Timestamp-based Result Cache Invalidation for Web Search Engines

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

Ismail Sengor Altingovde[*]
L3S Research Center
Hannover, Germany
altingovde@l3s.de

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

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

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

ABSTRACT

The result cache is a vital component for efficiency of large-scale web search engines, and maintaining the freshness of cached query results is the current research challenge. As a remedy to this problem, our work proposes a new mechanism to identify queries whose cached results are stale. The basic idea behind our mechanism is to maintain and compare generation time of query results with update times of posting lists and documents to decide on staleness of query results. The proposed technique is evaluated using a Wikipedia document collection with real update information and a real-life query log. We show that our technique has good prediction accuracy, relative to a baseline based on the time-to-live mechanism. Moreover, it is easy to implement and incurs less processing overhead on the system relative to a recently proposed, more sophisticated invalidation mechanism.

Categories and Subject Descriptors

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

General Terms

Design, Experimentation, Performance

Keywords

Web search, result cache, cache invalidation, freshness

1. INTRODUCTION

Large-scale web search engines maintain a cache of previously computed search results [5]. Successive occurrences of a query are served by the cache, decreasing the average query response latency as well as the amount of query traffic hitting the backend search servers. So far, most works in literature addressed issues related to cache eviction [23], admission [6], and prefetching [18], assuming limited capacity caches and a static web index. In practice, however, search engine caches are stored on disk and hence can be very large, resembling a cache with almost infinite capacity [10]. Moreover, the web index is not static; it is continuously updated due to new document additions, deletions, or modifications.

It has been recently shown that, when an incrementally updated web index is coupled with a very large result cache, the staleness of cache entries becomes an issue since cache hits may result in stale search results to be served [8, 10], potentially degrading user satisfaction. A simple solution to this problem is to associate with every cache entry a time-to-live (TTL) value so that the validity of an entry is expired after some time, i.e., hits on expired cache entries are considered misses and lead to reevaluation of the query. The TTL solution can be coupled with proactive refreshing of stale entries, i.e., recomputing cached results when backend servers have low user query traffic [10]. An alternative is to couple the TTL solution with cache invalidation mechanisms, where the cache entries whose results are predicted to change due to index updates are detected and invalidated [8].

This paper introduces a new mechanism, which utilizes timestamps to facilitate invalidation decisions. The proposed mechanism maintains a separate timestamp for each document in the collection and posting list in the index. Timestamps indicate the last time a document or posting list became stale, decided based on an update policy. Similarly, every query result in the cache is time-stamped with its generation time. In case of a cache hit, the invalidation mechanism compares the timestamp of the query result with timestamps of associated posting lists and documents to decide whether the query result is stale or not, based on a certain invalidation policy.

The proposed approach does not involve blind decisions, as in the case of TTL-based invalidation [10], or techniques that are computationally expensive [8]. In terms of computation, the proposed approach incurs little overhead on the system. Moreover, our approach can be easily integrated into a real search engine, due to its distributed nature. Finally, its accuracy in identifying stale queries and its suc-

[*]Work done while the author was at Bilkent University.

SIGIR'11, July 24–28, 2011, Beijing, China.
Copyright 2011 ACM 978-1-4503-0757-4/11/07 ...$10.00.

cess in reducing redundant query executions at the backend search system is better than that of TTL and reasonably close to that of a sophisticated mechanism [8]. In addition to the proposed invalidation mechanism, our work provides a detailed study of the cache invalidation problem, with several important parameters, such as query length, query frequency, result update frequency, and query execution cost, as real life query streams exhibit different characteristics for different applications (e.g., query results containing news pages may become stale more often than others).

The rest of the paper is organized as follows. Section 2 provides some preliminaries. The proposed cache invalidation framework and policies are presented in Section 3. We discuss our experimental setup in Section 4. Experimental results are presented in Section 5. Section 6 provides a cost analysis for the overhead incurred on the search system. Section 7 provides a survey of related work. We conclude and point to future research directions in Section 8.

2. PRELIMINARIES

2.1 Incremental Indexing Framework

We assume that every search node is responsible for storing and indexing a subset of documents in the collection as well as processing queries over its index (i.e., document-based partitioning). Mapping of documents to search nodes is through hashing of document ids into search node ids. Future modifications in the original document collection (i.e., document addition, deletion, and updates) are communicated to search nodes by the crawler, which continuously monitors the changes in the Web. Search nodes incrementally reflect these changes to their local indexes. We model updates on already indexed documents as the deletion of the old version succeeded by an addition of the new version.

In an ideal incremental indexing setup, changes on the document repository are immediately reflected to local indexes in search nodes. In practice, depending on the freshness requirements, the changes can be accumulated for a small time period and then reflected to index at once [8]. If the majority of the entire index is kept in main memory [13], this update process does not require a strict locking mechanism, i.e., updates can be applied on copies of inverted lists and do not affect queries that are concurrently processed on the lists. After all lists of a particular update are processed, the index pointers are set to point to the new lists. Document properties (such as length, link and content scores, etc.) are also updated accordingly. In our setup, we also maintain and update some timestamp values for affected documents and terms, as we will discuss in Section 3.

In Fig. 1, we illustrate a simplified version of the search system architecture we consider in this work. Typically, a result cache is placed within a broker machine. If the query is found in the cache, the answer is immediately served by the cache; otherwise, the query is processed at the backend search system. Note that, in the incremental indexing framework described above, the underlying index may be modified after query results are generated and stored in the cache. In this paper, we introduce mechanisms to detect and invalidate such stale query result entries in the cache.

2.2 Root Cause Analysis of Staleness

Before discussing our invalidation mechanism, we take a closer look at the causes that make a result in the cache stale. We consider a query result stale if there is a change in the ids or the order of documents in the result [8]. In this respect, we claim that at least one of the following cases should hold to make the cached result R of a query q stale:

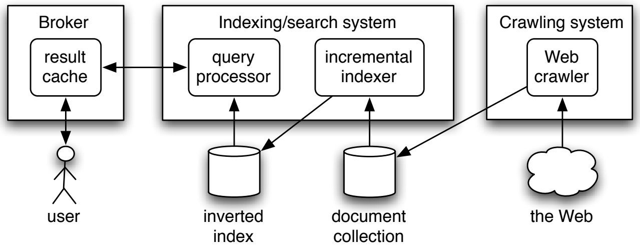


Figure 1: Architecture of the search system.

Case (i). At least one document d that was initially in R is either deleted, or revised in such a way that its rank in R is changed. In the latter case, some query terms that were previously found in d could have been deleted, their frequency could have been modified (i.e., by an increase or decrease), or document length could have been changed (i.e., terms that are not in q can be added to or deleted from d, or their frequency in d can be modified).

Case (ii). At least one document d that was not previously in R can qualify to enter R. In this case, a new document including all query terms (and yielding a high-enough score) could have been added to the collection, or an existing document could have been revised in such a way that its new score qualifies for R. In such an update, some query terms that were not previously in d could have been added to d, the frequency of query terms that appear in d could have been modified or, as in the previous item, the document length could have been changed due to modifications in other terms that are not in q.

We note that our discussion assumes a ranking function that is essentially based on basic document and collection statistics (e.g., TF-IDF, BM25). In practice, a revision on a document can also change the term distances within the text and subsequently, the document score, if a proximity-based scoring function is employed [24]. Similarly, changes on the graph-based features of a document (such as its PageRank score) may also change its overall score. In this paper, we assume a basic scoring function while evaluating the proposed timestamp-based invalidation framework to keep our experimental setup tractable (as in [8]). However, throughout the discussions, we point to possible extensions to our policies to cover more sophisticated ranking functions.

For handling case (i), the primary source of required information is the query result R (in addition to deleted and revised documents). However, handling case (ii) requires some knowledge of documents that are not in R, i.e., all other candidate documents for q in the collection. Obviously, this constraint is harder to satisfy. In the following section, we introduce our timestamp-based invalidation framework (TIF), which involves various policies that attempt to detect if one of the above cases hold for a cached query result. Note that, since it is not always possible to guarantee if any of these cases really occurred (without reexecuting the query), all invalidation prediction approaches involve a factor of uncertainty, and subsequently, a trade-off between prediction accuracy and efficiency. Hence, while tailoring our policies, our focus is on both keeping them practical and efficient to be employed in a real system and as good as the approaches in the literature in terms of prediction accuracy.

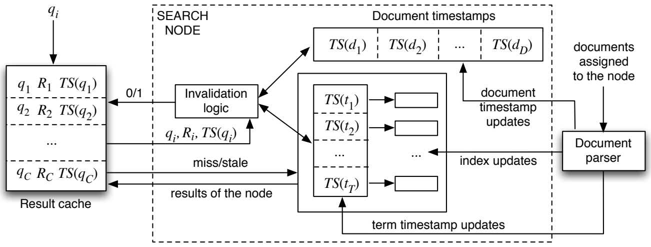


Figure 2: Proposed cache invalidation architecture.

3. INVALIDATION FRAMEWORK

Our framework has an offline (i.e., indexing time) and an online (i.e, query time) component (see Fig. 2). 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 decision for terms and documents are given based on the policies discussed in Section 3.1.

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 at the backend. Our invalidation policy (Section 3.2) aims to predict whether any one of the cases discussed in Section 2.2 hold for a cached 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, and can render the cached result invalid. To predict results 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.

3.1 Timestamp Update Policies

Updating document timestamps. In Fig. 2, for the sake of simplicity and saving space, we present the data stored at a single node in the search cluster. At each index node, in addition to the inverted index and other auxiliary data structures that are typically used for query processing, we keep timestamp values for documents and terms.

We set the timestamp of newly added documents to the current date. For all deleted documents, we set TS to a predefined infinite value.[1] 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 [8]). This parameter is intended to allow a level of flexibility in when a document can be considered as updated. When L is set to 0, every single modification of document would cause a TS update.

For document revisions, it is also possible to consider other features relevant to the underlying score function while determining the new TS. For instance, when the DOM structure or PageRank of an existing document changes (e.g., more than a predefined threshold), the document TS can also be updated. We leave exploring alternative score functions and their impact on invalidation as a future work.

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. That is, keeping track of document TS values is a simple operation and since its cost would be amortized during the in-place index update it would yield almost no additional burden on the system.

Updating term timestamps. For each term in the index (again, on a certain node), we update the timestamp value when a term's list is significantly modified in a time period. Analogous to the document TS case, our decision is guided by the amount of change in the length of a posting list. Furthermore, for terms, we can not only keep track of the number of modifications (addition and deletion of postings) but also estimate which of these modifications are more important in terms of the ranking score. In this respect, we describe two alternative policies, as follows.

i) Frequency-based update: In this policy, we keep an update counter that is incremented whenever the term's posting list is modified by addition or deletion of postings. Here, we only take into account the modifications due to postings that are newly added to a term's list (due to our anticipation that deletions from a revised document may less often make a result stale, in comparison to the addition of new content). When the value of a term's update counter exceeds a certain fraction (F) of its initial posting list length, the term is said to be stale. Then, a new timestamp is assigned to the term, and its update counter is set to zero.

ii) Score-based update: For each term's posting list, we initially sort the postings (in descending order) using the ranking function of the search system and store the score threshold, i.e., the score of posting at rank P ($S@P$). The parameter P can be set to a constant value (such as 10), or adaptively determined for each term as some percentage of the term's posting list length. At each modification to a list, we compute the score of the newly added posting, S_{new}. If $S_{\text{new}} > S@P$ for this list, we assign a new TS to the term, and recompute $S@P$.

In some sense, the latter policy resembles the posting list pruning method proposed by Carmel et al. [11]. In [11], posting lists are again sorted based on their ranking scores; and those postings with scores smaller than the score of the Pth postings are decided to be less worthy, i.e., can be pruned safely. Here, conversely, we imply that only those postings that can enter among the top-P postings of a term are valuable enough to update this term's timestamp.

Note that, both of these timestamp-update polices require only an additional field to be stored per index term, which a very modest requirement. Clearly, the score-based policy is more expensive than the frequency-based one as the former requires ranking of the postings in a list at each timestamp update for that term. In return to its higher cost, we anticipate that the score-based policy may identify those update operations that can change query results more accurately. This expectation is experimentally justified in Section 5.

It is possible to reduce the cost of the score-based update policy by employing a hybrid approach. For instance, it is possible to apply the score-based policy only for terms that appear in the most frequent queries (as can be obtained from previous query logs), whereas the frequency-based policy can be applied to other terms. Alternatively, the terms

[1] This choice simply allows a uniform presentation of our invalidation policy in the next section. In practice, it is also possible to set the timestamp value of a deleted document to a null value, or the deletion can be inferred by the system if it can not be found in the document TS data structure.

to apply the score-based policy can be determined based on the collection frequency or update frequency. These promising ideas are not evaluated in this work due to lack of space, and left as a future work.

3.2 Query Result Invalidation Policies

As seen in Fig. 2, the result cache stores the query string q, result R and timestamp value $TS(q)$.[2] For each cache hit, the triplet $\langle q, R, TS(q)\rangle$ is sent to index servers. Each node, in parallel, predicts whether the cached result is stale or not, using the term and document timestamps and the triplet $\langle q, R, TS(q)\rangle$ for the query in question. A node decides that a result is stale if one of the two conditions holds:

- C_1: If $\exists d \in R$, s.t. $TS(d) > TS(q)$ (i.e., document is deleted or revised after the generation of the query result), or
- C_2: If $\forall t \in q$, s.t. $TS(t) > TS(q)$ (i.e., each query term appeared in some new documents after generation of the query result)

Each node sends its prediction to the result cache, located in the broker. If at least one index server returns a stale decision, the query is re-executed at the index nodes, and R and $TS(q)$ information are updated in the cache; otherwise, the result in the cache is served to the user.

Note that the first condition of our policy can correctly detect all stale results due to deletion of documents in R. For result documents whose scores may have changed due to a revision, we adopt a conservative 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 would most probably change and we decide that the result would be stale. We also propose to relax this latter case by introducing a parameter M, which is a threshold for the number of revised documents to be in R to predict its staleness. That is, we predict a query result as stale if at least M documents in R is found to have a larger TS values than the query's TS value.

The second condition is intended to (partially) handle the stale results that are caused by a newly added document or a revised document (e.g., after addition of query terms), which was not in R but now qualifies to enter. For this case, we again take a conservative approach and decide that a query is stale if each one of its terms now appear in a sufficiently large number of new documents.

We note that the first condition, C_1 may cause some false positives (i.e., predict some results as stale that are, in reality, not), however does not yield any stale results to be served (when $L = 0$ and $M = 1$). That is, all stale results caused by the first case discussed in Section 2.2 would be caught. On the other hand, the second condition, C_2, can yield both false positives and stale results to be served, as it cannot fully cover the second case discussed in the previous section. For instance, assume a document that includes all terms of a particular query but its score does not qualify for top-10 results. Then, during a revision, if some terms that are irrelevant to the query are deleted from this document (so that it is significantly shortened), its score can now qualify for top-10. Clearly, this situation cannot be deduced by either conditions of our invalidation policy. 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 [8] and augment our policy with an invalidation scheme based on TTL. Thus, in case of a cache hit, a query TS is first compared to a fixed TTL value, and if expired, it is reexecuted; otherwise, our timestamp-based invalidation policy is applied.

4. EXPERIMENTAL SETUP

For the sake of comparability, we use the simulation setup introduced in [8] as blueprint and strictly follow that setup in terms of the dataset and query set selection as well as the simulation logic. In what follows, we discuss the details of our experimental setup.

Data. We obtain a public dump of the English Wikipedia site. This dump[3] includes all Wikipedia articles along with their revision history starting from Jan 1, 2006 to April, 2007. We omitted certain pages (e.g., help, user, image, and category pages) from our data as they are not useful for our experiments. The remaining data includes 3.5 million unique pages and their revision information. Since the information about deleted pages is not available, we construct the list of deleted pages on each day by querying Wikipedia using the MediaWiki API.[4]

Simulation setup. We consider all modifications on the collection (additions, deletions and updates) for the first 30 days following Jan 1, 2006. The initial snapshot on that date includes almost one million unique pages. For our dataset, the average number of page additions, revisions and deletions per day are 2,050 (0.2% of the initial dataset), 41,000 (4.1%) and 167 (0.02%), respectively. These numbers are similar to those reported in [8].

Following the practice in [8] (also to reduce the complexity of the simulation setup), we assume that all modifications on the collection are applied as a batch separately for each day. Thus, we create an index for each day by reflecting all changes of a particular day on top of the index constructed for the previous day. We used the open source Lucene library[5] for creating the index files and processing queries.

Query set. As the query set, we sample 10,000 queries from the AOL query log [28] such that each query has at least one clicked answer from Wikipedia.[6] The queries span a period of 2 weeks, and 8,673 of them are unique. We also verify that the query frequency distribution of our sample follows a power-law, which is typical for the Web.

We assume that, in each day, the set of queries is submitted to the search system. Using a fixed set of queries in each day allows evaluating the invalidation approaches in a way independent from other cache parameters (e.g., size, replacement policies, etc.), as discussed in [8]. Therefore, for each day in our simulation, we execute the query set on the current day's index and retrieve top-10 results per query, which constitutes the ground truth set. During query processing, we enforce the conjunctive query processing semantics.

Evaluation metrics. Each invalidation strategy predicts whether a query result in the cache is stale for each day within our evaluation period and accordingly decides either to return the cached result or reexecute the query. As defined in [8, Table 2], we evaluate cache invalidation strategies in terms of the stale traffic (ST) ratio (i.e., the percentage of stale query results served by the system) versus the false positive (FP) ratio (i.e., the percentage of redundant query

[2] Snippets are omitted as they are irrelevant to this problem.

[3] http://www.archive.org/details/enwiki-20070402

[4] http://www.mediawiki.org/wiki/API.

[5] http://lucene.apache.org/

[6] http://en.wikipedia.org

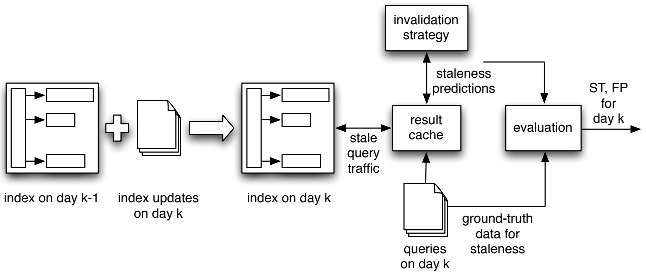


Figure 3: The simulation framework used.

executions). If the returned query result for a particular day differs from that day's ground truth in terms of the listed documents or their order, it is said to be stale. The entire simulation framework is illustrated in Fig. 3. All reported results are averages over the evaluation period.

Note that we do not present results with respect to the false negative (FN) ratio metric, which is also defined in [8], since we consider the ST ratio, which takes into account the frequency of the stale results that are served and stale results accumulated in the cache, as a more realistic metric to evaluate the success of an invalidation approach. Nevertheless, we also obtain plots with the FN ratio and when compared to plots with the ST ratio, we observe almost the same trends reported in [8]. These plots are discarded to make space for our presumably more interesting findings.

Baseline strategies. We compare our timestamp-based invalidation framework (TIF) to two baseline approaches in the literature. The most-straightforward baseline is assigning a fixed time-to-live (TTL) value to each cached query. As a stronger baseline, we implement the cache invalidation predictor (CIP) with its best-case parameters (i.e., using complete documents and score thresholding) [8, Fig. 6].

In our adaptation of the CIP strategy, a cached result is deemed to be stale if it includes at least one deleted document in a particular day (similar to the first case of our result invalidation policy discussed in Section 3.2). While handling additions, first all cached queries that match to the document at hand are determined, using conjunctive query processing semantics. Next, the score of the document with respect to each such query is computed, and if this score exceeds the score of the top-10th document in the cached result, the query result is marked as stale. For each day, we assume that collection statistics (such as inverse document frequency (IDF)) from the previous day are available to CIP.

5. EXPERIMENTAL RESULTS

In Table 1, we summarize the invalidation approaches and related parameters that are investigated in the following experiments. In the baseline TTL strategy each query result is associated with an expiration period (τ), and the result is decided to be stale at the end of this period. Since our simulation setup reflects all updates to the index in batch, each day, the $\tau = 1$ case simply corresponds to no caching at all, i.e., each query is executed everyday. This means that there is no stale traffic, however a very high fraction of queries are executed redundantly (i.e., about 86% in this setup). Not surprisingly, with increasing values of τ, the ST ratio increases while FP ratio decreases.

For the other baseline, CIP, we have two relevant parameters. We employ the document length parameter L in the same way as it is used for our timestamp-based policies; i.e., a revised document is considered for invalidation only if its older and newer versions differ more than L% in terms of the number of terms contained. This is similar to the document modification threshold used in the CIP setup [8]. We also augment the CIP policy with the TTL strategy such that each cached query is also associated with τ.

Table 1: Invalidation approaches and parameters

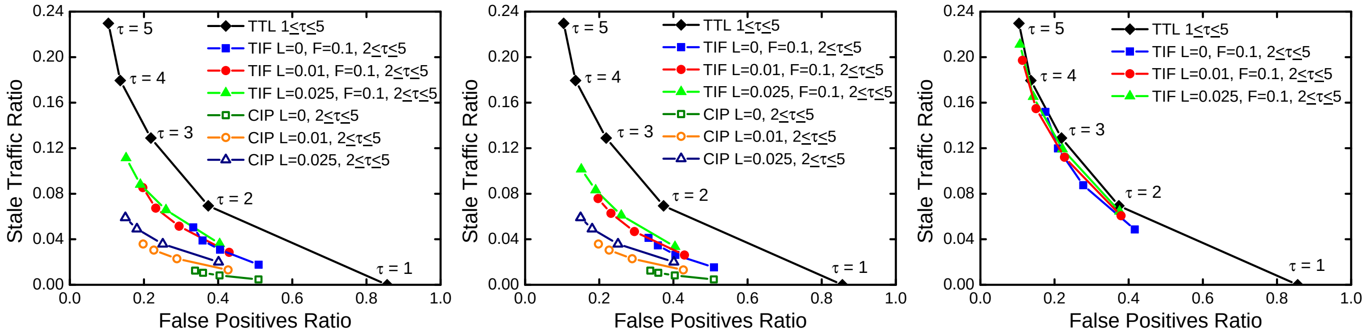
Approach	Parameter	Value range
TTL	τ	$1-5$
CIP [8]	τ	$2-5$
	L	0%, 1%, or 2.5%
TIF	τ	$2-5$
	L	0%, 1%, or 2.5%
	F	10%
	P	top-10
	M	1 or 2

Finally, our timestamp-based invalidation framework (TIF) uses the following parameters. For the document TS update policy, we experiment with L values of 0% (i.e., all revisions to a document causes an update on TS values), 1%, and 2.5%. For updating term timestamps, we use either the frequency-based or score-based TS update policy, as discussed in Section 3.1. For the former case, we set F to 10% (i.e., a term is assigned a new TS when the number of newly added postings exceed 10% of the initial count). For the latter case, we set P to 10 indicating that a term is assigned a new TS if a new posting has a score that can enter the top-10 postings of its list. Recall that, while computing posting scores, we can safely use the statistics from the previous day since the computation actually takes place at the index nodes and during the incremental index update process. Finally, the parameter M takes the values 1 or 2, indicating that a query result R at a particular day is predicted to be stale if it includes either one or two documents updated on that day (see Section 3.2). Our strategy is also augmented with TTL, i.e., the parameter τ is again associated with cached queries.

In Fig. 4(a), we compare the performance of TIF that uses a frequency-based term TS update policy (with L ranging from 0% to 2.5% and $M = 1$) to those of the baselines strategies. The figure reveals that our invalidation approach is considerably better than the baseline TTL strategy. In particular, for each TTL point, we have a better ST ratio with the same or lower FP ratio, and vice versa. For instance, when τ is set to 2 for TTL policy, an ST ratio of 7% is obtained while causing an FP ratio of 37%. Our policy halves this ST ratio (i.e., to less than 4%) for an FP ratio of 36%. Similarly, for τ values 3 and 4, we again provide ST ratio values that are relatively 31% and 38% lower than those of the TTL policy at the same or similar FP ratios.

In Fig. 4(b), we compare the performance of TIF that uses a score-based term TS update policy (with L ranging from 0% to 2.5% and $M = 1$) to those of the baselines strategies. A quick comparison of Figs. 4(a) and (b) reveals that TIF performs better when term update decision are given based on the scores, as expected. In this case, TIF yields about a half of the ST ratio values produced by TTL for the corresponding FP ratios for all values of τ. For instance, TIF yields around an ST ratio of 6% for an FP ratio of 23%, whereas TTL causes an ST ratio of 13% for almost the same number of false positives, i.e., at 22%.

While TIF can significantly outperform TTL baseline, its



(a) (b) (c)

Figure 4: ST vs. FP for TIF: (a) frequency-based, (b) score-based, and (c) frequency-based ($M=2$).

performance is inferior to CIP, although with a smaller margin at the smaller ST ratios. Our implementation of CIP is quite successful for reducing ST ratio; and it is even about 30% better than the best case reported in [8], which might be due to minor variations in the setup or other factors. However, this accuracy does not come for free, as CIP also involves some efficiency and practicality issues (of which discussion is deferred to Section 6). On the other hand, our approach is tailored to provide a compromise between prediction accuracy and efficiency, and not surprisingly, placed between TTL and CIP strategies in Figs. 4(a) and (b). In the following experiments, we also present cases where the gaps between TIF and CIP are further narrowed.

Impact of M. In Fig. 4(c), we report results with $M = 2$ for the case corresponding to Fig. 4(a). As expected, a higher value of M yields smaller false positive predictions but causes higher ST ratios. Surprisingly, even such a slight increase in M drastically affects the performance of TIF, rendering it almost same as the TTL strategy. This indicates that, document revisions are the most important causes of stale results, at least in our setup.

In the rest of this section, we analyze the performance of TIF regarding other aspects such as the query length, frequency, result update frequency and query processing cost.

Impact of query length. Previous works show that queries that are repeatedly submitted to a search engine have smaller lengths and, indeed, single-term queries constitute a large portion of them(e.g., see [32]). Therefore, it would be valuable to investigate the performance of cache invalidation approaches for single-term queries.

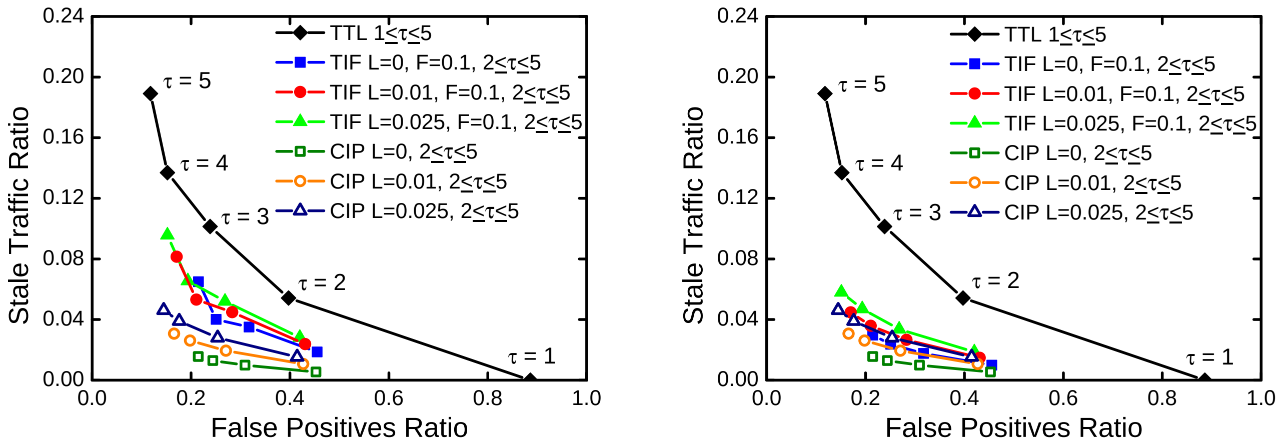
In Fig. 5(a), we compare the performance of TIF with the frequency-based TS update policy to TTL and CIP for single-term queries. As a quick comparison of Fig. 4(a) and Fig. 5(a) reveals, all strategies are more successful for single-term queries, as the absolute values of ST ratios drop. It also seems that the relative gain of our TIF over TTL is slightly improved for this case. For instance, in Figure 4(a), the ST ratios are around 9% and 13% for TIF and TTL for an FP ratio around 22%, respectively. Thus, the relative improvement of TIF over TTL is 31%. For the same FP ratio, Fig. 5(a) reveals ST ratios of 10% to 5%, for TTL and TIF, respectively, indicating a relative improvement of 50%.

For TIF with the score-based TS update policy, performance relative to baseline strategies are even better. In this case, TIF is not only superior to TTL, but it also performs very closely to the CIP strategy, as shown in Fig. 5(b). As the TIF approach employed in this experiment takes into account the changes in top-10 postings per each term, it can more accurately predict the changes in the results of single-term queries. Nevertheless, this experiment implies that for a real-life search engine where a significant amount of repeated queries include a single-term, the achievements of all invalidation strategies would be better and our TIF would provide better prediction accuracy.

Impact of query frequency. The simulation setting that we adapted from [8] involves several simplifications to be able to cope with the dynamicity and complexity of the overall system. One such simplification is for the query set, which is assumed to be repeated everyday. In fact, among 10,000 sampled queries used in our experiments, only 610 of them are repeated more than once in the original log (within the sampling period of two weeks). So, in a separate experiment, we investigated the performance of invalidation strategies for these queries that are more amenable to be repeated in the future. This is important, as repeated queries have different characteristics than those submitted only once [32]; i.e., they are shorter, may include more popular terms, etc.

We find that the performance trends are similar to those of single-term queries for all cases and therefore omit a graphical presentation of the results. The similarity of trends means that most of the frequent queries may involve only a single-term and we found that the fraction of single-term queries in queries repeated more than once reaches to 43%, whereas it is only 20% for our original query set. Again, this is an encouraging result indicating that the performance of invalidation polices would be better in more realistic settings, and the gains of TIF would be even more emphasized.

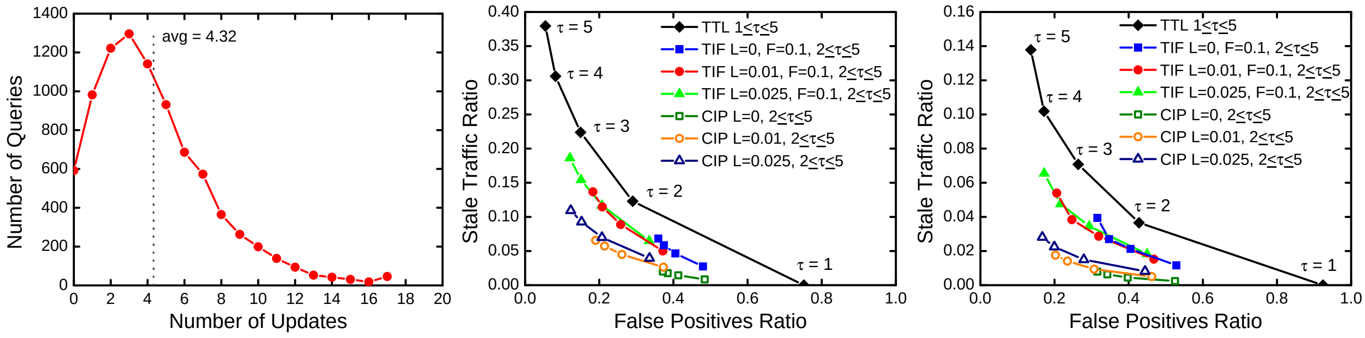
Impact of result update frequency. Depending on the collection change dynamics and the content of the queries, results of some queries may change rapidly (e.g., for a query about an event in news) while the results of some others may remain the same for a much longer time. We investigate the effects of the result update frequency on the invalidation policies. To this end, for each query, we computed the number of times the query result changes within our evaluation period (see Fig. 6(a)), and obtained their average, which is 4.32. In Figs. 6(b) and (c), we report the performance for queries that have higher or lower update frequency than this average value, respectively. In the former case, for all invalidation approaches, ST ratios significantly increase for frequently changing query results (see Fig. 6 (b)). This is an interesting finding, and it implies that more intelligent mechanisms should be developed for such queries. We also note that, while TTL gets considerably worse in this setup (e.g., for $\tau = 2$, ST ratio rises from 7% to 12%, a relative increase of 70%) , the drop in the performance of TIF is more



(a)

(b)

Figure 5: ST vs. FP for $|q|=1$ and TIF: (a) frequency-based and (b) score-based.



(a)

(b)

(c)

Figure 6: (a) Update frequency of queries (30 days), (b) ST vs. FP for $updateFreq > avg$, and (c) ST vs. FP for $updateFreq < avg$ (TIF with the frequency-based TS update policy).

reasonable, as TIF gets closer to CIP for small ST ratios, which would be more preferable in practice. Conversely, for query results that change less frequently, a comparison of Figs. 6(b) and 4(a) shows that all invalidation strategies perform better in this case (i.e., absolute ST ratios drop).

Effects of the query cost. In all experiments up to this point, we report false positive ratio, which is basically the fraction of queries that are executed redundantly. However, not all such queries have the same processing costs, and invalidation strategies may make different choices which may result in the same FP ratio but different processing burden on the search cluster. We investigate whether this phenomenon exists by modeling the processing cost of each query as the sum of its terms' posting list lengths (as in [16]) and repeating our experiments. In Fig. 7, we report FP-cost ratio vs. stale traffic ratio. A comparison of Fig. 4(a) to Fig. 7 reveals that the FP and FP-cost ratios are positively correlated for most cases. Therefore, integrating query costs into the decision mechanisms of the invalidation mechanisms may not yield significant improvements.

6. COST ANALYSIS

Since our timestamp-based invalidation framework and the CIP approach [8] share common underlying assumptions (i.e., most importantly, an incremental index update scheme) and have similar experimental setups, it is natural

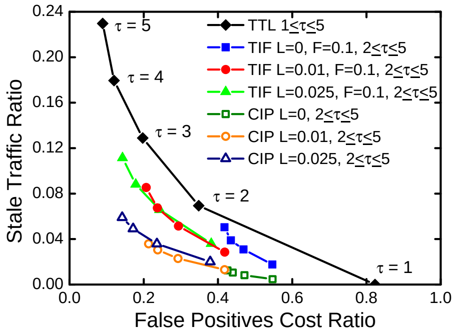


Figure 7: ST vs. FP-cost ratio (TIF with the frequency-based TS update policy)

to compare their efficiency. Recall that, our aim in this study is tailoring an efficient and practical invalidation strategy while providing prediction accuracies higher than the basic TTL scheme and close to that of the CIP. In the previous section, we showed that the latter goal is attainable, i.e., although CIP is generally superior to TIF, there are certain cases (especially for low ST values) where the prediction accuracy of TIF gets close to CIP. Here, we turn our attention to the cost of making invalidation decisions and compare TIF and CIP in terms of practicality and efficiency.

A major difference between the two approaches is the underlying architecture. Our invalidation framework is distributed, i.e., each index node updates document and term timestamps for its own subset of collection (offline) and checks staleness of results in case of a cache-hit (online). In contrast, CIP involves one or more centralized modules that find all matching queries in the cache to every modified (added or updated) document (offline), which may cause a bottleneck in the system. In this respect, we envision that our approach is more practical to fit in a real search engine. Moreover, our architecture allows the changes on the underlying index to affect further staleness predictions as soon as they are reflected. In contrast, CIP should match each document synopsis to the cached queries; and since it may not be possible to process all of the arriving synopses concurrently at a CIP module (especially if the collection changes are accumulated and propagated to CIP in batches), some queries might be served stale until all predictions are completed.

To evaluate efficiency, we formalize the cost of TIF and CIP in terms of the communication volume they cause on the network and the number of comparisons they need to make a prediction. We also provide a back-of-the-envelope calculation using some representative values of involved parameters in cost formulas, as described in Table 2.

i) Communication volume: The network cost of our policy involves the transfer of $\langle q, R, TS(q)\rangle$ triplets between the cache and index nodes for each cache hit. On the other hand, for CIP, the indexer should create a synopsis[7] for each change on the collection (i.e., added or updated document) and propagate it to the CIP module. We assume that the synopses include the term string, its frequency in the document and IDF value [8]. Note that the information in the synopses is needed to compute scores with matching queries in the CIP module. Moreover, the CIP module needs to transfer all cached queries and their results, so that it can make invalidation predictions and then forward its prediction for each query to the cache. In the formulas in Table 3, we neglected these latter costs for simplicity and only consider the cost of transmitting synopses.

ii) Number of comparisons during prediction: Another metric for comparing TIF and CIP is the number of in-memory operations to make a prediction. TIF (with the frequency-based TS update policy) simply compares query term timestamps and result document timestamps to the query timestamp, which is a negligible number of comparisons in the order of $r(q)+u(q)$. In contrast, the CIP module should make expensive score computations between all document synopses and matching queries in the cache. This computation requires an inverted index on the cached queries and accessing the posting list of each term in a synopsis.

As a further complication, invalidating queries with deleted result documents would indeed requires another inverted index on the cached results, i.e., an index mapping document ids to queries. This is not considered in cost formulas as its complexity would be similar to the score computation stage discussed above.

In Table 3, we provide the corresponding formulas for each cost metric discussed above. For a better comparison of TIF and CIP, we also consider a numerical example using the representative values provided in Table 2. We believe that the values presented in the table reflect the state-of-the-art for a large scale search engine. In particular, we consider a collection size of 50 billion documents that is distributed over 5,000 index nodes. For simplicity, we assume that the textual part (after excluding mark-up etc.) of a document includes 500 unique terms and a query includes 2 unique terms, on the average.

Table 2: The parameters and representative values for a large-scale search engine

	Parameter	Value
D	no. of documents in the collection	50 billion
N	no. of index servers	5K
Q	no. of queries per day	100M
C	no. of changed documents in D per day	$0.003\times D$
H	no. of cache-hits per day	$0.5\times Q$
$u(d)$	no. of unique terms in document d	500
$u(q)$	no. of unique terms in query q	2
$l(q)$	length of query q (in bytes)	20
$r(q)$	no. of cached results for query q	10
$p(t)$	posting list length of a term t (in the index over the cache)	10
$l(r)$	length of a unique result identifier (in bytes)	8
$l(p)$	length of a posting (in bytes)	8
$l(s)$	length of a timestamp (in bytes)	4
$l(t)$	length of a term (in bytes)	8

Two key parameters in Table 2 are the collection change-rate and cache hit-rate per day, as they essentially determine the cost of CIP and TIF strategies, respectively. The former is hard to determine and there are several works reporting different rates of change for different subsets obtained from the Web (e.g., see [15, 1, 25]). Among these, the largest scale study of Web change has been conducted by Fetterly et al. [15], which reports that almost 3% of all Web documents change weekly. Relying on this finding, we set the daily change-rate of Web documents as 0.3% of the entire collection. That is, for a collection of 50 billion documents, we assume the number of page additions, deletions and revisions add up to 150M per day, which seems like a reasonable -or, even conservative- estimation (e.g., we anticipate that even the news sites all over the world can be adding millions of new pages in a daily basis). For the cache-hit rate, most works in the literature report a value around 50% depending on the cache parameters (e.g., see [5]), so we also rely on this value. Thus, for a daily load of 100M queries, which is, again, a reasonable assumption for state-of-the-art search engines, 50M queries result in as cache-hits. Finally, for the inverted index constructed over the cached query results in CIP, we assume average posting list length is 10; i.e., a term appears in 10 different queries, on the average.

When we plug the numbers of Table 2 into the formulas presented in Table 3, we see that CIP is more efficient than TIF in terms of the communication volume metric. The daily bandwidth usage of CIP is 5% of that of TIF. The disadvantage of TIF is caused since result triplets are sent to each of the N index nodes. However, in practice, the updates in the collection can be accumulated for a short time period and reflected to the index in batch (as discussed in Section 2). In this case, it is not necessary to resend the queries that occur frequently within a short time (such as "wikipedia") to the index nodes if the query timestamp is larger than the last batch's update time. We anticipate that this would significantly reduce the bandwidth usage of TIF in practice. Moreover, our calculation favors CIP as we as-

[7] We assume the synopsis is created for the entire document content since it yields the best prediction accuracy [8].

Table 3: The cost formulas for CIP and TIF

Cost Metric (per day)	CIP	TIF (with freq.-based TS update)
Communication volume (bytes)	$C \times (l(t) + l(p)) \times u(d)$	$H \times N \times (l(q) + r(q) \times l(r) + l(s))$
No. of comparison operations	$C \times \sum_{t \in d} p(t)$	$H \times (r(q) + u(q))$

sume just a single dedicated server for this purpose. In practice, there maybe several CIP servers in the system, which makes the bandwidth usage of both approaches comparable.

In terms of the number of comparison operations for invalidation predictions, TIF is a clear winner over CIP. In this latter case, CIP makes 1500 times more daily comparisons (i.e., by traversing the posting lists of the index over the cached queries) than TIF, which makes only a constant number of TS comparisons (e.g., 12 comparisons per cache hit according to Tables 2 and 3).

Finally, note that, although we assume a daily load of 100M queries, a search engine may cache a much larger number of queries, maybe all queries seen within a month, mimicking an infinite cache as discussed before. In this case, TIF performance would remain the same, as it only depends on the daily hit rate, but not the queries stored in the cache. In contrast, CIP has to access all cached queries to be able to invalidate them, which may further complicate the synchronization with cache servers and increase the costs.

Our analysis and example calculations show that although TIF causes a higher bandwidth usage, its prediction mechanism is very fast. CIP (at a single server) has lower bandwidth requirements, but actual prediction is much slower, which would be a bottleneck if the change-rate of the collections is high. Furthermore, TIF can be applied on top of a distributed search setup without additional burden, whereas CIP needs to synchronize CIP and cache servers. Hence, we conclude that TIF is a more practical and efficient policy than CIP, while providing better accuracy than TTL strategy, and a reasonably good accuracy in comparison to CIP.

7. RELATED WORK

Index updates. There are three basic approaches to keep a Web index fresh [20]. In the first approach, a new web index is periodically built from scratch over the current document collection and replaces the existing index. For mostly static document collections, this is the preferred approach. In the second approach, newly discovered documents are maintained in a so-called delta index [19]. Queries are processed over both main and delta indexes. Once the delta index is large enough, it is merged with the main index and a new delta index is grown. In the third approach, all modifications are performed on the main index [12, 31]. Modifications may be accumulated and performed in batches [33]. This approach requires maintaining some unused free space at the end of inverted lists, and the efficiency of update operations is important. It is preferred mainly for indexing document collections that are frequently updated. A hybrid strategy between the last two techniques is also possible [9].

Result caching. The first work on result caching in the context of search engines is [23], which provides a comparison between static and dynamic result caching approaches. A probabilistic caching algorithm is introduced in [18]. A two-level cache architecture that combines result and posting list caching is presented in [30]. A similar idea is discussed in [7], in the context of static result caching. A three-level cache architecture, which incorporates a cache for intersection of posting lists besides result and list caches, is proposed in [21]. A few recent works propose cache architectures that combine multiple item types [17, 22, 27]. A hybrid architecture, which combines static and dynamic result caches is discussed in [14]. Many issues in static and dynamic caching are covered by [5]. Several works [26, 3, 4, 16] consider query execution costs in caching. An incremental result cache is proposed in [29] for collection selection.

Staleness of result caches. We are aware of two recent works dealing with staleness of result caches in search engines. A TTL-based invalidation mechanism is presented in [10]. The same work also proposed proactively refreshing cached query results that are predicted to be stale by the TTL mechanism, before they are reissued by the users. Unfortunately, the impact of almost blindly reissuing queries to the backend servers on financial costs is unclear. This work is very different from ours in that it does not take into account the updates on the index when making staleness decisions. The closest work to ours is that in [8], which proposes an invalidation mechanism based on index updates. That work proposes identifying all stale queries by building an inverted index on the queries in the result cache and evaluating updated documents on this index.

In our work, we do not use a sophisticated invalidation mechanisms as proposed in [8]. Instead, our invalidation mechanism relies on very simple heuristics using timestamps assigned to queries, terms, and documents. Yet, we show that reasonable accuracies can be achieved by this mechanism, which incurs less overhead on the system and is much easier to implement. More detailed comparisons to this earlier work is provided in previous sections.

Finally, we have reported some preliminary findings in [2]. This paper extends our previous study in several ways. First, we introduce a new and effective score-based term TS update policy. Second, we investigate the performance of our approach for several parameters, such as query length and frequency, result update frequency, and query execution cost. Third, we implement and compare another closely related invalidation strategy, namely CIP [8], from the literature. Finally, we provide a detailed cost analysis and comparison of our timestamp based invalidation framework and CIP.

8. CONCLUSION

We investigated the trade-off between a cheap-but-inaccurate time-to-live based (TTL) strategy and a recently proposed accurate-but-expensive strategy [8] for invalidating stale results in a web search engine cache. To this end, we presented a simple yet effective approach that maintains timestamps for posting lists and documents, indicating their last revision times, which are compared with the generation time of cached query results to give invalidation decisions. Through a realistic and detailed simulation setup, we verified that the invalidation accuracy of our approach is better than TTL and reasonably close to that of the sophisticated invalidation strategy. Moreover, our approach is easier to implement and incurs less overhead on the system, rendering sophisticated invalidation strategies less attractive.

Result cache invalidation is a recent and active area of research, open to significant improvements. Existing works so far only concentrate on the accuracy of staleness decisions, ignoring other factors, such as the financial cost of these decisions on the search engine company or the satisfaction of users. As a future work, we plan to work on a unified invalidation framework that takes into account all these factors.

9. ACKNOWLEDGMENTS

This work is partially supported by The Scientific and Technological Research Council of Turkey (TÜBİTAK) under grant no. 110E135, FP7 EU Project LivingKnowledge (contract no. 231126) and the European Community funded project COAST-ICT-248036.

10. REFERENCES

[1] E. Adar, J. Teevan, S. T. Dumais, and J. L. Elsas. The Web changes everything: understanding the dynamics of web content. In *Proc. 2nd ACM Int'l Conf. Web Search and Data Mining*, pages 282–291, 2009.

[2] S. Alici, I. S. Altingovde, R. Ozcan, B. B. Cambazoglu, and Ö. Ulusoy. Timestamp-based cache invalidation for search engines. In *Proc. 20th Int'l Conf. World Wide Web (Companion Volume)*, pages 3–4, 2011.

[3] I. S. Altingovde, R. Ozcan, B. B. Cambazoglu, and Ö. Ulusoy. Second chance: A hybrid approach for dynamic result caching in search engines. In *Proc. 33rd European Conference IR Research*, pages 510–516, 2011.

[4] I. S. Altingovde, R. Ozcan, and Ö. Ulusoy. A cost-aware strategy for query result caching in web search engines. In *Proc. 31th European Conference on IR Research*, pages 628–636, 2009.

[5] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri. The impact of caching on search engines. In *Proc. 30th Int'l ACM SIGIR Conf. Research and Development in Information Retrieval*, pages 183–190, 2007.

[6] R. Baeza-Yates, F. Junqueira, V. Plachouras, and H. Witschel. Admission policies for caches of search engine results. In *Proc. 14th Int'l Symposium on String Processing and Information Retrieval*, pages 74–85. 2007.

[7] R. Baeza-Yates and F. Saint-Jean. A three level search engine index based in query log distribution. In *Proc. 10th Int'l Symposium on String Processing and Information Retrieval*, pages 56–65. 2003.

[8] 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. Research and Development in Information Retrieval*, pages 82–89, 2010.

[9] S. Büttcher, C. L. A. Clarke, and B. Lushman. Hybrid index maintenance for growing text collections. In *Proc. 29th Int'l ACM SIGIR Conf. Research and Development in Information Retrieval*, pages 356–363, 2006.

[10] 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. World Wide Web*, pages 181–190, 2010.

[11] D. Carmel, D. Cohen, R. Fagin, E. Farchi, M. Herscovici, Y. S. Maarek, and A. Soffer. Static index pruning for information retrieval systems. In *Proc. 24th Int'l ACM SIGIR Conf. Research and Development in Information Retrieval*, pages 43–50, 2001.

[12] D. Cutting and J. Pedersen. Optimization for dynamic inverted index maintenance. In *Proc. 13th Int'l ACM SIGIR Conf. Research and Development in Information Retrieval*, pages 405–411, 1990.

[13] J. Dean. Challenges in building large-scale information retrieval systems. In *Proc. 2nd ACM Int'l Conf. Web Search and Data Mining*, page 1, 2009.

[14] 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):51–78, 2006.

[15] D. Fetterly, M. Manasse, M. Najork, and J. Wiener. A large-scale study of the evolution of web pages. In *Proc. 12th Int'l Conf. World Wide Web*, pages 669–678, 2003.

[16] Q. Gan and T. Suel. Improved techniques for result caching in web search engines. In *Proc. 18th Int'l Conf. World Wide Web*, pages 431–440, 2009.

[17] S. Garcia. *Search engine optimisation using past queries.* PhD thesis, RMIT University, 2007.

[18] R. Lempel and S. Moran. Predictive caching and prefetching of query results in search engines. In *Proc. 12th Int'l Conf. World Wide Web*, pages 19–28, 2003.

[19] N. Lester, A. Moffat, and J. Zobel. Efficient online index construction for text databases. *ACM Trans. Database Syst.*, 33(3):1–33, 2008.

[20] N. Lester, J. Zobel, and H. E. Williams. In-place versus re-build versus re-merge: index maintenance strategies for text retrieval systems. In *Proc. 27th Australasian Conf. Computer Science*, pages 15–23, 2004.

[21] X. Long and T. Suel. Three-level caching for efficient query processing in large web search engines. In *Proc. 14th Int'l Conf. World Wide Web*, pages 257–266, 2005.

[22] M. Marin, V. Gil-Costa, and C. Gomez-Pantoja. New caching techniques for web search engines. In *Proc. 19th ACM Int'l Symp. High Performance Distributed Computing*, pages 215–226, 2010.

[23] E. P. Markatos. On caching search engine query results. *Comput. Commun.*, 24(2):137–143, 2001.

[24] D. Metzler and W. B. Croft. A Markov random field model for term dependencies. In *Proc. 28th Int'l ACM SIGIR Conf. Research and Development in Information Retrieval*, pages 472–479, 2005.

[25] A. Ntoulas, J. Cho, and C. Olston. What's new on the Web?: the evolution of the Web from a search engine perspective. In *Proc. 13th Int'l Conf. World Wide Web*, pages 1–12, 2004.

[26] R. Ozcan, I. S. Altingovde, and Ö. Ulusoy. Cost-aware strategies for query result caching in web search engines. *ACM Trans. Web*, 5(2):9:1–9:25, May 2011.

[27] R. Ozcan, I. S. Altingovde, B. B. Cambazoglu, F. P. Junqueira, and Ö. Ulusoy. A five-level static cache architecture for web search engines. *Information Processing & Management*, in press, 2011.

[28] G. Pass, A. Chowdhury, and C. Torgeson. A picture of search. In *Proc. 1st Int'l Conf. Scalable Information Systems*, 2006.

[29] D. Puppin, F. Silvestri, R. Perego, and R. Baeza-Yates. Tuning the capacity of search engines: load-driven routing and incremental caching to reduce and balance the load. *ACM Trans. Inf. Syst.*, 28(2):1–36, 2010.

[30] 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 *Proc. 24th Int'l ACM SIGIR Conf. Research and Development in Information Retrieval*, pages 51–58, 2001.

[31] W.-Y. Shieh and C.-P. Chung. A statistics-based approach to incrementally update inverted files. *Inf. Process. Manage.*, 41(2):275–288, 2005.

[32] G. Skobeltsyn, F. Junqueira, V. Plachouras, and R. Baeza-Yates. Resin: a combination of results caching and index pruning for high-performance web search engines. In *Proc. 31st Int'l ACM SIGIR Conf. Research and Development in Information Retrieval*, pages 131–138, 2008.

[33] A. Tomasic, H. García-Molina, and K. Shoens. Incremental updates of inverted lists for text document retrieval. In *Proc. 1994 ACM SIGMOD Int'l Conf. on Management of Data*, pages 289–300, 1994.