Space Efficient Caching of Query Results in Search Engines

Rifat Ozcan
Dept. of Computer Engineering
Bilkent University
06800, Bilkent Ankara, TURKEY
Email: rozcan@cs.bilkent.edu.tr

Ismail Sengor Altingovde
Dept. of Computer Engineering
Bilkent University
06800, Bilkent Ankara, TURKEY
Email: ismaila@cs.bilkent.edu.tr

Özgür Ulusoy
Dept. of Computer Engineering
Bilkent University
06800, Bilkent Ankara, TURKEY
Email: oulusoy@cs.bilkent.edu.tr

Abstract- **Web search engines serve millions of query requests per day. Caching query results is one of the most crucial mechanisms to cope with such a demanding load. In this paper, we propose an efficient storage model to cache document identifiers of query results. Essentially, we first cluster queries that have common result documents. Next, for each cluster, we attempt to store those common document identifiers in a more compact manner. Experimental results reveal that the proposed storage model achieves space reduction of up to 4%. The proposed model is envisioned to improve the cache hit rate and system throughput as it allows storing more query results within a particular cache space, in return to a negligible increase in the cost of preparing the final query result page.**

I. INTRODUCTION

Web search engines (WSEs) serve millions of query requests everyday. They are optimized to answer as many queries per second as possible with most relevant set of results and within a reasonable amount of time. Such query load and response time constraints require immense amount of computing resources to be available. Furthermore, several intelligent mechanisms are employed to improve the efficiency and scalability of the overall system.

One of the most important techniques used for this purpose is caching. It is observed that, despite the very high volume of the requests per day, the diversity of the queries does not change dramatically. That is, a relatively small portion of the queries are asked frequently and they dominate the query requests. This fact leads to caching mechanisms for popular query results and/or terms to reduce the query request load on servers.

Search engines typically cache either the query results or posting lists for query terms, or both [4]. For each case, static or dynamic caching (and even a hybrid of both) can be applied. In this paper, we focus on the static caching of the query results. Static caching approach typically stores the results of the most popular (i.e., frequent) queries that are obtained from the previous query logs of that WSE (please see [12] for alternative query selection strategies to fill the static cache). It is a read-only cache and no entry of the cache is replaced until the next refresh time of the static cache.

A static cache can store the query results in two ways. In a, so-called, *docID cache* [6] only the document identifiers of the query results are stored. The snippets and the final result page are generated each time a query request yields a cache-hit. An alternative to this approach is a *snippet cache*, which stores the final HTML result pages (including snippets, etc.) to be displayed upon a request that yields a cache-hit. Clearly, in the former approach, the cache can store more items; but a cache-hit still needs some processing for preparing the result page, whereas the latter approach stores less items in the same space, but the results can be sent immediately to the user once a query is found in the cache.

In this study, we propose a storage mechanism for static docID caching. Our approach exploits the overlaps in the results of similar queries, which are identified by clustering queries. Intuitively, we presume that there may exist several query clusters that share a set of documents in their member queries' results, and we try to encode these common document identifiers in a compact form, to better utilize the static cache space and increase the hit rate. In the literature, query clustering is essentially exploited for better answer ranking and query recommendation purposes (see Section II for a detailed discussion). To the best of our knowledge, our approach is the first attempt to use query clustering for efficient storage of query results.

The rest of the paper is organized as follows: Section II presents the related work on caching and clustering queries. We describe the details of our storage approach in Section III. Experimental evaluation and discussion of the results are provided in Section IV. Finally, we conclude the paper and point future research directions in Section V.

II. RELATED WORK

A. *Caching for Search Engines*

Caching is one of the key techniques for search engines to cope with the high query loads. Typically, search engines cache the query results or the posting lists for query terms, or both. Caching the posting lists may lead higher cache hit ratios, simply because the same query terms may appear in several different queries (e.g., see [4]). On the other hand,

978-1-4244-2881-6/08/$25.00 ©2008 IEEE

caching query results would provide more gains in terms of efficiency, especially when the network communication dominates the query processing costs. Furthermore, in the latter case, there is no need for query processing, and it is adequate to simply send the results to the user.

In one of the earliest works, Markatos [11] analyses the EXCITE query log and shows that static query result caching is a good choice for small cache sizes, but dynamic caching is better for large cache sizes. In [6], a static-dynamic caching policy is proposed: cache is divided into two parts and one part is reserved for static caching and the other is used for dynamic caching of the result pages. This work also mentions two possible ways of caching results: either document identifiers (docID cache) or snippets (i.e., HTML cache); however they do not discuss the actual storage details for these cases.

Several other works [13, 4, 3, and 10] further investigate the possibility of caching posting lists. To the best of our knowledge, [13] is the first work in the literature which mentions about two-level caching idea which combines caching of the query results and posting lists. [2] proposes a three level search index structure using the query log distribution.

[15] proposes another caching framework for databases using Bloom filters [5] as a lookup mechanism in the cache. Bloom filters [5] are space efficient data structures which achieves set membership check. In this study, our focus is not to propose another lookup structure for query result caching but we try to optimize the actual storage of document ids in the cache by exploiting the overlaps in similar queries formed by query clustering.

As far as we know, none of the above works discuss how the results (docIDs or snippets pages) are actually stored in a static or dynamic cache. Such practical details of the commercial search engines are also not publicly available. Thus, we basically assume that the docID cache includes a simple list of top-*K* integers (as provided by the query processor) per query and the snippet cache includes HTML pages that can be directly displayed to the user for a given query. Our goal in this paper is to devise a more compact storage scheme for storing docID's in a static cache.

B. *Query Clustering*

Query clustering is previously proposed for two main goals: a) enhancing result ranking, and b) query recommendation. [1] uses clicktrough data, which consists of <query, url> pairs, in order to find related queries and related URLs by clustering. The proposed approach is to see the query log as a bipartite graph between the sets of queries and sets of URLs. Then, two particular vertices (one from the queries and the other from the URLs) are connected by an edge if such a pair occurs in the query log. Finally, clustering is performed on this bipartite graph. The proposed clustering is a kind of hierarchical agglomerative clustering [7]. The similarity of two vertices in this graph is calculated by the overlap on neighbors. If two queries have more common URLs then their similarity is high. After finding query clusters, they can be used to assist users by suggesting alternative (and potentially related) queries during Web search. That is, for a given user query, the system determines its cluster and suggest other queries from the same cluster. The proposed approach is evaluated by the number of user clicks on this suggested alternative query links.

In [3], queries are clustered by using the content of documents that are clicked by the users. These documents are represented by the well-known vector space model and clustered by using the k-means algorithm. After query clustering, this structure is used for 2 applications: a) Answer ranking: The popularity of the results in a query's cluster are employed to re-rank the original results of that particular query. b) Query recommendation: When a query is submitted, its query-cluster is found and queries are recommended based on their similarity to the query and their support based on the query log.

[9] also performs query clustering using the logs. Similar to the above approaches, they also claim that using only query keywords to do query clustering is not successful since queries are very short and words have polysemic meanings (e.g. "java"). They propose to use the common documents which are clicked for the queries to measure the similarity between those queries. This cross-reference information between the queries and clicked documents is shown to be effective for query clustering and better than using the either one of the query keywords or logs alone.

In this paper, we use the result lists for forming query clusters and exploit these clusters for utilizing the storage of these results. Since we focus on storing the entire result list per query, we do not use solely the clicked documents, differing from the other works mentioned above. To the best of our knowledge, query clusters are not used previously for utilizing the cache storage space in this manner.

III. Our Approach

Our approach consists of two steps: a) Query clustering, and b) Storage of query results. Each of these steps is explained in detail in the following subsections.

A. *Query Clustering*

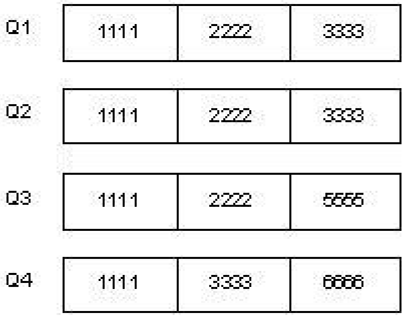
We applied the single link (linkage) hierarchical clustering algorithm [7] for query clustering. The aim of the clustering in our context is to find overlaps between result lists of similar queries. Therefore, each query is represented by its *n* result document numbers. Clustering algorithm works as follows:

At the beginning of the clustering phase, one cluster is formed for each query in the dataset. Assume two queries Q_i

Documents = {1111, 2222, 3333, 4444, 5555, 6666}

Q1: texas lotto = {1111, 2222, 3333}
Q2: texas lottery = {1111, 2222, 3333}
Q3: texas lottery results = {1111, 2222, 5555}
Q4: texas lottery numbers = {1111, 3333, 6666}

Shared documents = {1111, 2222, 3333}



Q1	1111	2222	3333
Q2	1111	2222	3333
Q3	1111	2222	5555
Q4	1111	3333	6666

Fig. 1. Conventional cache storage mechanism for queries Q1, Q2, Q3 and Q4

and Q_j have the following result sets, each containing top-*n* result document identifiers.

$$R_i = \{r_{i1}, r_{i2}, \ldots, r_{in}\}$$
$$R_j = \{r_{j1}, r_{j2}, \ldots, r_{jn}\}$$

Then the similarity between these two queries (actually clusters in the context of the algorithm) is computed by the following formula which considers the fraction of intersection:

$$sim\_measure_{ij} = \frac{|R_i \cap R_j|}{\min(|R_i|, |R_j|)} \qquad (1)$$

At each step of the single link clustering, most similar cluster pair is chosen whose similarity value is greater than a predefined minimum similarity threshold. If such a cluster pair is found, they are merged and the union of result lists in each cluster in this pair constitutes the result list for the new cluster. This process continues until no cluster pair satisfying the minimum similarity threshold can be found or all clusters are merged to one cluster, which is practically not possible for a real query log.

B. Storage of Query Results

In this section, we present the details of our storage mechanism exploiting the query clusters obtained in the previous stage. Note that our focus is to improve the actual storage of result lists of queries in the static cache and we do not address any lookup mechanisms in this paper.

We present our storage mechanism by the following simple example. Assume that we have obtained the query cluster containing queries Q1, Q2, Q3 and Q4 as given in Fig. 1. For the sake of simplicity, let each query store only top-3 results.

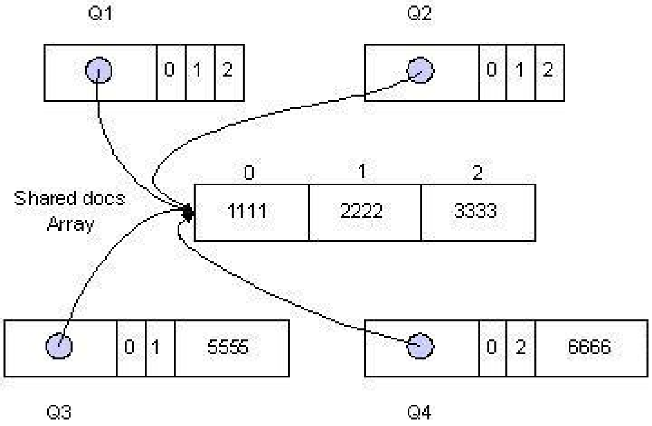


Fig. 2. Our storage mechanism exploiting query clustering

We also give hypothetical document ids for each result document. The shared documents in this cluster are {1111, 2222, 3333}.

In Fig.1, we illustrate the conventional storage scheme for the results of these four queries. In this case, it is assumed that the query results are simply kept as a list of document identifiers, each requiring 4 bytes of storage. Then, the total storage space required for query results is 48 bytes (excluding the space required for lookup mechanism and the queries themselves). Note that, as it is mentioned before, there is no earlier work on storage mechanisms for query result lists in the context of caching. Although commercial Web search engines definitely employ static and dynamic caching mechanisms, details are not exposed. So, we use the simple storage scheme described above as the *baseline* in this paper.

In the above scenario, it is seen that the shared document ids are stored several times in the cache. Our storage approach exploits this overlap of document ids in the query clusters. Fig. 2 shows the general structure of our approach. A *shared-documents* array is constructed for the overlapping result document ids in a cluster. For each cluster, its shared-documents array includes the top-256 documents that have the highest frequency among the results of the queries in that cluster. The result list of a query stores the array index for the shared documents, which can be expressed in only 1-byte (as there are at most 256 entries in the array). For instance, in Fig. 2, the result list of Q3 starts with the 1-byte identifier 0, which will be resolved to the first element of the array, i.e., document 1111. The result documents that are unique to each query (e.g., 5555 for Q3 and 6666 for Q4) will be stored as is, i.e., in 4-bytes. Clearly, as the degree of overlap in the results of the queries in a cluster increase, our storage scheme will yield more gains. That is, shared documents will be stored with 4-bytes only once, and will be pointed by 1-byte entries in each result list.

The proposed storage scheme employs a shared-documents array per cluster, which implies that each query should know

TABLE I
STORAGE PERFORMANCES

Number of Queries in the cache	Baseline Storage	Baseline Storage (with global shared array)	% Reduction	Cluster-Based Storage (sim_threshold=0.2)	% Reduction	Cluster-Based Storage (sim_threshold=0.1)	% Reduction
1000	112,212	110,868	1,20	108,954	2.90	108,616	3.20
3000	335,500	332,953	0,76	324,487	3.28	323,280	3.64
5000	558,216	555,029	0,57	539,967	3.27	537,980	3.63
10000	1,109,212	1,104,220	0,45	1,069,220	3.61	1,064,961	3.99
15000	1,664,260	1,657,619	0,40	1,603,909	3.63	1,597,907	3.99
20000	2,206,984	2,199,540	0,34	2,127,752	3.59	2,119,242	3.98
30000	3,282,072	3,273,717	0,25	3.164,765	3.57	3,152,692	3.94
40000	4,296,000	4,286,139	0,23	**4,138,939**	**3.66**	**4,123,945**	**4.01**
50000	5,325,428	5,315,378	0,19	5,133,331	3.61	5,116,419	3.92

the location of this array. In Fig. 2, an extra 4-byte entry is added to the beginning of each query's result list to store the address of this array. Furthermore, we also need a mechanism to encode whether an entry in the result list should be interpreted as a 1-byte pointer or a 4-byte document identifier, as they are in a mixed order in our scheme. In the literature, it is reported that Web users very rarely see more than top-30 results ([14, 8]). Thus, we assume that for each query a result list of at most 30 entries are stored. In this case, another 4-byte entry is added to the beginning of the result list, to encode whether the succeeding entries should be interpreted as 1-byte or 4-byte values. For instance, for Q3, the corresponding bit sequence would start with 110, which means that the first two entries in the result list are pointers to the shared array, and the third entry is actually a document identifier. As a result, our scheme incurs a cost of 8-bytes (4-bytes for the address of shared-documents array and 4-bytes for the entry interpretation mask) per query result list. For the simple scenario outlined above, the proposed storage scheme can not compensate these costs; but in real life our approach would compensate the costs and yield space gains even when a query cluster have a few documents in common among the top-30 results. For instance, in a cluster of three queries, an intersection of 5 results would be enough for compensating the additional costs. Finally, since static caching is an offline process, we can decide whether to apply our storage scheme or not, considering the cost/gain trade-off for each cluster. In Section IV, we provide experimental evidence supporting our claims.

IV. EXPERIMENTS

Dataset: We use a subset of the AOL Query Log (http://imdc.datcat.org/collection/1-003M-5) which contains around 20 million queries of about 650K people for a period of 3-months. Our subset contains 1,127,894 query submissions and 661,791 of them are distinct queries.

We used Yahoo! search engine's "Web search" web service [16] to get Top-100 results including titles, urls and snippets, for all distinct queries. This resulted in a 13.8 GB dataset. In our experiments, top-30 query results are cached in static cache since most users only check a few result pages [14, 8]. For instance, [14] reports that in 95.7% of queries, users requested up to only three result pages.

Following the practice in the literature, static cache is populated with the most popular query results. We select most frequent *K* queries from our query log. Next, single link clustering algorithm is executed on this *K* query set. After obtaining query clusters, we distinguish clusters as "useful" and "useless" according to the space consumed by using our approach. If storing a cluster in our scheme requires more space than its baseline storage space (i.e., when there is not enough overlap in the result documents among the queries of the cluster), then it is identified as "useless" cluster and we store the queries in that cluster as in the case of baseline. Additionally, since clustering process is terminated based on a minimum similarity threshold value at some point, there may also exist single-query clusters left apart from the "useless" clusters. These single-query clusters could not be merged with any other cluster during the query clustering. Baseline storage model is also applied for those types of queries. For useful clusters that yield space gains, we apply the storage scheme proposed in this paper.

In Table I we provide the overall reduction rates in cache sizes where the cache is filled with the most-frequent *K* queries. The column "baseline storage" denotes the case where each document id in the result lists is stored by using 4-bytes. The columns "cluster-based storage" denote the cases in which our storage scheme is applied as described above. We experiment with two different values of minimum similarity threshold that is used to terminate the clustering process. Finally, we also conducted an additional experiment where we kept a shared-document array for the *entire* set of queries in the cache. That is, instead of clustering queries we determine the top-*256* most frequent result documents for all queries in the cache and store in a global shared-documents array. Again,

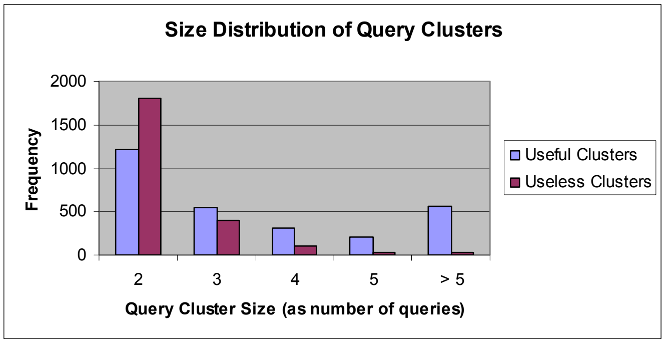


Fig. 3. Size distribution of query clusters for most frequent 40,000 queries (clustering similarity threshold is 0.1)

the shared documents in the result lists are encoded with a 1-byte pointer. For this case, there is no need to store the address of array per query, since there is only one global array.

We draw the following observations from Table I. First of all, encoding shared documents in a compact manner is a beneficial approach even in the global case. When a global array of 256 documents is used, we observe a slight reduction in the space wasted. However, the gains are more emphasized when queries are clustered. For the case with clustering similarity threshold is set to 0.1, we obtain the space reductions up to 4%. For all values of *K*, cluster-based storage scheme outperforms the baseline storage and the baseline with a global shared-documents array.

Fig. 3 shows the size distribution of query clusters (with similarity threshold 0.1) for *K*=40,000 queries case, for which our storage scheme achieves the highest reduction (i.e., 4.01%). As it can be seen from the graph, clusters involving two queries dominate. For a more detailed analysis, we also report the number of "useful", "useless" and single-query clusters in this case. Out of 40,000 queries, 2,840 "useful" clusters and 2,340 "useless" clusters are formed. These clusters contain 13,652 and 5,548 queries, respectively. 20,800 queries are left as single-query clusters. Average cluster size of "useful" clusters is 4.81 queries, whereas "useless" clusters have 2.35 queries per cluster on average. This is expected since more queries should be overlapping in "useful" clusters.

The queries in the useless and single-query clusters (summing up to 26,348 queries) are stored in the conventional manner whereas the remaining queries (13,652 of them) are stored by using our scheme. Thus, 34.13 % of all queries are stored using our mechanism. The storage space used only for these queries drops from 1,513,212 bytes to 1,359,157 bytes; resulting an 11% reduction in the consumed space. This implies that better clustering of queries may also yield higher overall reductions.

Note that, our approach may also cause a slight increase in the preparation of final query result page in case of a cache hit, due to relatively more complicated handling of the query result lists. In turn, the gains in the storage space would improve the cache hit rate and throughput with respect to the baseline scheme, as more queries can be filled to the same cache space with our approach. As a result, we envision that the former cost of processing would be negligible and compensated by the latter gains in hit rate and throughput.

V. Conclusion

We presented a storage mechanism for caching of query results by exploiting the query clustering. In particular, we store the documents identifiers that are shared by the queries in a cluster in a more compact manner and improve storage utilization.

In our current framework, we do not consider the ranks of the overlapping result documents during similarity computation. As a future research direction, it might be a good idea to incorporate the rank information into the similarity formula so that cluster pairs having overlaps at high ranks could be merged earlier. Additionally, we plan to use a more efficient clustering algorithm than single-link, which has $O(n^2)$ complexity.

Acknowledgment

This work is supported by The Scientific and Technological Research Council of Turkey (TÜBİTAK) by the grant numbers 108E008 and 105E065. We also thank anonymous referees for their valuable comments.

References

[1] D. Beeferman, and A. Berger, "Agglomerative clustering of a search engine query log", In Proceedings of the Sixth ACM SIGKDD international Conference on Knowledge Discovery and Data Mining (Boston, Massachusetts, United States, August 20 - 23, 2000). KDD '00. ACM, New York, NY, pp. 407-416, 2000.

[2] R. Baeza-Yates, and F. Saint-Jean, "A three level search engine index based in query log distribution", In Proceedings of 10th International Symposium on String Processing and Information Retrieval (Manaus, Brazil) , Springer, pp. 56-65, 2003.

[3] R. Baeza-Yates, C. Hurtado, and M. Mendoza, "Improving search engines by query clustering", *J. Am. Soc. Inf. Sci. Technol.* 58, 12 (Oct. 2007), pp. 1793-1804, 2007.

[4] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri, "The impact of caching on search engines", In Proceedings of the 30th Annual international ACM SIGIR Conference (Amsterdam, The Netherlands). SIGIR '07. ACM Press, New York, NY, pp. 183-190, 2007.

[5] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors", *Commun. ACM* 13, 7 (Jul. 1970), pp. 422-426, 1970.

[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 Trans. Inf. Syst.*, 24, 1 (Jan. 2006), pp. 51-78.

[7] A. K. Jain, and R. C. Dubes, "*Algorithms for Clustering Data*", Prentice-Hall, Inc., 1988.

[8] B. J. Jansen, and A. Spink, “An analysis of web documents retrieved and viewed”, In Proc. of the 4th International Conference on Internet Computing. Las Vegas, Nevada, pp. 65-69. 23 – 26 June 2003.

[9] J. Wen, Y. Jian and H. Zhang, “Query clustering using user logs”, *ACM Transactions on Information Systems (ACM TOIS)*, 20(1), pp. 59-81, January, 2002

[10] X. Long, and T. Suel, “Three-level caching for efficient query processing in large web search engines”, In Proceedings of the 14th international Conference on World Wide Web (Chiba, Japan). WWW '05. ACM Press, New York, NY, pp. 257-266, 2005.

[11] E. P. Markatos, “On caching search engine query results”, Computer Communications, 24, 2 (2001), pp. 137-143.

[12] R. Ozcan, I. S. Altingovde, and Ö. Ulusoy, “Static query result caching revisited”, In Proceedings of WWW 2008 conference (Beijing, China), pp. 1169-1170, 2008.

[13] P. C. Saraiva, E. Silva de Moura, N. Ziviani, W. Meira, R. Fonseca, and B. Riberio-Neto, “Rank-preserving two-level caching for scalable search engines”, In Proceedings of the 24th Annual international ACM SIGIR Conference (New Orleans, Louisiana, United States). SIGIR '01. ACM Press, New York, NY, pp. 51-58, 2001.

[14] C. Silverstein, H. Marais, M. Henzinger, and M. Moricz, “Analysis of a very large web search engine query log”, *SIGIR Forum* 33, 1 (Sep. 1999), pp. 6-12, 1999.

[15] P. Triantafillou, N. Ntarmos, and J. Yannakopoulos, “A cache engine for e-content integration”, *IEEE Internet Computing* 8, 2 (Mar. 2004), pp. 45-53, 2004.

[16] Yahoo! Search engine’s “Web search” Web service, http://developer.yahoo.com/search/web/V1/webSearch.html