Timestamp-based Cache Invalidation for Search Engines

Sadiye Alici
Computer Engineering Dept.
Bilkent University, Turkey
sadiye@cs.bilkent.edu.tr

Ismail Sengor Altingovde
Computer Engineering Dept.
Bilkent University, Turkey
ismaila@cs.bilkent.edu.tr

Rifat Ozcan
Computer Engineering Dept.
Bilkent University, Turkey
rozcan@cs.bilkent.edu.tr

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

Özgür Ulusoy
Computer Engineering Dept.
Bilkent University, Turkey
oulusoy@cs.bilkent.edu.tr

ABSTRACT

We propose a new mechanism to predict stale queries in the result cache of a search engine. The novelty of our approach is in the use of timestamps in staleness predictions. We show that our approach incurs very little overhead on the system while its prediction accuracy is comparable to earlier works.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Design, Performance, Experimentation

Keywords

Result cache, time-to-live, cache invalidation, freshness

1. INTRODUCTION

Result caching is vital to reduce the backend query workload in search engines. In the past, eviction, admission, and prefetching issues were studied, assuming that result caches have limited capacities. Abundance of cheap storage, however, makes it attractive for search engines to cache practically all past query results. This leads to a new challenge: identifying queries with stale results in the cache [1, 2].

In this work, we propose a new mechanism, based on the use of timestamps, to predict staleness of queries. We assume an incrementally updated index (as in [1]) to which all modifications (addition, deletion, and update of documents) are continuously reflected. Our approach does not involve blind decisions (e.g., the TTL-based invalidation approach in [2]) and, in terms of computation and communication, it incurs very little overhead on the system (unlike [1]). Our experiments indicate that its accuracy in predicting stale queries is comparable to previous works.

2. INVALIDATION FRAMEWORK

Our framework has an offline and an online component (Fig. 1). The offline component is responsible for reflecting document updates on the index and deciding on stale terms and documents. To this end, each term t in the vocabulary and each document d in the collection are associated with timestamps $TS(t)$ and $TS(d)$, respectively. The value of a timestamp shows the last time a term (or document) is deemed to be stale. The staleness of terms and documents are decided based on the policies given in Section 2.1.

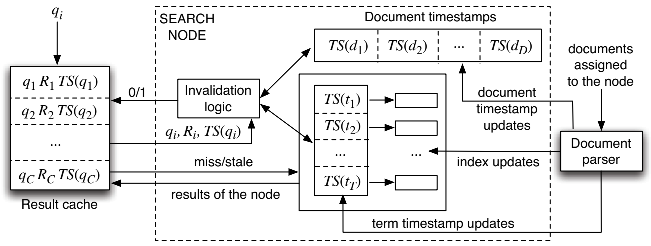

Figure 1: Result cache invalidation architecture.

The online component is responsible for deciding on staleness of a query result. Each query q in the result cache is associated with a timestamp $TS(q)$, showing the last time the query results are computed on the backend. A query result R in the cache can be stale due to two reasons: i) at least one of the documents in R is either deleted or updated (therefore, its rank in R is changed), or ii) at least one document that was not previously in R obtained a score high enough to enter R. The latter case is possible after the addition of a new document or update of an existing document. Our invalidation policy (Section 2.2) aims to predict whether any of these cases holds for a cached query result. In a nutshell, we compare documents' TS values to query's TS value to identify the documents that are deleted or updated after the query result was generated. To identify queries that became stale due to the second reason, we compare query terms' TS values to query's TS value to identify queries whose terms start to appear in some new documents.

2.1 Timestamp Update Policies

We set the timestamp of newly added documents to the current date. For all deleted documents, we set TS to an infinite value. Finally, for a revised document, we compare the old and new versions of the document and set the timestamp to the new version's date only if their lengths (the total number of terms) differ by more than a fixed percentage L (this is similar to [1]). Since each document is assigned to a certain index node via a hash function, we store a document's TS value only on the associated index node.

For each term in the index (on each node), we keep an update counter that can be incremented whenever the term's posting list is modified by addition or deletion of postings (an update is modeled as a deletion followed by an addition). Here, we only consider the modifications due to postings that are newly added to a term's list. When the value of a term's update counter exceeds a certain fraction (F) of its initial posting list length, the term is considered to be stale. Then, a new timestamp is assigned to the term, and its update counter is set to zero.

2.2 Query Result Invalidation Policy

In case of a cache hit, the triplet $\langle q, R, TS(q)\rangle$ is sent to all nodes (Fig. 1). Then, each node concurrently decides whether the cached result is stale or not. A result is considered to be stale if one of the two conditions hold:

- C1: If $\exists d \in R$, s.t. $TS(d) > TS(q)$ (i.e., document is deleted or revised after result generation), or
- C2: If $\forall t \in q$, $TS(t) > TS(q)$ (i.e., all query terms appear in new documents after result generation)

If at least one node decides that the result is stale, the query is re-executed and $TS(q)$ is updated. Otherwise, the cached result is served to the user. Note that the first condition of our policy can correctly identify all results that are stale due to deletion of documents in R. For result documents whose scores may have changed due to an update, we take a pessimistic approach. If a document in R is found to have a larger TS value than the query's TS value, we assume that its rank in R is likely to change and predict the query result as stale. The second condition is intended to (partially) handle the stale results that are caused by a newly added document or an updated document (e.g., after addition of query terms) that was not in R, but now qualifies to enter R. For this case, we take a conservative approach and consider only newly added posting for each term (since deletions are mostly handled by the first condition) and predict a query as stale if each one of its terms now appear in a sufficiently large number of new documents.

We note that our approach is approximate and may miss some invalidations. We anticipate that such cases would be rather rare in practice. Nevertheless, to handle such cases and prevent accumulation of stale results in the cache, we adapt the practice in [1] and couple our policy with an invalidation scheme based on TTL. Thus, in case of cache hit, the TS of a query is first compared to a fixed TTL value, and if it is found to be expired, it is re-executed. Otherwise, our timestamp-based invalidation policy is applied.

3. EXPERIMENTS

Our setup is very similar to [1]. We obtain the Wikipedia snapshot on Jan 1, 2006 as well as all added, revised, and deleted documents in the next two months. We also obtain, from the AOL query log, 10K queries that have at least one clicked answer in the Wikipedia domain. Queries span a period of two weeks, and there are 8630 unique queries. Each day, all document updates are processed in batch, and then, the same set of 10K queries are executed over the updated index to retrieve the top-10 results (i.e., the ground truth). We also obtain staleness predictions from our strategy and the TTL strategy for each day (after the updates) and query.

We evaluate invalidation strategies in terms of the stale traffic (ST) ratio and false positive (FP) ratio, i.e., percent of redundant query executions [1]. For the document TS update policy, we experiment with L values of 0% (i.e., all revisions to a document cause an update on TS values), 1%, and 2.5%. For the term TS update policy, we set F to 10%.

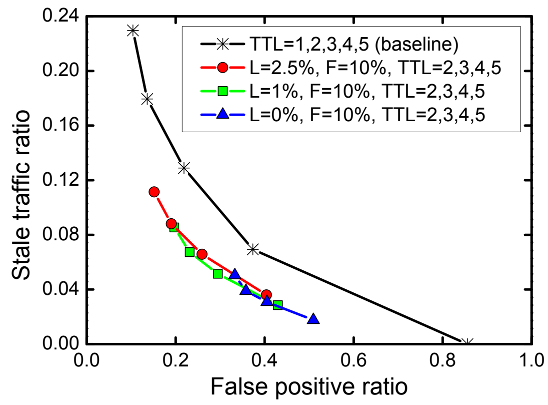

Figure 2: Prediction accuracy (the TTL value increases from right to left in each curve).

According to Fig. 2, our invalidation policy is considerably better than the baseline TTL approach. In particular, for each TTL point, we have a better ST ratio with a similar or lower FP ratio. For instance, when TTL is set to 2, an ST ratio of 7% is obtained with an FP ratio of 37%. Our policy almost halves this ST ratio (i.e., around 4%) for a lower FP ratio of 36%. As expected, for larger values of L, we obtain fewer redundant executions but higher ST ratios.

4. CONCLUDING DISCUSSION

Our work achieves prediction accuracies comparable to [1] (e.g., see [1, Fig. 6]) while the relative improvement of [1] over the TTL scheme is better than ours. However, our goal here is to devise an efficient and practical invalidation policy while providing a prediction accuracy better than the basic TTL approach and comparable to [1]. In this respect, our work has significant efficiency advantages over [1]. First, our invalidation framework operates in a distributed manner, i.e., each index node updates document and term timestamps for its own subset of the collection (offline) and checks staleness of results in case of a cache hit (online). In contrast, [1] involves one or more centralized invalidation predictors that find *all* matching queries in the cache to *every* revised (added or updated) document (offline), which may cause a bottleneck in the system. Second, the network cost of our policy involves the transfer of $\langle q, R, TS(q)\rangle$ triplets between the cache and index nodes for cache hits, whereas the approach in [1] must transfer all synopses for revisions from the parser and further interact with the result cache to find the matching queries. Finally, the timestamp-based invalidation policy has the processing cost of comparing $|R|+|q|$ timestamps, whereas the approach in [1] requires expensive score computations between all revised document synopses and matching queries in the cache.

5. REFERENCES

[1] R. Blanco, E. Bortnikov, F. Junqueira, R. Lempel, L. Telloli, and H. Zaragoza. Caching search engine results over incremental indices. In *Proc. 33rd Int'l ACM SIGIR Conf. on Research and Development in IR*, pages 82–89, 2010.

[2] 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 *Proc. 19th Int'l Conf. on World Wide Web*, pages 181–190, 2010.