Strategies for Setting Time-to-Live Values in Result Caches

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

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

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

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

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

ABSTRACT

In web query result caching, staleness of queries are often bounded via a time-to-live (TTL) mechanism, which expires the validity of cached query results at some point in time. In this work, we evaluate the performance of three alternative TTL mechanisms: time-based TTL, frequency-based TTL, and click-based TTL. Moreover, we propose hybrid approaches obtained by pair-wise combination of these mechanisms. Our results indicate that combining time-based TTL with frequency-based TTL yields superior performance (i.e., lower stale query traffic and less redundant computation) than using a particular mechanism in isolation.

Categories and Subject Descriptors

H.3.3 [**Information Storage Systems**]: Information Retrieval Systems

General Terms

Design, Performance, Experimentation

Keywords

Web search engines; result caching; time-to-live

1. INTRODUCTION

Query result caching is a commonly used technique in web search engines [3]. In this technique, the results of previously processed user queries are stored in a cache. The results for the subsequent occurrences of a query are served by this cache, eliminating the need to process the query and generate its results using the computational resources in the backend search system. This technique helps reducing the query processing workload incurred on the search engine while reducing the response time for queries whose results are served by the cache.

CIKM'13, Oct. 27–Nov. 1, 2013, San Francisco, CA, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2263-8/13/10 ...$15.00.
http://dx.doi.org/10.1145/2505515.2507886.

In practice, commercial web search engines deploy result caches that are large enough to store practically all query results computed in the past by the search engine [8]. Having a very large result cache renders basic caching techniques unnecessary (e.g., admission of queries [4], eviction of old cache entries [9], or prefetching of successive result pages [11]). In case of very large result caches, the main problem is to preserve the freshness of cached query results. This is because commercial web search engine indexes are frequently updated as more recent snapshots of the Web are crawled and new pages are discovered. Eventually, the cached results of a query may differ from its actual results that could be obtained by evaluating the query on the current version of the index. Queries whose cached results are not consistent with those that would be provided by backend search system are referred to as stale queries. Identifying such queries and improving the overall freshness of a result cache is crucial because presenting stale query results to the users may have a negative effect on user satisfaction for certain types of queries [2]. So far, two different lines of techniques addressed the freshness issue mentioned above: refreshing [8, 10] and invalidation [1, 5, 6, 7].

In the first set of techniques, cached query results that are predicted to be stale are refreshed by evaluating the associated queries at the search backend. The main motivation behind these techniques is to use the idle cycles of the backend search system to recompute the results of a selected set of supposedly stale queries. In general, the techniques based on refreshing are easy to implement as they do not require any interaction between the result cache and the backend search system to decide which queries' results to refresh. On the other hand, identification of stale queries is a rather difficult task and this leads to an increase in the volume of queries whose results are unnecessarily recomputed with no positive impact on the freshness of the cache.

In the second set of techniques, the result cache is informed by the indexing system about the recent updates on the index. This information is then exploited at the cache side to identify cached query results that are potentially stale. More specifically, upon an update on the index, an invalidation module located in the backend system transfers these changes to the result caching module. The result caching module then decides, for each query in the cache, if the received changes on the index may render the results of the query stale. In this case, the query is marked as invalid, i.e., its results are considered to be not cached.

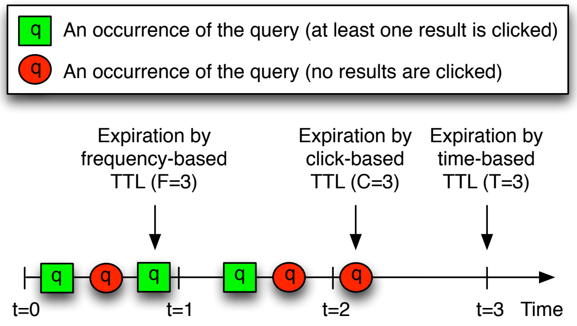

q An occurrence of the query (at least one result is clicked)
q An occurrence of the query (no results are clicked)
Expiration by frequency-based TTL (F=3)
Expiration by click-based TTL (C=3)
Expiration by time-based TTL (T=3)
q
t=0
t=1
t=2
t=3
Time

Figure 1: The points at which the results of a query are expired in different TTL approaches (the query results are assumed to be cached at time $t=0$).

In all techniques mentioned above, the staleness decisions are given via heuristics that do not yield perfect accuracy. In particular, staleness of certain queries may not be identified on time after their results were changed. Such queries may remain in the cache for a long time with stale results. As a remedy to this problem, all of the above-mentioned techniques rely on a complementary mechanism known as time-to-live (TTL). In this mechanism, the validity of selected cache entries are expired based on a fixed criterion with the aim of setting an upper-bound on the possible staleness of a cache entry. In fact, in the absence of more sophisticated refreshing or invalidation techniques, this simple mechanism can provide freshness to a certain degree, just on its own.

The focus of this work is on mechanisms for setting the TTL values of entries in result caches. We consider three alternative approaches: time-based TTL [8], frequency-based TTL [7], and click-based TTL. We evaluate the performance of these alternatives in terms of attained cache freshness and redundant query workload incurred to the backend. Moreover, we propose hybrid approaches that combine the above-mentioned basic approaches. Our results indicate that the best performance can be achieved when time-based TTL is combined with frequency-based TTL.

The rest of the paper is organized as follows. In Section 2, we present three basic TTL approaches and propose two hybrid approaches that combine these basic approaches. The details of our experimental setup are presented in Section 3. Section 4 provides the experimental results. We conclude the paper in Section 5.

2. TTL APPROACHES

2.1 Basic Approaches

In this section, we present three different strategies for expiring the results of a cached query: time-based TTL [2, 6, 8], frequency-based TTL [7], and click-based TTL. The functioning of these three approaches is illustrated in Fig. 1. We note that, throughout the paper, we assume that the evaluated TTL approaches are not accompanied by more sophisticated refreshing or invalidation mechanisms.

Time-based TTL. Time-based TTL is commonly used in result caches in search engines [8] as well as other types of caching systems. In this approach, every cached query result is associated with a fixed lifetime T. Given a query whose results are computed and cached at time t, the cached results are expired at time point $t+T$. Hence, the expiration point for the query results are known at the time of caching. Beyond time point $t+T$, the results of the query are considered to be invalid (if the query results are not refreshed or already invalidated before that time by some other mechanism) and any request for the results leads to a cache miss. The time-based TTL strategy is especially useful for bounding the staleness of the results of infrequent (tail) queries. In general, larger T values increases the fraction of stale queries served by the cache while smaller values lead to a larger fraction of queries whose results are unnecessarily recomputed. In Fig. 1, the results of query q are expired $T=3$ time units after the results were cached.

Frequency-based TTL. A recently proposed alternative is the frequency-based TTL (or virtual TTL) approach [7]. In this approach, unlike the time-based TTL approach where the expiration point (i.e., $t+T$) is fixed, the expiration point for the results of a query is determined depending on the number of recent occurrences of the query. In particular, the results of a query are assumed to be expired if the query was issued to the search engine F times since its results were cached. The frequency-based TTL approach is effective in bounding the staleness of very frequent (head) queries. In Fig. 1, the results of query q are expired after the query was issued to the search engine for $F=3$ times.

Click-based TTL. To the best of our knowledge, the click-based TTL approach is not proposed before. This approach is somewhat similar to the frequency-based TTL approach in that it relies on the number of recent occurrences of the query. In this approach, however, the expiration is determined taking into account only the query occurrences in which no search results are clicked by the user. In particular, the results of a query are expired after C occurrences with no clicks (such occurrences do not have to be consecutive). The rationale here is to use the absence of clicks on search results as an indication of the staleness. In a sense, every occurrence of the query with no clicks on search results increases the confidence on that the query results are not fresh. In Fig. 1, the results of query q are expired once there were $C=3$ occurrences with no clicks.

2.2 Hybrid Approaches

In this section, we describe two hybrid approaches, where the TTL is set based on a combination of the two or more of the basic TTL approaches presented in the previous section. We evaluate two different logical operators in the combinations: conjunction and disjunction. We omit other operators that we experimented with (e.g., multiplication) since they did not provide superior performance.

Conjunction. In this approach, the cached results are expired only if every basic approach in the combination agrees. In a sense, this approach seeks for consensus to make an expiration decision. For instance, in a combination involving the frequency- and time-based approaches, the cached results of a query are expired once both the frequency and age of the query reaches the specified threshold values. In the example given in Fig. 1, the query results would be expired at time point $t=3$.

Disjunction. This approach is more aggressive than the conjunction approach in that the results are expired as soon as one of the combined approaches raises a flag. Using the same example before, the cached results would be expired right after the frequency of the query has reached three.

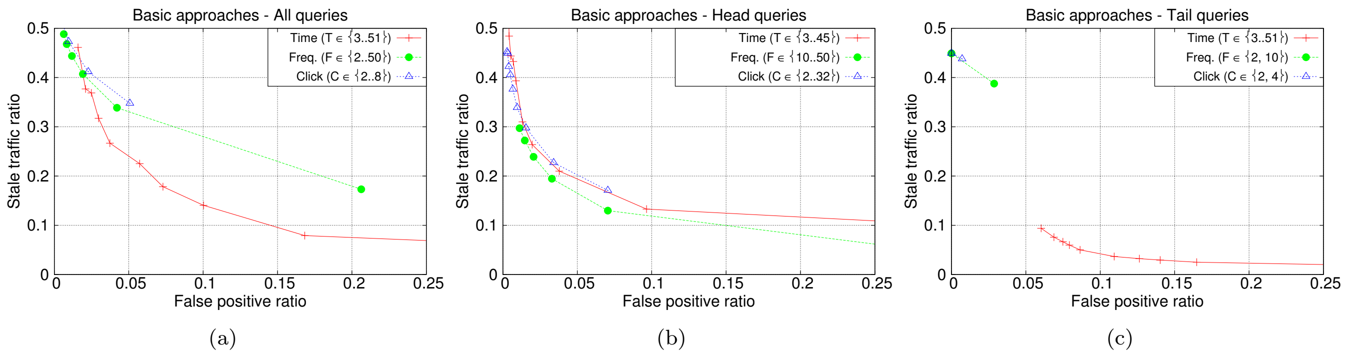


Figure 2: Stale traffic and false positive ratios for the basic TTL approaches using a) all, b) head, and c) tail queries.

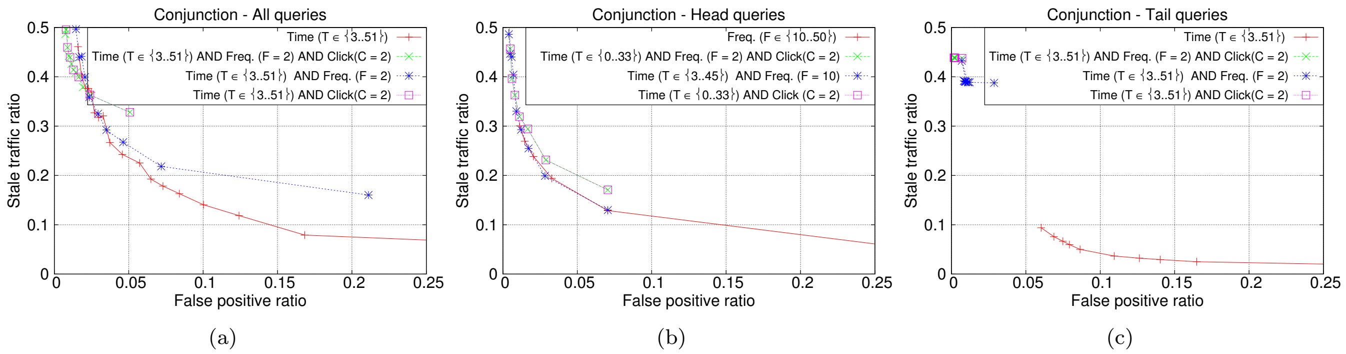


Figure 3: Stale traffic and false positive ratios for the conjunction approach using a) all, b) head, and c) tail queries.

3. SETUP

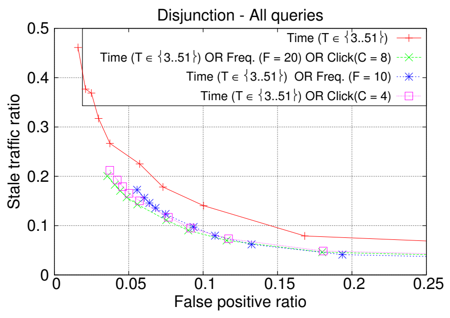
Data. We use a sample of queries submitted to the Spanish web search frontend of Yahoo!. The sample constitutes 2,044,531 queries sorted in timestamp order. We use the queries in the first half of the sample to warm up the cache and those in the remaining half to evaluate the performance. We also provide performance results for head and tail queries. To this end, we sort all unique queries in our query sample in decreasing order of their frequencies and label those in top 1% and bottom 90% as head and tail queries, respectively. We then construct the corresponding head and tail query samples that include only the labeled queries. As before, we use the first half of queries for warming up the cache and the second half for performance evaluation.

Simulation setup. We assume an infinitely large cache so that we can evaluate the performance independent of various parameters, such as 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^*_t (i.e., the fresh answer for q at time t is R^*_t). During the simulations, when a query is first encountered, say at time t, its result R^*_t is cached. In a subsequent submission of the same query at time t', if the TTL was not expired, we assume that the result of the query is served by the cache. Otherwise, we replace the cached result with the result in the query log ($R^*_{t'}$). A result R served by the cache at some time point t is said to be stale if it differs from the result in the query log (R^*_t). As in [1, 6], we consider two query results to be different if the same URLs are not present in exactly the same order.

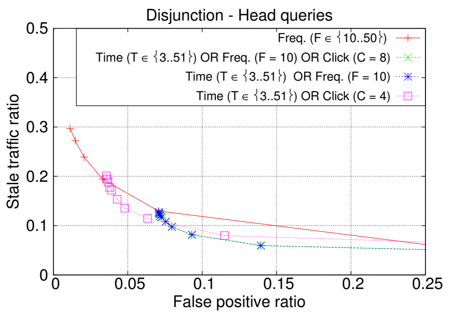
Evaluation metrics. We evaluate the basic and hybrid TTL approaches in terms of the stale traffic (ST) ratio and the false positive (FP) ratio (as in [1, 6]). The stale traffic ratio is the percentage of queries for which the result served from the cache turns out to be stale. The false positive ratio is the percentage of redundant query executions, i.e., the fraction of queries for which the refreshed result is found to be the same as the cached result. We report the performance only for the parameters and combinations that achieve the largest reduction in the sum of ST and FP ratios.

4. RESULTS

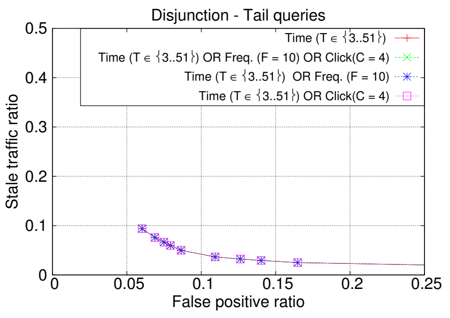
Fig. 2a shows the performance results for the basic TTL approaches using the entire query sample. According to the figure, the time-based approach is superior to the frequency- and click-based approaches since it yields reasonably low ST ratios also with low FP ratios. In contrast, for head queries (Fig. 2b), we observe that the time- and click-based approaches are comparable and the frequency-based approach outperforms both. This is a somewhat intuitive finding. Since head queries are extremely popular, using a fixed time interval as the TTL results in lots of stale results in case of a sudden change in query results. The frequency-based approach, on the other hand, sets an upper bound on the number of stale queries that can be served by the cache (indeed, this is the underlying motivation in [7]). In case of tail queries (Fig. 2c), we find that the frequency- and click-based approaches both perform rather poorly. This is mainly because of the fact that the frequency of tail queries is very low, leading to long time periods before an expiration decision can be made by the frequency- and click-based



(a)



(b)



(c)

Figure 4: Stale traffic and false positive ratios for the disjunction approach using a) all, b) head, and c) tail queries.

approaches, even for the smallest F and C values (see the corresponding points where F and C are equal to two in Fig. 2c). During this long time period, the underlying index and query results are likely to be updated, resulting in very high ST ratios on the cache side.

In order to evaluate the performance of hybrid TTL approaches, we consider three combinations: (time-based, frequency-based), (time-based, click-based), and (time-based, frequency-based, click-based). We prefer to omit the (frequency-based, click-based) combination since it was found to be inferior to all other combinations. We evaluate conjunction and disjunction, separately. In the plots, we display the best-performing basic TTL approach as a baseline.

In Figs. 3a, 3b, and 3c, we present the results for the conjunction of the expiration decisions made by the basic TTL approaches using all, head, and tail queries, respectively. According to the figure, none of the hybrid TTL approaches can outperform the best-performing basic TTL approach in any of the query samples. This is potentially because seeking a consensus among the basic TTL approaches delays the expiration decisions, leading to a larger number of stale queries in turn.

In case of combinations using the disjunction of individual expiration decisions, the picture is different. Fig. 4a indicates that the hybrid approaches can considerably outperform the baseline time-based TTL approach when all queries are used. Figs. 4b and 4c demonstrate the reason behind this. In Fig. 4b, we observe that all combinations are superior to the baseline in case of head queries, the best approach (i.e., the one with the lowest ST ratios) being the (time-based, frequency-based) combination. In the mean time, the hybrid approach does not degrade the performance for tail queries (Fig. 4c), and hence the improvement observed for head queries is reflected to the entire query sample. In other words, using the disjunction of the decisions made by the time- and frequency-based TTL approaches, we combine the best of the two worlds: we improve the performance for head queries without any adverse effect on tail queries. Consequently, we can achieve better overall performance.

5. CONCLUSIONS

We evaluated the performance of three basic time-to-live (TTL) approaches for result caching: time-based TTL, frequency-based TTL, and click-based TTL. We further proposed hybrid TTL approaches that combine these basic approaches. We measured the attained stale query traffic ratio and redundant computation overhead via simulations on a real-life query log obtained from a commercial web search engine. Our experimental results indicate that the best performance is achieved when the time-based TTL approach is combined with the frequency-based TTL approach using a disjunction of the expiration decisions made by these two approaches. We also found that combining the click-based TTL approach with the other two approaches does not bring further improvement.

6. 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'11*, 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'12*, pages 401–412, 2012.

[3] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri. The impact of caching on search engines. In *SIGIR'07*, pages 183–190, 2007.

[4] R. Baeza-Yates, F. Junqueira, V. Plachouras, and H. Witschel. Admission policies for caches of search engine results. In *SPIRE'07*, LNCS, pages 74–85. 2007.

[5] X. Bai and F. P. Junqueira. Online result cache invalidation for real-time web search. In *SIGIR'12*, pages 641–650, 2012.

[6] R. Blanco, E. Bortnikov, F. Junqueira, R. Lempel, L. Telloli, and H. Zaragoza. Caching search engine results over incremental indices. In *SIGIR'10*, pages 82–89, 2010.

[7] E. Bortnikov, R. Lempel, and K. Vornovitsky. Caching for realtime search. In *ECIR'11*, pages 104–116, 2011.

[8] 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'10*, pages 181–190, 2010.

[9] Q. Gan and T. Suel. Improved techniques for result caching in web search engines. In *WWW'09*, pages 431–440, 2009.

[10] S. Jonassen, B. B. Cambazoglu, and F. Silvestri. Prefetching query results and its impact on search engines. In *SIGIR'12*, pages 631–640, 2012.

[11] R. Lempel and S. Moran. Predictive caching and prefetching of query results in search engines. In *WWW'03*, pages 19–28, 2003.