Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 00 (2010) 1–8

Procedia Computer Science

International Conference on Computational Science, ICCS 2010

A vector model for routing queries in web search engines

Mauricio S. Oyarzun[a], Senen Gonzalez[b], Marcelo Mendoza[c], Flavio Ferrarotti[a,c], Max Chacon[a], Mauricio Marin[a,c,*]

[a] *Universidad de Santiago de Chile*
[b] *Universidad de Chile*
[c] *Yahoo! Research Latin America, Santiago de Chile*

Abstract

This paper proposes a method for reducing the number of search nodes involved in the solution of queries arriving to a Web search engine. The method is applied by the query receptionist machine during situations of sudden peaks in query traffic to reduce the load on the search nodes. The experimental evaluation based on actual traces from users of a major search engine, shows that the proposed method outperforms alternative strategies. This is more evident for systems composed of a large number of search nodes which indicates that the method is also more scalable than the alternative strategies.

Keywords: Parallel and Distributed Computing, Information Retrieval, Web Search Engines

1. Introduction

Web search engines are composed of a large set of P *search nodes* that communicate each other via message passing. Their objective is to quickly determine the top N documents that best match a given user query. This is achieved by means of parallel query processing upon the master/slaves paradigm. The overall steps involved in the solution of a query are as follows. A query receptionist master machine, called *broker*, receives a user query and checks if the query is stored in a results cache. This broker's cache contains the answers to most frequent queries. If the query is not in the results cache, the broker sends the query to the P search nodes (slaves) for solution. Each search node logically holds a different $1/P$ fraction of the collection of Web documents (html files, pdf files, etc.). Thus each search node calculates in parallel its best N results for the query and sends them to the broker. Finally, the broker merges the P sets of results, determines the top N results and produces the answer Web page.

This paper proposes a method for reducing the average number of search nodes participating in the solution of queries. The method calculates approximated results for queries. Since for a large scale system it holds that $N < P$, the non-trivial problem consists on predicting in advance what are the best search nodes for solving a query. The importance of the problem is given by the fact that data centers hosting major search engines can easily contain thousands of search nodes, which are set to efficiently process tens of thousands of user queries per second. In this context, reducing the amount of hardware resources devoted to the solution of queries is a relevant issue for a number reasons such as power consumption.

*

Email address: mmarin@yahoo-inc.com (Mauricio Marin)

The proposed method finds its best application in cases in which the search engine is subjected to sudden peaks in query traffic. Data centers work with enough redundancy to keep search nodes operating at 30% or less utilization. This allows them to cope with peaks without compromising user query latency. Sudden peaks are usually of short duration since queries that become very popular, quickly become trapped in different special-purpose caches located in the broker machine and search nodes. Our method allows the search engine to operate in steady state with a smaller number of search nodes than the standard case, and provide good-quality approximated query results during periods of peaks in query traffic. In other words, use less search nodes (each operating at a higher utilization in steady state) and treat momentary peaks by responding approximated answers to queries. The quality of the approximation critically depends on the smart selection of search nodes. This paper proposes a method for this purpose.

The state of the art methods for our setting have been presented in [9, 16, 19]. Our method outperforms previous ones for large number of search nodes, which shows that the method is more scalable. In addition, it can be used in combination with others to improve quality of results.

2. Related Work

Regarding caching strategies, one of the first ideas studied in literature was having a static cache of results (RCache abridged) which stores queries identified as frequent from an analysis of a query log file. Markatos et al. [17] showed that the performance of the static caches is generally poor mainly due to the fact that the majority of the queries put into a search engine are not frequent ones and therefore, the static cache reaches a low number of hits. In this sense, dynamic caching techniques based on replacement policies like LRU or LFU achieved a better performance. In another research, Lempel and Moran [13] calculated a score for the queries that allows for an estimation of how probable it is that a query will be made in the future, a technique called Probability Driven Caching (PDC). Lately, Fagni *et al.* [6] proposed a structure for caching where they maintain a static collection and a dynamic one, achieving good results, called Static-Dynamic Caching (SDC). In SDC, the static part stores frequent queries and the dynamic part handles replacement techniques like LRU or LFU. With this, SDC achieved a hit ratio higher than 30% in experiments conducted on a log from Altavista. Long and Suel [15] showed that upon storing pairs of frequent terms determined by the co-occurrence in the query logs, it is possible to increase the hit ratio. For those, the authors proposed putting the pairs of frequent terms at an intermediate level of caching between the broker cache and the end-server caches. Baeza-Yates *et al.* [2] have shown that caching posting lists is also possible, obtaining higher hit rates than when just doing results and/or terms. Recently, Gan and Suel [10] have studied weighted result caching in which the cost of processing queries is considered at the time of deciding the admission of a given query to the RCache.

A common factor that is found in the previous techniques is that they attempt to give an exact answer to the evaluated queries through the cache exactly. The literature shows that the use of approximation techniques like semantic caching can be very useful for distributed databases. A seminal work in this area is authored by Godfrey and Gryz [11]. They provide a framework for identifying the distinct cases that are presented to evaluate queries in semantic caches. Fundamentally, they identify cases which they call semantic overlap, for which it is possible for the new query to obtain a list of answers from the cache with a good precision. In the context of Web search engines, Chidlovskii *et al.* [4, 3] poses semantic methods for query processing. The proposed methods store clusters of co-occurring answers identified as regions, each of these associated to a signature. New frequent queries are also associated to signatures, where regions with similar signatures are able to develop their answer. Experimental results show that the performance of the semantic methods differs depending on the type of semantic overlap analyzed. Amiri *et al.* [1] deals with the problem of the scalability of the semantic caching methods for a specific semantic overlap case known as query containment.

A related work to our objective is authored by Puppin et al. [19]. The authors proposed a method where a large query log is used to form P clusters of documents and Q clusters of queries by using a co-clustering algorithm proposed in [5]. This allows defining a matrix where each entry contains a measure of how pertinent a query cluster is to a document cluster. In addition, for each query cluster a text file containing all the terms found in the queries of the cluster is maintained. Upon reception of a query q the broker computes how pertinent the query q is to a query cluster by using the BM25 similarity measure. These values are used in combination with the matrix co-clustering entries to compute a ranking of document clusters. The method is used as a collection selection strategy achieving good precision results. At the moment this method is considered as state-of-the-art in the area. This method, namely PCAP, will be evaluated against our method in the experimental results section.

The another similar work to our objective is our prior work [9]. Here we propose using an LCache as a semantic cache. The method reduces the visited processors for queries not found in an LCache. The semantic method for evaluating new queries uses the inverse frequency of the terms in the queries stored in the cache (Idf) to determine when the results recovered from the cache are a good approximation to the exact answer set. The precision of the results differs depending on the semantic overlap case analyzed. In the context of this paper we call this method SEMCACHE and it will be evaluated against our method in the experimental results section. Recently [16], we show that the combination of an LCache and an RCache using dynamic caching strategies is able to achieve efficient performance under severe peaks in query traffic, as opposite to the strategy that only uses the RCache that gets saturated as they are not able to follow the variations in traffic.

3. Proposed Method

In this section, we propose a new query routing method that elaborates at the broker side a list of search nodes capable of producing a good approximated answer to queries. To do this, we use the contents of the location cache (LCache) described in the previous section of this paper. Each entry in the location cache consists of a list of frequent queries and for each of the queries, the list of search nodes that allow them to calculate the top-k answers. To avoid that evaluation in the actual search nodes when a query makes a hit in the location cache, though at the expense of more use of space, it is recommended to store the ID of each document in the location cache as well.

When an user query arrives to the search engine and has been previously stored in the location cache (hit) we only visit the search nodes that the location cache has indicated for the query. In this way, we avoid evaluating the query using all of the search nodes (broadcast) obtaining identical results for top-k results as in the case in which the query is sent to all search nodes. When the query is not stored in the location cache we are forced to evaluate it in all of the search nodes. To avoid this situation, we propose a method that is capable of predicting which search nodes contain the best answers for each new query.

To make a list of likely search nodes for each new query, we will explore the relationships between the query terms stored in the location cache and the lists of search nodes. Logically, when a search node responds to more queries that contain a given term, we can affirm that this term describes the documents stored in that search node well. Upon exploring the relationships between the queries and the lists of search nodes stored in the location cache, we avoid constructing representations of the contents of each search node using the text from the documents stored in each search node. This idea has been previously explored in [19]. The main inconvenience of it is the high computational cost involved in processing the whole collection and the large amount of space required to represent the vocabulary of the collection in the broker.

Instead, the method proposed in this paper utilizes the terms stored in the location cache to construct a representation of each search node. Let V be the vocabulary of the location cache, meaning the collection of terms that make up the queries stored in the cache. Let M be the size of V (the number of terms that make it up). We construct a vector representation for each search node on the M-dimensional space of terms formed by V. Let S be the vector representation of a search node. Let t_i be the i-th term that belongs to V. The i-th component of S is given by $S[i] = Q_{i,s}$ where $Q_{i,s}$ represents the number of queries in the location cache that have at least one document inside its top-N best answers in the server S and also contains the term t_i. When t_i has been used in more queries that contain at least one relevant document in S, t_i will be a better descriptor of the documents stored in S.

To determine how relevant each search node is for each new query, we calculate a score that evaluates the similarity between the query and the vector representation of each search node. Let q be a query not stored in the location cache. Let $\mathrm{Sim}(q, S)$ be the similarity between q and the search node S. We calculate $\mathrm{Sim}(q, S)$ in the following way:

$$\mathrm{Sim}(q, S) = \sum_{\forall t_i \in q} \frac{Q_{i,s}}{Q_i},$$

where Q_i corresponds to the sum of the values of $Q_{i,s}$ for all of the search nodes. We observe that we add the components from the vector of each search node S considering all the terms of q. This means that when q has more terms in the location cache, we display more evidence to construct the ranking of the search nodes. Upon incorporating the variable Q_i what we have evaluated is the relative support from each final server for each term, ignoring that they are the most frequent terms in the collection which may bias the vector model. We can also observe that in the case that the query does not share any terms with the location cache, Sim(q, S) is zero. This represents the situation in which we do not have enough information in the location cache to be able to produce the ranking of search nodes. In this case, these queries are sent to all of the search nodes.

Once we have calculated Sim(q, S) for each search node, we generate a list of search nodes ordered decreasingly by Sim(q, S). This list defines the order in which the method indicates to the broker should visit the search nodes. Also, we can truncate the size of this list by using some threshold criteria for the value of Sim(q, S).

The location cache works in tandem with the results cache. It is recommendable to store frequent queries in the results cache that are solved using many search nodes. This is because for this type of queries the location cache offers little benefits in cost savings. In the location cache it is convenient to store frequent and light queries to evaluate. This means it is recommendable to store queries that are associated to short lists of search nodes, in a way in which the benefits in cost savings with respect to broadcast are significant.

To determine which queries are recommended to store in the location cache, it is adequate to process query logs periodically. In this way we can recalculate the collections of queries that are recommendable to be stored in the results cache and the location cache. When updated often, the queries stored in the broker will be capable of reflecting the dynamics of the user preferences.

In terms of space complexity, it is necessary to store the location cache and the M-dimensional vectors in the broker. If we store only the non-null coefficients, the space occupied by the vector representation of each search node is $O(MP)$, where P represents the number of search nodes. In the case of the location cache the additional space required is $O(QP)$ where Q represents the number of queries stored in the broker.

4. Experimental Evaluation

4.1. Dataset and Document Distribution

The dataset we will use in this paper corresponds to a query log file of Yahoo! from the year 2009. The query log contains 2,109,198 distinct queries and 3,991,719 query instances. 1,629,113 distinct queries were considered as training data, leaving the 602,862 remaining queries for evaluation. The testing set considers 999,573 query instances in the query log trace. The training set considers 2,992,146 query instances. The vocabulary of the 3.9 million query instances is composed of 239,274 distinct query terms.

We initialize the location cache with the 150,000 most frequent queries in the training query log trace. The location cache contains 204,485 different terms. Out of the 602,862 distinct queries in the testing dataset, there are 453,354 queries that have at least one term in the set of terms of the location cache.

For each query considered in the query log data, we obtained the top-50 results retrieved using the Yahoo! Search BOSS API [21]. The generated document collection corresponds to 50,864,625 documents. The API uses the Yahoo! services to get the same answers that are presented to actual users by the production Yahoo! search engine.

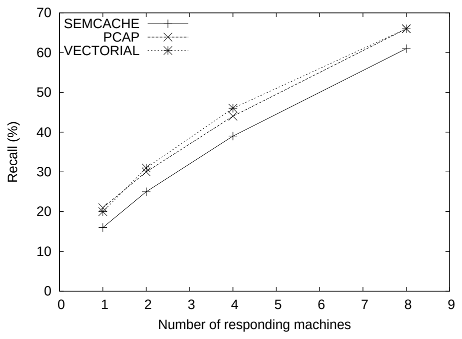
The document collection was distributed using a query-document co-clustering algorithm. The co-clustering algorithm allows us to find clusters of related documents. We choose to use this clustering algorithm because we wanted to compare our results with those of PCAP - the strategy of collection selection that is considered as state of the art in the area [19]. A **C++** implementation of the algorithm was used for the experiments. The source code of this algorithm is available in [18].

The document collection was distributed over two sets of search nodes considering 16 and 128 machines respectively. Each entry in the location cache stores the set of search nodes that have the top-20 results for the corresponding query.

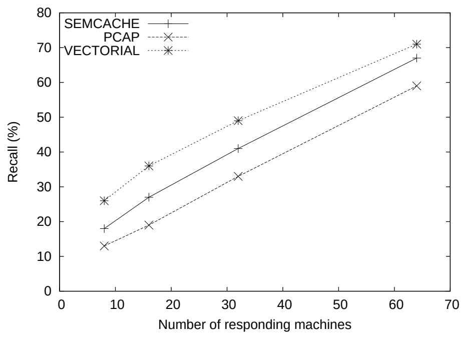
4.2. Performance Evaluation

To evaluate the performance of our method we compare the results coming from the predictive model with the results coming from the Yahoo! Search Engine. This is an effective approach to evaluate the performance in our distributed search engine because the document collection was built using the Yahoo! Search BOSS API. Thus, the best case for our method is to predict the machines where the top-N results for the testing queries are.

For the evaluation we will consider the state of the art approximated methods PCAP and SEMCACHE, described in Section 2. We tested the correctness of our PCAP implementation by using the query training set over the 16 search nodes. In this setting, PCAP obtained a $INTER_{20}$ measure [19] of 31.40%, 43.25%, 59.19%, and 77.68%, when we



16 search nodes



128 search nodes

Figure 1: Percentage of exact results obtained by retrieving top 20 results from the selected search nodes.

visit 1, 2, 4, and 8 search nodes, respectively. These $INTER_{20}$ values are very similar to those obtained by Puppin *et al.* [19] in their experiments. Regarding the SEMCACHE method, we use an Idf threshold equals to 10, as it is recommended in [9].

4.3. Comparing effectiveness

In Figure 1 we show results that compare our method, called VECTORIAL, against the PCAP and SEMCACHE methods for a total number of 16 and 128 search nodes. We define recall as the percentage of exact results retrieved by the search nodes involved in the solution of a query. The exact results for each query are obtained by using all of the search nodes. The Figure 1 shows that the proposed method is more effective in obtaining a better approximation to the exact answer to queries. The margin is wider for 128 processors which indicates a better scalability of the proposed method. Similar trend is observed for top 5 and top 10 results.

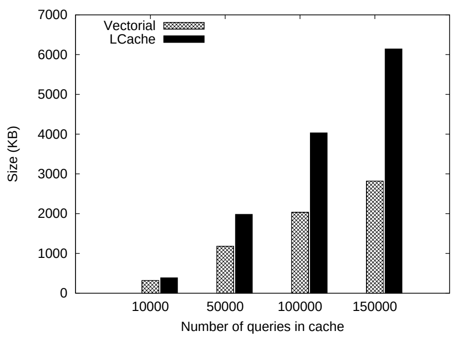
A relevant conclusion from the results is that by using half of the search nodes it is possible to respond queries with a good precision with respect to the exact results obtained by sending the query to all of the search nodes.

4.4. Comparing space used

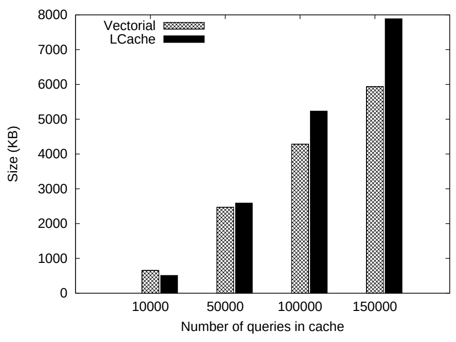
The proposed method (VECTORIAL) requires extra main memory space in the broker machine to rank the search nodes where to send a given query. In Figure 2 we show the space used by the method in comparison with the space used by the location cache (LCACHE for short). For 128 search nodes the space demanded by the VECTORIAL method increases relative to the LCACHE space because it has to use more bits to store the search node IDs than the case for 16 search nodes.

The SEMCACHE works on top of the location cache to rank in an on-line manner the set of search nodes. The PCAP method is an off-line method and does not require the location cache, but it uses more space than SEMCACHE [9]. The VECTORIAL method is *semi* off-line as we use the location cache to build the M-dimensional vectors. That is, the contents of the location cache are used to periodically re-build the vectors and follow in this way the changes in the user preferences. The results of Figure 1 show that with this relatively small set of data the method is able to achieve good results. Instead, the PCAP method requires a very large set of queries (millions) to be properly initialized.

Also notice that results cache maintained in the broker machine is expected to consume much more space than the space consumed by the location cache. Each entry in the results cache is of the order of several KBs whereas the location cache stores query terms and a list of integers representing search nodes IDs. Namely it consumes tens of bytes. This means that the space occupied by the location cache and the M-dimensional vectors is neglectable as compared with the results cache.

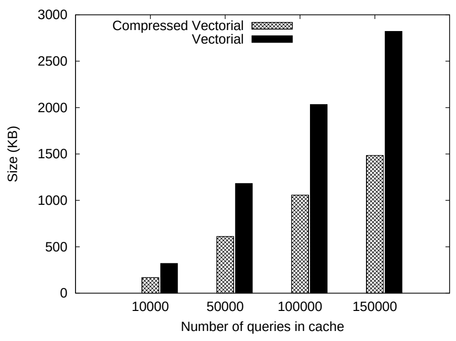


16 search nodes

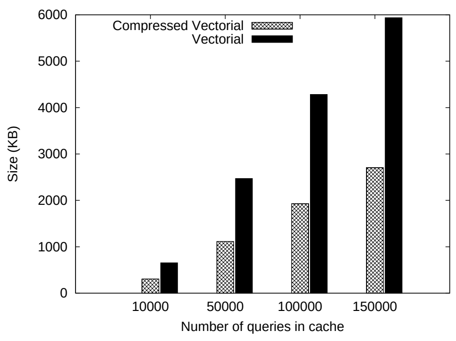


128 search nodes

Figure 2: Space used in the main memory of the broker machine.



16 search nodes



128 search nodes

Figure 3: Compressing the main memory space used by the VECTORIAL strategy.

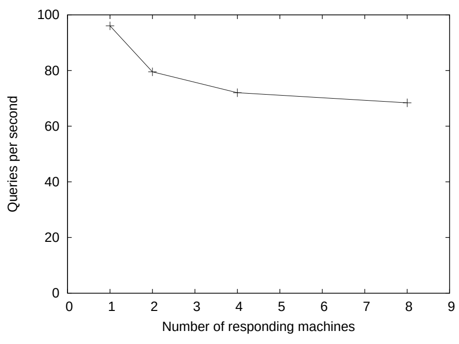
4.5. Compressing data

A relevant observation is that the M-dimensional vector are highly compressible. This can be observed in Figure 3. In all cases, the space occupied can be halved. To achieve this compression rate we applied the following procedure. For each M-dimensional vector we maintain two arrays of bits. The first one is a vector where each bit represents a different search node. If the bit is set to 1, then the associated search node has participated in the solution of the query with a certain frequency. The second array of bits is used to represent the frequencies in compressed format. We use delta-encoding to compress integers. The first bits array indicates to which search node corresponds the frequency represented in compressed format in the second bits array.

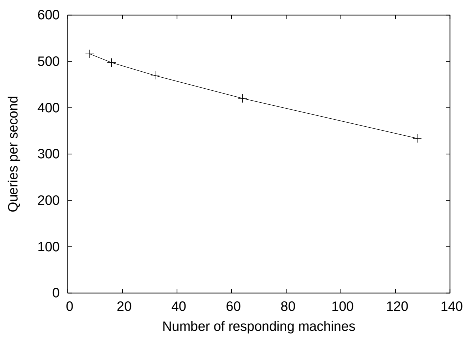
4.6. Impact in query throughput

Upon sudden peaks in query traffic, search nodes can be subjected to a high load which can increase user latency as a consequence of queuing at the different hardware devices. In this case, sending the queries to a reduced set of search nodes contributes to prevent the machines from being saturated, which contributes to increase query throughput in such a situation. We define query throughput as the ratio of total number of processed queries to total running time.

Figure 4 presents the average throughput achieved when the queries are sent to a few search nodes. The results show that there is an increase of about 30% in overall throughput when queries are sent to one or two search nodes. This indicates that with our approach there is room for search nodes utilization to be increased by at least 20% in steady state operation.



16 search nodes



128 search nodes

Figure 4: Query throughput achieved by routing queries to different number of search nodes.

5. Conclusions

This paper has proposed a vectorial method to solve Web queries in an approximated manner by using a fraction of the resources provided by a search engine composed of *P* search nodes. We have shown that a combination between a location cache used in the broker machine and our method to predict promising search nodes, can be a good alternative to efficiently react upon sudden peaks in query traffic. During this period the search engine responds approximated answers to ease the load on search nodes. This is made at reasonable precision by using less than a half of the search nodes. The implication for a data center is that it can operate less search nodes at a higher utilization than the standard approach, which is beneficial to overall power consumption, rate of device failures and maintenance costs.

The experimental evaluation based on actual user query traces shows that the proposed vectorial method outperforms alternatives approaches for the same problem. The gain is more evident for large number of search nodes. The space used by the vectorial method is not significant when compared against the PCAP method, whereas it requires reasonable extra space when compared against the SEMCACHE method. In this case the total space may be no more than twice the SEMCACHE space. Certainly, the gain is better quality approximated answers to queries. Nevertheless, we have observed that the space used by the vectorial method can be efficiently compressed (halved for large number of processors) by using fast delta-encoding at negligible running-time overheads.

However, a key drawback of the proposed method is that it requires proper training to produce good quality results. The experiments presented in the paper considered as training set the same location cache (LCache) used by the SEMCACHE method. This showed that the method is able to deliver good results with reduced training sets. We detected that the training period is short, namely upon a few hundred thousand queries the method started to perform fairly well. Certainly, user preferences can drastically change upon some sudden big-news event and this can potentially change the set of terms and documents over which the method has been trained in the recent past. Here the vectorial method loses effectiveness whilst it is re-trained. Hence, a reasonable strategy to follow is to use SEMCACHE during this period. As future work we plan to devise a strategy able to dynamically use both methods.

References

[1] K. Amiri, S. Park, R. Tewari, and S. Padmanabhan. Scalable template-based query containment checking for web semantic caches. In *ICDE*, 2003.

[2] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri. Design trade-offs for search engine caching. ACM TWEB, 2(4), 2008.

[3] B. Chidlovskii, C. Roncancio, and M. Schneider. Semantic Cache Mechanism for Heterogeneous Web Querying. Computer Networks, 31(11-16):1347-1360, 1999.

[4] B. Chidlovskii, and U. Borghoff. Semantic Caching of Web Queries. VLDB Journal, 9(1):2-17, 2000.

[5] I. Dhillon, S. Mallela and D. Modha. Information-theoretic co-clustering. In *KDD*. 2003.

[6] T. Fagni, R. Perego, F. Silvestri, and S. Orlando. Boosting the performance of Web search engines: Caching and prefetching query results by exploiting historical usage data. ACM TOIS, 24(1):51-78, 2006.
[7] F. Falchi, C. Lucchese, S. Orlando, R. Perego, and F. Rabitti. A Metric Cache for Similarity Search. In *LSDS-IR*, 2008.
[8] R. Fan, K. Chang, C. Hsieh, X. Wang, and C. Lin. LIBLINEAR: A library for large linear classification. Journal of Machine Learning Research, 9:1871-1874, 2008
[9] F. Ferrarotti, M. Marin and M. Mendoza. A Last-Resort Semantic Cache for Web Queries. In *SPIRE*, 2009.
[10] Q. Gan, T. Suel, Improved Techniques for Result Caching in Web Search Engines. In *WWW*, 2009.
[11] P. Godfrey, and J. Gryz. Answering Queries by Semantic Caches. In *DEXA*, 1999.
[12] S. Keerthi, S. Sundararajan, K. Chang, C. Hsieh, and C. Lin. A sequential dual method for large scale multi-class linear SVMs. In *SIGKDD*, 2008.
[13] R. Lempel, and S. Moran. Predictive caching and prefetching of query results in search engines. In *WWW*, 2003.
[14] C. Lin, R. Weng, and S. Keerthi. Trust region Newton method for large-scale logistic regression. Journal of Machine Learning Research, 9:627-650, 2008.
[15] X. Long, and T. Suel. Three-level caching for efficient query processing in large Web search engines. In *WWW*, 2005.
[16] M. Marin, F. Ferrarotti, M. Mendoza, C. Gomez, and V. Gil-Costa Location Cache for Web Queries. In *CIKM*, 2009.
[17] E. Markatos. On caching search engine query results. Computer Communications, 24(7):137-143, 2000.
[18] D. Puppin, and F. Silvestri. C++ implementation of the co-cluster algorithm by Dhillon, Mallela, and Modha. Available on *http://hpc.isti.cnr.it*
[19] D. Puppin, F. Silvestri, R. Perego, and R. Baeza-Yates. Load-balancing and caching for collection selection architectures. In *INFOSCALE*, 2007.
[20] G. Tsoumakas, I. Katakis. Multi-label Classification: An Overview. International Journal of Data Warehousing and Mining, 3(3):1-13, 2007.
[21] Yahoo! Search BOSS API. Available on *http://developer.yahoo.com/search/boss/* , 2009.
[22] H. Yan, S. Ding and T. Suel. Inverted index compression and query processing with optimized document ordering. In *WWW*, 2009.