A RESULT CACHE INVALIDATION SCHEME FOR WEB SEARCH ENGINES

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING
AND THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULLFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Şadiye Alıcı
October, 2011

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Özgür Ulusoy (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Fazlı Can

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Nihan Kesim Çiçekli

Approved for the Graduate School of Engineering and Science:

Prof. Dr. Levent Onural
Director of the Graduate School

ABSTRACT

A RESULT CACHE INVALIDATION SCHEME FOR WEB SEARCH ENGINES

Şadiye Alıcı
M.S. in Computer Engineering
Supervisor: Prof. Dr. Özgür Ulusoy

October, 2011

The result cache is a vital component for the efficiency of large-scale web search engines, and maintaining the freshness of cached query results is a 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 the generation time of query results with the 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. Throughout the experiments, we compare our approach with two baseline strategies from literature together with a detailed evaluation. We show that our technique has good prediction accuracy, relative to the baseline based on the time-to-live (TTL) mechanism. Moreover, it is easy to implement and it incurs less processing overhead on the system relative to a recently proposed, more sophisticated invalidation mechanism.

Keywords: Web search, result cache, cache invalidation, time-to-live, freshness, adaptive.

ÖZET

WEB ARAMA MOTORLARI İÇİN CEVAP ÖNBELLEĞİ TAZELEME YÖNTEMİ

Şadiye Alıcı
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Prof. Dr. Özgür Ulusoy

Ekim, 2011

Cevap önbelleği, büyük ölçekli Web arama motorlarının verimi için anahtar bileşen konumundadır ve önbellekte bulunan sorgu cevaplarının tazeleğinin korunması güncel araştırma konularından birisidir. Bu probleme çözüm olarak, gerçekleştirdiğimiz çalışma önbellekte bayat cevaba sahip olan sorguların tespit edilmesi için yeni bir yöntem önermektedir. Önerdiğimiz yöntemin temelindeki ana fikir, sorgu cevaplarının taze olup olmadığına karar vermek amacıyla sorgular için cevap oluşturulma zamanının, terim listeleri ve dökümanlar için de güncellenme zamanlarının tutulmasıdır.

Önerilen yöntemin başarımı, gerçek güncellenme zaman bilgisi içeren Wikipedia doküman kümesi ve yine gerçek bir sorgu kümesi kullanılarak değerlendirilmiştir. Gerçekleştirilen deneylerde, önerilen teknik literatürdeki referans yaklaşımlarla karşılaştırmalı olarak incelenmiş ve detaylı bir şekilde değerlendirilmiştir. Bu yöntem ile literatürdeki son-kullanma-süresi (SKS) yaklaşımından çok daha başarılı tahmin sonuçları elde edilmiştir. Buna ilave olarak, önerdiğimiz yöntem literatürdeki gelişmiş bir yönteme göre de daha kolay gerçeklenebilir ve sistem üzerinde merkezi bir darboğaz yaratmayacak şekildedir.

Anahtar sözcükler: Arama motoru, cevap önbelleği, önbellek tazeleme, son-kullanma-süresi, tazelik, uyarlanabilen.

Acknowledgement

I would like to express my sincere gratitude to my supervisor Prof. Dr. Özgür Ulusoy for his invaluable guidance and understanding during this thesis.

I am thankful to Prof. Dr. Fazlı Can and Prof. Dr. Nihan Kesim Çiçekli for kindly accepting to be in the committee and also for giving their precious time to read and review this thesis.

I am very grateful to Dr. İsmail Sengör Altıngövde for his endless support, guidance, and encouragement during this research. Furthermore, I would also like to thank to my colleagues Dr. Rifat Özcan and Dr. B. Barla Cambazoğlu.

I am grateful to The Scientific and Technological Research Council of Turkey (TÜBİTAK-BİDEB) for the financial support they provided during the timespan of this thesis. Moreover, this work is partially supported by The Scientific and Technological Research Council of Turkey (TÜBİTAK) under grant no. 110E135, FP7 EU Project Living Knowledge (contract no. 231126) and the European Community funded project COAST-ICT-248036.

I would like to thank to my office mates Oğuz and Erdem for their friendship and for the enjoyable office times. I would also like to thank my friends Burçin, Elif, Seher, Özlem, and Başak for their valuable friendship and understanding. I would also like express my gratitude to my parents and my brothers.

Last but not least, I would like to thank my husband, Kamil Boratay Alıcı, for being with me all the time. With very special thanks, I dedicate this thesis to him.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The development of world-wide web appeared at the end of 1980s [1] and one could have hardly imagined its current impact those days. Although the time passed from the introduction of the web is not long, the amount of its content has grown rapidly since then. Right after the introduction of the web, at the beginning of 1990s search engines have become an issue and it did not take much time before the development of the first generation web search engines between 1995 and 1997 [2]. The search engine technology has significantly improved in this period and, as web content has become the main source of information, web search engines have become the main entry point to this large amount of information.

At this point it would be appropriate to describe how web search engines work. Web search engines basically download a portion of the available web content and answer user queries based on this downloaded content. The big picture of a web search engine architecture is presented in Figure 1.1. There are three main components of a web search engine. The first component is *crawler* where web content is downloaded continuously. The second component is *indexer* where downloaded documents are indexed after some preprocessing such as parsing or tokenizing. The component at the top level is *query processor*

where users come into action. Here, the queries submitted by users are searched on the indexes and the results are returned to users.

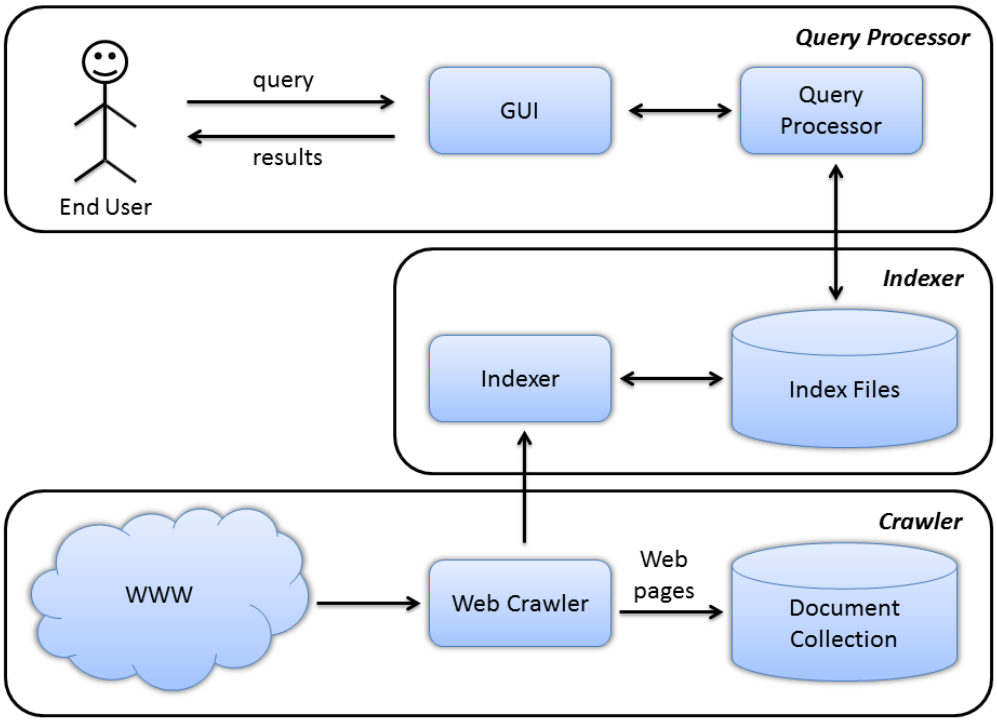


Figure 1.1: General architecture of a web search engine.

Since search engines have become an absolute necessity for users to find information on the web, the number of people using them increases at a very high rate. While an incredible number of users search the web, they all want to get high quality results with a low response time. In order to be able to serve the large number of users within acceptable latency constraints, large-scale search engines maintain a cache of previously computed search results [3]. Successive occurrences of a query are served by the cache (cache hit), decreasing the average query response latency as well as the amount of query traffic hitting the backend search servers. Therefore, successful caching can both lower the number of query executions and shorten the search engine's response time.

Caching is an effective technique for web search engines; however it also comes with its drawbacks. So far, the issues addressed about cache technology are related to cache eviction [4], admission [5], and prefetching [6], assuming limited capacity caches and a static web index. In practice, however, search engine caches are stored on disk and hence they can be very large, resembling a cache with almost infinite capacity [7]. Moreover, the web content is not static; on the contrary, an important aspect of world-wide web is its temporality. Web content is continuously updated with new document additions, modifications on existing documents, and deletions of existing documents.

With decreasing locality of reference, caching mechanism may cause some problems regarding the quality of search results. Examples of situations with decreasing locality of reference include:

- Engines that personalize search results, and
- Engines with incremental or rapidly changing indices.

We know that web content changes continuously. Web search engines continuously download web content and update their indices accordingly. The problem resulting from decreasing locality of reference is that, when the indices of web search engines change, the corresponding cached entries become stale. It has been recently shown that, when an incrementally updated web index is coupled with a very large result cache, the staleness of cached entries becomes an issue since cache hits may result in stale search results to be served [7, 8], potentially degrading user satisfaction.

A simple solution to this problem is to associate a time-to-live (TTL) value with every cache entry so that the validity of an entry will be expired after this amount of time. By this method, hits on expired cache entries are considered as misses and lead to reevaluation of the query. However, this is a blind approach since it invalidates cached entries without any knowledge of changes to the index. This basic solution can be coupled with proactive refreshing of stale

entries, i.e., recomputing cached results when backend servers have low user query traffic [7].

An alternative approach is to couple the TTL solution with cache invalidation mechanisms. In this case, the cache entries whose results are predicted to change due to index updates are detected and invalidated [8]. This is referred to as a sightful mechanism since it provides the cache information about index changes. The mechanism in this work includes a separate module, which is called cache invalidation predictor (CIP), for selectively invalidating cached results. Although this is a more sophisticated mechanism and performs better than the TTL solution in terms of accuracy, the approach proposed in this work is very costly and not practical.

In this thesis, we introduce a new mechanism which utilizes timestamps to facilitate invalidation decisions. Our aim is to devise an invalidation mechanism which is better than TTL and close to CIP in detecting stale results, and better than CIP and close to TTL in efficiency and practicality. For this purpose, the proposed mechanism (based on [9]) maintains a separate timestamp for each document in the collection and posting list in the index [9, 10]. 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 approach proposed in this thesis has several contributions. First of all, this approach does not involve blind decisions, as in the case of TTL-based invalidation [7], or techniques that are computationally expensive [8]. In terms of computation, it incurs little overhead on the system. Moreover, it can be easily integrated into a real search engine, due to its distributed nature. Finally,

the accuracy of the proposed approach in identifying stale queries and success 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 invalidation mechanism, our work also provides a detailed evaluation of the proposed approach 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 thesis is organized as follows. Chapter 2 summarizes studies related to the field of this work and also gives some background information about incremental indexing framework and the root cause analysis of staleness. The proposed cache invalidation framework and related policies are presented in Chapter 3. The experimental setup is explained in detail in Chapter 4. In addition, the experimental results together with the cost analysis are presented in the same chapter. Finally, Chapter 5 concludes our discussions and presents possible future research directions.

Chapter 2

Related Work and Background

In the following sections, we first provide a summary of related research efforts that deal with the quality of web search engines. We then present the background information that is needed to understand the concepts discussed in the following chapters.

2.1 Related Work

Temporality of world-wide web affects the quality of search engines. There are two main research dimensions in the literature regarding this issue. The first dimension deals with the quality of search engines' databases and the other is related to the quality of search engines' results. The following two subsections discuss the research efforts in these two dimensions.

2.1.1 Quality of Search Engines' Databases

We have stated that web content is changing frequently with documents being added, deleted, and updated. This means that new terms need to be added to the index and posting lists of existing terms need to be updated. There are three

basic approaches to keep a web index fresh [11]. The first approach is to periodically reconstruct the index from scratch. This is the simplest approach and it is a preferable solution if the number of changes over time is small. Also, the delay in making new documents searchable should be acceptable for this approach. In the second approach, the necessary modifications are performed on a delta index, meaning that new documents are added to this separate index and updates and deletions of existing documents are also maintained in this delta index [12]. In this case, query processing is performed on both the main index and the delta index. After a specific amount of time or when the delta index reaches a specific amount of size, 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 [13, 14]. Modifications may be accumulated and performed in batches also [15]. This approach is preferable for frequently changing document collections. Lastly, a hybrid strategy between the last two techniques is also possible [16].

The quality of search engines' databases is an important issue since users will find the information they search for only if the search engine index is up to date and contains that information. For this reason, the crawler of a web search engine should operate in continuous mode and it should obtain fresh copies of previously fetched pages [17]. Bias in web crawling and therefore in the indices of web search engines is discussed in [18-20]. The freshness of the databases of three most popular web search engines is studied in [21] and their index update frequency is measured. As a result Google search engine is found to be the best as it updates most of its web pages on a daily basis.

A work that deals with the temporal quality of the data collected for a search engine claims that in order to have 80% freshness over the database, we need to synchronize with at least twice the actual update frequency [22]. Another finding in this paper is that, if real world elements (web documents) change once a day and search engine synchronizes also once a day, then freshness of the

database is 63%. In addition, another work [23] tries to estimate how often web search engines must reindex the web in order to keep current with the changing web content. As a result, it is shown that one-day-current web search engine needs a reindexing period of 8.5 days, and one-week-current web search engine needs a reindexing period of 18 days.

2.1.2 Quality of Search Engines' Results

Result caching in search engines has been an active research area in recent years. The first work on result caching in the context of search engines is [4], which provides a comparison between static and dynamic result caching approaches. Several other works present new methods for static, dynamic, and hybrid result caching approaches [24-27]. Many issues in static and dynamic caching are covered by [3].

In [28], it is claimed that previous works on search engines focused on the crawler ignoring the other parts of a search engine, and they somehow change the point of view from search engine databases to search engine results. In that work, they propose metrics for measuring staleness of a page with respect to when and how many times it is clicked by users. Their main claim is that a stale page clicked on by many users will have a bigger impact on freshness of a search engine than many pages that rarely show up in user search results.

In the previous subsection, we have presented the works that try to maintain up to date databases for web search engines in order not to return stale results to users. However, without careful real-time management of result caches, stale results might be returned to users despite the efforts invested in keeping the database and the index up to date. There are two recent works which deal with the staleness of result caches in web search engines. The first one [7] uses a blind approach while invalidating the queries. In this context, using the blind approach means invalidating cached results without any knowledge of changes

to the index. Cached results are expired based on a time-to-live (TTL) value and the invalidated results are selectively refreshed before they are reissued by the users. For this, the results to refresh are prioritized and they are refreshed in the idle cycles of the servers. However, the impact of almost blindly reissuing queries to the backend servers on financial costs is unclear. The second work dealing with the staleness of result caches is [8], which uses a sightful approach. The sightful approach means that cache is provided information about index changes and invalidation decisions are based on this information. In that work, a cache invalidation predictor (CIP) framework is presented which selectively invalidates cached results of queries whose results are affected by the updates to the index. However, this approach is computationally expensive since it builds an inverted index on the queries in the result cache and evaluates updated documents on this index [8]. Lastly, there is the extension of that work which uses authentic web-scale workloads and standard system metrics [29]. It presents new CIPs that are claimed to better adapt to real workloads. However, in that work a finite size cache is assumed which is not really realistic for search engines. Because search engine caches are stored on disk, they can be very large resembling a cache with almost infinite capacity [7].

2.2 Background

In this section, the preliminary information needed to understand the concepts discussed in the following chapters is provided. This background information includes the general architecture of our incremental indexing framework, and the root cause analysis of staleness.

2.2.1 Incremental Indexing Framework

Large scale search engines have very huge indices which need to be accessed at very high speed. For this reason, their indices are generally distributed across

multiple machines referred to as nodes. By this way, each node searches some part of the distributed index. There are two types of distribution methods for inverted indices which are document-based partitioning (local index organization) and term-based partitioning (global index organization). In document-based partitioning, every node is responsible for storing and indexing a disjoint set of documents in the collection as well as processing queries over its index. In term-based partitioning, each node holds the posting lists of a disjoint set of terms, and performs query processing only if it holds inverted lists relative to the query terms. The two methods are compared for distributed query processing on a real machine in [30] and the conclusion of this work is that the global index organization outperforms the local index organization. However, it is also stated that the local index organization provides high parallelism. So, in our approach we use the local index organization approach which is also the state-of-the-art for search engines.

Furthermore, we employ the incremental indexing approach for updating the search engine database. In this method, firstly every document in the collection should be assigned to one node. This mapping of documents to search nodes is obtained through hashing of document ids into search node ids. After the initial assignment, future modifications occur in the original document collection (i.e., document addition, deletion, and updates). These modifications are communicated to search nodes by the crawler, which continuously monitors the changes in the Web. Every search node incrementally reflects these changes to its local index. 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, 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 [31], 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 addition to these, we also maintain and update some timestamp values for updated documents and affected terms. These will be discussed in Chapter 3.

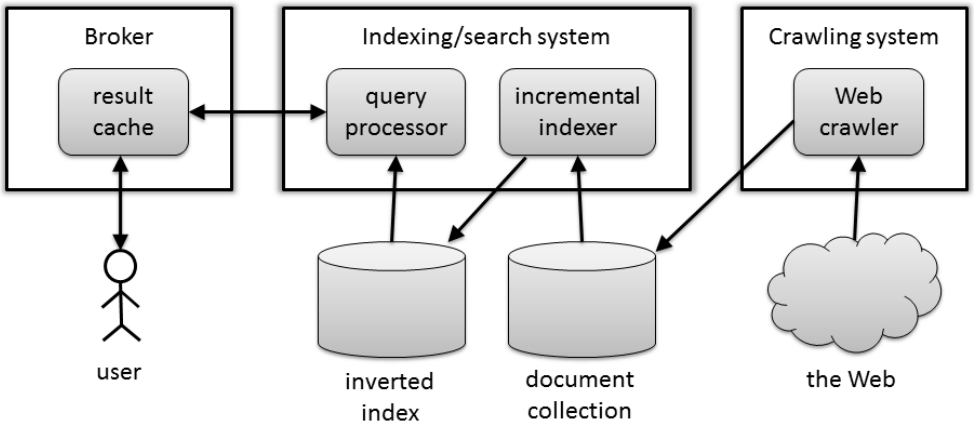


Figure 2.1: Architecture of the search system (S. Alici, I. S. Altingovde, R. Ozcan, B. B. Cambazoglu, and O. Ulusoy, "Timestamp-based result cache invalidation for web search engines," Proceedings of the 34th international ACM SIGIR conference on Research and development in Information, ©2011 ACM, Inc. http://dx.doi.org/10.1145/2009916.2010046. Reprinted by permission.)

In Figure 2.1, we illustrate a simplified version of the search system architecture we consider in this work. This architecture is generally same as the search engine architecture presented in Chapter 1 and it corresponds to one search node's indexing/search system. First of all, there is the crawling system that continuously downloads information from the web into the document collection. Then there is the indexing/search system, in which downloaded information is indexed incrementally and queries are processed on the inverted indices. The result cache is presented as a separate component placed within a broker machine. The important operation we want to focus here is that, when the

user submits a query, we first check if it is found in the cache. If it is found in the result cache, then this is referred to as a “cache hit” and the answer is immediately served by the cache. Otherwise a “cache miss” occurs where the query is sent to all of the nodes and processed at their own indices. After query processing is completed at all nodes, the results from all of these nodes are collected and merged in the broker node, and the final ranked answer is returned to the user. 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. As a result, we see that while the search engine decreases its response time by serving results from the cache, it may serve stale results to users.

2.2.2 Root Cause Analysis of Staleness

Before discussing our invalidation mechanism, we will take a closer look at the causes that make a result in the cache stale. We consider a query result as 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:

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).

An example scenario for these possibilities is presented in Figure 2.2. In this example, there is the query q and its result set R which contains three documents: d1, d2, and d3. In **(a)**, d1 is deleted from the collection, as a result another document replaces d1 in the result set. In **(b)**, d3 is updated and the

query terms' frequency in d3 has changed (increased in this example). As a result of this, rank of d3 changes from 3 to 1. In **(c)**, d3 is updated and one of the query terms (t1) is completely deleted from d3. For this reason, score of d3 decreases for query q and d3 does not appear in the result set anymore. In the last option, **(d)**, document length of d3 changes and it becomes a shorter document. So, the importance of q's terms in d3 increases which then results in a rank change for document d3.

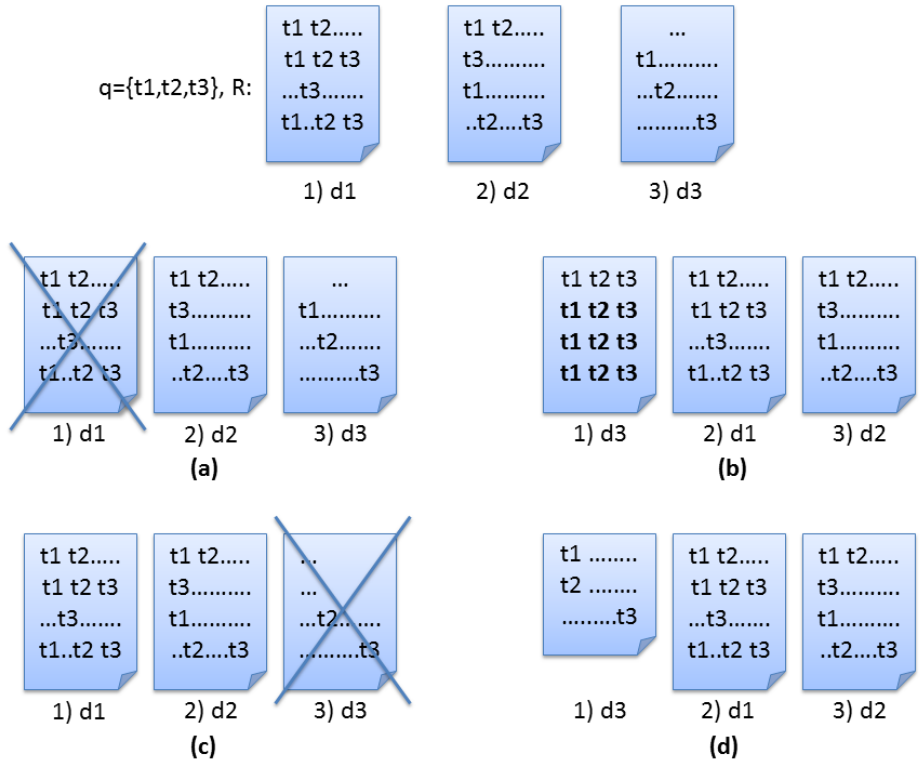


Figure 2.2: Example staleness scenarios that depend on document deletions/updates for documents appearing in the result set (R).

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.

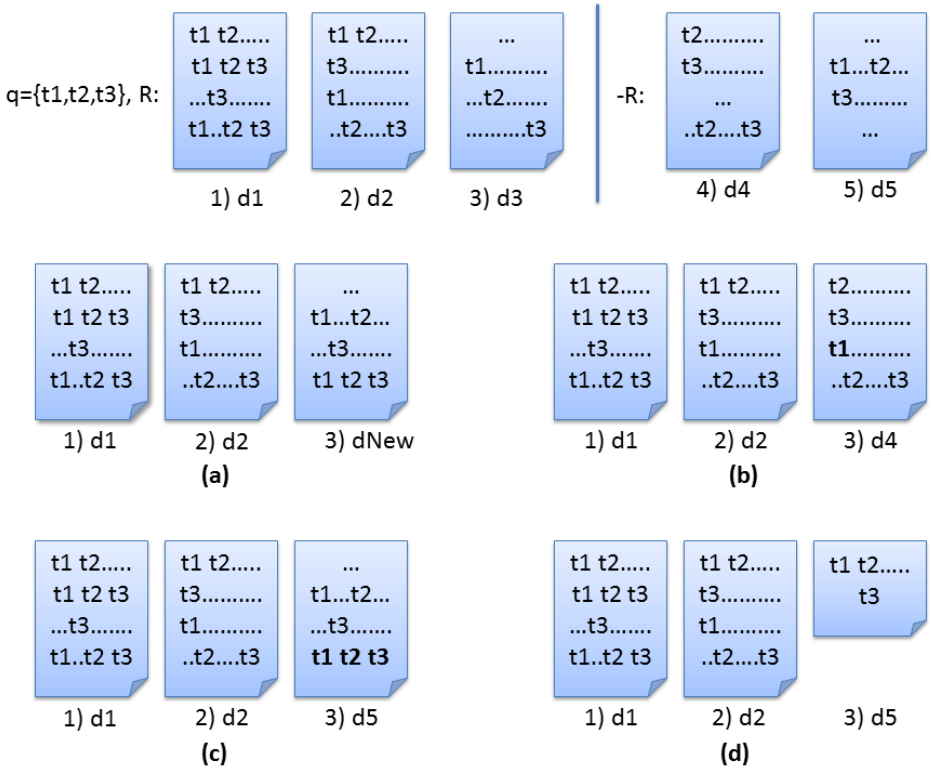


Figure 2.3: Example staleness scenarios that depend on document additions/updates for documents not appearing in the result set (R).

An example scenario for these possibilities is presented in Figure 2.3. In this scenario, again we have query q whose result set (R) includes documents d1, d2, and d3. Moreover, there are two documents d4 and d5 which contain q's terms but do not have a high enough score to enter R. In **(a)**, a totally new document is added to the collection and its score qualifies to enter R for query q. In **(b)**, document d4 is updated (one query term, t1 in this case, is added to d4) and it now has a higher score than d3 to enter R. In **(c)**, d5 is updated so that frequency of query terms increases. Lastly, in **(d)**, document length of d5 changes and as a result it qualifies to enter R.

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 [32]. Similarly, changes on the graph-based features of a document (such as its PageRank score) may also change its overall score. In this thesis, we assume a basic scoring function while evaluating the proposed invalidation framework to keep our experimental setup tractable (as in [8]). However, throughout the discussions, we point to possible extensions of our policies to cover more sophisticated ranking functions.

For handling case (i), the primary source of the required information is the query result R (in addition to the 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 chapter, 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.

Chapter 3

Timestamp-based Result Cache Invalidation Schemes

Our timestamp-based invalidation framework has an offline (i.e., indexing time) and an online (i.e., query time) component (see Figure 3.1). 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 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 aims to predict whether any one of the cases discussed in Section 2.2.2 hold for a cached result.

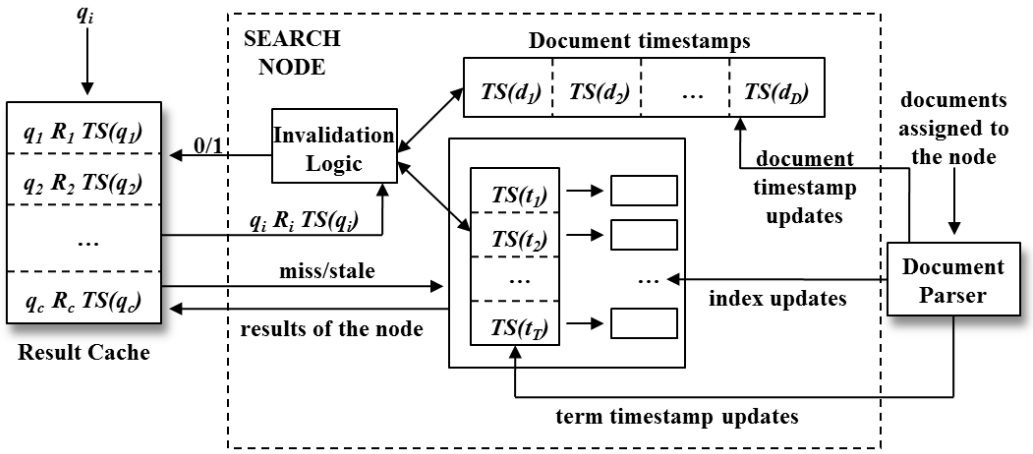


Figure 3.1: Timestamp-based result cache invalidation architecture (S. Alici, I. S. Altingovde, R. Ozcan, B. B. Cambazoglu, and O. Ulusoy, "Timestamp-based result cache invalidation for web search engines," Proceedings of the 34th international ACM SIGIR conference on Research and development in Information, ©2011 ACM, Inc. http://dx.doi.org/10.1145/2009916.2010046. Reprinted by permission.)

In order to predict the staleness of a cached result; we compare documents' *TS* values to query's *TS* value. By this, we try 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 in Section 2.2.2, 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

3.1.1 Updating document timestamps

In Figure 3.1, for the sake of simplicity, 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.

The timestamps of documents are defined as follows. We set the timestamps of the 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 a document causes a *TS* update.

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.

3.1.2 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.

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

[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 cannot be found in the document *TS* data structure.

that are newly added to a term's list. The reason of this decision is 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 (see Figure 3.2). Then, a new timestamp is assigned to the term which reflects the current date, and its update counter is set to zero.

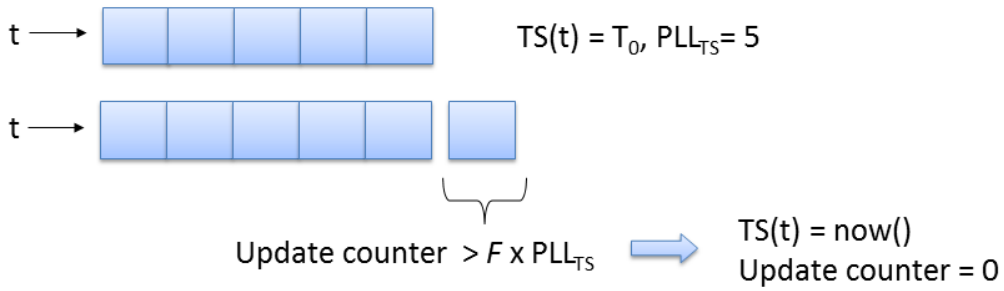


Figure 3.2: Frequency-based term *TS* update policy

Score-based update. In this policy, for each term's posting list, we initially sort the postings (in descending order) using the ranking function of the search system. The working mechanism of this update policy can be seen in Figure 3.3. The figure shows that, after the sort operation, we 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 $S_{new} > S@P$ for this list, we update the *TS* of this term, and recompute *S@P*.

In some sense, the latter policy resembles the posting list pruning method proposed by Carmel et al. [33]. In this work, posting lists are again sorted based on their ranking scores; and those postings with scores smaller than the score of the *P*th 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.

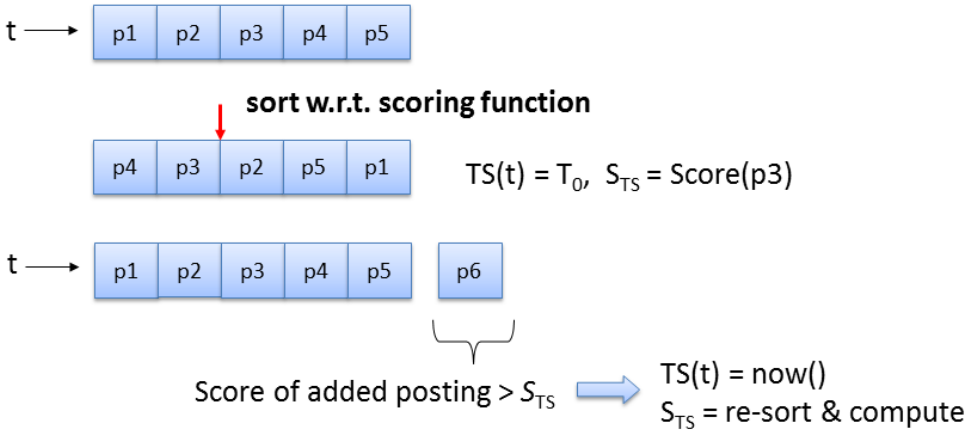


Figure 3.3: Score-based term *TS* update policy

Both of our timestamp-update polices require only an additional field to be stored per index term, which is 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 Chapter 4.

3.2 Query Result Invalidation Policies

As seen in Figure 3.1, the result cache stores the query string *q*, result set *R* and timestamp value *TS(q)*.[2] For each cache hit, the triplet <*q*, *R*, *TS(q)*> 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 <*q*, *R*, *TS(q)*> for the query in question. A node decides that a result is stale if one of the two conditions holds:

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

- 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 the generation of the query result).

After deciding on staleness of a 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.

The first condition of our policy can correctly detect all stale results due to the 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 are found to have a larger TS value 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 appears 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 it 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 2.2.2 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 Section 2.2.2. 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, some terms that are irrelevant to the query are deleted from this document (so that it is significantly shortened) and 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.

Chapter 4

Experimental Setup and Results

In Chapter 3 we have presented our invalidation approach together with the developed policies in detail. Now, we provide an evaluation of our approach. We give a detailed explanation of our experimental setup and results in the following subsections.

4.1 Experimental Setup

In order to evaluate the cache invalidation strategy defined in the previous chapter, there is the need for a detailed, realistic, and high-scale framework. To comply with these requirements and also for the sake of comparability, we use the simulation setup introduced in [8] as blueprint. We strictly follow this setup in terms of the dataset, and query set selection as well as the simulation logic.

4.1.1 Dataset

In order to use throughout the experiments, we obtain a public dump of the English Wikipedia site. This dump[3] includes all Wikipedia articles along with

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

their revision history starting from Jan 1, 2006 to April 30, 2007. The size of the dataset is 85GB compressed and 1.7TB actual. The dataset contains around eight million web pages, but some of these are for purposes other than publishing information on the web. We omitted these 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.

The information regarding to the revision of existing pages and the addition of new pages are obtained using the Wikipedia dataset. However, the information about deleted pages is not available in this dataset. For this reason, we obtained the deleted page information by querying the Wikipedia database using the MediaWiki API.[4] For instance, when the following query "*http://en.wikipedia.org/w/api.php?action=query&list=logevents&letype=delete&lestart=20060101000000&ledir=newer&lelimit=500*" is submitted, we get the deleted page information of up to 500 pages that are deleted on or after January 1, 2006 at 00:00:00 a.m. The results are sorted starting from the first deleted page. By sending such queries we construct the list of deleted pages on each day of the dataset's time interval.

4.1.2 Simulation setup

In order to perform the experiments we need an incremental indexing framework. For this, we consider all modifications on the collection (additions, deletions, and updates) for the first 30 days following Jan 1, 2006. Firstly, we generate an initial index using the Wikipedia content present on this first day. This initial snapshot contains 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].

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

Following the practice in [8] and 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 modifications of a particular day on top of the index constructed for the previous day. By this way, we have an initial index and 30 more incremental indices for the next 30 days. We used the open source Lucene library[5] for creating the index files and processing queries.

4.1.3 Query set

As the query set, we sample 10,000 queries from the AOL query log [34] such that each query has at least one clicked answer from the English Wikipedia domain[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.

4.1.4 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. We decide on the staleness of a

[5] http://lucene.apache.org
[6] http://en.wikipedia.org

query by comparing the returned result for a particular day with the ground truth of that day. If the two result sets differ in terms of the listed documents or their order, then the returned result is said to be stale. The entire simulation framework is illustrated in Figure 4.1. In this figure, we see the index on a day, namely on day *k-1*. When the index updates on day k come (i.e., the document additions, updates, and deletions), the index of day *k* is generated. After this index generation, the queries are run on this index while our invalidation strategy is also in operation. Comparison of the results of our strategy with the ground-truth data gives us the stale ratio and false positive ratio for day *k*.

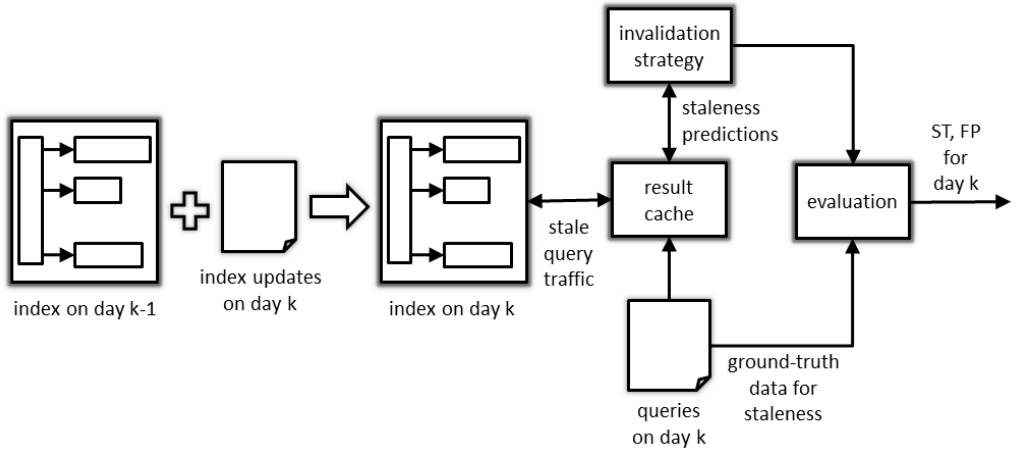


Figure 4.1: The simulation framework used (S. Alici, I. S. Altingovde, R. Ozcan, B. B. Cambazoglu, and O. Ulusoy, "Timestamp-based result cache invalidation for web search engines," Proceedings of the 34th international ACM SIGIR conference on Research and development in Information, ©2011 ACM, Inc. http://dx.doi.org/10.1145/2009916.2010046. Reprinted by permission.)

When we compare the two result sets, one based on our invalidation strategy and the other based on the ground-truth data, one of the following cases will occur.

1. **False Positive (FP):** The cached query result is not stale and the invalidation strategy decides that it is stale.

2. **False Negative (FN):** The cached query result is stale and the invalidation strategy decides that it is not stale.
3. **True Positive (TP):** The cached query result is stale and the invalidation strategy decides that it is stale.
4. **True Negative (TN):** The cached query result is not stale and the invalidation strategy decides that it is not stale.

$$\textbf{False Positive Ratio} = \frac{\text{Redundant query executions}}{\text{Number of unique queries}}$$

$$\textbf{Stale Traffic Ratio} = \frac{\text{Stale results returned}}{\text{Number of query occurrences}}$$

Figure 4.2: Evaluation metrics used in the experiments

As defined in [8], 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 FP ratio (i.e., the percentage of redundant query executions). The formulas for these metrics are shown in Figure 4.2. False positive ratio is calculated by dividing the FP count by the number of cached queries. Here, the FP count corresponds to the redundant query executions. The stale traffic ratio is calculated as sum of the frequencies of the stale queries divided by the overall frequency sum. We also present the FN ratio versus the FP ratio and observe the same trends reported in [8]. However, 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.

4.1.5 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].

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. This means that the newly added document has a high enough score to enter the top-10 result set of query. For each day, we assume that collection statistics (such as inverse document frequency (IDF)) from the previous day are available to CIP.

4.2 Experimental Results

Throughout the experiments, we try to see if our invalidation scheme performs good enough to beat the baseline strategies. We also experiment with different parameters in order to see their effect on different strategies. We summarize the invalidation approaches and related parameters that are investigated in Table 4.1. 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 τ = 1 case simply corresponds to no caching at all. In this case, each

query result is expired every day and thus each query is executed every day. 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.

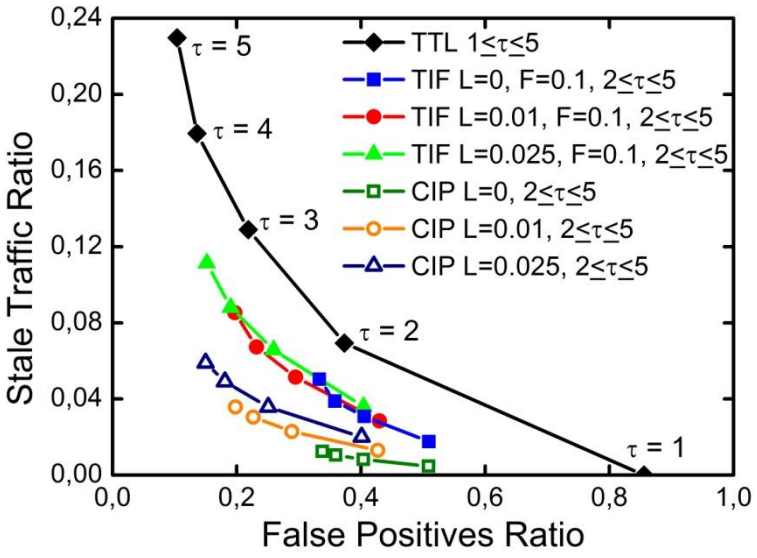
For the other baseline, CIP, we have two relevant parameters. We employ the document length parameter L in the same way as it is used for our timestamp-based policies; i.e., a revised document is considered for invalidation only if its older and newer versions differ by more than L% in terms of the number 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. By this way, if the strategy itself does not invalidate the cached query for a period of τ, then the TTL strategy invalidates this query.

Table 4.1: Invalidation approaches and parameters (S. Alici, I. S. Altingovde, R. Ozcan, B. B. Cambazoglu, and O. Ulusoy, "Timestamp-based result cache invalidation for web search engines," Proceedings of the 34th international ACM SIGIR conference on Research and development in Information, ©2011 ACM, Inc. http://dx.doi.org/10.1145/2009916.2010046. Reprinted by permission.)

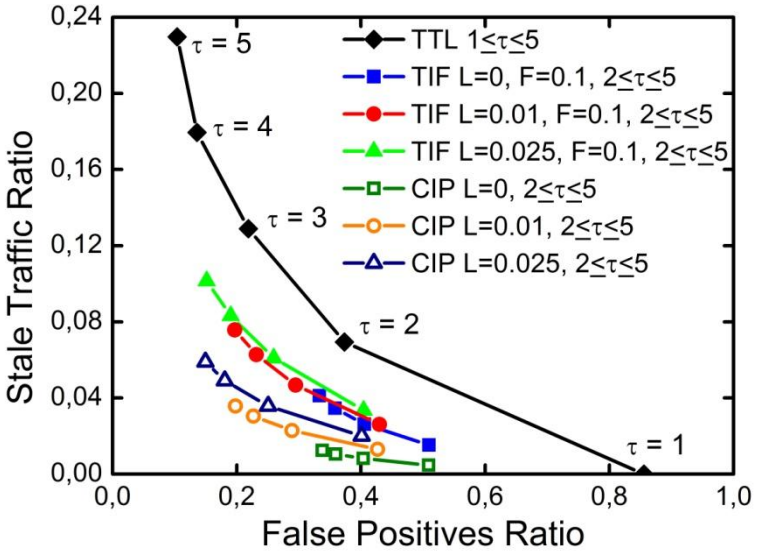
Approach	Parameter	Value range
TTL	τ	1 – 5
CIP [8]	τ	2 – 5
	L	0%, 1%, or 2.5%
TIF	τ	2 – 5
	L	0%, 1%, or 2.5%
	F	10%
	P	top-10
	M	1 or 2

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 the score-based *TS* update policy. For the former case, we set *F* to 10%. This means that, a term is assigned a new *TS* when the number of newly added postings exceed 10% of the initial posting list size. For the latter case, we set *P* to 10 indicating that a term is assigned a new *TS* if a newly added posting has a score that can enter the top-10 postings of its list. We note 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 Figure 4.3(a), we compare the performance of TIF that uses a frequency-based term *TS* update policy to those of the baseline strategies. In this experiment, *L* ranges from 0% to 2.5% and $M = 1$. 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.



(a)



(b)

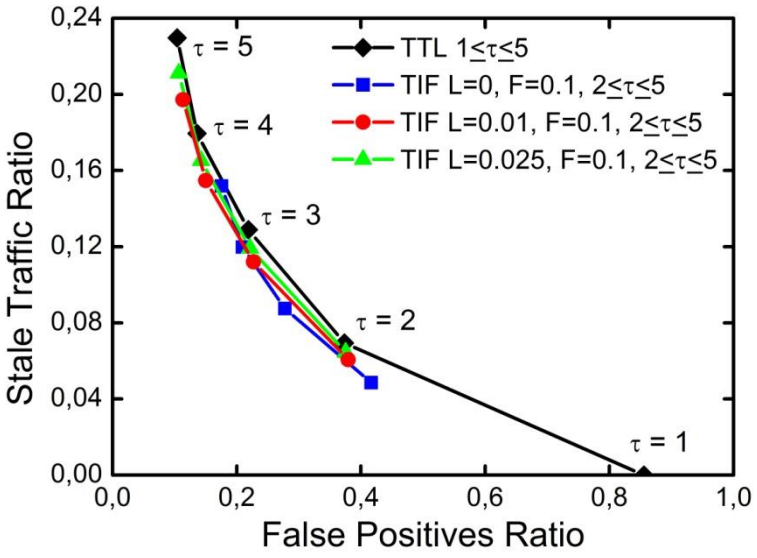
Figure 4.3: ST vs. FP for TIF: (a) frequency-based term *TS* update policy and (b) score-based term *TS* update policy (S. Alici, I. S. Altingovde, R. Ozcan, B. B. Cambazoglu, and O. Ulusoy, "Timestamp-based result cache invalidation for web search engines," Proceedings of the 34th international ACM SIGIR conference on Research and development in Information, ©2011 ACM, Inc. http://dx.doi.org/10.1145/2009916.2010046. Reprinted by permission.)

In Figure 4.3(b), we compare the performance of TIF that uses a score-based term *TS* update policy to those of the baseline strategies. Again, *L* ranges from 0% to 2.5% and *M* is set to 1. When we compare the results in (a) and (b), we

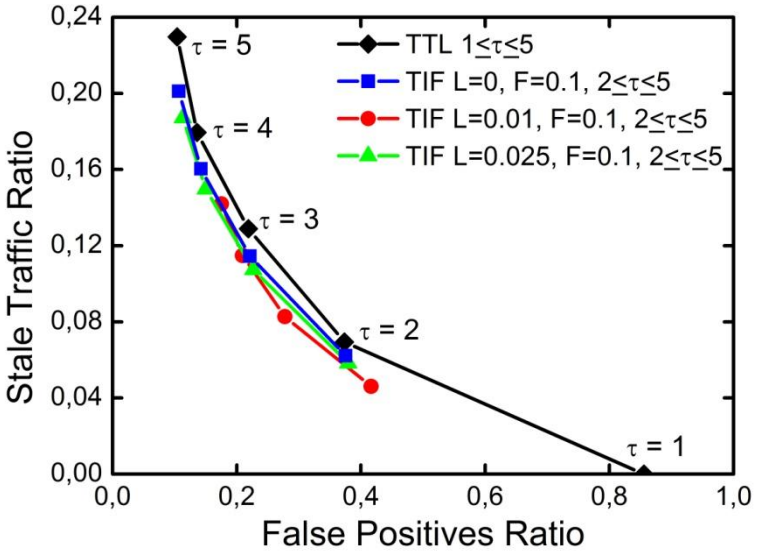
see that TIF performs better when the term update decisions are given based on the scores, as expected. In this case, TIF yields about a half of the ST ratio values produced by TTL for the corresponding FP ratios for all values of τ. For instance, TIF yields around an ST ratio of 6% for an FP ratio of 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 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. 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 Figures 4.3(a) and 4.3(b). In the following experiments, we also present cases where the gaps between TIF and CIP are further narrowed. In the rest of this chapter, we analyze the performance of TIF regarding: M, query length, frequency, result update frequency, and query processing cost.

Impact of M. In Figures 4.4(a) and 4.4(b), we report results with $M = 2$ for the cases corresponding to Figure 4.3. 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 the same as the TTL strategy. This indicates that, document revisions are the most important causes of stale results, at least in our setup.



(a)



(b)

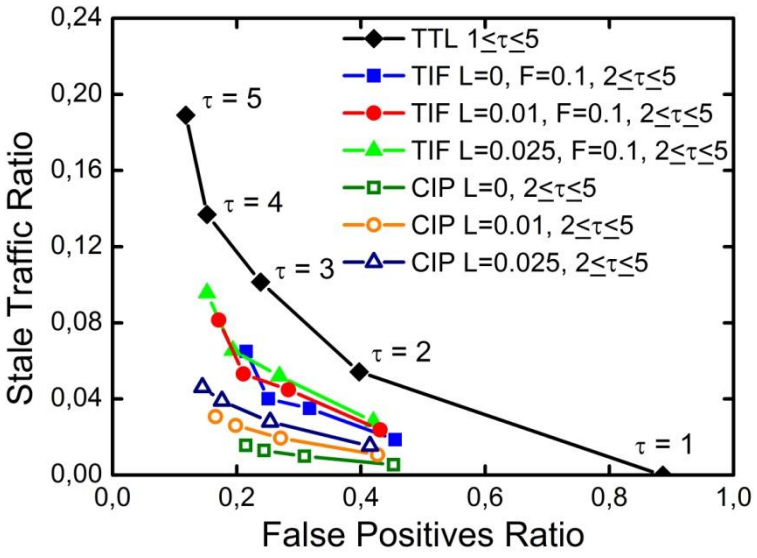
Figure 4.4: ST vs. FP for TIF when *M* = 2: (a) frequency-based term *TS* update policy and (b) score-based term *TS* update policy (S. Alici, I. S. Altingovde, R. Ozcan, B. B. Cambazoglu, and O. Ulusoy, "Timestamp-based result cache invalidation for web search engines," Proceedings of the 34th international ACM SIGIR conference on Research and development in Information, ©2011 ACM, Inc. http://dx.doi.org/10.1145/2009916.2010046. Reprinted by permission.)

Impact of query length. Previous works show that queries that are repeatedly submitted to a search engine have smaller lengths and, indeed, single-

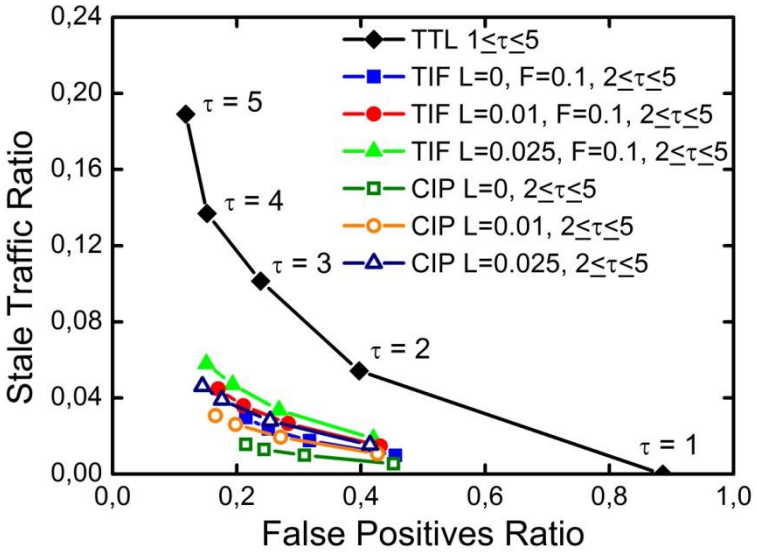
term queries constitute a large portion of them (e.g., see [35]). Therefore, it would be valuable to investigate the performance of cache invalidation approaches for single-term queries.

In Figure 4.5(a), we compare the performance of TIF with the frequency-based *TS* update policy to TTL and CIP for single-term queries. When we compare Figure 4.3(a) and 4.5(a), we see that 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 TIF over TTL is slightly improved for this case. For instance, in Figure 4.3(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, Figure 4.5(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, the performance relative to baseline strategies is 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 Figure 4.5(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, the results of 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 strategy would provide better prediction accuracy.

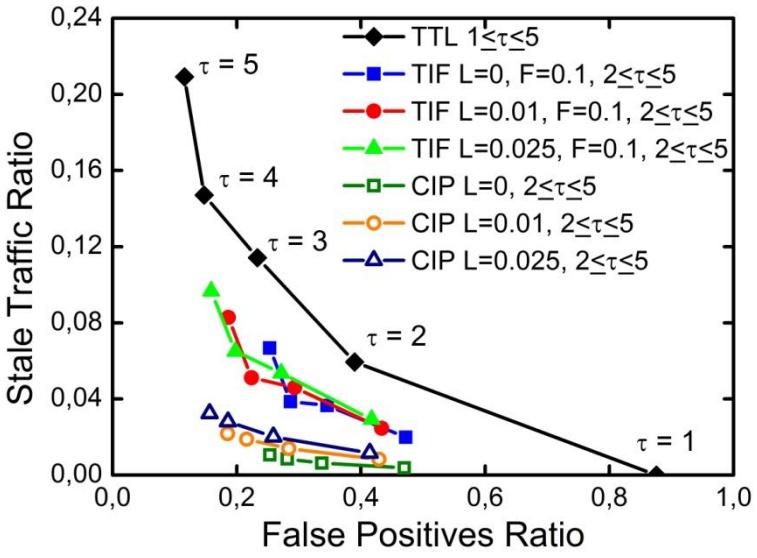


(a)

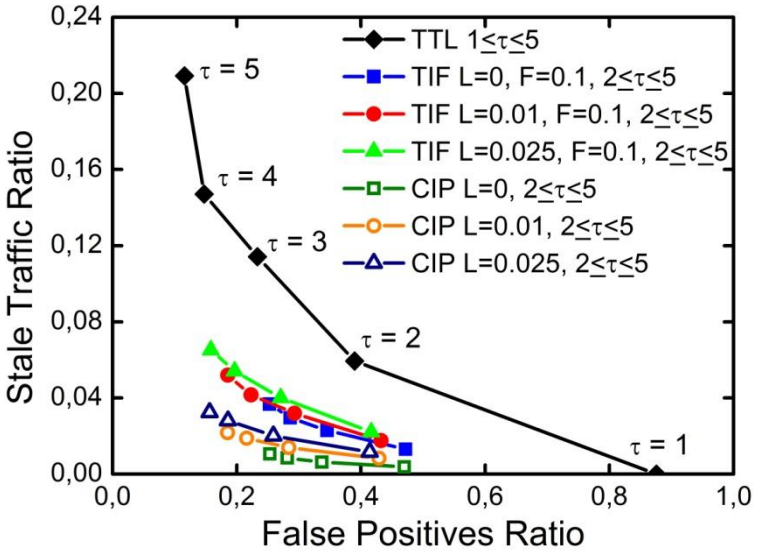


(b)

Figure 4.5: ST vs. FP for $|q| = 1$: (a) frequency-based and (b) score-based (S. Alici, I. S. Altingovde, R. Ozcan, B. B. Cambazoglu, and O. Ulusoy, "Timestamp-based result cache invalidation for web search engines," Proceedings of the 34th international ACM SIGIR conference on Research and development in Information, ©2011 ACM, Inc. http://dx.doi.org/10.1145/2009916.2010046. Reprinted by permission.)



(a)



(b)

Figure 4.6: ST vs. FP for *query frequency > 1*: (a) frequency-based and (b) score-based (S. Alici, I. S. Altingovde, R. Ozcan, B. B. Cambazoglu, and O. Ulusoy, "Timestamp-based result cache invalidation for web search engines," Proceedings of the 34th international ACM SIGIR conference on Research and development in Information, ©2011 ACM, Inc. http://dx.doi.org/10.1145/2009916.2010046. Reprinted by permission.)

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 every day. 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. These are basically the queries with a frequency higher than one. This is important, as repeated queries have different characteristics than those submitted only once [35]; i.e., they are shorter, may include more popular terms, etc.

In Figures 4.6(a) and 4.6(b) we present the results for TIF with frequency-based term *TS* update policy and with score-based term *TS* update policy, respectively. In these experiments, we see that the performance trends are similar to those of single-term queries for all cases. 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. For each query, we computed the number of times the query result changes within our evaluation period (see Figure 4.7). Then, we obtained the average of these, which is 4.32.

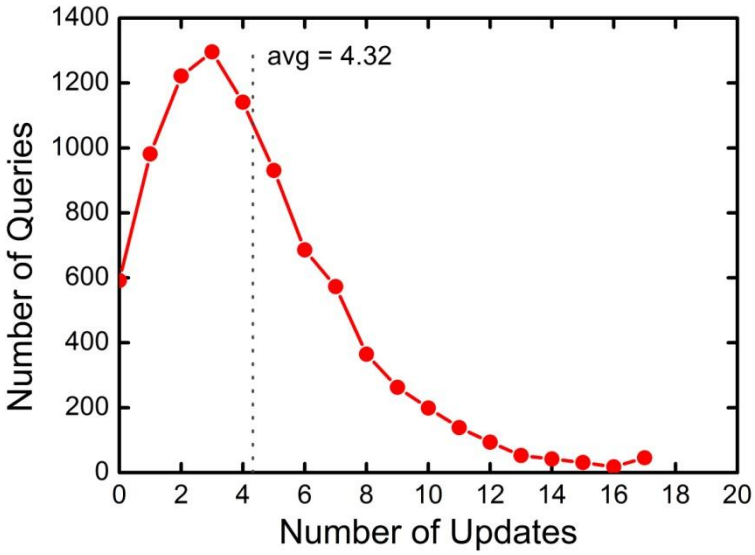
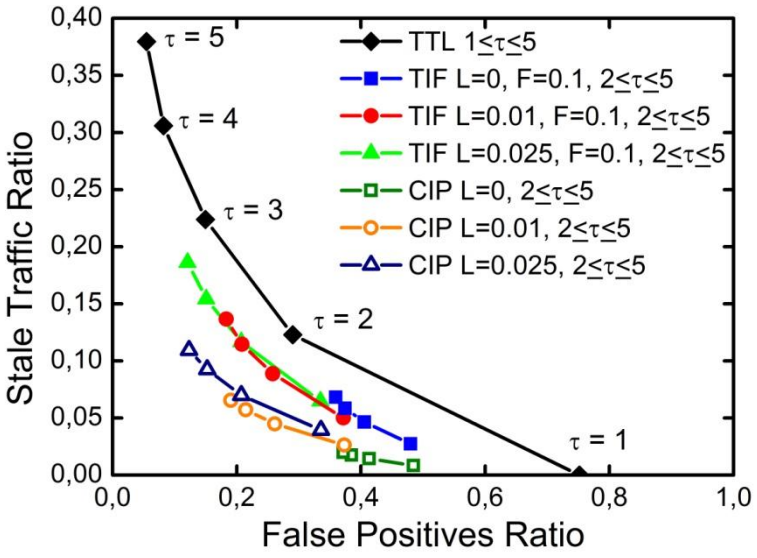
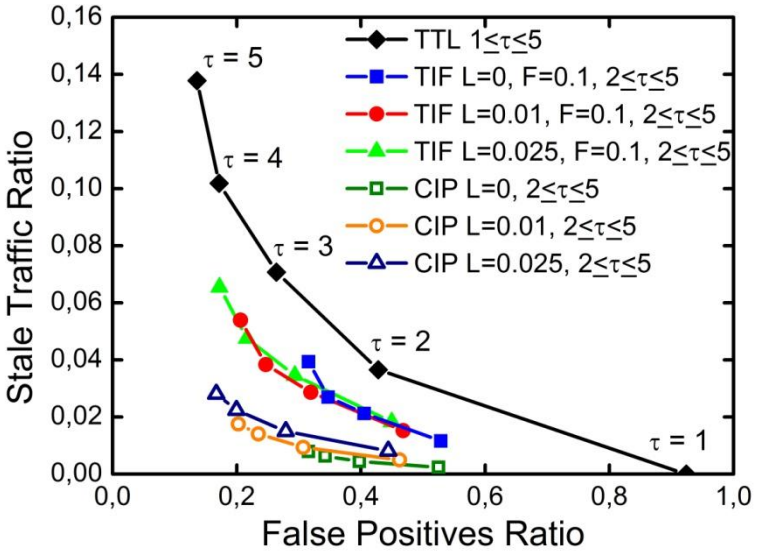


Figure 4.7: Update frequency of queries (S. Alici, I. S. Altingovde, R. Ozcan, B. B. Cambazoglu, and O. Ulusoy, "Timestamp-based result cache invalidation for web search engines," Proceedings of the 34th international ACM SIGIR conference on Research and development in Information, ©2011 ACM, Inc. http://dx.doi.org/10.1145/2009916.2010046. Reprinted by permission.)

In Figures 4.8(a) and 4.8(b), we report the performance for queries that have higher or lower update frequency than this average value for the frequency-based term *TS* update policy for TIF, respectively. In the former case, we see that for all invalidation approaches, ST ratios significantly increase for frequently changing query results (see Figure 4.8(a)). 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 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 Figures 4.8(b) and 4.3(a) shows that all invalidation strategies perform better in this case (i.e., absolute ST ratios drop).

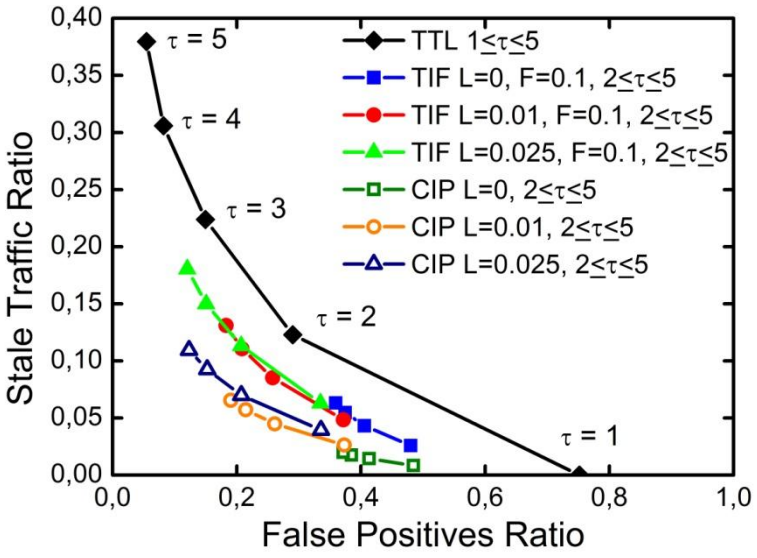


(a)

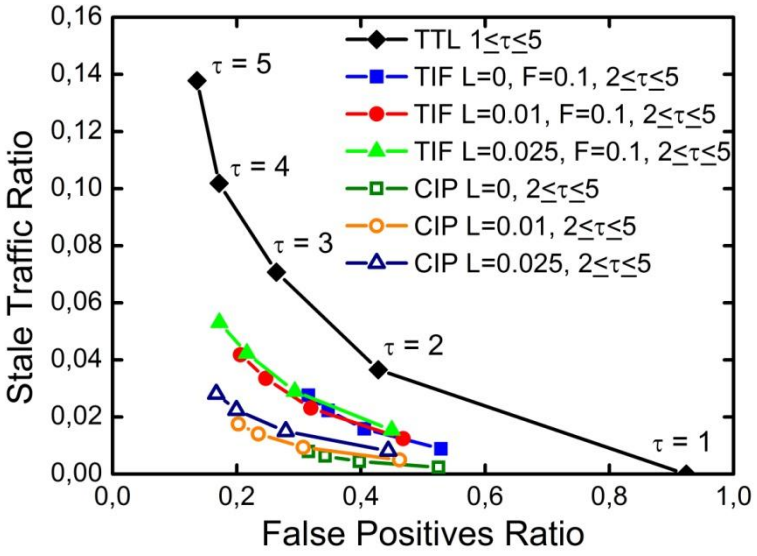


(b)

Figure 4.8: ST vs. FP for frequency-based term *TS* update policy (a) *updateFreq* > *avg* and (b) *updateFreq* < *avg* (S. Alici, I. S. Altingovde, R. Ozcan, B. B. Cambazoglu, and O. Ulusoy, "Timestamp-based result cache invalidation for web search engines," Proceedings of the 34th international ACM SIGIR conference on Research and development in Information, ©2011 ACM, Inc. http://dx.doi.org/10.1145/2009916.2010046. Reprinted by permission.)



(a)



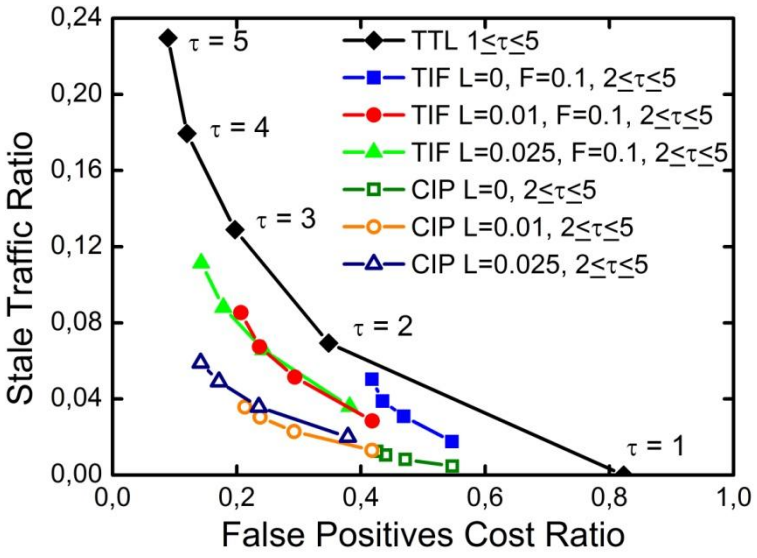
(b)

Figure 4.9: ST vs. FP for score-based term *TS* update policy (a) *updateFreq* > *avg* and (b) *updateFreq* < *avg* (S. Alici, I. S. Altingovde, R. Ozcan, B. B. Cambazoglu, and O. Ulusoy, "Timestamp-based result cache invalidation for web search engines," Proceedings of the 34th international ACM SIGIR conference on Research and development in Information, ©2011 ACM, Inc. http://dx.doi.org/10.1145/2009916.2010046. Reprinted by permission.)

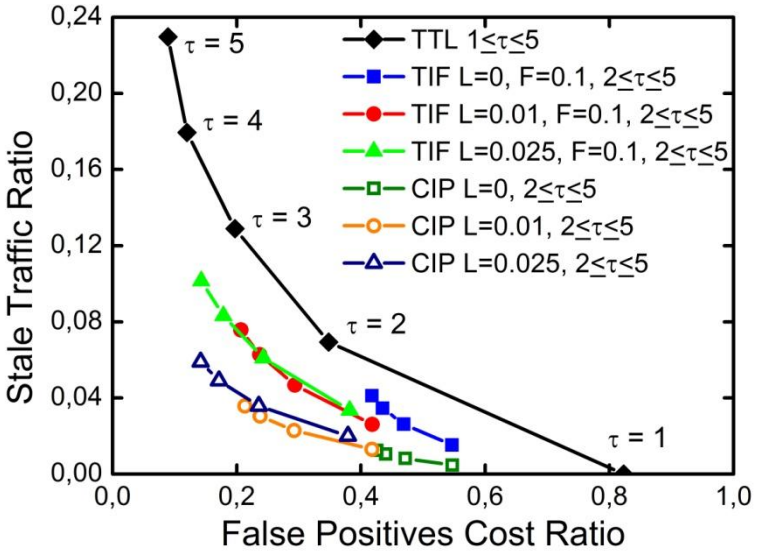
We also present the performance for queries that have higher or lower update frequency than the average for the score-based term *TS* update policy in Figures 4.9(a) and 4.9(b), respectively. The trends of these results are very similar to

those of the frequency-based results. When we compare Figures 4.8 and 4.9, we see that for all cases the absolute ST ratios drop in the score-based results. This drop is better recognized for the smaller ST ratio values. In addition, TIF is again closer to CIP for the score-based update policy results.

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 [36]) and repeating our experiments. In Figure 4.10(a) and 4.10(b), we report FP-cost ratio vs. stale traffic ratio, for the frequency-based and score-based update policies for TIF, respectively. A comparison of Figure 4.3(a) to Figure 4.10(a) reveals that the FP and FP-cost ratios are positively correlated for most cases. Furthermore, the results for the score-based update policy again follow the same trend and are closer to CIP than the frequency-based update policy. Therefore, we conclude that integrating query costs into the decision mechanisms of the invalidation mechanisms may not yield significant improvements.



(a)



(b)

Figure 4.10: ST vs. FP-cost ratio (a) frequency-based and (b) score-based (S. Alici, I. S. Altingovde, R. Ozcan, B. B. Cambazoglu, and O. Ulusoy, "Timestamp-based result cache invalidation for web search engines," Proceedings of the 34th international ACM SIGIR conference on Research and development in Information, ©2011 ACM, Inc. http://dx.doi.org/10.1145/2009916.2010046. Reprinted by permission.)

4.3 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 to compare their efficiency. We should note 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. In this section, we turn our attention to the cost of making invalidation decisions and compare TIF and CIP in terms of practicality and efficiency. Here, we present the comparison between TIF with frequency-based term *TS* update policy and CIP.

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.

Table 4.2: The cost formulas for CIP and TIF (S. Alici, I. S. Altingovde, R. Ozcan, B. B. Cambazoglu, and O. Ulusoy, "Timestamp-based result cache invalidation for web search engines," Proceedings of the 34th international ACM SIGIR conference on Research and development in Information, ©2011 ACM, Inc. http://dx.doi.org/10.1145/2009916.2010046. Reprinted by permission.)

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	$C \times \sum_{t \in d} p(t)$	$H \times (r(q) + u(q))$

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 4.2.

Communication volume. The network cost of our policy involves the transfer of *<q, R, TS(q)>* 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 the 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 4.2, we neglect

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

these latter costs for simplicity and only consider the cost of transmitting synopses.

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 require 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 4.2, 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 4.3. 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.

Two key parameters in Table 4.3 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 [37-39]). Among these, the largest scale study of Web change has been conducted by Fetterly et al. [37], 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 [3]), 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 4.3 into the formulas presented in Table 4.2, 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. 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 assume just a single dedicated server for this purpose. In practice, there may be several CIP servers in the system, which makes the bandwidth usage of both approaches comparable.

Table 4.3: The parameters and representative values for a large-scale search engine (S. Alici, I. S. Altingovde, R. Ozcan, B. B. Cambazoglu, and O. Ulusoy, "Timestamp-based result cache invalidation for web search engines," Proceedings of the 34th international ACM SIGIR conference on Research and development in Information, ©2011 ACM, Inc. http://dx.doi.org/10.1145/2009916.2010046. Reprinted by permission.)

	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

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 4.2 and 4.3).

Finally, we 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 an 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.

Chapter 5

Conclusion

In this thesis, we aim to achieve a compromise between a cheap-yet-inaccurate time-to-live based (TTL) strategy and a recently proposed accurate-yet-expensive strategy [8] for invalidating stale results in a web search engine's cache. Throughout the chapters, we first present introductory information about web search engines. Then, we provide detailed information about the related works in literature and the background information. Following that, we present our timestamp-based approach. Lastly, we investigate the performance of our approach by comparing it to the baseline strategies.

From a general point of view, we present a simple yet effective approach that maintains timestamps for posting lists and documents, indicating their last revision times. These timestamps are compared with the generation time of cached query results to give invalidation decisions. Through a realistic and detailed simulation setup, we verify that the invalidation accuracy of our approach is better than TTL and reasonably close to that of the sophisticated invalidation strategy [8]. Moreover, our approach is easier to implement and incurs less overhead on system resources, 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. As another future work, for document revisions, it is also possible to consider other features relevant to the underlying score function while determining the new *TS* value. 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.

Furthermore, it is also 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 to apply the score-based policy can be determined based on the collection frequency or update frequency. These promising ideas are also left as a future work.

Bibliography

[1] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret, "The World-Wide Web," *Commun. ACM,* vol. 37, pp. 76-82, 1994.

[2] A. Broder, "A taxonomy of web search," *SIGIR Forum,* vol. 36, pp. 3-10, 2002.

[3] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri, "The impact of caching on search engines," presented at the Proceedings of the 30th annual international ACM SIGIR conference on research and development in information retrieval, Amsterdam, The Netherlands, 2007.

[4] E. P. Markatos, "On caching search engine query results," *Computer Communications,* vol. 24, pp. 137-143, 2000.

[5] R. Baeza-Yates, F. Junqueira, V. Plachouras, and H. F. Witschel, "Admission policies for caches of search engine results," presented at the Proceedings of the 14th international conference on String processing and information retrieval, Santiago, Chile, 2007.

[6] R. Lempel and S. Moran, "Predictive caching and prefetching of query results in search engines," presented at the Proceedings of the 12th international conference on World Wide Web, Budapest, Hungary, 2003.

[7] B. B. Cambazoglu, F. P. Junqueira, V. Plachouras, S. Banachowski, B. Cui, S. Lim, and B. Bridge, "A refreshing perspective of search engine

caching," presented at the Proceedings of the 19th international conference on World wide web, Raleigh, North Carolina, USA, 2010.

[8] R. Blanco, E. Bortnikov, F. Junqueira, R. Lempel, L. Telloli, and H. Zaragoza, "Caching search engine results over incremental indices," presented at the Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval, Geneva, Switzerland, 2010.

[9] S. Alici, I. S. Altingovde, R. Ozcan, B. B. Cambazoglu, and Ö. Ulusoy, "Timestamp-based result cache invalidation for web search engines," presented at the Proceedings of the 34th international ACM SIGIR conference on Research and development in Information, Beijing, China, 2011.

[10] S. Alici, I. S. Altingovde, R. Ozcan, B. B. Cambazoglu, and Ö. Ulusoy, "Timestamp-based cache invalidation for search engines," presented at the Proceedings of the 20th international conference companion on World wide web, Hyderabad, India, 2011.

[11] N. Lester, J. Zobel, and H. E. Williams, "In-place versus re-build versus re-merge: index maintenance strategies for text retrieval systems," presented at the Proceedings of the 27th Australasian conference on Computer science - Volume 26, Dunedin, New Zealand, 2004.

[12] N. Lester, A. Moffat, and J. Zobel, "Efficient online index construction for text databases," *ACM Trans. Database Syst.*, vol. 33, pp. 1-33, 2008.

[13] D. Cutting and J. Pedersen, "Optimization for dynamic inverted index maintenance," presented at the Proceedings of the 13th annual international ACM SIGIR conference on Research and development in information retrieval, Brussels, Belgium, 1990.

[14] W.-Y. Shieh and C.-P. Chung, "A statistics-based approach to incrementally update inverted files," *Inf. Process. Manage.*, vol. 41, pp. 275-288, 2005.

[15] A. Tomasic, H. Garcia-Molina, and K. Shoens, "Incremental updates of inverted lists for text document retrieval," presented at the Proceedings of the 1994 ACM SIGMOD international conference on Management of data, Minneapolis, Minnesota, United States, 1994.

[16] S. Büttcher, C. L. A. Clarke, and B. Lushman, "Hybrid index maintenance for growing text collections," presented at the Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval, Seattle, Washington, USA, 2006.

[17] P. R. Christopher D. Manning, Hinrich Schütze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008.

[18] A. Mowshowitz and A. Kawaguchi, "Assessing bias in search engines," *Inf. Process. Manage.*, vol. 38, pp. 141-156, 2002.

[19] L. Vaughan and M. Thelwall, "Search engine coverage bias: evidence and possible causes," *Inf. Process. Manage.*, vol. 40, pp. 693-707, 2004.

[20] V. Cothey, "Web-crawling reliability," *J. Am. Soc. Inf. Sci. Technol.*, vol. 55, pp. 1228-1238, 2004.

[21] D. Lewandowski, H. Wahlig, and G. Meyer-Bautor, "The freshness of web search engine databases," *J. Inf. Sci.*, vol. 32, pp. 131-148, 2006.

[22] J. Cho and H. Garcia-Molina, "Synchronizing a database to improve freshness," presented at the Proceedings of the 2000 ACM SIGMOD international conference on Management of data, Dallas, Texas, United States, 2000.

[23] B. E. Brewington and G. Cybenko, "Keeping Up with the Changing Web," *Computer*, vol. 33, pp. 52-58, 2000.

[24] R. Baeza-Yates, F. Saint-Jean, M. Nascimento, E. de Moura, and A. Oliveira, "A Three Level Search Engine Index Based in Query Log

Distribution String Processing and Information Retrieval." vol. 2857, ed: Springer Berlin / Heidelberg, 2003, pp. 56-65.

[25] 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.*, vol. 24, pp. 51-78, 2006.

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

[27] M. Marin, V. Gil-Costa, and C. Gomez-Pantoja, "New caching techniques for web search engines," presented at the Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, Chicago, Illinois, 2010.

[28] A. Dasdan and X. Huynh, "User-centric content freshness metrics for search engines," presented at the Proceedings of the 18th international conference on World wide web, Madrid, Spain, 2009.

[29] E. Bortnikov, R. Lempel, and K. Vornovitsky, "Caching for realtime search," presented at the Proceedings of the 33rd European conference on Advances in information retrieval, Dublin, Ireland, 2011.

[30] C. Badue, B. Ribeiro-Neto, R. Baeza-Yates, and N. Ziviani, "Distributed query processing using partitioned inverted files," in *String Processing and Information Retrieval, 2001. SPIRE 2001. Proceedings.Eighth International Symposium on*, 2001, pp. 10-20.

[31] J. Dean, "Challenges in building large-scale information retrieval systems," presented at the Proceedings of the Second ACM International Conference on Web Search and Data Mining, Barcelona, Spain, 2009.

[32] D. Metzler and W. B. Croft, "A Markov random field model for term dependencies," presented at the Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval, Salvador, Brazil, 2005.

[33] D. Carmel, D. Cohen, R. Fagin, E. Farchi, M. Herscovici, Y. S. Maarek, and A. Soffer, "Static index pruning for information retrieval systems," presented at the Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval, New Orleans, Louisiana, United States, 2001.

[34] G. Pass, A. Chowdhury, and C. Torgeson, "A picture of search," presented at the Proceedings of the 1st international conference on scalable information systems, Hong Kong, 2006.

[35] 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," presented at the Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval, Singapore, Singapore, 2008.

[36] Q. Gan and T. Suel, "Improved techniques for result caching in web search engines," presented at the Proceedings of the 18th international conference on World wide web, Madrid, Spain, 2009.

[37] D. Fetterly, M. Manasse, M. Najork, and J. Wiener, "A large-scale study of the evolution of web pages," presented at the Proceedings of the 12th international conference on World Wide Web, Budapest, Hungary, 2003.

[38] A. Ntoulas, J. Cho, and C. Olston, "What's new on the web?: the evolution of the web from a search engine perspective," presented at the Proceedings of the 13th international conference on World Wide Web, New York, NY, USA, 2004.

[39] E. Adar, J. Teevan, S. T. Dumais, and J. L. Elsas, "The web changes everything: understanding the dynamics of web content," presented at the Proceedings of the Second ACM International Conference on Web Search and Data Mining, Barcelona, Spain, 2009.