A Method for Searching Keyword-lacking Files Based on Interfile Relationships

Tetsutaro WATANABE[1], Takashi KOBAYASHI[2], and Haruo YOKOTA[3]

[1] Tokyo Institute of Technology, Department of Computer Science
tetsu@de.cs.titech.ac.jp
[2] Nagoya University, Graduate School of Information Science
tkobaya@is.nagoya-u.ac.jp
[3] Tokyo Institute of Technology, Global Scientific Information, and Computing Center
yokota@cs.titech.ac.jp

Abstract. Current information technologies require file systems to contain so many files that searching for desired files is a major problem. To address this problem, desktop search tools using full-text search techniques have been developed. However, those files lacking any given keywords, such as picture files and the source data of experiments, cannot be found by tools based on full-text searches, even if they are related to the keywords. It is even harder to find files located in different directories from the files that include the keywords. In this paper, we propose a method for searching for files that lack keywords but do have an association with them. The proposed method derives relationship information from file access logs in the file server, based on the concept that those files opened by a user in a particular time period are related. We have implemented the proposed method, and evaluated its effectiveness by experiment. The evaluation results indicate that the proposed method is capable of searching keyword-lacking files and has superior precision and recall compared with full-text and directory-search methods.

1 Introduction

Advances in information technologies have led to many types of multimedia data, including figures, images, sounds, and videos, being stored as files in computer systems alongside conventional textual material. Moreover, the recent price drop for magnetic disk drives [1] has accelerated the explosive increase in the number of files within typical file systems [2].

Most current operating systems adopt hierarchical directories to manage files. A very large number of files makes the structure of such a directory very extensive and complex. Therefore, it is very hard to classify the many files into appropriate directories. Even if all the files are classified appropriately, it is still difficult to find a desired file located at a deep node in the directory tree.

To address this problem, several desktop search tools using full-text search techniques have been developed, such as Google Desktop, Windows Desktop Search, Spotlight, Namazu [3], and Hyper Estraier [4]. However, their target is restricted to text-based files such as Microsoft Word documents, PDFs, and emails.

Other types of files, such as picture files, image files, and source data files for experiments and field work, cannot be found by these full-text search tools because they lack search keywords. Even for text-based files, they cannot be found if they do not include directly related keywords. It becomes even harder if these files are located in different directories from the files that contain the keywords.

To address the demand for searching for these keyword-lacking files, there has been much research aiming to append metadata to files [5]. However, it is practically impossible to assign "perfect" metadata to a large number of files. On the other hand, Google Image Search [7] is capable of searching image files in web sites associated with the keywords by using the reference information in HTML files within the site. However, this method does not directly apply to files in the file system because they rarely contain information relevant to the sources of the included objects.

In this paper, to provide the function of searching for keyword-lacking files that match with given keywords, we focus on the relationship between files that have been frequently accessed at about the same time. For example, when a user of a computer system edits the file for a research paper containing conceptual figures and graphs of experiments, the user frequently opens files for the figures and data sources of the experiments at the same time. Of course, other files that do not directly relate to the paper, such as reports for lectures and emails to friends, are also opened simultaneously. However, the frequency of opening related files at the same time should be higher than the frequency for nonrelated files.

To achieve this function, we propose a method for mining the file access logs in a file server to find interfile relationships. We have implemented the proposed method using access logs for Samba, using the name FRIDAL (File Retrieval by Interfile relationships Derived from Access Logs). To evaluate the method, we compared the search results for FRIDAL with a full-text search method, directory search methods, and a related method used in Connections [6]. The evaluation results, using actual file access logs for testers indicated that the proposed method is capable of searching the keyword-lacking files that cannot be found by other methods, and that it has superior precision and recall compared with the other approaches.

The remainder of this paper is organized as follows. First, we review related work in Section 2. Then we propose a method for mining the file access logs in Section 3, and describe the implementation of FRIDAL in Section 4. In Section 5, we compare the search results for FRIDAL with the other methods to evaluate FRIDAL. We conclude the paper and discuss future work in Section 6.

2 Related Work

The Semantic File System [9] enables a file to have a number of attributes, instead of placing it in a directory, to help share its information. The method proposed in the article [10] uses the Resource Description Framework to describe the attributes of files, and applies semantic web technology to manage files by maintaining their consistency. Our method and these other approaches have the same goal: solving the problem of the hierarchical directory. However, our approach uses file access logs instead of semantics.

In time-machine computing [8], all files are put on the desktop and gradually disappear over time. If a user inputs a date to the system, the user can see the desktop at the appointed date. The system also supports keyword searches by using an "electronic post-it" note created by the user. Their time-centric approach differs from our keyword-centric approach.

OreDesk [11] derives user-active records from OS event logs and installed plug-ins. Then it calculates user-focused degrees for web pages and files (called *Datas*) based on the active records, and also calculates relationships between *Datas*. It provides a search function for related files, given a Data name, and also provides a viewer for *Datas* and relationships. Whereas OreDesk changes the user's environment to derive the active records, our method does not change it because we use the access logs for the file server. Also, whereas OreDesk uses the *Data* name for searching, our method uses keywords. Furthermore, OreDesk takes account of the start time of *Data* only, using it to calculate the relationships, whereas our method takes account of total time, number, and separation of co-occurrences. Our method differs from OreDesk, therefore, in its ability to discriminate between relationships.

Connections [6] obtains system calls to files, such as read() and write(), and constructs a directed graph comprising the files (as its nodes) and the relationships (as its edges). Following a search request, it performs a context-based search, then searches for related files in the result of the context-based search, by tracing the directed graph. The aim of Connections is the same as that of our method. However, whereas Connections aims to derive the reference/referenced relationships via system calls, our method derives information of file usage via open-file/close-file information in access logs. It also differs in calculating the relationships of files in the search results. Since the logs of Samba are obtained without any modifications for the target system, our method is easier to be implemented than the method based on system-calls' logs. Moreover, overhead of obtaining logs in a file server is very small compared with the system calls.

We will describe the calculations of Connections because we will be comparing our method to the calculations of Connections in Section 5. In [6], various calculation methods are proposed. We will explain the most efficient of these, as reported by the authors. First, Connections makes an edge whose weight is 1 from the read file to the written file in a time window specified by the constant *TimeWindows*. If the edge already exists, its weight is incremented by 1. In the search phase, it first performs a context-based search. Let w_{n_0} be the point of the files, as scored by the context-based search. If the file is not to be included in the result, then $w_{n_0} = 0$. Let E_m be the set of edges going to the node m, where $e_{nm} \in E_m$ is the weight of the edge from n to m divided by the sum of the weights of the edges leaving node n. If $e_{nm} <$*Weight Cutoff*, then the edge is ignored. The point of node w_n is calculated as follows when the number of repeats is P.

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

$$w_n = \sum_{i=0}^{P} w_{n_i} \tag{2}$$

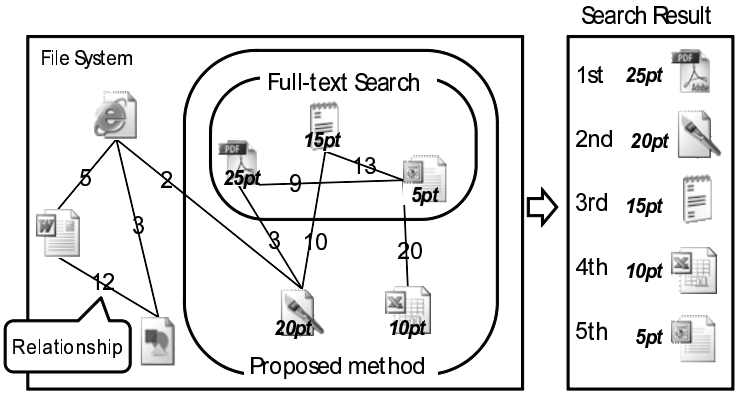

Fig. 1. Overview of the proposed method

3 Proposed Method

In this paper, we propose a method capable of searching keyword-lacking files using keywords, which has both a preparing and a searching phase.

Preparing Phase: Derive information about files usage from file access logs, and calculate interfile relationships.

Searching Phase: Obtain keywords, perform a full-text search using the keywords, calculate the point of files related to the results of the full-text search based on the relationships, and show the files ordered by point (Fig. 1).

The proposed method assumes that the files are stored in the file server, and are used on personal desktops. In this paper, we assume that the files are not copied, moved, or renamed.

3.1 Preparing Phase

We focus on the relationship between files that are frequently used at the same time. When a user edits a file during a task, the user often opens related files to refer to or to edit. In this case, the same files are used frequently every time the task is performed. Therefore, we suppose that files that are frequently used at the same time are related.

In our research, to identify such files, we derive information about file usage from file access logs.

Derive Approximate File Use Duration First, we obtain the "open-file" and "close-file" for all files for all users from the file access logs for a file server. Next, we define the File Use Duration "FUD" as the time between open-file and close-file. If a close-file is not recorded, we remove the corresponding open-file. For access logs in Samba, there

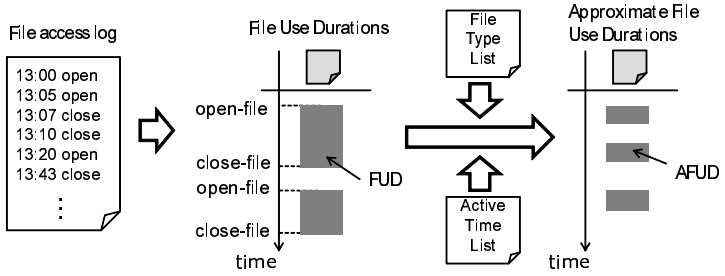

Fig. 2. Derive approximate file use duration

are two problems that mean that the actual duration of file use differs from the FUD. We propose a method for solving these problems to obtain an Approximate File Use Duration "AFUD", as shown in Fig2). Finally, we filter out any AFUD that has a time span less than T_f.

We now describe the two access log problems and how to solve them.

Problem A: A user sometimes leaves his or her seat with files open.

Problem B: There are two types of applications. One application locks a file when it opens file, whereas the other copies it to memory and does not lock it. The latter type causes a difference between the FUD and the actual duration.

To solve these problems, we prepare the following two pieces of information by analyzing the access logs.

Active Time List: This list comprises information about the existence of logs in all time windows T_w. "Logs exist" means a user was active at that time.

File Type List: This is a list of file types whose average FUD is less than T_l. An item in this list is possibly used by an application that unlocks files.

For Problem A, we mask the FUD with the Active Time List. Then, if the span of active time is greater than T_s, we delete the corresponding FUD, as shown in Fig. 3, because we think the user forgot to close the file and went home. For Problem B, for the file types in the File Type List, we extend the FUD from the first FUD to the last FUD for each item in the Active Time List, as shown in Fig. 4, because we think the first FUD is "open" and the last FUD is "save and close".

Calculate Interfile Relationships We assume that strongly related files are used at the same time when executing the same task. We calculate the interfile relationships by the AFUDs described in the previous section. To achieve this, we introduce four "relationship elements", where the term "CO" (meaning co-occurrence) is defined as the overlap of two AFUDs.

T : Total time of COs

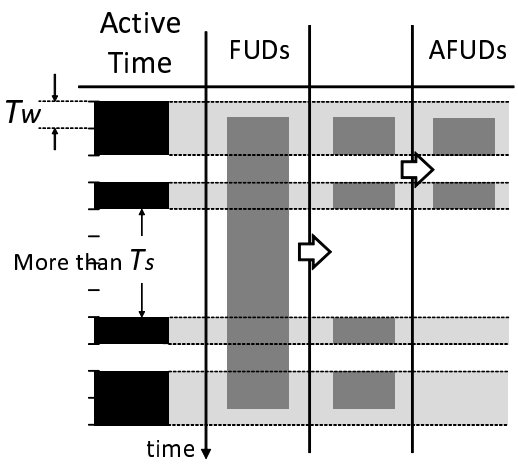

Fig. 3. Managing Problem A

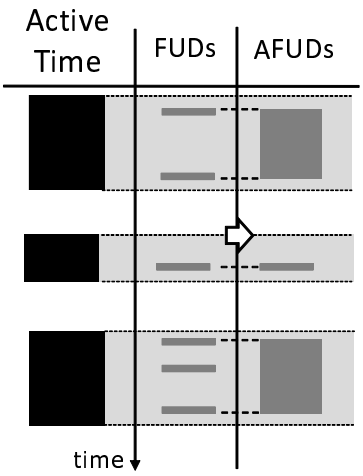

Fig. 4. Managing Problem B

C : Number of COs
D : Total time of the time span between COs
P : Similarity of the timings of the open-file operations

By using these four relationship elements and four parameters, α, β, γ, and δ, we define the interfile relationship R as follows.

$$R = T^{\alpha} \cdot C^{\beta} \cdot D^{\gamma} \cdot P^{\delta} \tag{3}$$

We now describe how to calculate relationship elements, by reference to Fig. 5. Let $\{z_1, z_2, \cdots, z_n\}$ be n COs of two files, let $\{t_1, t_2, \cdots, t_n\}$ be their times, let $\{d_{12}, d_{23}, \cdots, d_{(n-1)n}\}$ be the time spans between each CO, and let $\{p_1, p_2, \cdots, p_n\}$ be the difference in the start time of two AFUDs. Each relationship element is calculated as follows.

$$T = \sum_{i=1}^{n} t_i \qquad D = \begin{cases} 1 & n = 0 \\ \sum_{i=1}^{n-1} d_{i(i+1)} & otherwise \end{cases}$$

$$C = n \qquad P = \begin{cases} 1 & \forall i \;\; p_i = 0 \\ (\sum_{i=1}^{n} p_i)^{-1} & otherwise \end{cases}$$

3.2 Searching Phase

In the searching phase, we first run the full-text search using the input keywords. Then we score the file point by using the TF–IDF (Term Frequency–Inverse Document Frequency) and interfile relationships for all files related to the files found in the full-text search.

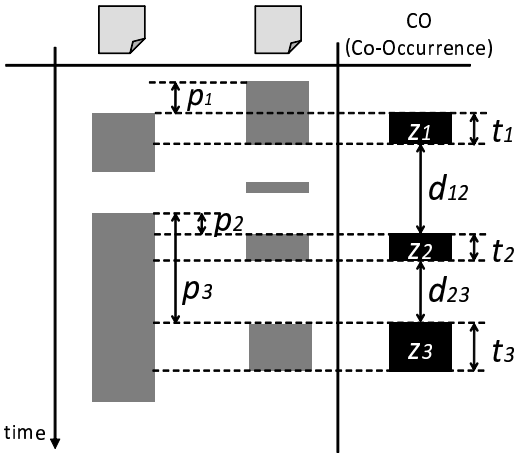

Fig. 5. The values used in the calculation of relationship elements

The file point is calculated as a high value when the file has a strong relationship with a high TF–IDF file. Let U be the set of all files, and let $R(f_i, f_j)$ be the interfile relationship between file f_i and file f_j.

For the case of a single input keyword, we run a full-text search using the keyword k. Let $F(k)$ be the resulting set of files. Then, for each file $f \in F(k)$, we calculate the TF–IDF $S_T(f, k)$ of f using k. Note that the TF–IDF is zero if a file is not in the full-text search result. Next, we normalize $R(f_i, f_j)$ to $\hat{R}(f_i, f_j, k)$ under keyword k as follows.

$$\hat{R}(f_i, f_j, k) = \frac{\log[R(f_i, f_j)]}{\log[\max_{\substack{f \in F(k) \\ f' \in U}} \{R(f, f')\}]} \tag{4}$$

In this normalization, the highest possible relationship degree becomes 1. Then the added point $S_R(f_i, f_j, k)$ to file f_i, based on the relationship with f_j, is calculated as follows.

$$S_R(f_i, f_j, k) = \hat{R}(f_i, f_j, k) \cdot S_T(f_j, k) \tag{5}$$

Finally, the point $S(f_i, k)$ of file f_i is calculated as follows.

$$S(f_i, k) = S_T(f_i, k) + \sum_{f \in F(k)} S_R(f_i, f, k) \tag{6}$$

We can explain this by using a concrete example. Refer to the left-hand side of Fig. 6. After searching using the keyword k, f_1, f_2, f_4 are found to include the keyword. The TF–IDFs for these files are $S_T(f_1, k) = 10$, $S_T(f_2, k) = 20$, $S_T(f_3, k) = 0$, and $S_T(f_4, k) = 30$. The normalized relationships are $\hat{R}(f_1, f_2, k) = 0.5$, $\hat{R}(f_2, f_3, k) = 0.75$, and $\hat{R}(f_3, f_4, k) = 1$. We calculate the point $S(f_3, k)$ of file f_3 as follows. Refer to the

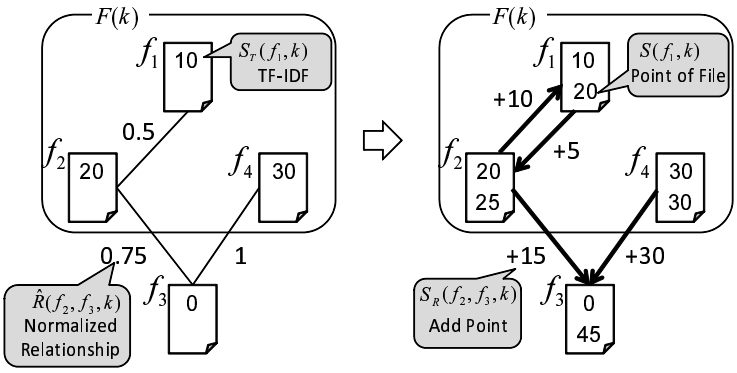

Fig. 6. Calculation of the point of files

right-hand side of Fig. 6. The added point $S_R(f_3, f_2, k)$, based on the relationship with f_2, is $\hat{R}(f_3, f_2, k) \cdot S_T(f_2, k) = 20 \cdot 0.75 = 15$. The added point $S_R(f_3, f_4, k)$, based on the relationship with f_4, is $\hat{R}(f_3, f_4, k) \cdot S_T(f_4, k) = 30 \cdot 1 = 30$. Therefore, $S(f_3, k) = 45$, namely the sum of the TF–IDF and both added point components.

For cases involving multiple keywords, the point is calculated as a sum of the point based on each keyword. If K is a set of keywords, the point $S(f_i, K)$ of file f_i is calculated as follows.

$$S(f_i, K) = \sum_{k \in K} S(f_i, k) \tag{7}$$

4 Implementation

We implemented the proposed method as an experimental system called FRIDAL.

Figure 7 shows the architecture of FRIDAL. FRIDAL comprises four components, namely the Web UI, the RDBMS, the Full-text search engine, and the Controller. The Controller implements two main functions of our proposed method. First, it mines the interfile relationships from the access logs of Samba, which is widely used as a CIFS (Common Internet File Services) file server. Second, it performs the file point calculations by using interfile relationships and a full-text search engine.

In an initial setup, administrators provide the access log files of Samba, and FRIDAL starts the mining process with the logs as described in Section 3 and stores relationships to the RDB. When a user retrieves files with FRIDAL, the user first supplies the keywords via the Web UI. The Controller passes the keywords to Hyper Estraier to obtain the results of a full-text search. Next, the Controller searches related files with interfile relationships from the RDB and calculates the file points. Finally, the Controller returns the list of files to the user via the Web UI. The results comprise the point, the file path, and basis of the point, such as the TF–IDF or the added points.

We implemented FRIDAL as a web application written in Java. We used Tomcat 5.5.9 as the web application container, Oracle Database 10g as the RDBMS and Hyper

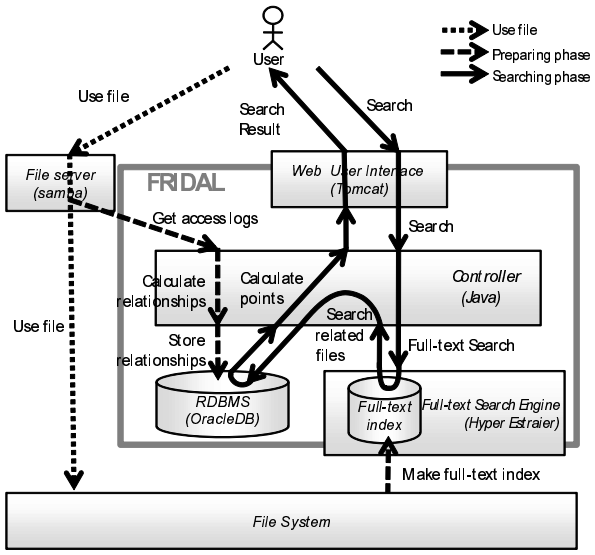

Fig. 7. The architecture and behavior of FRIDAL

Estraier 1.4.13[4] as the Full-text search engine. Since Hyper Estraier uses an N-gram index, we can avoid Japanese language morphological analysis problems, and obtain a perfect recall ratio by the N-gram method.

FRIDAL derives the file usage information from the access logs of Samba 2.2.3a, configured at debug level 2. Because we adopted Samba as the file system access logger, we can install FRIDAL with only slight modifications to the existing environment. In addition, we can use historic access log files even before installation, because we adopt the default format for Samba.

5 Evaluation

We evaluated FRIDAL in two different experiments. In Experiment 1, we investigate FRIDAL's ability to find keyword-lacking files. In Experiment 2, we compare FRIDAL to other search methods with respect to precision and recall.

5.1 Experimental Environment

We use access logs recording the file access of three testers, A, B, and C. Table 1 shows information about the logs, number of files appearing in the logs, and the client OS for each tester. The file extensions targeted for analysis are bib, doc, docx, gif, htm, html, jpg, mpg, mpeg, ppt, pptx, pdf, txt, tex, xls, and xlsx. The constants described in Section 3.1 are $T_f = 1[m]$, $T_w = 30[m]$, $T_s = 5[h]$,$T_l = 10[s]$. We specify the parameters $(\alpha, \beta, \gamma, \delta) = (1, 1, 0.5, 0.5)$, used in Equation (3), by performing a preparatory experiment.

Table 1. Information about the testers

	Start date of recording	End date of recording	#Log files	Total log lines	#Files	#Relationships	Client OS
Tester A	2007/01/22	2007/12/07	49	4873703	1100	17472	Windows XP
Tester B	2007/01/22	2007/12/07	53	4323090	713	5692	Windows XP
Tester C	2007/01/18	2007/12/07	71	7863206	793	5236	Windows Vista

Table 2. The search requests used in Experiment 1

Search request	Desired file
Q_1	The paper of tester A
Q_2	The source of the image files in the paper of tester A
Q_3	The eight data files for the paper of tester A
Q_4	The paper of tester C
Q_5	The source of the image files in the paper of tester C
Q_6	The data file for the paper of tester C

5.2 Experiment 1

Experimental Method In this experiment, we have a tester find specific files in another user's home directory. We examine the number of times a query is input and files are checked when using either FRIDAL or Full-text search. Here, "query" means keywords plus a file extension to narrow down the search. For search requests, we take account of the fact that the testers are students and use the six requests listed in Table 2.

Experimental Results Table 3 gives the experimental results when using the requests in Table 2. In Table 3, "#Queries" means the number of times the query was made, "#Check files" means the number of the files checked until the tester either finds the file or gives up the search, and "Can find" means that if the tester can find the file then "T" is marked, otherwise "F" is marked. If there are multiple relevant files, the ratio of the number of found files to all files is given.

Discussion In Table 3, for Q_2, Q_3, and Q_5, Full-text search cannot find the files at all, whereas FRIDAL can. With this result, we can confirm that FRIDAL can find keyword-lacking files.

For Q_1, Q_4, Q_6, whereas both methods can find the file, the numbers of Queries and Check files of FRIDAL are less than those of Full-text search. Therefore, it is apparent that FRIDAL can search more efficiently than Full-text search in the case of searching data related to a paper. Note that, in Q_6 the tester found the data file by using Full-text search. This was because the data file included the keyword accidentally.

Table 3. The cost to search

	FRIDAL			Full-text search		
Search request	#Queries	#Check files	Can find	#Queries	#Check files	Can find
Q_1	2	1	T	2	15	T
Q_2	1	9	T	1	6	F
Q_3	1	4	3/8	1	0	0/8
Q_4	1	2	T	1	11	T
Q_5	1	2	T	1	14	F
Q_6	1	15	T	2	14	T

5.3 Experiment 2

Experimental Method First, we prepared relevant sets by selecting files related to keywords from all files in the target file system by testers. Table 4 shows the keywords actually used. Next, we retrieved by those keywords with following four methods.

- FRIDAL
- Full-text search
- Directory search
- Connections calculation

Then we calculate the averaged 11-points precision of the search results, and also calculate the precision and recall of the top 20 of the results. If the averaged 11-points precision is high at one recall, the search result is good because the result will include few mismatched files at that recall.

Directory search is a method that searches the directory that includes the full-text search result. We suppose this method is the usual action by users when files cannot be found by a full-text search. The results for directory search are ordered as: the 1st rank of the full-text search, all files in the same directory as the 1st rank of the full-text search, the 2nd rank of the full-text search, all files in the same directory as the 2nd rank of the full-text search (except for files already counted), and lower ranks similarly.

Connections calculation is a method that uses the calculation method of Connections[6], introduced in Section 2. However because we cannot obtain the actual file system calls, we use the read/write attribute for file access in the access logs instead of read()/write(). The constant parameters in Equations (1) and (2) are set to *Time Window*=30[s], $P = 2$, $\alpha = 0.25$, and *Weight Cutoff*=0.1[%], which are the optimal parameter values reported in [6].

Experimental Results Fig. 8 is the averaged 11-points precision/recall graph of the results for six keywords and Fig. 9 is the 11-points precision/recall graph of the results without the full-text search results. The precision at recall = 0 is the precision at the point when the first relevant file is found. Table 5 is the average of the precision and recall of the top 20 of the search results.

Table 4. Evaluated query keywords and Relevant files for Experiment 2

Keywords	Relevant files	#Relevant files
"Revocation for Encrypted Data Stored with Replica"	The files related to the paper written by tester A	203
"DO–VLEI"	The files related to the proposed method named "DO-VLEI" of tester B	206
"DO–VLEI" and "adss"	The files related to the proposed method named "DO-VLEI" of tester B, to the conference "ADSS2007"	106
"patent"	The files related to the patent of tester C	48
"Letters"	The files related to paper published in "DBSJ Letters" of tester C	54
"access log" and "file search"	The files related to the research of tester C	75

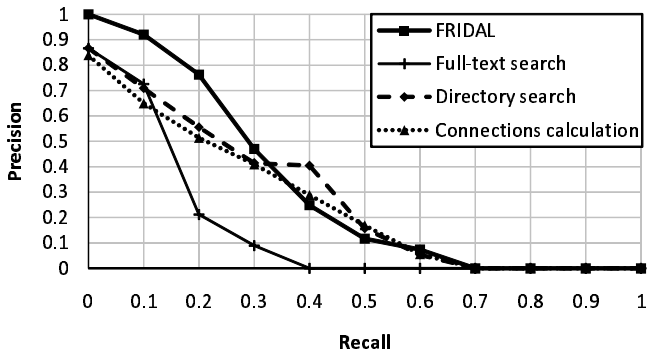

Fig. 8. The averaged 11-points precision/recall curve of the results

Discussion The precision of FRIDAL is higher than the other methods at low recalls, as shown in Fig. 8 and Fig. 9, and the recall and precision are also better, as seen from Table 5. These result show that FRIDAL can retrieve more relevant files than the others in the high orders of the results, and so we can find the desired files efficiently by using FRIDAL.

We discuss these results in detail in the following.

In Fig. 8, the precision of FRIDAL is slightly lower than for Directory search and Connection calculation at high recalls. Therefore, the results for FRIDAL include fewer relevant files in the low orders of the results. However, because the results of Directory search and Connections calculation include vast numbers of files, the user cannot check the files in the lower orders of the results. Thus, the utility of FRIDAL becomes apparent because the results for FRIDAL include more relevant files in the higher orders.

In Fig. 8, the reason why the precision of Director search, at 0.4, is much higher than that of FRIDAL is that the precision of Director search is higher than FRIDAL on one keyword. We investigated this, and found that files in the relevant set corresponding

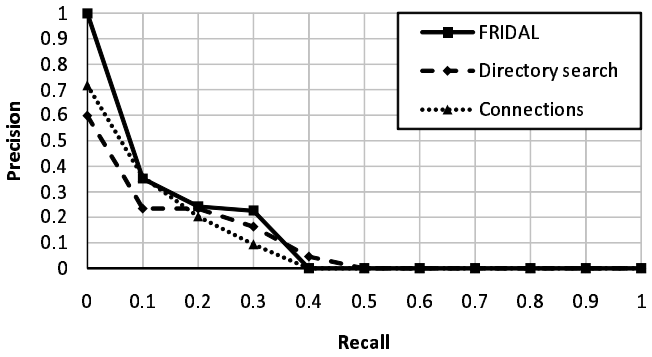

Fig. 9. The averaged 11-point precision/recall curve of the results without full-text search results

Table 5. The averaged precision and recall of the top 20

	Ave. precision	Ave. recall
FRIDAL	0.72	0.16
Full-text search	0.62	0.12
Directory search	0.62	0.13
Connections calculation	0.48	0.10

to the keyword are organized in just one directory. For organized files, searching with FRIDAL is less efficient than Directory search. Since we can predict that there are such partially organized files, we will improve FRIDAL to prepare for such cases in future work.

There are two causes for the values of Connections being lower. First, whereas Connections calculates the point of a file based on the number of reading/writing actions for the file, it does not take any account of the duration of file use. Second, it makes a directed edge from the read file to the written file, and calculates file point based on the graph. The directed edges cannot express interfile relationships adequately, because reading/writing files is not related to the reference/referenced file but based on the application that uses the file. However, in this experiment, we use the read/write attributes of Samba instead of the system calls that Connections actually uses. Therefore, it is not an accurate comparison.

In Table 5, all recalls are of low value because there are many relevant files which we cannot find relationship between files include keywords. As shown in Table 4, there are some correct sets that include more than 200 files. However, the number of searched files is 20. One of the reasons why we cannot find relationship is that a LaTeX file of a paper were divided into sub LaTeX files and each files had several renamed-variant for

version control. Since our method can not deal with file copy and rename, we cannot treat access to these file as file access to the same file. To track file system changing, such as file copy, rename, move, is an issue in the future.

Fig. 9 shows how many keyword-lacking files, which Full-text search cannot find, can be found by three methods. In these results, the precision of FRIDAL is also higher than the others. Therefore, we conclude we will be able to find keyword-lacking files efficiently by using FRIDAL.

6 Conclusion and Future Work

Traditional full-text searches cannot search keyword-lacking files, even if the files are related to the keywords. In this paper, we have proposed a method for searching keyword-lacking files, and we have described a system, FRIDAL, that implements our method. FRIDAL requires no modification in target systems, and has very small overhead to obtain the information of interfile relationships. In addition, we have performed experiments to evaluate FRIDAL via testers' answers. As a result, FRIDAL was found to be capable of searching for keyword-lacking files, with FRIDAL searching more efficiently than a full-text search. In addition, the 11-points precision and the recall/precision of the top 20 of FRIDAL's results were higher than those of the other methods.

In future work, we are considering the following four points. The first is to improve FRIDAL to be able to deal with the copying, moving, and renaming of files. Because there are files whose paths are recorded in the access logs but that do not exist in the file system, it is necessary to deal with the copying, moving, and renaming of files, by, for example, obtaining system calls to files. The second point concerns the various experiments. Because the experimental target involved the access logs of students, we only evaluated search requests such as those in Table 2. We should perform experiments using a wider variety of access logs. The third point is to improve the calculation of the interfile relationships. Although one relationship is a reference relationship, we did not take account of the direction of the relationship. Therefore, we should formulate the reference relationship by deriving the direction of the relationship from the access logs. The fourth point is to improve the method of displaying the search results. Currently, FRIDAL displays the files ordered by file point on a line. However, there are cases where the user would like a graphic structure comprising files and relationships. Therefore, we should consider ways of displaying the search result for all search requests.

Acknowledgment

Part of this research was supported by CREST of JST (Japan Science and Technology Agency), a Grant-in-Aid for Scientific Research on Priority Areas from MEXT of the Japanese Government (#19024028), and the 21st Century COE Program Framework for Systematization and Application of Large-Scale Knowledge Resources.

References

[1] B.Hayes: Terabyte territory. American Scientist,90(3):212-216, 2002.

[2] Nitin Agrawal, William J. Bolosky, John R. Douceur, Jacob R. Lorch: A Five-Year Study of File-System Metadata, 5th USENIX Conference on File and Storage Technologies(FAST'07), 2007.
[3] Namazu: a Full-Text search Engine http://www.namazu.org/index.html.en
[4] Hyper Estraier, http://hyperestraier.sourceforge.net/
[5] Ka-Ping Yee,Kirsten Swearingen,Kevin Li,Marti Hearst: Faceted Metadata for Image Search and Browsing, In Proc. of CHI ' 03, 401-408, 2003.
[6] Soules, C. A. N., and Ganger, G. R. Connections: using context to enhance file search. In Proceedings of the 20th ACM Symposium on Operating Systems Principles, pp.119-132. 2005.
[7] Google Image Search, http://images.google.com/
[8] Jun Rekimoto: TimeMachine Computing: A Timecentric Approach for the Information Environment, ACM UIST'99, 1999.
[9] D.K.Gifford, P.Jouvelot,M.A.Sheldoon, J.W.O'Toole,Jr: Semantic File Systems, 13th ACM Symposium on Operating Systems Principles, 1991.
[10] ISHIKAWA Kenichi, MORISHIMA Atsuyuki,TAJIMA Keishi: Development of a Semantic File System for Managing Large Document Spaces IEICE technical report. DE2006-115 pp.139-144, 2006. (in Japanese)
[11] Ryo Ohsawa, Kazunori Takashio, Hideyuki Tokuda: OreDesk: A Tool for Retrieving Data History Based on User Operations Eighth IEEE International Symposium on Multimedia (ISM'06) pp. 762-765, 2006.