DATE-BASED DYNAMIC CACHING MECHANISM

Christos Bouras,
Research Academic Computer Technology Institute, N. Kazantzaki, Panepistimioupoli and
Computer Engineering and Informatics Department, University of Patras
26504 Rion, Patras, Greece
bouras@cti.gr

Vassilis Poulopoulos
Research Academic Computer Technology Institute, N. Kazantzaki, Panepistimioupoli and
Computer Engineering and Informatics Department, University of Patras
26504 Rion, Patras, Greece
poulop@cti.gr

Panagiotis Silintziris
Computer Engineering and Informatics Department, University of Patras
26504 Rion, Patras, Greece
silingir@ceid.upatras.gr

ABSTRACT

News portals based on the RSS protocol are becoming nowadays one of the dominant ways that Internet users follow in order to locate the information they are looking for. Search engines, which operate at the back-end of a big portion of these web sites, receive millions of queries per day on any and every walk of web life. While these queries are submitted by thousands of unrelated users, studies have shown that small sets of popular queries account for a significant fraction of the query stream. A second truth has to do with the high frequency that a particular user tends to submit the same or highly relative search requests to the engine. By combining these facts, in this paper, we design and analyze the caching algorithm deployed in our personalized RSS portal, a web-based mechanism for the retrieval, processing and presentation in a personalized view of articles and RSS feeds collected from major Internet news portals. By using moderate amounts of memory and little computational overhead, we achieve to cache query results in the server, both for personalized and non-personalized user searches. Our caching algorithm operates not in a stand-alone manner but it co-operates and binds with the rest of the modules of our Portal in order to accomplish maximum integration with the system.

KEYWORDS

query results caching, search engine, personalized search, query locality, data retrieval, date-based caching

1. INTRODUCTION

The technological advances in the World Wide Web combined with the low cost and the ease of access to it from any place in the world has dramatically changed the way people face the need for information retrieval. More and more users migrate from traditional mass media to more interactive digital solutions such as Internet news portals. As the number of users exponentially increases and the volume of data involved is high, it is very important to design efficient mechanisms to enable search engines to respond fast to as many queries as possible. One of the most common design solutions is to use caching, which may improve efficiency if the cached queries occur in the near future.

Regarding earlier classic works in the topic of caching, Markatos investigated the effectiveness of caching for Web search engines (Markatos E.P. 2001). The reported results suggested that there are important efficiency benefits from using caches, due to the temporal locality in the query stream. Xie and O'Hallaron also found a Zipf distribution of query frequencies (Xie and Halaron, 2002), where different users issue very popular queries, and longer queries are less likely to be shared by many users. On cache management policies, Lempel & Moran (Lempel and Moran, 2003) proposed one that con- siders the probability distribution over

all queries submitted by the users of a search engine. (Fagni et al. 2006) described a Static Dynamic Cache (SDC), where part of the cache is read-only or static, and it comprises a set of frequent queries from a past query log. The dynamic part is used for caching the queries that are not in the static part. Regarding locality in search queries, in their earlier works, Jansen & Spink (Jansen and Spink, 2006) provide insights about short-term interactions of users with search engines and show that there is a great amount of locality in the submitted requests. Teevan et al. (Teevan et al. 2006) examined the search behaviour of more than a hundred anonymous users over the course of one year. The findings were that across the year, about one third of the user queries were repetitions of queries previously issued by the same user. Although these studies have not focused on caching search engine results, all of them suggest that queries have significant locality, which particularly motivates our work.

In our work, we take advantage of this space and time locality, and we cache the results from very recently used queries in order to reduce the latency on the client and the database-processing load from the server. Because of the fact that the caching is server side, both registered and unregistered users of the portal can take benefit. Furthermore, for registered users, the algorithm takes into account the dynamic evolution of their profile and provides them with results even more close to their preferences and interests. The rest of the paper is structured in the following way: in the next section description of the architecture of the system with the focus in the algorithm of caching is presented. The caching algorithm is analyzed in Section 3. In section 4, we present some of the experimental results and evaluation of our work, regarding algorithmic performance, results accuracy and storage space requirements. We conclude in Section 5 with some remarks about the described techniques and future.

2. ARCHITECTURE

The architecture of the system is distributed and based on standalone subsystems but the procedure to reach at the desired result is actually sequential, meaning by this that the data flow is representative of the subsystems of which the mechanism consists. Another noticeable architectural characteristic is the existence of modularity throughout the system's lines. This section is a description of how these features are integrated into the mechanism. We are putting the focus on the subsystem responsible for caching, though analysis of the other modules is presented in order to cross-connect the features of our system.

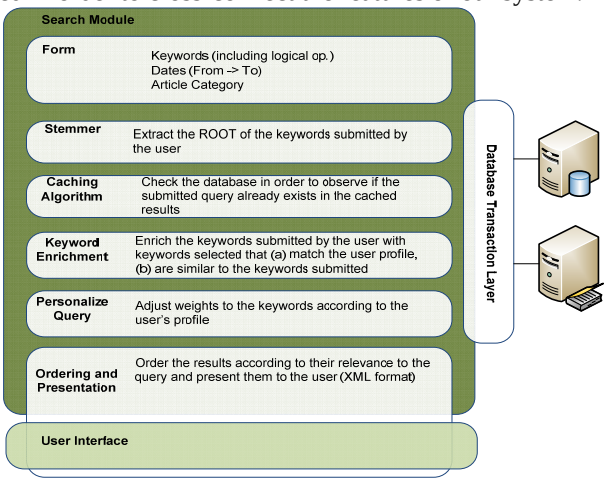

Figure 1. Search Module Architecture

The general procedure of the mechanism is as follows: first web pages are captured and useful text is extracted from them. Then, the extracted text is parsed followed by summarization and categorization. Finally we have the presentation of the personalized results to the end user. For the first step, a simple web crawler is deployed, which uses as input the addresses extracted from the RSS feeds. Theses feeds contain the web links to the sites where the articles exist. The crawler fetches only the html page, without elements such as referenced images, videos, CSS or JavaScript files. Thus, the database is filled with pages ready for input to the 1st level of analysis, during which, the system isolates the "useful" text from the html source.

Useful text contains the article's title and main body. This procedure analysis can be found in (self reference). In the 2nd level of analysis, XML files containing the title and the body of articles are received as input, targeting at applying pre-processing algorithms on this text in order to provide as output the keywords, their location in the text together with their absolute frequency in it. These results are the primary key to the 3rd level of analysis. In the 3rd level of analysis, the summarization and categorization technique takes place. Its main scope is to characterize the articles with a label (category) and come up with a summary of it. This procedure is described in (self reference).
In Figure 1, we can see the general schema and flow of the advanced and personalized search subsystem of our portal. The caching algorithm, which is the core of our work and will be analyzed in the next section, is used to search for identical queries submitted in the past from the same or other users and if matches are found, cached results are directly obtained improving in this way the search speed and reducing the workload on the server. The cached data is stored in the server's database.

3. ALGORITHMIC ASPECTS

In this section of the paper, we shall analyze the caching algorithm of our search implementation.

3.1 Search Configuration and Keyword's Stemming

Before the engine triggers the search procedure, the user has first to configure the query request. Apart from the specified keywords, a few other options are offered, including the date period for the target result (DATE FROM – DATE TO), the selection of the logical operation ("OR" and "AND), which will be performed in the articles matching and the thematic category of the desired output. Before proceeding with the query search operation, we should also notice that the engine passes the keywords through a stemmer, which implements the Porter Stemming Algorithm on the English language. Thus, we enable the integration of the search engine with the rest of the system, which is implemented by using stems rather than full words for the articles categorization. Additionally, simple duplicate elimination is executed on the stems.

3.2 Caching Algorithm

Prior to searching for the result articles, the system searches for cached data from previous search sessions. All cached data are stored on the server's storage space and the caching algorithm also operates in the server's memory so the procedure, which will be described, will benefit both registered users (members) as well as unregistered users Search Module (guests) of the portal without creating any computational overhead for their machines.
For each submitted query, we store in a separate table in our database information about the configuration of the search request. This information includes the id of the user who submitted the search, the exact time of the search (as a timestamp), the keywords used in the query formatted in a comma-separated list and a string containing information about the desired category of the results and the logical operation, which was selected for the matching. For the above data, our caching algorithm operates in a static manner. For example, if a user submits a query containing the keywords "nuclear technology", by selecting the "science" category as the target category for the returned articles, this query will not match against an existing (cached) query which contains the same keywords but which was in the first case cached for results on the "politics" category. Also, when a query containing more than one keyword is submitted, it will not match against cached queries containing subsets or supersets of the keyword set of the submitted query. For example, if the incoming query contains the keywords "Monaco circuit formula" probably referring to the famous Grand Prix race, it will not be considered the same with a cached query containing the keywords "circuit formula" which probably refers to an electrical circuit formula of physics. This decision for the implementation was taken in order to avoid semantic ambiguities in the keywords matching process.
The dynamic logic of our caching algorithm lies in the target date intervals of a search request, which are represented by the DATE FROM and DATE TO fields in the search configuration form of the portal. This perspective of caching was chosen after considering the fact that is very common for many web users to submit identical queries repeatedly in the same day or during a very short period of days. The algorithm

designed for this reason takes into account the following 4 cases for the cached DATE FROM and DATE TO fields and submitted DATE FROM and DATE TO fields:

- **1st Case**: the DATE FROM-TO interval of the submitted query is a subset of the DATE FROM-TO interval of the cached query. In this case, we have the whole set of the desired articles in our cache plus some articles out of the requested data interval. The implementation fetches all the cached results and it filters out the articles, which were published before DATE FROM and these, which were published after DATE TO attribute of the submitted request. The server's cache is not updated with new articles because in this case no search is performed in the articles database.
- **2nd Case**: the DATE FROM of the submitted query is before the DATE FROM of the cached query and the DATE TO of the submitted query is after the TO DATE TO of the cached query. In this case, the desired articles are a superset of the articles, which are cached in the database. As a consequence, the algorithm fetches all the cached results but it also performs a new search for articles in the date intervals before and after the cached date interval. When the search procedure finishes, the algorithm updates the cache by extending it to include the new articles and by changing the DATE FROM and DATE TO attributes so that they can be properly used for future searches.
- **3rd Case**: the DATE FROM of the submitted query is before the DATE FROM of the cached query and the DATE TO of the submitted query is between the DATE FROM and DATE TO of the cached query. In this case, a portion of the desired articles exists in the cache. The algorithm first fetches all the results and then it filters out the articles, which are after the DATE TO date of the submitted request. Furthermore, a new search is initiated for articles not existing in the cache memory. For the new search the DATE FROM and DATE TO dates become the DATE FROM date of the submitted query and the DATE FROM date of the cached query.
- **4th Case**: The form case is similar to the third case but in the opposite date direction. The final results consist of the cached results between DATE FROM date of the submitted request, the DATE TO date of the cached request and the new articles coming from a new search between the DATE TO date of the cached query and the DATE TO date of the submitted query.

We should notice that for the cached results data, an expiration mechanism is deployed. Every cached query is valid for a small number of days, in order to keep the engine's output to the end user as accurate as possible. Whenever a search for a matching with the cached results is performed, cached date that have expired are deleted from the database and are replaced with new ones. It is also possible for the same query to exist more than one cached records as long as they have not expired. The selection of the proper expiration time for the cached data will be discussed in section 5 of the paper.

By examining the cache matching algorithm, operating in the server machines, we can see that in all four cases, we achieve to limit the computational overhead of a fresh search in the database by replacing it with some overhead for cached results filtering. However, this filtering is implemented with simple XML parsing routines and cannot be considered as a heavy task for the server. The most significant improvement happens in the first case, where no new search is performed and all the results are fetched directly from the cache. This is a great benefit to our method as this is the most common case, where the user submits the same query over and over without changing the DATE FROM and DATE TO date fields or by shrinking the desired date borders. The worst case is the second, where the user expands his query in both time directions (before and after) in order to get more results in the output. In this case, the engine has to perform two new searches, followed by an update query in the database cache. However, this is the rarest case, as the average user tends to shrink the date interval rather than expanding it, when he repeatedly submits an identical query in order to get more date-precise and date-focused results. In the other two situations, one new search is executed each time and one update is committed in the database. This means that in an average case, we can save more than 50% of our computational overhead when the expansion of the date borders (with the newly submitted query) are not bigger than the cached results date interval.

4. EXPERIMENTAL EVALUATION

In our experiment to evaluate the caching algorithm, which was described in the previous section, we create a virtual user to submit queries to the server. The executed queries consist of keywords from several thematic categories (sports, science, politics, etc) used throughout the articles database of our system. We choose to test caching performance on queries containing no more than three keywords, in order for the

output to contain a big number of articles and for the overall procedure to last as much as needed for our time measurements to be sufficient and capable of analysis and conclusions.

4.1 Caching Algorithm

In the previous section of this paper, we analyzed the way in which the algorithm tries to match a submitted query to find an identical cached record. During the experiment, we tested several queries, requesting articles from different categories, covering the period of the last six months. In the first phase, we used an empty cache memory and the server was configured to have the caching feature disabled. As it was expected, queries consisting of very focalized and specific keywords were processed very quickly. These queries are not of high interest concerning our analysis, as the number of articles containing such keywords are always quite limited and require small computational time to process.
The major problem exists with queries consisting of generic keywords, which can be found on a plethora of articles in the database. This class of queries makes heavier usage of system resources and can be considered as a good starting point to evaluate our method. In Figure 2, we can examine the results of caching on execution procedure speedup for three generic queries (‘sports’, ‘computers’, ‘health or body’), which returned over 5000 articles. This graphic depicts the time in seconds that the system needed to fetch the matching output from the database. The cases considered in this figure are cases 2, 3 and 4 of our algorithm, where only a subset of the results for the submitted query exists in the cache memory and the system will initiate a new search in the database to fetch articles for the missing date periods. The selection of the date period, for which the results were cached in the first place, before the actual queries were submitted, was a random number of days varying from 60 to 90. The actual query, which was to be evaluated, required articles published in the last 180 days. This means that the system had still to search for more articles than the number of articles it had already stored in its cache memory. In the results presented, we can notice that under some situations, the benefit reached almost half the time of the actual (without caching) time needed. As it was expected, the worst case is case 2, where two new un-cached searches have to be executed, one before and the other after the date period of the cached set. After that, we come up with three different sets of articles. Prior to presenting them to the end user, we have to resort them according to their degree of relevance to the initial query. For cases 3 and 4, the results are almost similar. The higher times in case 4, could be a consequence of a possibly high concentration of desired articles in the date period, for which the new search was initiated, combined with a reduced concentration of articles in the date period stored in the cache memory of the server.

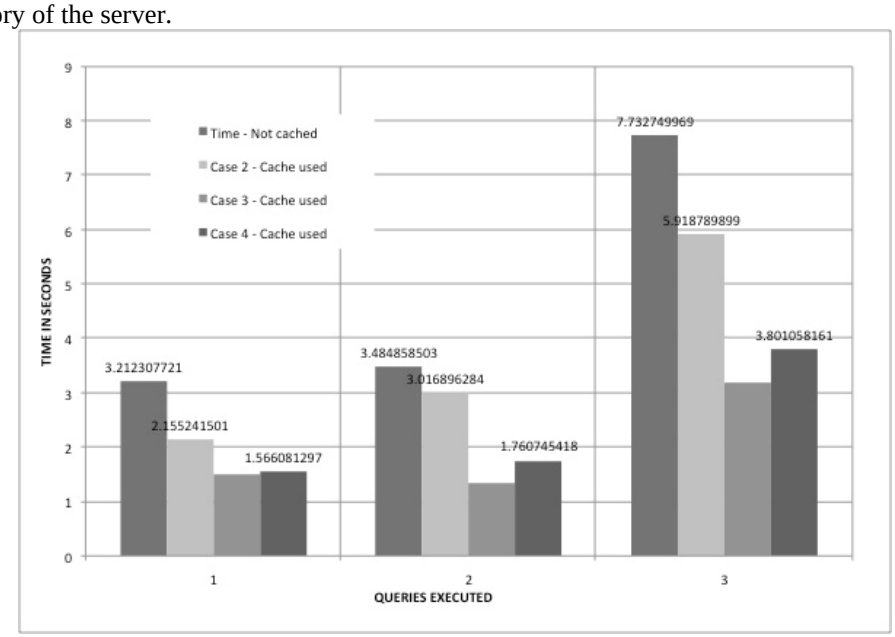

Figure 2: Time in seconds for un-cached searches and cached searches for Cases 2, 3 and 4

In the execution times measured throughout the experiment, an average 0.1 seconds were needed to fetch the articles from the cache memory, which is at average almost 3% of the overall time needed. Another 2% of the time was spent on resorting the two or three sets of results, according to their relevance to the query, in order to present them to the end user in the right order of relevance to the query. This said, it is expected for the case 1 of our algorithm to achieve an almost 95% speed up on the search. After the first execution of these queries, every next submission of the same request is serviced in under 0.1 seconds. Whenever results are cached for a query, every following identical one which demands articles inside the date period of the cached result, will be processed in almost zero time – only the time needed to fetch the results from the cache - no resorting is required in this case as we have only one already sorted set of articles. This reduces the computational overhead on the server for time demanding queries to the cost of the search procedure for only the first time they are executed. Every next time they are processed through the cache memory and the algorithm operating on it.

4.2 Cache Memory Size

Our second concern was to examine how the number of the cached articles per query in our cache, affects the overall algorithm performance and the size of the database table used to store the cached data. We executed a generic query for several numbers of cached articles by increasing each time the date period in which the caching occurred. The total number of articles for this query was 4782 over a period of 4 months. For this experiment, we tested cases 2,3 and 4 of our algorithm, so that in every submitted request, a part of the results were not contained in the cache and the engine could not rely only on the cached data to create the output.

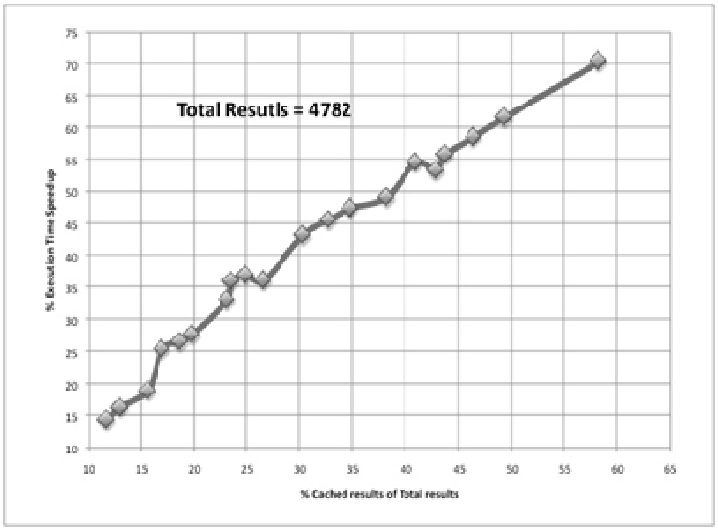

Figure 3: How the number of cached articles affects the speedup of a new search

From the graphical representation of Figure 3, relating the percentage of execution time speedup with the percentage of cached results on total results, we can notice that the search execution time reduces at an average 50% when a little less than 40% of the output has been cached. As the total number of articles in this test covered a period of four months, we can say, by statistic, that the 40% of the results would be retrieved by a search in a period of less than two months, which, speaking modestly, is a rather limited date interval on a common search. By that, it is meant that if a user submitted a search query, requesting articles for a period of more than two months, then every next time he submits an identical request, it would take at most half the time to be processed. If we add to this the fact that the algorithm updates the cache memory with new results, every time an extended (in terms of dates) version of an already cached query is submitted (the percentage of cached results probably increases and never decreases in every search), we could get even more improved execution times.

Due the fact that the algorithm stores for each query in the cache a limited set of information relative to the retrieved articles, such as ids, dates and relevance factors, the size of the cache per record in the server

memory is kept at minimum. As an example, for caching the 4782 results of the above query, which is a rather generic one with a lot of articles to be found relative, the corresponding row size in the cache database table was measured to be less than 150KB. If we combine the small row size with the periodical deletion of cached query records that expire, the technique can guarantee low storage space requirements in the server. The selection of the expiration date will be discussed in the next paragraph

4.3 Expiration Date and Result Accuracy

In the last phase of the experiment, we will examine the impact of selecting a proper expiration time for the cached records on the accuracy and the quality of the final output to the end user. As it was mentioned in the previous paragraph, the proposed algorithm periodically deletes cached records from the corresponding table in the database. The implementation of such an expiration mechanism in the algorithm is essential not only because it helps in keeping the storage space of the server's cache low, but mainly for keeping the accuracy and the quality of the search results at high levels.

Our purpose in this last step of the experiment is to examine how extending the expiration time of the cached records degrades the accuracy of the output result. For this reason, we created a virtual user and constructed a profile for him with favourite thematic categories and keywords. Having no cached data for this user on the first day of the experiment, we had him submit several queries to the system and we cached the results for some of these queries. For the next days of the month, we had the user navigating inside the system by submitting every day several different queries, this time, without caching any of them or expanding the already existing cached results. Among the submitted queries, we included queries identical to the cached ones for comparison to be feasible. The personalization mechanism the portal takes into account the daily behavior of each registered user (articles he reads, articles he rejects, time spent on each article) and dynamically evolves the profile of the user. For example, it is possible for a user to choose the sports as his favorite category upon his registration, but he may occasionally show an increased interest over science related news. The personalization system then evolves his profile and starts feeding him with scientific news among the sport news and this evolution has an obvious impact in his search sessions inside the system.

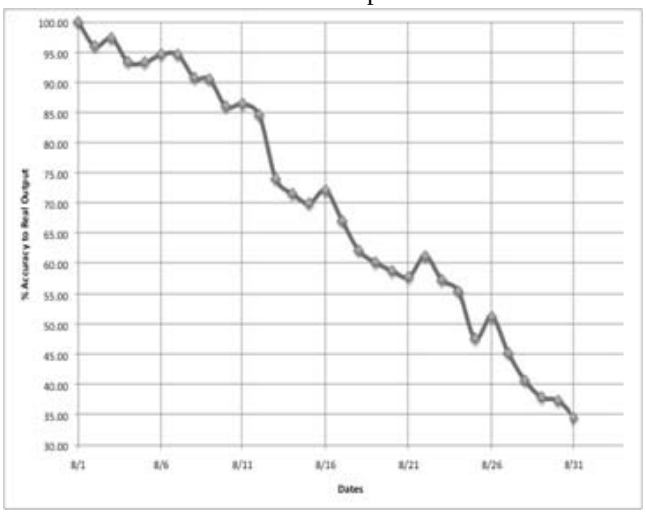

Figure 4: How extending date expiration affects results accuracy

In figure 4, we can see how is the accuracy of the search result degraded over the days passing when comparing the actual search results with the cached ones. For our virtual user, on the first day, the average accuracy is obviously 100% as it is the day that the actual queries are cached. Every next day, we get the results (uncached) of the actual queries, which have relevance over 35% to the submitted requests, and we count at average, how many of them existed in the cached queries results. As time passes, the output of the actual queries change (according to the user's evolving profile) and the average percentage of the cached results in the actual output decreases. Until the tenth day of the experiment, we can see that the accuracy is close to 90% to the actual results. However, after the first two weeks the accuracy is degraded at 70% and toward the end of the third week, it is close to 55%. In other words, if the user were to be presented, at this

point, with the cached results (cached on the first day), instead of the actual results for his queries, he would see almost, only half of the results that match his changed (since first day) profile.
As a conclusion, caching the results of a search for more than two weeks is not a preferable solution for a registered user, as it might significantly produce invalid results, not matching his evolving profile and preferences. However, for unregistered users (guests) of the system, for whom no profile has been formed, an extended expiration date could be used. In our implementation, there is a distinction for registered and unregistered users when checking for cached data, which makes the caching algorithm more flexible.

5. CONCLUSION AND FUTURE WORK

Due to the dynamism of the Web, the content of the web pages change rapidly, especially when discussing about a mechanism that fetches more than 1500 articles on a daily basis and presents them personalized back to the end user. Personalized portals offer the opportunity for focalized results though, it is crucial to create accurate user profiles. Based on the profiles that are created from the mechanism we created a personalized search engine for our web portal system in order to enhance the searching procedure of the registered and the non-registered. In this paper, we discussed about the caching algorithm of the advanced search sub-system. We presented the algorithmic procedure that we follow in order to cache the results and how to utilize the cache in order to enhance the speed of the searching procedure. Finally, we described experimental procedures that prove the aforementioned enhancement in speed. Comparing the results to the generic search's results it is obvious that the system is able to enhance the searching procedure and help the users locate the desired results quicker. For the future what we would like to do is to further enhance the whole system with a more accurate search personalization algorithm supporting "smarter" caching of data in order to make the whole procedure faster and in order to omit any results that are of very low user interest.

REFERENCES

Beitzel S. M., Jensen E. C., Chowdhury A., Grossman D. A. and Frieder O., 2004. Hourly analysis of a very large topically categorized web query log. *ACM SIGIR, pp. 321-328.*

Bouras C., Poulopoulos V., Tsogkas V. 2008. PeRSSonal's core functionality evaluation: Enhancing text labeling through personalized summaries. Data and Knowledge Engineering Journal, Elsevier Science, 2008, Vol. 64, Issue 1, pp. 330 - 345

DMOZ. *Open directory project*. http://www.dmoz.com

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 24, pp. 51–78*.

Google. *Google Search Engine*. http://www.google.com (last accessed March 2009)

Jansen B. J., Spink A., 2006. How are we searching the World Wide Web? A comparison of nine search engine transaction logs. *Information Processing and Management, 42(1) ,pp. 248-263.*

Lempel, R., Moran, S., 2003. Predictive caching and prefetching of query results in search engines. Proceedings of the 12th WWW Conference, pp. 19–28

Markatos, E.P., 2001. On caching search engine query results. Computer Communications 24, pp. 137–143.

Teevan J., Adar E., Jones R. and Pott M., 2006. History repeats itself: Re-peat queries in Yahoo's logs. *ACM SIGIR, pp. 703-704.*

Xie, Y., O'Hallaron, D.R., 2002. Locality in search engine queries and its implications for caching. IEEE Infocom 2002, pp. 1238 – 1247.

Yahoo. *Yahoo Web Search.* http://www.yahoo.com (last accessed March 2009)