to the server as plain text words. On the server side, received metadata is compared with the last version of metadata for the file. If the client's version is newer, that metadata is used.

If only the server has modifications, the client retrieves meta-

[8]Nokia website. `http://www.nokia.com`

[Example file: excerpt from
enron/quenet-j/inbox/10.eml,
last modified 2004-02-04 04:27]

```
From: cory.willis@enron.com
To: stephanie.sever@enron.com
Subject:

I will be managing Joe Quenet's stack
and will need his password reset.
Also, I request trader access
to manage my own products.

Thank you,

Cory Willis
East Power Trading
```

[Example Dessy Metadata for the file]

```
from: cory.willis@enron.com
subject:
to: stephanie.sever@enron.com
text: password, access, product,
servic, east, thank, manag, joe,
trader, reset, stack, need, trade,
cori, willi, quenet, request
mod-date: week_6, 2004-02-04, 2004-02-04-04:27:29
name: 10
ext: eml
```

Figure 3: An example file and its metadata.

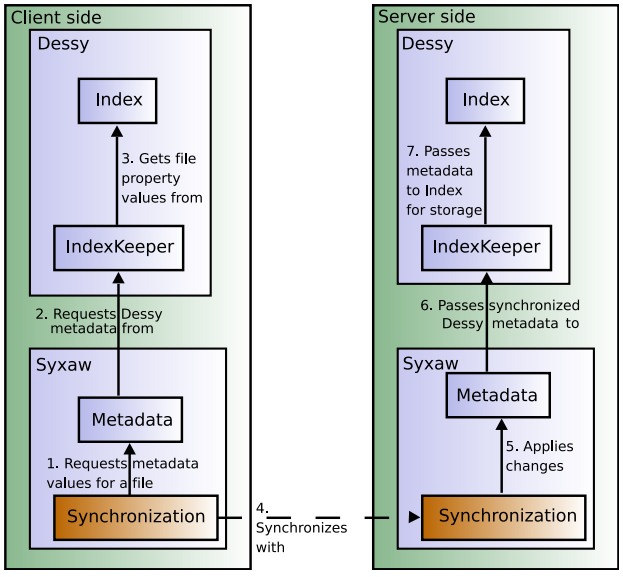

Figure 4: A sequence diagram of synchronization of Dessy metadata.

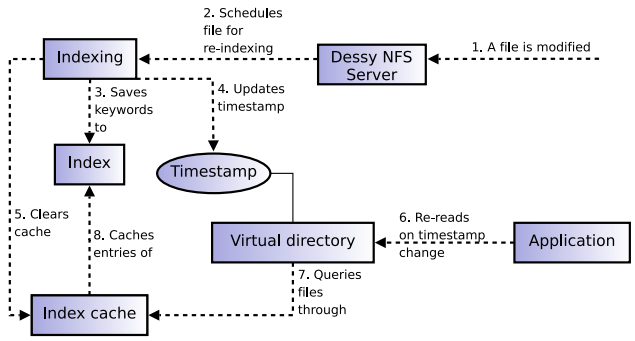

Figure 5: Sequence diagram from file modification to the updating of the application's view.

data from the server and stores it in its Index implementation.

In the case that both ends have modifications, instead of using either version, file content is synchronized, and then the file is re-indexed on the client side. The resulting metadata is synchronized.

Dessy search results are shown in virtual directories. The search system is browseable by regular file managers on desktop systems and through a Java API on mobile devices. Synchronization of files contained in a virtual directory is straightforward. A set of results in a virtual directory is a set of Syxaw UIDs. UIDs are independent from file paths, so files in a virtual directory are identified the same way as files in physical directories. To synchronize the files, Dessy just tells Syxaw to synchronize the set of UIDs.

In the indexing process, gathering of file properties is done by indexing helpers. The MIME [1] type of each file is determined and indexing helpers registered for that type are used to gather properties. The property names are used as the names of property virtual directories and their values as value virtual directories when finding files. For example, the PDF file helper produces the property `author:` among others. To find files with the author `bob`, we look in the virtual directory called `author:/bob`. New indexing helpers can be added by implementing the `IndexingHelper` interface.

Figure 5 shows the sequence of events from modifying a file to the view of the application being updated. File modifications are detected by the `Dessy NFS` server. In the mobile version, the NFS server is replaced by a Java API. Then the server tells `Indexing` to re-index the modified file. Files scheduled for indexing are indexed after a period of one second without file modifications. The `Index` updates the time stamp of the virtual directories that correspond to the properties of the file, and `Indexing` clears old results from the `Index Cache`. The next time a corresponding virtual directory is read, the time stamp has been changed so that file managers can get new entries instead of their cached information. The read request is passed to Dessy and new results are retrieved from the `Index`.

The user can add or remove context tags and metadata using the symbolic link facility of the Dessy NFS server. If a

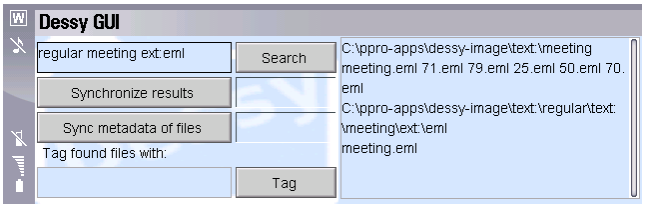

Figure 6: A screenshot of the Dessy mobile version GUI.

symbolic link pointing, for example, to the file `william-k/123.eml` is created in the virtual directory `tag:/board_meeting`, the file `william-k/123.eml` is added the value `board_meeting` for the property `tag` and can then be found by the query `tag:board_meeting`. Similarly, if a user deletes the file `tag:/board_meeting/123.eml`, the original file will not be deleted, but the tag `tag:board_meeting` is removed. Handling tags is done by the `tag()` and `untag()` operations of the API on mobile platforms.

To speed up queries, a query is not evaluated when entering a virtual directory, only when a directory listing is read.

6. IMPLEMENTATION

The current implementation of the Dessy prototype runs on Personal Profile Java (we use a Nokia 9500) and Linux and Windows desktops. Figure 6 shows a screenshot of the mobile version graphical user interface. The GUI allows finding files, listing choices for property values, synchronizing the metadata of all files, synchronizing the data of query result files, and tagging files with custom properties. The names of query result files are shown on the right side, with the full virtual directory path of the last few queries. Figure 7 shows a Nokia 9500 running the Dessy mobile version.

The Dessy prototype indexes these file MIME types:

- text/plain - Plain text files are currently found by words contained in them.
- text/x-mail - Email files are searched for subject, from and to fields, and their text is searched for words. Email attachments of non-text types are ignored. Emails are identified by text content and the .eml suffix.
- application/pdf - PDF files' author, creator, keywords, number of pages, subject and title fields are indexed. A PDF file's extracted text is used for the `text:` property.
- all - The modification dates of all files are checked and a virtual directory called `mod-date:` allows to search for them. Also, file extensions and names are indexed. Note that custom tags can be added to the index for files of any type.

6.1 Indexing Performance

Figure 8 shows the indexing performance of Dessy using a simple file-based index and a MySQL[9] index, without lim-

[9]The MySQL open-source database. `http://www.mysql.com/`

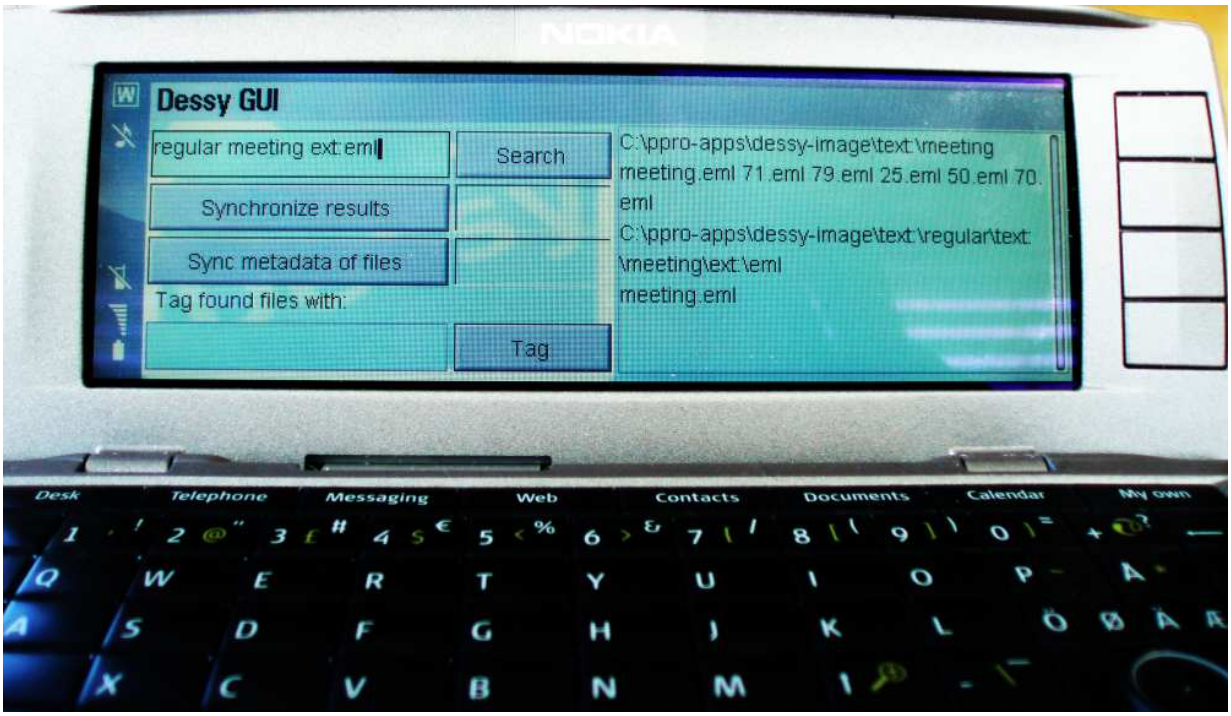

Figure 7: A photo of a 9500 running the Dessy mobile version.

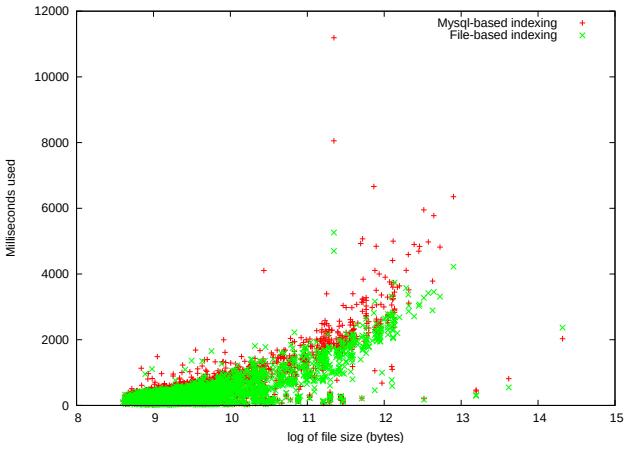

Figure 8: Indexing performance.

iting the number of words stored. Files are represented by points; the x-axis shows the logarithm of the file length in bytes, and the y-axis displays the number of milliseconds it took to index the file. In the two indexing methods, as new words are met, they are added into a word dictionary and given a unique number. This way the words of a file can be stored efficiently as numbers. The high variability might be due to the construction of the word dictionary.

The results suggest that both indexing methods scale well. The file-based index seems to have performance comparable to that of the Mysql-based index, if not better. On mobile devices, Dessy uses the file-based indexing, since it is more lightweight and fully implemented in Java.

The experiments were carried out by indexing a subset of the Enron Email Data set[10]. The subset contained approximately 300 000 files, and amounted to a total of 1.4 GB. The size of the largest file was 1.6 MB, while the smallest files were around 4 KB. The machine running the experiments was a PC with an AMD Athlon 64 X2 Dual Core 4200+ processor running GNU/Linux.

6.2 Querying performance

This section shows query performance measurements which were obtained from the mobile version of Dessy running on a Nokia 9500. In this case, the index of Dessy is not compressed and inverted lists of matching documents for a given query word are stored in the file-based index. This makes

[10]Publicly available from `http://www.cs.cmu.edu/~enron/`. Contact the authors for information on reconstructing the subset.

Properties	Results	Time (ms)	σ
1	151,30	2649,37	2977,28
2	17,44	532,13	544,08
3	4,71	308,38	222,50
4	3,00	311,74	206,19
5	2,40	313,97	220,98
6	2,40	315,67	194,00
7	2,50	335,43	199,46
8	2,46	369,00	193,70
9	2,00	497,14	268,87
10	1,17	515,67	233,82

Table 1: Query performance

query evaluation faster for devices with limited CPU power, such as the 9500.

Table 1 shows the measurement results. The first column displays the number of (property, value) pairs used. The second column shows the average number of results returned for the queries, the third shows the average time taken in milliseconds, while the last shows the standard deviation of the time. The measurements were taken from a small subset of the Enron data set, with a total of approximately 14 000 files and a size of 66 MB. The measurements were obtained in the following way: first, 100 files of the data set were chosen at random. The choice of a file depended on its size: Larger files were given higher probability than small files. Values of all properties of the file were used as queries of one (property, value) pair (the first row in the table). Multiple values were obtained for the `text:` property only (rows after the first in the table), since most other properties, such as `author:`, `subject:`, `mod-date:` and `ext:`, have just one value for a given document and are highly document type - specific. `text:` has multiple values for documents with text content and is available for all files that contain a form of text in the dataset. Queries of 2 to 10 values were constructed using the first 2 `text:` values in a document, then the first 3, and so on, finally querying with the first 10 values. All queries that returned results were included in the measurements.

The high standard deviation in one-property queries is probably due to the high variability in the number of results for queries of one property. For example, the query for the modification date of a file would return all the files, since the dataset was created on the same day.

The measurements suggest that the increase in queried pairs does not significantly increase the time taken by the query. This might be because the queries were Boolean **and** queries, and with more query terms, the result set diminishes rapidly. This makes the combining of further query results faster. Also, the caching of previous results made subsequent queries with the same pairs faster. For example, a query with the first 2 pairs of a file was executed before the one with the first 3, so the results of the first 2 pairs were already cached.

The Dessy prototype will be released in 2008 under the MIT License. Contact the authors [11] for details.

[11] `http://www.hiit.fi/fi/fc`

7. CONCLUSIONS AND FUTURE WORK

This paper presented Dessy, a flexible desktop search system for mobile devices. Dessy separates metadata extraction, exchange and use. It has separate synchronization support for file metadata and data. It uses virtual directories to find files. Dessy is extensible through changeable indexing helpers, indexes and query plugins. Our work on the Dessy prototype supports the conclusion that desktop search is feasible on mobile devices.

We will continue to explore desktop search on mobile platforms through improving Dessy. As the emphasis of Dessy has not been on efficient indexing, this is something to concentrate on next. Index compression should be implemented. Implementing an NFS-like system for mobile devices and a Dessy-aware file manager are among future plans for the Dessy project.

8. REFERENCES

[1] N. Borenstein and N. Freed. *RFC 1521: MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies*. Internet Engineering Task Force, Sept. 1993.

[2] E. Cutrell, S. T. Dumais, and J. Teevan. Searching to eliminate personal information management. *Communications of the ACM*, 49(1):58–64, Jan. 2006.

[3] E. Cutrell, D. Robbins, S. Dumais, and R. Sarin. Fast,flexible filtering with phlat. In R. E. Grinter, T. Rodden, P. M. Aoki, E. Cutrell, R. Jeffries, and G. M. Olson, editors, *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 261–270. ACM Press, Apr. 2006.

[4] D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. James W. O'Toole. Semantic file systems. In *SOSP '91: Proceedings of the thirteenth ACM symposium on Operating systems principles*, pages 16–25. ACM Press, 1991.

[5] D. Harman, R. Baeza-Yates, E. Fox, and W. Lee. Inverted files. In W. B. Frakes and R. Baeza-Yates, editors, *Information Retrieval: Data Structures and Algorithms*, pages 28–43. Prentice Hall, Upper Saddle River, New Jersey, USA, 1992.

[6] C. K. Hess and R. H. Campbell. An application of a context-aware file system. In G. Cockton and P. Korhonen, editors, *CHI '03 extended abstracts on Human factors in computing systems*, pages 339–352, Apr. 2003.

[7] J. Kubiatowicz et al. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems*, Nov. 2000.

[8] T. Lindholm. XML three-way merge as a reconciliation engine for mobile data. In *Third ACM International Workshop on Data Engineering for Wireless and Mobile Access*, pages 93–97, Sept. 2003.

[9] T. Lindholm. A three-way merge for XML documents. In E. V. Munson and J.-Y. Vion-Dury, editors, *ACM Symposium on Document Engineering*, pages 1–10, Milwaukee, Wisconsin, USA, Oct. 2004. ACM Press.

[10] C. P. Mah and R. J. D'Amore. Complete statistical

indexing of text by overlapping word fragments. *ACM SIGIR Forum*, 17(3):6–16, Jan. 1983.

[11] L. Mummert, M. Ebling, and M. Satyanarayanan. Exploiting weak connectivity for mobile file access. In *Proceedings of the Fifteenth ACM Symposium on Operating System Principles*, pages 143–155, Dec. 1995.

[12] B. Nowicki. *RFC 1094: NFS Network File System Protocol Specification*. Internet Engineering Task Force, Mar. 1989.

[13] K. Petersen, M. Spreitzer, D. Terry, M. Theimer, and A. J. Demers. Flexible update propagation for weakly consistent replication. In *Proceedings of the sixteenth ACM Symposium on Operating Systems Principles*, pages 288–301, Sept. 1997.

[14] M. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, July 1980.

[15] Y. Saito and M. Shapiro. Optimistic replication. *ACM Computing Surveys*, 37(1):42–81, 2005.

[16] C. A. N. Soules and G. R. Ganger. Connections: using context to enhance file search. In A. Herbert and K. P. Birman, editors, *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 119–132, Oct. 2005.

[17] Sun Microsystems Inc., Santa Clara, California, USA. *JSR 62: Personal Profile Specification*, Sept. 2002. [JCP Final].

[18] S. Tarkoma, J. Kangasharju, T. Lindholm, and K. Raatikainen. Fuego: Experiences with mobile data communication and synchronization. In *17th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, Sept. 2006.

[19] J. Zobel, A. Moffat, and K. Ramamohanarao. Inverted files versus signature files for text indexing. *ACM Transactions on Database Systems*, 23(4):453–490, Dec. 1998.