ADEPT SEARCH: A DESKTOP SEARCH ENGINE

[1]Prof. Akshay Agrawal, [2]Venkatachalam.S, [3]Vicky Vijay Shirkar, [4]Gunjan Vikas Chaudhary

[1]Asst. Professor, [2,3]BE Student, [1,2,3]Comp. Engg. Dept, SSJCET, Asangaon, India.

[1]akshay1661@gmail.com, [2]heartbreakkid4454@gmail.com, [3]vshirkar077@gmail.com, [4]gunj143@gmail.com

Abstract: This system refers to this index files when processing the searching for user's request in a fast and produces effective results. Searching files on personal computer is becoming more popular as both personal and enterprise computer users are finding that sorting and organizing files that are getting bigger in both size and quantity is getting more difficult. The algorithms are experimentally evaluated with synthetic and real data. The results show that their relative performance depends on the problem characteristics.

Keywords: *Data crawler, Task scheduler, Data indexer, Data Searcher, Namazu indexer, NSE Query*

I. INTRODUCTION

With the upcoming of modern technology, computer can complete many kinds of complex tasks. Therefore, the number of files stored in an individual's computer is increasing very rapidly. At the same time, even the cost of storage equipment with large capacity is getting lower and lower, the number of various documents stored in the personal computer, such as video, audio files, digital photos, text files, is increasing very quickly. However, the problem arises is computer users have to spend much time searching their information in the sea of the computer data, and sometimes, even seen or used files cannot be found easily.

Desktop search engine emphasizes on mining all available information in an individual's computer, including web browser history, email files, documents, multimedia files and so on. Desktop search tools search within a user's own computer files as opposed to searching the Internet. A desktop search program is that search results are displayed quickly due to the use of proper indexes. Desktop search emerged as a concern for large firms for two main reasons untapped productivity and security. Most desktop search engines build and maintain an index database to achieve reasonable performance when searching several gigabytes of data. Indexing usually takes place when the computer is idle and most search applications can be set to suspend indexing if a portable computer is running on batteries, in order to save power. In addition to not requiring persistent storage, more powerful queries can be issued, whereas indexed search engines are limited to keyword-based queries. The benefits to not having indices is that, in addition to not requiring persistent storage, whereas indexed search engines are limited to keyword-based queries.

II. LITERATURE SURVEY

A. PERSONAL INTEREST ANALYZER ON END-USER-SIDE AND PERSONALIZED COMPUTING

Personalized search, the key is the use of personal interest information; the personal interest information is produced by personal interest analyzer through analysis and personalized computing of the user's web clicks and browsing, and save it into the personalized file. So, the personal interest information of each target field in personalized file wills directly affects the search result produced by personalized search engines though information filtering of crawled web pages.

When build a personalized file, the personal interest analyzer can provide users a wizard interface to guide the user to enter target fields, like woke field and life target field, and so on. Each target field contains some basic personal interest information. It can track the user's query keywords, the target field being put into use currently and extract pertinent keywords form the user's clicks and browsing of the search result based on this keyword, record the time of this visit, and update these data into the corresponding target field in the personalized file. In the process of updating, there is a need to separate the layers of the keywords, each layer as a field of the personalized file, and to limit the number of the keywords in each layer. Each layer set the granularity of the number of occurrences and the up limit of the number of the keywords, used to debase the layer of the keywords in the personalized file that occurrences are lower and wash out the keywords that have no occurrence for a long time.

B. INTEGRATION OF WEB-BASED INTERFACE WITH NAMAZU INDEXER

Namazu is working in the terminal or console that requires users to type the command to run the searching process. The command used require user to know the location of index file as well as the options available with the command. Hence, the motivation of this work: by integrating Namazu indexer with a web-based interface via java script, an interactive and user-friendly desktop search tools would be developed. The indexer will open and enter the required folders or partition recursively and crawls through all the text files to extract the file information.

C. INDEXER

The Indexer is the box labeled NSE in the right-hand portion. It is managed by the DB2 Net Search Extender (NSE). NSE is a featured text search engine integrated with the IBM DB2 DBMS product. NSE supports the creation of text indexes, and can index the output on individual columns of applying a user-defined function to a column, which is the feature we exploit. NSE manages IR-style inverted index over the virtual documents returned by the Crawler UDF described in the previous section.

D. KEYWORD SEARCH PROCESSOR

The Keyword Search Processor from box labeled “Translate Keywords to NSE Query”. A user’s keyword-based query is translated to an NSE invocation in the form of a SQL query invoking the special contains function.

For example, after indexing our input: Schema graph G where nodes are relations and edges are foreign key constraints, set of root relations RS, and an inserted, deleted, or updated tuple r from Relation Rout put: Root tuples RT whose text objects and virtual documents need to be recomputed.

III. PROPOSED SYSTEM

With the increase of personal computer, desktop search is now an important aspect; researches on desktop search are a common aspect now-a-days. Currently, there are two approaches used for desktop searching: one is to find the file location directly, and the other way of searching files is by their file name, type, text content. Basically, these methods are based on basic information of a file, such as file name, location, and updated time, which are considered as the main hints to complete the search function. However, they cannot effectively manage a large number of data if provided.

Therefore, here some new technologies on desktop search have been proposed, including the following aspects:

- Desktop search engine can be extended to the internal network through PnP protocol, so you can search all files in all computers which are connected over LAN network.
- The retrieval precision of results in desktop search has been excelled by using the relationship between computer data and user's schedule.
- In order to make better search accuracy, integrating semantic expertise to the desktop search process, questioning with natural language technology in desktop search and concept-based search are given.
- Based on the technologies of logic and context, for example, tenet and user mining, desktop search outcome can be enhanced.

A. THE ARCHITECTURE

The proposed system mainly uses the following modules for searching viz :

- *Data Crawler:*

The main function is to scan the local file-system constantly and collect different formatted documents as quickly as possible from local file-system. The main function of Data Crawler is to scan the local file-system regularly and collect different form of documents quickly as possible from local file-system. The collected data may be from different data sources, such as register of entities of files folder, Dynamic File Monitor, mails of various clients, browsing history and so on. Therefore, data collection needs distinct components of different data sources, and all data is modified into the same format for further scheduled process.

- *Task scheduler:*

Task Scheduler acknowledges the data in Prioritized Queue to the Indexer effectively in order to improve the competence of index which is the prime function of Task Scheduler. According to the stated scheduling algorithm, Task Scheduler submits the data in preeminence Queue to the Indexer effectively in regulation to improve the efficiency of index. It is the prime function of Task Scheduler.

- *Data indexer:*

The primary function of Data Indexer is to excerpt text and information of the files collected by the Crawler and then index them. The accustomed items of index are including filename, author's name, file path updated time, text. Text files need word segmentation. Indexing process usually does inverted index (Inversion Index), i.e., items forms the index to find the appropriate document. Text files needs word bisection. Indexes are usually used for inverted index from which it can find the appropriate document from given items of index.

- *Data searcher:*

Data Searcher search the watchword acknowledged by user in FirteX Index Database, and submits the results obtained to the end user through user interface. Then Data Searcher will catch related documents, and gives each document a mark to represents relevant degree between the document and the queries. The search results will be ranked according to rank algorithm used in Rank module and later the final results will be displayed to the user's.

Syntax Constructor is based on FirteX query syntax. It implements the function of multi-keyword search and multi-field search. Different types of files have different fields. For example, text files have text information, while pictures have pixel information but no text, and music files have titles, artists and genre information. Users can search for these fields while FirteX support multi-field search

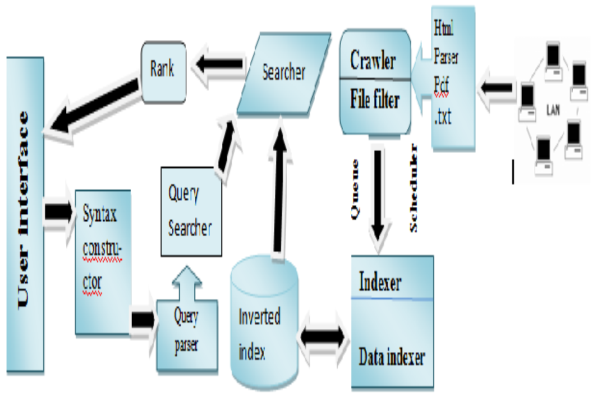


Fig 1: ARCHITECHTURE

IV. MATHEMATICAL MODEL

Suppose the user poses a query q resulting in a large set of results Cand(q,F), and we have (i) q _ content(f) for all files f 2 Cand(q,F), and (ii) there is only one file f0 2 Cand(q,F) such that q \ name(f0) 6= ;. Ignoring for a moment all the other features, and assuming that the user strives to formulate a selective query, it is reasonable to conjecture that the user formulated q while having in mind (either implicitly or explicitly) the filename of f_ q|F. And from this, we have that f0 is more likely to be the desired file f_q|F than any other file in Cand(q,F).

Extending this intuition to typically more fuzzy situations that happen in practice, Selective combines (i) the information carried by the textual properties of the files in Cand(q,F), and (ii) the (observed on Cand(q,F)) frequency of textual connection between each such property and query q. Formally, Selective is computed as follows. We use nz(Featureq) to denote the number of files f 2 Cand(q,F) that have a non-zero (that is, some non-trivial) value Featureq(f).

Given that,

$$\text{Selective } q(f) \text{ def} = \sum_{\text{Name,Path,Content,Query Log}} \frac{\text{Featureq}(f)}{\text{nz}(\text{Featureq})}$$

V. ALGORITHM

The algorithm for merge of inverted lists is based on the index structure. We denote variant of B-tree as SB-tree in following manner.

Let *P*(*w; t*) be a predicate: the word *w* is present in the given document *t*.

Queries of given form are: find all documents *t*, where ^*n i*=1_*mij*=1 *P* (*wij; t*).

Let *Q* = *[n i*=1 *[mij*=1 *wij* be a set of all terms that occurs in the query.

For query evaluation jQjstacks where elements of SBtree entries contain text numbers in ascending order.

The entries of the leaf level nodes are placed into stacks: (*Elem; elem;* 0*; NULL*),

Given *elem* is a text number obtained, dead space is zero and pointer to child node is null.

The stacks referred in the algorithm are of two forms:

Stack(*i*) *i* = 1*; : : : ; jQj* and *Stack*(*wij*)

A. INITIALIZATION

(a) For *i* = 1*; : : : ; jQj* read root node of the SB-tree for *i*-th word in *Q* and place its entries in *Stack*(*i*). If there is no SB-tree for particular word its list numbers must be decoded and placed in stack as well as leaf entries.

(b) Initialize *PosZones* to be actual.

B. NORMALIZATION

(a) Delete all entries from word stacks which intersects no *PosZones* extents.

(b) If for text number *t* all word stacks have entry of the form (*t; t;* 0*;NULL*) then add *t* to the result set of the query. Delete this entry from stacks and adjust *PosZones* to be actual.

(c) If all stacks are empty then end of the algorithm.

C. SEARCH AND RETRIEVAL

(a) Let (*tmin; tmax*) be the top entry of *PosZones*.

Choose word stack which top entry extent (*t*0*; t*1) satisfy *t*0 < *tmin* < *t*1*;* and value of dead space is maximal.

If there is no such stack choose from ones satisfying *t*0 _ *tmin* _ *t*1.

(b) Read child node of the top entry of chosen stack and place its entries instead into stack.

(c) Adjust *PosZones* and go to the step 2.

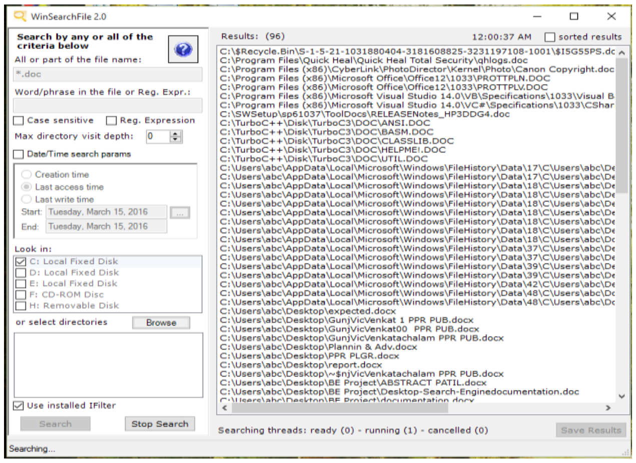
Actual state of *PosZones* is defined in the following way further an algorithm for computing *OrPosZones*(*i*) is described.

1. Clear *OrPosZones*(*i*), initialize *EStack*(*wij*) to be *Stack*(*wij*) for every *j*.

2. Find the tops of *EStack*(*wij*) with minimal left border. If stacks are empty then we have reached end of the algorithm.

3. Pop found extent form it's *EStack* and push it into *OrPosZones*(*i*) if *OrPosZones*(*i*)as it is empty.

VI. EXPECTED OUTPUT



All or part of the file name: *.doc
Word/phrase in the file or Reg. Expr.:
☐ Case sensitive ☐ Reg. Expression
Max directory visit depth: 0
☐ Date/Time search params
Creation time [✓ Last access time] Last write time
Start: Tuesday, March 15, 2016
End: Tuesday, March 15, 2016
Look in:
☑ C: Local Fixed Disk
☐ D: Local Fixed Disk
☐ E: Local Fixed Disk
☐ F: CD-ROM Disc
☐ H: Removable Disk
or select directories
☑ Use installed IFilter

Results: (96) 12:00:37 AM ☐ sorted results

```
C:\$Recycle.Bin\S-1-5-21-1031880404-3181608825-3231197108-1001\$I5G55PS.dc
C:\Program Files\Quick Heal\Quick Heal Total Security\qhlogs.doc
C:\Program Files (x86)\CyberLink\PhotoDirector\Kernel\Photo\Canon Copyright.doc
C:\Program Files (x86)\Microsoft Office\Office12\1033\PROTTPLN.DOC
C:\Program Files (x86)\Microsoft Office\Office12\1033\PROTTPLV.DOC
C:\Program Files (x86)\Microsoft Visual Studio 14.0\VB\Specifications\1033\Visual B
C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC#\Specifications\1033\CShar
C:\SWSetup\sp61037\ToolDocs\RELEASENotes_HP3DDG4.doc
C:\TurboC++\Disk\TurboC3\DOC\ANSI.DOC
C:\TurboC++\Disk\TurboC3\DOC\BASM.DOC
C:\TurboC++\Disk\TurboC3\DOC\CLASSLIB.DOC
C:\TurboC++\Disk\TurboC3\DOC\HELPME!.DOC
C:\TurboC++\Disk\TurboC3\DOC\UTIL.DOC
C:\Users\abc\AppData\Local\Microsoft\Windows\FileHistory\Data\17\C\Users\abc\De
C:\Users\abc\AppData\Local\Microsoft\Windows\FileHistory\Data\17\C\Users\abc\De
C:\Users\abc\AppData\Local\Microsoft\Windows\FileHistory\Data\18\C\Users\abc\De
C:\Users\abc\AppData\Local\Microsoft\Windows\FileHistory\Data\18\C\Users\abc\De
C:\Users\abc\AppData\Local\Microsoft\Windows\FileHistory\Data\18\C\Users\abc\De
C:\Users\abc\AppData\Local\Microsoft\Windows\FileHistory\Data\18\C\Users\abc\De
C:\Users\abc\AppData\Local\Microsoft\Windows\FileHistory\Data\18\C\Users\abc\De
C:\Users\abc\AppData\Local\Microsoft\Windows\FileHistory\Data\18\C\Users\abc\De
C:\Users\abc\AppData\Local\Microsoft\Windows\FileHistory\Data\18\C\Users\abc\Dc
C:\Users\abc\AppData\Local\Microsoft\Windows\FileHistory\Data\37\C\Users\abc\De
C:\Users\abc\AppData\Local\Microsoft\Windows\FileHistory\Data\37\C\Users\abc\De
C:\Users\abc\AppData\Local\Microsoft\Windows\FileHistory\Data\39\C\Users\abc\De
C:\Users\abc\AppData\Local\Microsoft\Windows\FileHistory\Data\39\C\Users\abc\De
C:\Users\abc\AppData\Local\Microsoft\Windows\FileHistory\Data\42\C\Users\abc\De
C:\Users\abc\AppData\Local\Microsoft\Windows\FileHistory\Data\48\C\Users\abc\Dc
C:\Users\abc\AppData\Local\Microsoft\Windows\FileHistory\Data\48\C\Users\abc\Dc
C:\Users\abc\Desktop\expected.docx
C:\Users\abc\Desktop\GunjVicVenkat 1 PPR PUB.docx
C:\Users\abc\Desktop\GunjVicVenkat00  PPR PUB.docx
C:\Users\abc\Desktop\GunjVicVenkatachalam PPR PUB.docx
C:\Users\abc\Desktop\Plannin & Adv.docx
C:\Users\abc\Desktop\PPR PLGR.docx
C:\Users\abc\Desktop\report.docx
C:\Users\abc\Desktop\~$njVicVenkatachalam PPR PUB.docx
C:\Users\abc\Desktop\BE Project\ABSTRACT PATIL.docx
C:\Users\abc\Desktop\BE Project\Desktop-Search-Enginedocumentation.doc
C:\Users\abc\Desktop\BE Project\documentation.docx
```

Searching threads: ready (0) - running (1) - cancelled (0)

Searching...

Fig 2: Expected Result

VII. CONCLUSTION

Hence the above project implemented is basically for the group of people whose personal computers consist of plenty of files including personal as well as official. It can also be used at various work places where data sharing is important for completion of group related works (for example Large Scale Projects, School, Colleges).

Also this project have implemented client server model wherein the server would be the parent system and the clients connected to server system are called as offspring's of parent system. Here clients can view each other's data but cannot overwrite it, where as the server can perform action.

REFERENCES

[1] Technology of The Institute of Computing Technology of Chinese Academy of Sciences, "FirteX-High Performance Search Platform",

[2] Y. MATSUBARA and I. KOBAYASHI, "Development of a Desktop Search System Using Correlation between User's Schedule and Data in a Computer," Proc. the 2007 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology.

[3] Wei Lun Huang, Tzao Lin Lee and Chiao Szu Liao, "Desktop Search in the Intranet with Integrated Desktop Search Engines," Proc. The 13th IEEE Asia-Pacific Computer Systems Architecture Conference (ACSAC 2008), IEEE Press, Aug. 2008, pp. 1-4.

[4] S. Cohen, C. Domshlak and N. Zwerdling, "On Ranking Techniques for Desktop Search," ACM Transactions on Information Systems, vol. 26, Mar. 2008, pp. 1183-1184.

[5] J. Gaugaz, S. Costache, P. Chirita, C. S. Firan1 and W. Nejdl, "Activity Based Links as a Ranking Factor in Semantic Desktop Search," Proc. Latin American Web Conference 2008(LA-Web 2008), IEEE Press, Oct. 2008, pp. 49-57.

[6] S. Chernov, "Task Detection for Activity-based Desktop Search," Proc. the 31st annual international ACM SIGIR conference on Research and Development in Information Retrieval (SIGIR 2008), ACM New York, Jul. 2008, pp. 894-894.

[7] C. Fluit, "Autofocus: Semantic Search for the Desktop," Proc. the 9th International Conference on Information Visualisation (IV 2005), London, IEEE Press, Jul. 2005

[8] P. A. Chirita, R. Gavriloaie, S. Ghita, W. Nejdl and R. Paiu, "Activity based Metadata for Semantic Desktop Search," Proc. the 2nd Annual European Semantic Web Conference (ESWC 2005), Springer Berlin, May. 2005, pp. 439-454.

[9] P. A. Chirita, S. Costache, W. Nejdl and R. Paiu, "Beagle++: Semantically Enhanced Searching and Ranking on the Desktop," Proc. the 3rd Annual European Semantic Web Conference (ESWC2006), sSpringer Berlin, Jun. 2006, pp. 348-362.