A Cost-Aware Strategy for Query Result Caching in Web Search Engines

Ismail Sengor Altingovde, Rifat Ozcan, and Özgür Ulusoy

Department of Computer Engineering, Bilkent University, Ankara, Turkey
{ismaila,rozcan,oulusoy}@cs.bilkent.edu.tr

Abstract. Search engines and large scale IR systems need to cache query results for efficiency and scalability purposes. In this study, we propose to explicitly incorporate the query costs in the static caching policy. To this end, a query's cost is represented by its execution time, which involves CPU time to decompress the postings and compute the query-document similarities to obtain the final top-*N* answers. Simulation results using a large Web crawl data and a real query log reveal that the proposed strategy improves overall system performance in terms of the total query execution time.

1 Introduction

Caching is one of the most crucial mechanisms that is employed in the large scale information retrieval (IR) systems and Web search engines (WSEs) for efficiency and scalability purposes. Search engines cache the query result pages and/or posting lists of the terms that appear in the queries. A search engine may employ a static or dynamic cache of such entries, or both [5]. In a static result cache, the cache is typically populated with the results of the most-frequent queries that are extracted from the previous logs of WSEs (see [10] for alternative strategies). The cache content remains intact until the next periodical update. In a dynamic cache, the content changes dynamically with respect to the query traffic.

In the context of WSEs, the literature involves several proposals concerning what and how to cache. However, especially for the case of the query result caching, the cost of a "miss" is usually disregarded, and all queries are assumed to have the same cost. In this paper, we essentially concentrate on the static caching of the query results and propose a cost-aware strategy that explicitly makes use of the query costs while determining the cache contents.

In the literature, non-uniform miss costs are exploited in the caching policies in several domains such as WWW proxies and memory systems [4, 7, 8]. For WSEs, we are aware of only one earlier study that uses the notion of costs in a similar manner to us. In that work [6], Garcia proposes to use a heterogeneous cache that can store all possible data structures (posting lists, accumulator sets, query results, etc.) to process a query and each of these entry types is associated with a cost function. However, the cost function in that work is essentially based on the disk access times and has to be recomputed for each cache entry after every modification of the cache. In contrast, our cost function is based on the actual query processing time and can be computed for once. In the other studies, the notion of the cost is usually employed for some other purposes (e.g., for computing the optimal cache space split in [1]) but not to decide the actual contents of a results cache.

Our research is motivated by the following hypotheses. First, the queries submitted to a search engine would have significantly varying costs in terms of several aspects (e.g., CPU processing time, network requirements, etc.). Thus, it is not realistic to assume that all cache misses would incur the same cost. Second, the frequency of the query may not always be an accurate indicator of its cost. Thus, using the popularity or recentness of a query alone (as in static and dynamic caching setups, respectively) may not always lead to the optimum performance, and a cost-aware strategy may provide further gains.

In this study, we first verify the validity of these hypotheses for our experimental setup. Next, we introduce a cost-aware strategy for the static caching of the query results. In the preliminary simulation setup discussed here, we define the query cost in terms of the CPU execution time, which involves decompressing the postings, computing the query-document similarities and determining the top-N document identifiers in the final answer set. Our simulation results using a large Web crawl data and real query logs reveal that the cost-aware strategy improves overall system performance in terms of the total query processing time.

The rest of the paper is organized as follows. In the next section, we describe the characteristics of our dataset and query log. Section 3 is devoted to the cost analysis of the queries in the log. The cost-aware static caching is discussed and evaluated in Section 4. We conclude and point to future research directions in Section 5.

2 Data Characteristics

Dataset. In this study, we use a subset of the terabyte-order crawl datasets provided by the Stanford University's WebBase Project Repository [12]. Our dataset includes pages collected from US government Web sites during the first quarter of 2007. The dataset is indexed by the Zettair search engine (http://www.seg.rmit.edu.au/zettair/) without stemming and stopword removal. The dataset includes approximately 4.3 million pages, yielding an index of 7 GBs on disk (including the term offset information in the posting lists).

Query log. We use a subset of the AOL Query Log (available at 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 around 1.1 million queries (700K of them are distinct) from the first six weeks of the log. Queries submitted in the first 3 weeks constitute the training set (to fill the static cache) whereas queries from the second three weeks are reserved as the test set.

In this paper, the requests for the next result page of a query are considered as a single query request, as in [2]. Another alternative would be interpreting each log entry as <query, result page number> pairs [5], which has been left as a future work. Accordingly, we presume that, a fixed number of *N* results are cached per query. Since *N* would be set to a small number in all practical settings, we presume that the actual value of *N* would not significantly affect the findings in this paper. Here, we set *N* as 30, as earlier works [11] report that in 95.7% of queries, users requested up to only three result pages.

3 An Analysis of the Query Processing Cost

3.1 The Setup for the Cost Measurement

The underlying motivation for employing the result caching in WSEs (at the server side) is reducing the burden of query processing. In a typical distributed environment, the cost of query processing would involve several components. For a given query, the central broker first consults its result cache, and if a cache-miss occurs, sends the query to index nodes. Each index node should then fetch the corresponding posting lists to main memory (if they are not already fetched) with the cost C_{DISK}. Next, the postings are processed and partial results are computed, with the cost C_{CPU}. More specifically, the CPU cost involves the decompression of the posting lists (as they are usually stored in a compressed form), computation of a similarity function between the query and the postings, and finally obtaining the top-*N* documents as the partial result. Then, each node sends its partial results to the central broker, with the cost C_{NET}, where they are merged. Finally, the central broker generates the snippets for the query results, with the cost C_{SNIP}, and sends the output page to the user. Thus, the cost of query processing is the sum of all of these costs, i.e., $C_{DISK} + C_{CPU} + C_{NET} + C_{SNIP}$.

For the purposes of this paper, we consider the CPU execution time (C_{CPU}) as the representative of the overall cost of a query. At the first sight, this decision seems to leave out two other major components, disk access and network communication costs. However, index nodes in a distributed architecture would probably store a large amount of posting lists, if not all; in the main memory. Indeed, given the current advances in the hardware, it is possible to store all posting lists in the main memory, an approach totally eliminating the cost of disk access (e.g., see [13]). Furthermore, the time spent for disk access for a particular query depends on the execution order of the queries and the current contents of operating system buffers, which may be unpredictable and hard to measure in an objective manner. For the issue of network costs, a recent work states that for a distributed system interconnected via a LAN, the network cost would only be a fraction of the query processing cost (see Table 2 in [1]). These factors make neglecting disk and network costs an acceptable choice at this stage of our work, though our research for incorporating these components to the cost model is already underway. Finally, the snippet generation cost is also left out, since the efficiency of the snippet generation is investigated in only a few previous studies, and none of these discuss how the cost of snippet generation compares to other cost components. This is another direction for future work.

In our setup, all distinct queries are processed using the Zettair search engine in batch mode to obtain the CPU execution times. To be more accurate, we describe the setup as follows.

- We use the Zettair in its default mode, which employs an early pruning strategy that dynamically limits the number of accumulators used for a query (see [9] for details). This is a crucial choice for the practicality of our proposal, since no real WSE would make a full evaluation and the CPU execution time clearly depends on the partial evaluation strategy employed in the system.

- All query terms in the log are converted to lower case. The queries are modified to include an additional “AND” conjunct between each term, so that the search engine runs in the “conjunctive” mode. This is the default search mode of the major search engines. Stopwords are not eliminated from the queries. No stemming algorithm is applied. Finally, all phrase queries are discarded.

3.2 Experiments

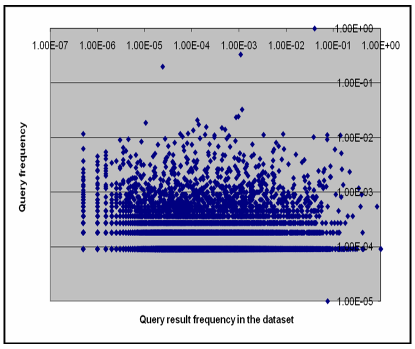
In this section, our goal is to obtain and analyze the query processing costs using the log and document collection described in Section 2. However, our initial experiments revealed that an unusually large number of queries return no answer for our dataset. We attribute this situation mostly to the fact that our dataset is a Web crawl from the *.gov* domain in 2007, whereas query log includes requests from a general domain search engine in 2006 (see [15] for a related discussion on the “appropriateness” of a log for a dataset). An additional factor can be the conjunctive processing of the queries. Nevertheless, we simply discarded all queries from our query log that return an empty answer. The remaining set, so called *original query log*, includes 700K query instances, 357K of which are distinct.

A further observation from the preliminary experiments is that some of the most frequent queries in the query log appear much less frequently in the dataset. For instance, “yahoo com” is in among the most frequent queries in the log, whereas there are relatively few documents with both words in our collection. To explore the possible affects of this situation on our experimental results, we obtained a more restricted subset, *semantically aligned query log*, from the original query log. In particular, following a similar approach discussed in a recent work [14], we first submitted all distinct queries in the original query log to *Yahoo!* search engine’s “Web search” service [16] to get the top-10 results. Next, we only kept those queries that yield at least one result from the *.gov* domain. In this latter set, there remain 83K queries and 48K of them are distinct.

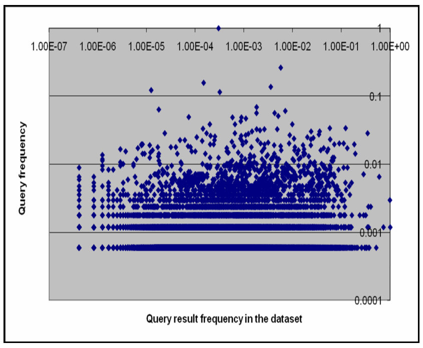
For each of these logs, we obtained the top-30 results and CPU execution time of all distinct queries using the setup described in the previous section. The experiments are repeated four times and the results revealed that the computed CPU costs are stable and can be used as the basis of the following discussions.

In Figure 1, we show the normalized log-log scatter plots that relate the query’s frequency in the log and result frequency in the collection. We can deduce that for both query logs, there is a low correlation between query frequency and result frequency. Interestingly, the plots for the two logs do not seem to differ a lot, implying that selecting only queries that are semantically more related to the data does not really change the distribution of the result frequency in our setup. This may be due to the conjunctive processing of the queries. Thus, we conclude that original query log does not have a bias in this sense and both query logs can be used for further experimentation.

As mentioned before, our research is motivated by the hypotheses that the queries submitted to a search engine can have very different processing costs and the cost of a query may not be correlated with its frequency. In Figure 2, we provide the normalized log-log scatter plots that relate the query’s frequency in the log and query evaluation time, i.e., C_{CPU}. These plots reveal adequate evidences to support the above hypotheses. First, we see that the query execution time covers a wide range from a fraction of a millisecond to a few thousand milliseconds. The majority of the queries are uniformly distributed between 1 and 100 milliseconds, but still showing large variations. Thus, it may be useful to devise cost-aware strategies in the caching mechanisms. Second, we cannot derive a high positive correlation between the query frequency and the processing times. That is, a very frequent query may be cheaper than a less-frequent query. This can be explained by the following arguments: in earlier works, it is stated that the query terms and collection terms may not be highly correlated (e.g., correlation between Chilean crawl data and query log is found to be only 0.15 [3]), which means that a highly frequent query may occur in less number of documents, and be cheaper to process. Furthermore, even for the cases where the reverse is true (i.e., the frequent queries appear in more number of documents), the processing time does not necessarily follow the proportion, due to the compression and pruning techniques applied during the query processing (e.g., see Fig. 10 in [1]).

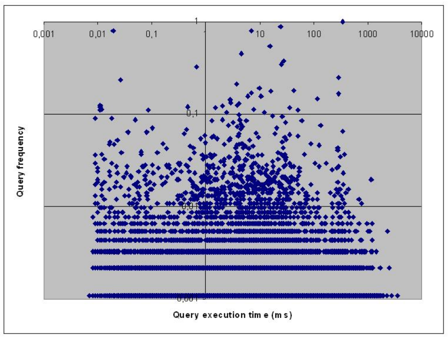


(a)

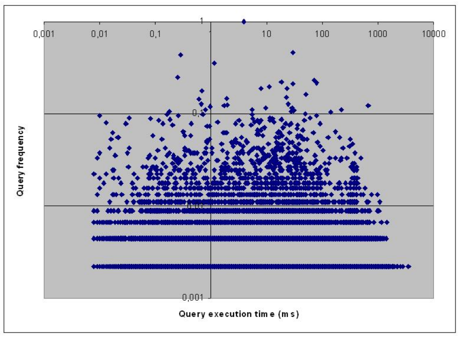


(b)

Fig. 1. Normalized log-log scatter plot of the query result frequency in the dataset and query frequency in the a) original, and b) semantically aligned, query logs



(a)



(b)

Fig. 2. Normalized log-log scatter plot of the query execution time and query frequency in the a) original, and b) semantically aligned, query logs

Our findings in this section are encouraging in the following ways. We observe that the query processing costs, and accordingly, miss costs are non-uniform and may vary a lot among different queries. Furthermore, this variation is not directly correlated to the query frequency, a feature already employed in the current caching strategies. These call for a cost-aware caching strategy, which we discuss next.

4 Cost-Aware Static Caching

4.1 Cost-Aware Caching (CAC) Policy

Typically, a static cache is populated with the most frequent queries in the previous logs of a WSE. We call this the frequency-based strategy. However, since the miss costs of the queries are not uniform, the improvement promises of such a strategy evaluated in terms of say, hit-rate, may not really translate to actual improvements that may be measured in terms of the query processing time or throughput. To remedy this problem, we embed the miss costs into the static caching policy, as follows. Similar to the approach taken in [1], filling a static cache with a predefined capacity can be reduced to the well-known knapsack problem, where queries are the items with certain sizes and values. In our case, we presume that cache capacity is expressed in terms of the number of queries, i.e., each query (and its results) is allocated a unit space. Then, the question is how to fill the knapsack with the items that are most valuable, where the value of an item is the execution time saving obtained by selecting this item. Thus, we define the cost-aware caching (CAC) strategy as follows.

For a query q with cost C_q and frequency F_q, the expected value of the query q can be simply computed as the product of these two figures, i.e., C_q x F_q. That is, we expect that the query would be as frequent in the future as in the past logs, and caching it would provide a gain as expressed by this formula. During the simulations reported below, we observed that this expectation may not hold in a linearly proportional manner, i.e., queries that have occurred with some high frequency still tend to appear with a high frequency whereas the queries with relatively less frequency may appear even more sparsely, or totally fade away in the future. For this reason, we use a slightly modified version of the formula that is biased to emphasize the higher frequency values and depreciate the lower ones, as shown below. In the next section, we provide experimental results for the cases where $k \geq 1$.

$$Value(q) = C_q x F_q^k$$

Given the above value function and a cache of size M, the cost-aware strategy adapts a greedy approach, i.e., locates the most valuable M queries in the cache.

4.2 Simulation Results

In this section, we compare two strategies for filling the static result cache of a WSE, namely the frequency-based and cost-aware strategies. As described in Section 2, the query log is split into training and test sets, and the former is used to fill the caches of varying sizes whereas the latter is used to evaluate the performance. The training set includes 211,854 and 28,794 distinct queries for the original and semantically aligned logs, respectively. Cache size is in terms of the number of the queries. Remarkably, we don't measure the cache hit rate due to non-uniform miss costs, but use the total query processing time for evaluation. For cache hits, we assume that the processing time is negligible, i.e., the cost is 0. To simulate the cache misses, we use the query processing costs computed by the search engine as described in Section 3. That is, we compute and record the CPU execution times (C_{CPU}) of all distinct queries in batch mode. Whenever a cache miss occurs, the cost of this query is retrieved and added to the total query processing time.

Table 1. Query processing times (in seconds) of different caching strategies for original query log. Percentage reductions for the CAC strategies are shown in the nearby columns.

Cache Size	Freq. Cache	CAC (k=1)	% red.	CAC (k=2.5)	% red.	CAC (k=3)	% red.	Optimal CAC	% red.
1000	18,412	17,798	3.34	17,702	3.85	17,769	3.49	17,145	6.88
5000	17,468	17,262	1.18	16,909	3.20	16,953	2.95	15,963	8.62
10000	17,099	17,000	0.58	16,655	2.60	16,638	2.69	15,651	8.47
30000	16,549	16,391	0.95	16,154	2.39	16,164	2.32	15,499	6.35
50000	16,357	16,007	2.14	15,910	2.73	15,910	2.73	15,498	5.25
70000	16,158	15,763	2.45	15,723	2.69	15,726	2.68	15,498	4.08
90000	16,081	15,653	2.66	15,636	2.77	15,637	2.76	15,498	3.62
150000	15,792	15,517	1.74	15,517	1.74	15,517	1.74	15,498	1.86
211854	15,498	15,498	0.00	15,498	0.00	15,498	0.00	15,498	0.00

Table 2. Query processing times (in seconds) of different caching strategies for semantically aligned log. Percentage reductions for the CAC strategies are shown in the nearby columns.

Cache Size	Freq. Cache	CAC (k=1)	% red.	CAC (k=2.5)	% red.	CAC (k=3)	% red.	Optimal CAC	% red.
1000	3,149	3,152	-0.09	3,067	2.62	3,075	2.37	2,899	7.94
2000	3,090	3,105	-0.50	3,034	1.80	3,031	1.90	2,846	7.88
3000	3,056	3,063	-0.26	3,017	1.26	3,014	1.37	2,835	7.22
4000	3,030	3,020	0.33	2,987	1.41	2,991	1.30	2,834	6.48
5000	3,001	2,996	0.17	2,952	1.63	2,959	1.41	2,834	5.57
10000	2,959	2,896	2.13	2,888	2.38	2,892	2.26	2,834	4.22
15000	2,927	2,857	2.40	2,854	2.49	2,854	2.47	2,834	3.18
20000	2,893	2,841	1.80	2,840	1.83	2,840	1.83	2,834	2.04
28794	2,834	2,834	0.00	2,834	0.00	2,834	0.00	2,834	0.00

In Tables 1 and 2, we provide the total query execution time using the frequency-based and cost-aware caching strategies for each query log. We also experimented with different values of k for CAC strategies. Our results reveal that using k values greater than 1 reflects the query repetition patterns better, which conforms to our discussion in Section 4.1. The best results are obtained for k=2.5 and using higher values do not provide further improvements.

We also provide the potential gains for the optimal cost-aware caching strategy, in which case the test queries are assumed to be known beforehand. Since we know the actual future frequencies of the training queries, we fill the cache with those queries that would yield the highest *C* x *F* values. Clearly, this is only reported to demonstrate how far away the proposed strategy is from the optimal.

In the experiments, cost-aware strategy decreases the overall execution time (up to 3%) with respect to the frequency-based strategy. It is also remarkable that the gains for the optimal cache is much higher, which implies that a better function for the cost-aware caching may provide higher improvements.

Another important observation is that the optimal cache saturates very early as it stores all of the training queries that would ever be seen in the test set. As this early saturation also implies, the majority of the requests in the test set yield compulsory misses, i.e., misses for the queries that have never seen in the training set. We envision that, coupling the static cache with a dynamic cache would partially remedy this situation, as the further requests for some of those newly seen queries can be answered from the dynamic cache. Thus, the gains of the cost-aware strategy would be more emphasized in a hybrid caching environment.

5 Conclusion

We propose a cost-aware caching strategy for the static caching of the query results in WSEs. Our experiments with a large crawl data and a real life query log reveal promising results and reduce the total query processing times up to 3%. The future work involves applying the cost-aware techniques for dynamic and hybrid caching.

Acknowledgments. This work is supported by The Scientific and Technological Research Council of Turkey (TÜBİTAK) by the grant number 108E008.

References

1. Baeza-Yates, R., Gionis, A., Junqueira, F., Murdock, V., Plachouras, V., Silvestri, F.: The impact of caching on search engines. In: Proc. of SIGIR 2007, Netherlands, pp. 183–190 (2007)
2. Baeza-Yates, R., Junqueira, F., Plachouras, V., Witschel, H.F.: Admission policies for caches of search engine results. In: Ziviani, N., Baeza-Yates, R. (eds.) SPIRE 2007. LNCS, vol. 4726, pp. 74–85. Springer, Heidelberg (2007)
3. Baeza-Yates, R., Saint-Jean, F.: A three level search engine index based in query log distribution. In: Nascimento, M.A., de Moura, E.S., Oliveira, A.L. (eds.) SPIRE 2003. LNCS, vol. 2857, pp. 56–65. Springer, Heidelberg (2003)
4. Cao, P., Irani, S.: Cost-aware WWW proxy caching algorithms. In: Proc. of the USENIX Symposium on Internet Technologies and Systems, Monterey, California (1997)
5. Fagni, T., Perego, R., Silvestri, F., Orlando, S.: Boosting the performance of Web search engines: Caching and prefetching query results by exploiting historical usage data. ACM TOIS 24(1), 51–78 (2006)
6. Garcia, S.: Search Engine Optimisation Using Past Queries. Ph.D thesis, RMIT (2007)
7. Jeong, J., Dubois, M.: Cache Replacement Algorithms with Nonuniform Miss Costs. IEEE Transactions on Computers 55(4), 353–365 (2006)

8. Liang, S., Chen, K., Jiang, S., Zhang, X.: Cost-Aware Caching Algorithms for Distributed Storage Servers. In: Pelc, A. (ed.) DISC 2007. LNCS, vol. 4731, pp. 373–387. Springer, Heidelberg (2007)
9. Lester, N., Moffat, A., Webber, W., Zobel, J.: Space-Limited Ranked Query Evaluation Using Adaptive Pruning. In: Ngu, A.H.H., Kitsuregawa, M., Neuhold, E.J., Chung, J.-Y., Sheng, Q.Z. (eds.) WISE 2005. LNCS, vol. 3806, pp. 470–477. Springer, Heidelberg (2005)
10. Ozcan, R., Altingovde, I.S., Ulusoy, Ö.: Static query result caching revisited. In: Proc. of WWW 2008, Beijing, China, pp. 1169–1170 (2008)
11. Silverstein, C., Marais, H., Henzinger, M., Moricz, M.: Analysis of a very large web search engine query log. SIGIR Forum 33(1), 6–12 (1999)
12. Stanford University WebBase Project, `http://www-diglib.stanford.edu/~testbed/doc2/WebBase`
13. Strohman, T., Croft, W.B.: Efficient document retrieval in main memory. In: Proc. of SIGIR 2007, Netherlands, pp. 175–182 (2007)
14. Tsegay, Y., Turpin, A., Zobel, J.: Dynamic index pruning for effective caching. In: Proc. of CIKM 2007, Lisbon, Portugal, pp. 987–990 (2007)
15. Webber, W., Moffat, A.: In Search of Reliable Retrieval Experiments. In: Proceedings of the Tenth Australasian Document Computing Symposium, ADCS, pp. 26–33 (2005)
16. Yahoo! "Web search" Web service (2008), `http://developer.yahoo.com/search/web/V1`