Propagating Expiration Decisions in a Search Engine Result Cache

Fethi Burak Sazoglu
Bilkent University
Ankara, Turkey
fethi.sazoglu@bilkent.edu.tr

Ismail Sengor Altingovde
Middle East Technical University
Ankara, Turkey
altingovde@ceng.metu.edu.tr

Rifat Ozcan
Turgut Ozal University
Ankara, Turkey
rozcan@turgutozal.edu.tr

B. Barla Cambazoglu
Yahoo Labs
Barcelona, Spain
barla@yahoo-inc.com

Özgür Ulusoy
Bilkent University
Ankara, Turkey
oulusoy@cs.bilkent.edu.tr

ABSTRACT

Detecting stale queries in a search engine result cache is an important problem. In this work, we propose a mechanism that propagates the expiration decision for a query to similar queries in the cache to re-adjust their time-to-live values.

1. INTRODUCTION

Result caching is a key technique that is widely employed by search engines, helping them to improve their efficiency by reducing the query processing workload on their backend search systems. Due to increasing storage capacities and dropping hardware prices, it is now possible to construct a cache that is large enough to contain the results of all queries submitted to a search engine for a very long time period, maybe for several months or even years [6]. Given that the web content and hence indexes of search engines are updated continuously, it now becomes important to correctly identify the stale query results in the cache.

Earlier work proposed methods to identify and refresh the stale results in the cache. The so-called *informed* strategies interact with the underlying index to detect the query results that are likely to be stale [1, 3, 4]. These techniques are known to be effective. Yet, they may be non-trivial to implement in existing search systems and may incur additional computational costs. In contrast, a straightforward strategy that is *blind* to changes in the underlying index is to assign a time-to-live (TTL) value to each result in the cache [6]. When a cache hit occurs, the corresponding result is served by the cache only if its TTL has not yet expired. Cambazoglu et al. [6] proposed such a proactive strategy, which exploits the idle cycles of the backend to refresh the query results that were expired (or close to expiration) and likely to be submitted again soon. Bortnikov et al. [5] introduced the frequency-based TTL strategy, which takes into account the number of cache hits for a query result before making an expiration decision. Alici et al. [2] used adaptive TTL values that are tailored for each query rather than assigning the same TTL value to all queries. Recently, Sazoglu et al. [7] showed that certain combinations of different TTL strategies perform better than using each strategy in isolation.

In this work, we seek to improve the performance of the TTL-based expiration strategy by propagating the expiration decision made for a query among similar queries in the cache. The underlying motivation is that, when a cached query result is found to be stale, it is very likely that the results of similar queries may also be stale, and the search engine should consider reducing the TTL values assigned to such queries. In Section 2, we provide the details of this similarity-based TTL strategy. In Section 3, we present our experimental setup and results.

2. SIMILARITY-BASED TTL STRATEGY

In the proposed expiration strategy, for each query q whose results are cached, we maintain a short list of most similar queries (S_q). Earlier work have shown that the number of overlapping results is a good indicator of query similarity. Therefore, we compute the similarity of two queries using the Jaccard similarity between their top-10 result sets.

As usual, a fixed TTL value is assigned to each query at the time of caching. Upon a cache hit on an expired query, the search engine computes the new (fresh) query results and, before replacing the old results in the cache, compares the two copies to see if the cached results were really stale (recall that TTL expiration does not necessarily imply that the cached results are stale). If the new results differ from the cached results, i.e., it is verified that the cached results were really stale, then this information is propagated to all queries in S_q. Each query $q_i \in S_q$ is "warned" proportional to its similarity to q. Finally, the current TTL of q_i is reduced if there is enough evidence that its results are stale. To this end, we associate each query q with a warning score w_q, which is initially set to 0, and increment w_q every time a similar query is found to be stale. If the warning score exceeds a threshold T, the TTL of q is reduced by half. We evaluate two alternative methods to set the warning score w_q: `BasicScore` and `AgeScore`. Given an expired query q_{e} and a similar query $q_{\mathrm{s}} \in S_{q_{\mathrm{e}}}$, `BasicScore` and `AgeScore` update $w_{q_{\mathrm{s}}}$ as shown in Eqs. 1 and 2, respectively:

WWW 2015 Companion, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3473-0/15/05.
http://dx.doi.org/10.1145/2740908.2742772.

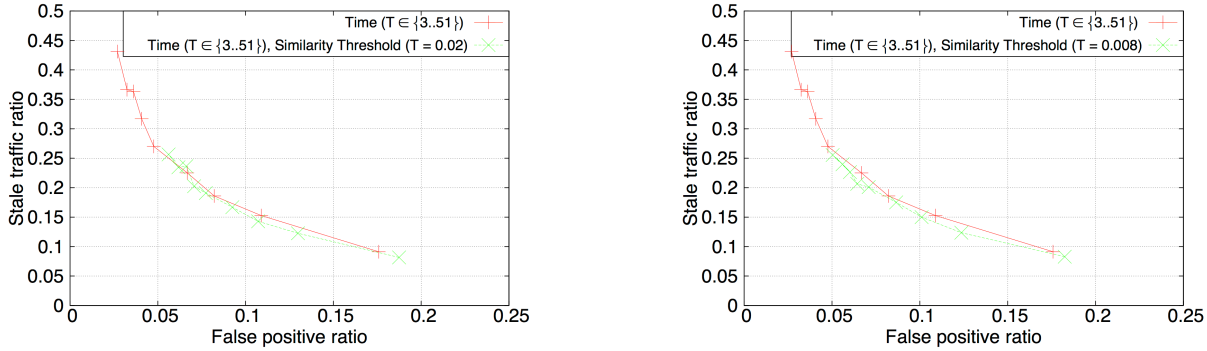

Figure 1: Performance of fixed TTL vs. similarity-based TTL with `BasicScore` (left) and `AgeScore` (right).

$$w_{q_s} \leftarrow w_{q_s} + \text{similarity}(q_e, q_s), \tag{1}$$

$$w_{q_s} \leftarrow w_{q_s} + \text{similarity}(q_e, q_s) \times \frac{\text{cacheAge}(q_s)}{\text{cacheAge}(q_e)}, \tag{2}$$

where cacheAge(q) denotes the difference between the current time and the time at which query q was cached.

`BasicScore` simply increments w_{q_s} by the similarity between q_e and q_s. `AgeScore` also takes into account the ratio between the times spent by q_s and q_e in the cache. If the ratio is greater than 1, i.e., the results of q_s were cached before the expired results of q_e were cached, we increase the weight of the similarity score added to w_{q_s}. This procedure is illustrated in Algorithm 1.

3. EXPERIMENTS

Data. In this study, we use a sample of 2,044,531 queries submitted to the Spanish frontend of Yahoo Web Search. The sample is sorted in timestamp order. We use the queries in the first half of the sample to warm up the cache and those in the remaining half to evaluate the performance.

Simulation setup. We assume an infinitely large cache so that we can evaluate the performance independent of various parameters, such as the cache capacity or eviction policy, as in [2]. We assume that, for a given query-timestamp pair (q, t), the corresponding top k ($k \leq 10$) URLs in the query log serve as the ground truth result $R^*_{q,t}$, i.e., the fresh result set for query q at time t is $R^*_{q,t}$. During the simulations, when query q is first encountered, say at time t, its result set $R^*_{q,t}$ is cached. When the same query is submitted again at time t', if the cached results were not expired, we assume that the results are served by the cache. Otherwise, the cached results are evicted and the results in the query log ($R^*_{q,t'}$) are inserted. A result set $R_{q,t}$ served by the cache at some time point t is said to be stale if it differs from the result set $R^*_{q,t}$ in the query log. We consider two query result sets to be different if the same URLs are not present in exactly the same order, following [1, 4].

Evaluation metrics. We evaluate the similarity-based TTL strategy in terms of the stale traffic (ST) ratio and the false positive (FP) ratio metrics [1, 4]. The stale traffic ratio is the percentage of queries for which the results served from the cache turn out to be stale. The false positive ratio is the percentage of redundant query executions, i.e., the fraction of queries for which the refreshed results are found to be the same as the cached results.

Results. The results shown in Fig. 1 reveal that the similarity-based TTL strategy (with both `BasicScore` or `AgeScore`) can outperform the fixed TTL baseline. `AgeScore` seems to be slightly better than `BasicScore` for larger values of TTL. Although improvements over the baseline are rather small, they are promising. As the future work, we plan to apply the similarity-based TTL idea to certain subsets of queries, e.g., based on the query frequency (head/tail), intent (informational/navigational), or query topic.

ALGORITHM 1: The similarity-based TTL strategy.

Input: q: query, C: cache, T: warning threshold, w_q: warning score of query q.

$R_q \leftarrow \emptyset$ ▷ initialize the result set of q;
if $q \notin C$ **then** `/* not cached */`
  evaluate q over the backend and obtain R_q;
  insert R_q into C;
else if $q \in C$ **then** `/* cached */`
  if cacheAge$(q) \geq TTL_q$ **then** `/* expired */`
    evaluate q over the backend and obtain R'_q;
    if $R_q \neq R'_q$ **then**
      foreach $q_i \in S_q$ **do**
        increment w_{q_i} using `BasicScore` or `AgeScore`;
        if $w_{q_i} \geq T$ **then**
          $TTL_{q_i} \leftarrow TTL_{q_i}/2$;
      end
      $R_q \leftarrow R'_q$;
return R_q;

Acknowledgments. This work is partially supported by the Ministry of Science, Industry and Technology of Turkey and Huawei Inc. under the grant no 0441.STZ.2013-2, and the Yahoo Faculty Research Engagement Program.

4. REFERENCES

[1] S. Alici, I. S. Altingovde, R. Ozcan, B. B. Cambazoglu, and O. Ulusoy. Timestamp-based result cache invalidation for web search engines. In *SIGIR*, pages 973–982, 2011.
[2] S. Alici, I. S. Altingovde, R. Ozcan, B. B. Cambazoglu, and O. Ulusoy. Adaptive time-to-live strategies for query result caching in web search engines. In *ECIR*, pages 401–412, 2012.
[3] X. Bai and F. P. Junqueira. Online result cache invalidation for real-time web search. In *SIGIR*, pages 641–650, 2012.
[4] R. Blanco, E. Bortnikov, F. Junqueira, R. Lempel, L. Telloli, and H. Zaragoza. Caching search engine results over incremental indices. In *SIGIR*, pages 82–89, 2010.
[5] E. Bortnikov, R. Lempel, and K. Vornovitsky. Caching for realtime search. In *ECIR*, pages 104–116, 2011.
[6] B. B. Cambazoglu, F. P. Junqueira, V. Plachouras, S. Banachowski, B. Cui, S. Lim, and B. Bridge. A refreshing perspective of search engine caching. In *WWW*, pages 181–190, 2010.
[7] F. B. Sazoglu, B. B. Cambazoglu, R. Ozcan, I. S. Altingövde, and Ö. Ulusoy. Strategies for setting time-to-live values in result caches. In *CIKM*, pages 1881–1884, 2013.