AN OPEN SOURCE SOFTWARE BASED LIBRARY CATALOGUE SYSTEM USING SOUNDEXING RETRIEVAL AND QUERY CACHING

Zhenghui Pan
Penrith City Library, 601 High Street Penrith Civic Centre, Penrith, NSW 2750, Australia

Yan Zhang, Jiansheng Huang
School of Computing and Math, University of Western Sydney, Sydney, NSW 1797, Australia

Keywords: Open Source Software, Library Catalogue System, Soundexing Retrieval, Query Caching.

Abstract: It has been a challenge to apply effective knowledge management tools in information systems for modern libraries that deliver the most up-to-date, relevant information to end users in a quick, efficient, and user-friendly manner. In this paper, the authors present the design of a library catalogue system using principally open source software. The integrated web-based library system takes client/server architecture with multiple tiers. For performance enhancement with respect to error tolerance, searching speed and scalability, techniques of soundexing retrieval and query caching were applied. With the support of an appropriately designed soundex algorithm, the catalogue system can largely increase its recall while not compromising the search precision. On the other side, the introduced query cache speeds up the system response significantly.

1 INTRODUCTION

Catalogue systems of modern libraries are getting more complicated than ever before. Collections of catalogue systems are now containing CD-ROMs, DVDs and e-books, in addition to the conventional printed materials. Not only collections of the catalogue systems have been expanded, but also their users and service time. All these call for a more efficient and user friendly service. Based on the survey on local council libraries in Western Sydney area, a wide existence of some common problems concerned with searching speed and accuracy, and user interface was found, all closely related to the performance of the catalogue system. The library information systems of these libraries are usually decades old. The current marketing price (including maintenance costs) for a library catalogue system can be prohibitively high. Improving and updating the information systems of these libraries using open source software might provide a better solution.

Soundex is an algorithm for encoding a word so that similarly sounding words are encoded the same. The soundex algorithm was initially invented by Robert Russell and Margaret Odell in 1918 (Wikipedia, 2007). Since then, the soundex algorithm has been updated with some variations and used principally in census systems for searching people's names (U.S. National Archives, 2007). To improve the performance of library catalogue systems with respect to response speed and scalability, database caching is a very cost-effective method. (Lempel, 2003).

This paper presents open source software based library catalogue system using soundexing algorithms and query caching. The paper is organised as follows: Section 2 outlines the library catalogue system. Section 3 briefs the soundexing algorithms for error tolerance. Section 4 describes the steps of embedding the soundex search into a traditional library catalogue system. The limitations of soundex search and the countermeasures are also discussed in the Section. Section 5 explains the adopted caching technique and the system design. The eviction and start-up processes are discussed in Section 6. Section 7 gives experiments results and discussions, and Section 8 summarizes and concludes the paper.

Pan Z., Zhang Y. and Huang J. (2010).
AN OPEN SOURCE SOFTWARE BASED LIBRARY CATALOGUE SYSTEM USING SOUNDEXING RETRIEVAL AND QUERY CACHING.
In *Proceedings of the 12th International Conference on Enterprise Information Systems - Databases and Information Systems Integration*, pages 182-189
DOI: 10.5220/0002891201820189

2 OPEN SOURCE SOFTWARE BASED SYSTEMS

The developed open source software-based library system takes a client/server architecture with multiple tiers. The prototype system runs on the Windows XP of a laptop computer with 1.5 GHz processor and 512 MB, see Figure 1.

2.1 MySQL Database

The database uses MySQL version 1.5. The database contains the following tables:

- Title
- Item
- Title-index
- Request-history
- Request-summary

2.2 Tom Cat Server and JVM Environment

The system is installed with the JRE 1.5 to provide the JVM environment. The Tomcat server is used as the application and the web server. One package of Java classes forms the core of the applications, containing two main parts: the search engine module and the background process module. The background process module runs three processes, system booting, query-log processing and cache items evicting.

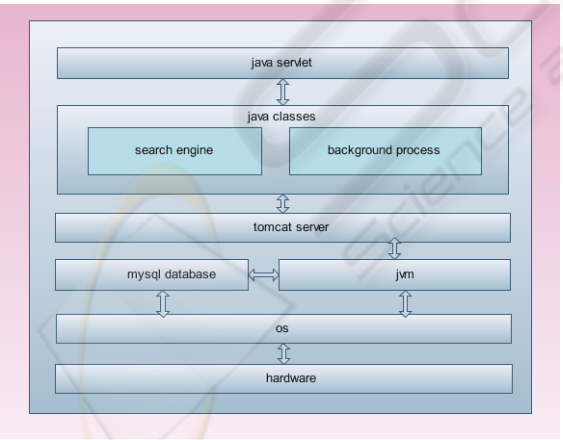

Figure 1: Outline of Library System.

3 SOUNDEXING ALRORITHM

Soundex algorithms convert words into their phonetic versions. For examples, ‘m’ and ‘n’, and ‘ph’ and ‘f’ are coded the same as the former is a common typing error, and the latter is pronounced the same.

The basic soundex algorithm is very straight forward as outlined below:

- Retain the first letter of the string (word)
- Remove all occurrences of the following letters, unless it is the first letter: a, e, h, i, o, u, w, y
- Assign numbers to the remaining letters (after the first) as follows
  - b, f, p, v = 1
  - c, g, j, k, q, s, x, z = 2
  - d, t = 3
  - l = 4
  - m, n = 5
  - r = 6
- If two or more letters with the same number were adjacent in the original word (before step 1), or adjacent except for any intervening h and w, omit all but the first.
- Return the first four characters, right-padding with ‘]’ if there are fewer characters.

As demonstrated in the following three case studies, the soundexing algorithm applied in a library catalogue system can greatly increase its recall while maintaining the search precision.

Table 1: Search results with input “hary poter rawling”.

#	Title	Author	Publication…
1	Harry Potter and the Half-Blood Prince / by J. K. Rowling	J. K. Rowling	…
2	Harry Potter and the Prisoner of Azkaban / by J. K. Rowling and read by Stephen Fry	J. K. Rowling; Stephen Fry	…
3	Harry Potter and the Order of the Phoenix / by J. K. Rowling and read by Stephen Fry	J. K. Rowling; Stephen Fry	…
4	…	…	…

Case 1: Keying in words “hary poter rawling” to search for the series of Harry Potter books by J.R. Rowling. Even though all the three input words were spelled wrongly, the search results using the soundex based catalogue system returned the correct results to the user, as given in Table 1 (including only the first 3 records). This was due to the fact that “hary” and “harry”, “poter” and potter”, “rawling” and “rowling” were converted into the same character string using the presented algorithm.

Case 2: Search results with the input “napaen rever water polootion”. By using the same algorithm, the

search results were returned as though the correct key words "Nepean River Water Pollution" were inputted.

Case 3: Search results with the input "enciclopedia scince". Again, the algorithm avoided search failure and returned results that were all the same as those for "encyclopaedia science".

Soundex searches, however, may have certain limitations, like bringing about irrelevant information, as to be addressed later in the paper.

4 SOUNDEX CATALOGUE SYSTEMS

By employing the soundex algorithms, the search on a library catalog system will be carried out with the specified "sounds" rather than the entered words. For such a soundex search, two main concerns should be focused upon in the system design: the performance and the accuracy.

4.1 Soundex Search

In the traditional catalog system, as illustrated by Figure 2(a), the index data is built as an additional part of the catalog database to improve the system performance. (Dvorský, Krátký, Skopal, and Snásel, 2003). The search will be carried out on the index data first to find the matched catalogue records, and the catalogue details will then be returned to the user as the search results.

It is straightforward to embed the soundex search into the catalog system: the word index data are just replaced by the soundex data with the latter being generated from the catalogue records by using the conversion algorithm given Section 3. Once the search engine receives the request from a user, it converts the request into the string of soundex first. Then it will search the soundex data in the database for the matching soundex and get the list of catalogue keys. The final search results, i.e., the catalogue details, are obtained according to the list of keys, as illustrated by Figure 2 (b).

4.2 Limitation of Soundex Algorithms and Countermeasures

In nowadays, the catalogue system is no longer a tool only for librarians; it is often used by the public in OPAC (Online Public Access Catalogue) systems that are widely accessible through the Internet. Consequently the search failures caused by spelling or typing errors become a major concern for the developers of such OPAC systems. Studies indicated that over 20% of catalog search failures come from typos of the catalogue users. (Jean, 1984).

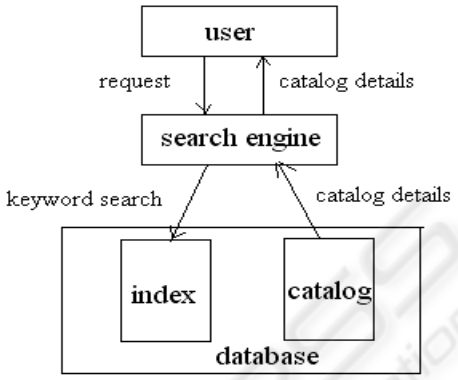

Figure 2 (a): Traditional Catalogue System.

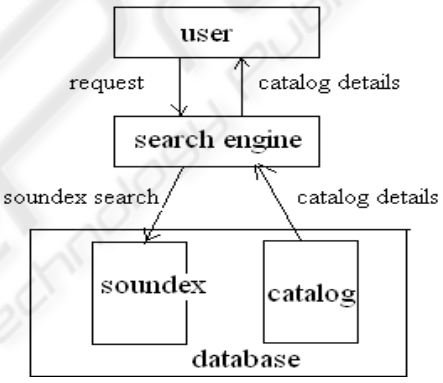

Figure 2 (b): Soundex Catalogue System.

To extend the soundexing search from conventional census systems to more general applications, ensuring the search accuracy is not a trivial task. As soundexing is a hashing system with one soundex mapping to many words, therefore, a soundex search may bring about irrelevant results and thus reduce the precision of the search. It is just this limitation that restrict the application of the soundexing techniques in general information systems. Some measures have been adopted in our project to ameliorate this limitation.

A multiple-word search will be conducted by performing a Boolean 'AND' operation. The search result is generated by combining all sub-results of the soundex search. Thus most of the irrelevant results will be filtered out as they hardly have any overlap. In most of the search cases, words entered by users are two or more so that the search results are quite acceptable. Study result from RankStat

indicated that more than 86.5% of queries were two or more word searches. (RankStat, 2007).

One-word searches remain problematic, which explains why the original soundex was only used implicitly for two or more word searches, such as in searching a person's name that usually contains at least the first and the last names. One countermeasure to this limitation is to turn off the soundex search automatically in the case of a one-word search. To avoid search failures, however, the soundex search could be turn on again, should no hit results be returned with the initial non-soundex search.

Another area that has been investigated in the project is the improvement of the soundex algorithm itself. The following measures are adopted to make the soundex more specific:

- Increasing the length of the soundex of a word from 4 to 5;
- Adopting Celko algorithms and/or Pfeifer algorithms. (Holmes and McCabe, 2002);
- Applying Stemming technique. (Porter, 2006);
- Using Phonix technique. (Gadd, 1990).

By adopting the above measures, search accuracy can be improved significantly.

4.3 Soundex Catalog Search Implementation

There are three major tasks in the implement of the soundex search:

- A set of program packages to convert a word string into soundex string;
- Persistent data structure (relational database tables) to store the soundex data of the catalogue records;
- A set of programs (the search engine) to carry out the soundex search.

In the project, all the major types of catalogue searches are included to store the soundex data, such as searches on titles, authors, subjects, or on full records. In the project, the study library catalogue system takes the following structure (columns):

- catalog_key
- title_description
- author_description
- series_description
- subject_description
- publication_details
- isbn
- physical_description
- cost
- dewey_number
- contents
- summary

The corresponding catalog_soundex table is thus created with the following columns:

- catalog_key
- title_soundex
- author_soundex
- series_soundex
- subject_soundex
- all_field_soundex

Data in all the above columns correspond respectively to the columns of title_description, author_description, series_description, subject_ description, and the catalogue table itself. By using the catalog_soundex table, the process of soundex search becomes very simple. Soundex based searches can be applied to title, author, series, subject or even all keywords from the whole catalogue record.

Suppose that the entered string is "hary poter rawling", the same as Case 1 in Section 3. The system will convert this string into a soundex string "H5]] P25] R346". Following that the system makes a sql-query on the database:

```
select catalog_key from
catalog_soundex where
match(all_field_soundex)
against('+H5]] +P25] +R346' in
Boolean mode)
```

The search results are as given in Table 1 (Section 3). It is apparent that the input "hry poter rawling" may lead to a search failure with a normal catalogue system.

The soundex search function can be easily embedded into the catalogue by following the steps given below:

- Add the soundex processing function into the cataloguing module;
- Generate the soundex data of a catalogue record;
- Commit it to the database after the catalogue record has been added to the system;
- Modify the search engine by adding the soundex processing function;

- Perform the sql-query on the database with generated soundex.

5 THE CACHING TECHNIQUE

While the soundexing is introduced for a better performance in error tolerance, the caching techniques are adopted to improve searching speed of the catalogue system. There are two main categories of cache, data cache and query cache. (Luo, Krishnamurthy, Mohan, Pirahesh, Woo, Lindsay and Naughton, 2002). In information systems, the query cache provides users with data in memory based on users' previous requests, and the caching levels can be at different places, either on client side, server side or proxy side, either at database tier, application server tier or web server tier (Fagni, Perego, Silvestri and Orlando, 2006). All these caches can be used separately and independently, or used together in certain combinations.

5.1 Database Design

To implement a cache in an information retrieval system the following mechanisms need to include: defining the structure of cache, adding items into cache, searching the cache, continuing a search when a cache missing occurs. Once the cache size exceeds the limit, the cache system should remove certain items out of the cache known as the cache replacement, during which, the system should keep the cached items that are likely to be requested in the near future.

The proposed library catalogue system adopts s query cache (Figure 3). There are two key components in the system, the search engine module and the cache manager module. Apart from the catalogue data, the database of the system includes some additional data like request history, request summary and cached requests that are used by the cache manger for updating the cache. There are two data structures kept in the main memory: the query cache and the query log. While the query cache contains cached catalogue objects, the query log temporarily holds the current catalogue search requests for further processing by the cache manager. The search engine module is responsible for retrieving user's required information. The cache manager module has two roles: using requests in the query-log to update the request-history and the request-summary table and adding new catalogue items to the cache. The module is also responsible for removing items out of the cache whenever the cache size is over the limit.

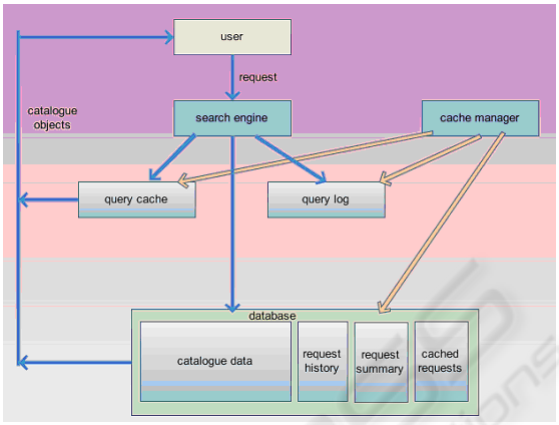

Figure 3: Catalogue System with Query Cache.

Basically the database includes the catalogue core data, catalogue index data (soundexing) and data for supporting the cache search. Thus the database contains five main tables: item, title, title-index, request-history and request-summary. The field "in_cache_flag" of request-summary table indicates whether a request has been cached. Full details of the database design are beyond the subject of this paper. Only the tables related to the caching processes will be described here. (Pan, Zhang and Huang, 2008).

The request-history has a very simple structure with two fields:

- request
- timestamp

The table will hold all requests together with the timestamp when the request entered. Data in the table is used for updating table request-summary.

The request-summary table of the database provides a list of all valid (a request that leads to at least one catalogue object being retrieved) and identical requests in conjunction with their properties, i.e., the most recent time and the total times this request was called. The request-summary table also provides the system criteria for adding or removing catalogue objects into or from the cache according to the field "activity rate". The table has the following structure:

- request_id
- request
- been_called_total
- last_called_time
- activity_rate

- in_cache_flag

With this database structure, the system can keep track of users' requests, and select the requests with the highest activity rates to cache.

5.2 Data Structures in Main Memory

There are two data structures staying in the system memory to support the cache catalogue search: query-log and query-cache. Like the request-history, the query-log has a very simple structure with only two fields:

- request
- timestamp

The query-log therefore can be implemented as a hash-table or an array-list since there is not much operations performed on it. If the query-log is implemented in an array-list, the following command can be used to write a request to the log:

```
log.add ({request, timestamp})
```

To get an object from the log, the command is like:

```
log.get(i)
```

where *i* is the index for the array-list. The following command is to empty the log:

```
log = new ArrayList ( )
```

The query-cache holds cached catalogue objects together with their correspondent requests. The query-cache is better to be implemented in a hash-table. With this data structure one can easily perform operations of search, add or remove to the query-cache. Search and retrieve data from the hash-table (named QC) use the command like:

```
data = QC.get(hash-key)
```

If the query-cache is in a hash-table format, it will have the following two fields:

- request, as the hash key and
- a set of catalogue objects corresponding to the request

With this data structure support, the command to retrieve required catalogue objects is like:

```
QC.get(request)
```

In this system context, both the query-log and the query-cache need to be the global variables. It makes the processes behave like the search engine and the cache manager and the system start-up processes be very simple since the global data structures are independent of any processes.

5.3 The Searching Process

When the search engine receives a search request, it uses the request as a hash-key and searches the cache (the hash-table). If the searching is successful, it gets the set of catalogue objects and returns the results to the user. Otherwise, the search engine continues to search the database. At the end of the process, the search engine always writes the request accompanied with the timestamp to the query-log.

The query log process can be scheduled to run in background. On each run, the cache manger moves all requests from the query-log to the request-history table. After adding new requests with the timestamps to the request-history table, the cache manager uses them to update the request-summary table. For a valid request (invalid entries will be ignored), the cache manager will further check if the request already exists in the request-summary table. If existing, the properties for this request (been_called_total and t_called_time) will be modified. If the request is not found in the table, a new entry for this request is created. The cache manager will then check if the new request was already loaded into the cache, and do this if not.

6 ADAPTIVE EVICTION PROCESS

The system defines two variables, the maximum-cache-size and minimum-cache-size. The maximum-cache-size is the maximum number of requests that can be loaded into the cache while the minimum-cache-size is the minimum number of requests kept in the cache at any time.

6.1 Eviction Process

There are a few of cache replacement algorithms often used in information retrieval systems like LRU, LFU and some hybrid replacement algorithms. Adaptive cache replacement algorithm is an algorithm with better performance. The algorithm, used in the designed system, is implemented by keeping track of both frequently used and recently used cached items in conjunction with the recent eviction history. (Santhanakrishnan, Amer, Chrysanthis and Li 2004).

As aforementioned, the cache manager is also responsible for replacement. Each time the process runs, the cache manager checks if the maximum-cache-size is exceeded. If it happened, the cache

manager starts the cache eviction process. The number of objects to be removed is:

```
Current_cache_size-inimum_cache_size
```

This work can be done in several steps. The process first calculates the activity-rate for all requests in the request-summary table. The requests with the smallest activity-rates will be removed. The activity-rate calculation is based on two attributes of a request, the been_called_total and last_called_time. If a visit one day more recent is equally important as three more visits, a variable time/times ratio @ can then be calculated :

```
@ = 3/86400000
```

where 86400000=24 * 3600 * 1000, i.e., million seconds in a day. If using *T* as the base time, a variable last_called_credit can be calculated as:

```
last_called_credit  =
(last_called_time – T) * @
```

Then the activity-rate of a request can be calculated as:

```
activity_rate =been_called_total +
last_called_credit
```

6.2 System Start-up Module

Like the search engine and the cache manager modules, the system start-up module is also responsible for cache loading when the system re-starts. The process works similar to the cache replacement process. It first calculates activity-rate for all the requests in the request-summary table. According to the initial loading level (minimum-cache-size), the module loads the requests with highest activity-rates into the cache.

7 SIMULATION RESULTS AND DISCUSSIONS

Once the Tomcat server is running, users can access the system through a web browser. If a user enters the following key words like “19th century”, “abc”, etc., the response time for each normal search is given in Table 2. The average response time = 664.18 ms. In the simulation, about 7,000 requests are initially selected and loaded into the cache (based on the activity-rate of the requests) from a total of 13,500 requests in the table request-summary. With the caching, the responding time is less than 1 ms if the required item is found in the cache.

Because about 50% of requests are loaded into the cache (with the corresponding catalogue objects), the minimum cache hits rate is about 50%. Although the cache missing rate is still high, the whole system throughput is improved significantly.

Table 2: Experiment on searching.

Request	Response time (ms)	Request	Response time (ms)
19th century	1828	Bats	468
abc	1343	bev harvey	203
aboriginal dreaming	609	bill hurter	156
aboriginal language	421	biochemistry	109
addict	375	bob marley	234
African tears	78	brown	2328
alexandre dumas	328	burke	1140
alternative medicine	797	cam jansen	94
amy jenkins	140	carol odell	109

8 CONCLUSIONS

Using open source software like MySQL and Tomcat could be a better solution in updating library catalogue systems for organizations with limited resources. To improve the performance, some techniques are investigated in the paper. Soundex algorithms are potentially a powerful tool applicable to library catalogue systems, as the algorithms, when appropriately developed, can greatly increase recalling capability while not compromising search precision. As verified in this paper, the soundex search can be cost-effectively embedded into a traditional library catalogue system using client/server and relational database technology. The proposed soundex library system will greatly improve efficiency and error tolerance.

Using caching techniques can effectively improve the catalogue system in terms of responding speed and scalability. To increasing the cache hit rate, some methods are investigated, such as taking into account more factors in the eviction process and introducing a caching-index to the system for query normalization.

While there is still room for further improvement, the test results are encouraging and meet the expectation of the project.

REFERENCES

Wikipedia, (2007). Soundex, from *http://en.wikipedia.org/wiki/Soundex#Rules*.

U.S. National Archives, (2007). from *http://www.archives.gov/genealogy/census/soundex.html*.

Lempel, R., Moran, S., (2003). Predictive Caching and Prefetching of Query Results in Search Engines. *WWW 2003*, May 20-24, 2003, Budapest, Hungary.

Dvorský, J., Krátký, M., Skopal, T., Snásel, V., (2003). Term Indexing in Information Retrieval Systems, *Communications in Computing*, 2003.

Jean, D., (1984). An Analysis of User Errors in Searching an Online Catalog. *Cataloging & Classification Quarterly*, Volume 4, Issue 3 March 1984 , 19 – 38.

RankStat, (2007). from *http://www.rankstat.com/html/en/seo-news1-most-people-use-2-word-phrases-in-search-engines.html*.

Holmes, D. and McCabe, M. C., (2002). Improving Precision and Recall for Soundex Retrieval. *Information Technology: Coding and Computing, 2002. Proceedings. International Conference on*, Volume, Issue , 8-10 April 2002, 22 - 26.

Porter, M. F., (2006). An algorithm for suffix stripping. *Program: electronic library and information systems*, 2006 Volume: 40 Issue: 3, 211 – 218, ISSN: 0033-0337.

Gadd, T., (1990). PHONIX: The algorithm. *Program: automated library and information systems*, 24(4): 363-366.

Luo, Q., Krishnamurthy, S., Mohan, C., Pirahesh, H., Woo, H., Lindsay, B. G., Naughton, J. F., (2002). Middle-Tier Database Caching for e-Business discussed. *ACM SIGMOD'2002*, June 4-6, 2002, Madison, Wisconsin, USA.

Fagni, T., Perego, R., Silvestri, F., Orlando, S., (2006). Boosting the Performance of Web Search Engines: Caching and Prefetching Query Results by Exploiting Historical Usage Data. *ACM Transactions on Information Systems*, Vol. 24, No.1, January 2006, 51-78.

Pan, Z., Zhang, Y., Huang, J., (2008). Application of Database Caching in a Library Catalogue System. *2008 International Conference on Computer Science and Software Engineering (CSSE 2008)*, Wuhan, China, 12-14, December 2008

Santhanakrishnan, G., Amer, A., Chrysanthis, P. K., Dan Li, (2004). GD-Ghost: A Goal-Oriented Self-Tuning Caching Algorithm. *SAC 2004*, March 14-17, Nicosia, Cyprus.