Caching Search Engine Results over Incremental Indices

Roi Blanco
Yahoo! Research
Barcelona, Spain
roi@yahoo-inc.com

Edward Bortnikov
Yahoo! Labs
Haifa, Israel
ebortnik@yahoo-inc.com

Flavio P. Junqueira
Yahoo! Research
Barcelona, Spain
fpj@yahoo-inc.com

Ronny Lempel
Yahoo! Labs
Haifa, Israel
rlempel@yahoo-inc.com

Luca Telloli
Barcelona Supercomputing Center
Barcelona, Spain
telloli.luca@bsc.es

Hugo Zaragoza
Yahoo! Research
Barcelona, Spain
hugoz@yahoo-inc.com

ABSTRACT

A Web search engine must update its index periodically to incorporate changes to the Web. We argue in this paper that index updates fundamentally impact the design of search engine result caches, a performance-critical component of modern search engines. Index updates lead to the problem of *cache invalidation*: invalidating cached entries of queries whose results have changed. Naïve approaches, such as flushing the entire cache upon every index update, lead to poor performance and in fact, render caching futile when the frequency of updates is high. Solving the invalidation problem efficiently corresponds to *predicting* accurately which queries will produce different results if re-evaluated, given the actual changes to the index.

To obtain this property, we propose a framework for developing *invalidation predictors* and define metrics to evaluate invalidation schemes. We describe concrete predictors using this framework and compare them against a baseline that uses a cache invalidation scheme based on time-to-live (TTL). Evaluation over Wikipedia documents using a query log from the Yahoo! search engine shows that selective invalidation of cached search results can lower the number of unnecessary query evaluations by as much as 30% compared to a baseline scheme, while returning results of similar freshness. In general, our predictors enable fewer unnecessary invalidations and fewer stale results compared to a TTL-only scheme for similar freshness of results.

Categories and Subject Descriptors

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

General Terms

Algorithms, Performance, Experimentation

SIGIR'10, July 19–23, 2010, Geneva, Switzerland.

Keywords

Search engine caching, Real-time indexing

1. INTRODUCTION

Search engines are often described in the literature as building indices in batch mode. This means that the phases of crawling, indexing and serving queries occur in generations, with generation $n+1$ being prepared in a staging area while generation n is live. When generation $n+1$ is ready, it replaces generation n. The length of each crawl cycle is measured in weeks, implying that the index may represent data that is several weeks stale [8, 9].

In reality, modern search engines try to keep at least some portions of their index relatively up to date, with latency measured in hours. News search engines, e-commerce sites and enterprise search systems all strive to surface documents in search results within minutes of acquiring those documents (by crawling or ingesting feeds). This is realized by modifying the live index (mostly by append operations) rather than replacing it with the next generation. Such engines are said to have *incremental indices*.

Caching of search results has long been recognized as an important optimization step in search engines. Its setting is as follows. The engine dedicates some fixed-size fast memory cache that can store up to k search result pages. For each query in the stream of user-submitted search queries, the engine first looks it up in the cache, and if results for that query are stored in the cache - a cache hit - it quickly returns the cached results to the user. Upon a cache miss - when the query's results are not cached - the engine evaluates the query and computes its results. The results are returned to the user, and are also forwarded to the cache. When the cache is not full, it caches the newly computed results. Otherwise, the cache's *replacement policy* may decide to evict some currently cached set of results to make room for the newly computed set.

An underlying assumption of caching applications is that the same request, when repeated, will result in the same response that was previously computed. Hence returning the cached entry does not degrade the application. This does not hold in incremental indexing situations, where the searchable corpus is constantly being updated and thus the results of any query can potentially change at any time. In such cases, the engine must decide whether to re-evaluate repeated queries, thereby reducing the effectiveness of caching

their results, or to save computational resources at the risk of returning stale (outdated) cached entries. Existing search applications apply simple solutions to this dilemma, ranging from performing no caching of search results at all to applying time-to-live (TTL) policies on cached entries so as to ensure worst-case bounds on staleness of results.

Contributions. This paper studies the problem of search results caching over incremental indices. Our goal is to selectively invalidate the cached results only of those queries whose results are actually affected by the updates to the underlying index. Cached results of queries that are unaffected by the index changes will continue to be served. We formulate this as a prediction problem, in which a component that is aware of both the new content being indexed and the contents of the cache, invalidates cached entries it estimates that have become stale. We define metrics by which to measure the performance of these predictions, propose a realizing architecture for incorporating such predictors into search engines, and measure the performance of several prediction policies. Our results indicate that selective invalidation of cached search results can lower the number of queries invalidated unnecessarily by roughly 30% compared to a baseline scheme, while returning results of equal freshness.

Roadmap. The remainder of this paper is organized as follows. Section 2 surveys related work on search results caching and incremental indexing. Section 3 defines the reference architecture on which this work is based. Section 4 presents schemes for selectively invalidating cached search results as the search index ingests new content. We also discuss in this section the metrics we use to evaluate cache invalidation schemes. Section 5 describes the experimental setup and reports our results. We conclude in Section 6.

2. RELATED WORK

Caching of search results was noted as an optimization technique of search engines in the late 1990s by Brin and Page [5]. The first to publish an in-depth study of search results caching was Markatos, in 2001 [17]. He applied classical cache replacement policies (e.g. LRU and variants) on a log of queries submitted to the Excite search engine, and compared the resulting hit-ratios, which peaked around 30%. *PDC* (Probability Driven Caching) [14] and *SDC* (Static Dynamic Caching) [10] are caching algorithms specifically tailored to the locality of reference present in search engine query streams, both proposed originally in 2003. PDC divides the cache between an SLRU segment that caches top-n queries, and a priority queue that caches deeper result pages (*e.g.*, results 11-20 of queries). The priority queue estimates the probability of each deep result page to be queried in the near future, and evicts the page least likely to be queried. SDC also divides its cache into two areas, where the first is a read-only (static) cache of results for "head" (perpetually popular) queries, while the second area dynamically caches results for other queries using any replacement policy (e.g. LRU or PDC).

The *AC* scheme was proposed by Baeza-Yates *et al.* in 2007 [3]. It applies a predictor that estimates the "repeatability" of each query. Several predictors and the features they rely on were evaluated, showing that this technique is able to outperform SDC.

Gan and Suel [12] study a weighted version of search results caching that optimizes the work involved in evaluating the cache misses rather than the hit ratios. They argue that different queries incur different computational costs.

Lempel and Moran studied the problem of caching search engine results in the theoretical framework of competitive analysis [15]. For a certain stochastic model of search engine query streams, they showed an online caching algorithm whose expected number of cache misses is no worse than four times that of any online algorithm.

Search results are not the only data cached in search engines. Saraiva *et al.* [19] proposed a two-level caching scheme that combines caching of search results with the caching of frequently accessed postings lists. Long and Suel extend this idea to also caching intersections of postings lists of pairs of terms that are often co-used in queries[16]. Baeza-Yates *et al.* investigate trade-offs between result and posting list caches, and propose a new algorithm for statically caching posting lists that outperform previous ones [2]. It should be noted, however, that in the massively distributed systems that comprise Web search engines, caching of postings lists and caching of search results may not necessarily compete on the RAM resources of the same machine. The work of Skobeltsyn *et al.* describes the *ResIn* architecture, which lines up a cache of results and a pruned index [20]. They show that the cache of results shapes the query traffic in ways that impact the performance of previous techniques for index pruning, so assessing such mechanisms in isolation may lead to poor performance for search engines.

The above works do not address what happens to the cached results when the underlying index, over which queries are evaluated, is updated. To this effect, one should distinguish between incremental indexing techniques, that incorporate updates into the "live" index as it is serving queries, and non-incremental settings. Starting with the latter case, we note that large scale systems may choose to not incrementally update their indices due to the large cost of update operations and the interference of incremental updates with the capability to keep serving queries at high rates [18, 7]. Rather, they manage content updates at a higher level.

Shadowing is a common index replacement scheme [1, 8]: while one immutable index is serving queries, a second index is built in the background from newly crawled content. Once the new index is ready, the engine shifts its service from the older index to the newly built one. In this approach, indexed content is fully updated upon a new index generation, and the results cache is often flushed at that time.

Another approach, that performs updates at a finer level of granularity than shadowing, uses *stop-press* or *delta* indices [7, 11, 21]. Here, the engine maintains a large main index, which is rebuilt at relatively large intervals, along with a smaller delta index which is rebuilt at a higher rate and reflects the new content that arrived since the main index was built. When building the next main index, the existing main index and the latest corresponding delta index are merged. Query evaluation in this approach is a federated task, requiring the merging of the results returned by both indices. The main index can keep its own cache, as its results remain stable over long periods of time.

We note that the vast literature on incremental indexing is beyond the scope of this paper. However, we are not aware of any work that addressed the maintenance of the search results cache in such settings. In incremental settings, systems typically either invalidate results whose age exceeds some threshold, or forego caching altogether.

3. SYSTEM MODEL

At a high level, Web search engines have three major components: a *crawler*, an *indexer*, and a *runtime component* that is dominated by the *query processor* (Figure 1). The crawler continuously updates the engine's document collection by fetching new or modified documents from the Web, and deleting documents that are no longer available. The indexer periodically processes the document collection and generates a new inverted file and auxiliary data structures. Finally, query processors evaluate user queries using the inverted file produced by the indexer [1, 5].

The runtime component of a Web search engine typically also includes a cache of search results, located between the engine's front-end and its query processor, as depicted in Figure 1. The cache provides two desirable benefits: (1) it reduces the average latency perceived by a user, and (2) it reduces the load on back-end query processors. Such a cache may run in the same machines as query processors or in separate machines. To simplify our discussion, we assume that caches of results reside in separate machines, and that most resources of those machines are available to the cache.

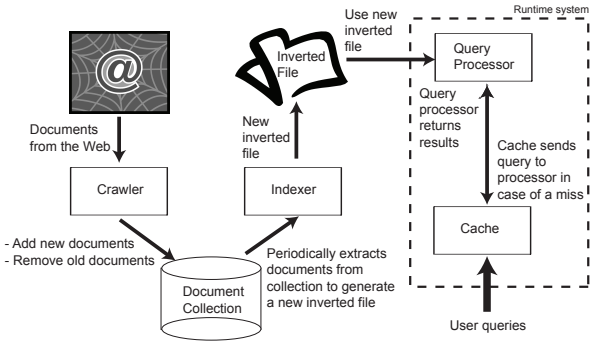

Figure 1: Overview of system model.

However, as the index evolves, the cached results of certain queries no longer reflect the latest content and become stale. By stale queries, we precisely mean queries for which the top-k results change because of an index update. In order to keep serving fresh search results, the engine must invalidate those cached entries. One trivial invalidation mechanism is to have the indexers indicate whenever the inverted index changes, thereby prompting the cache to invalidate all queries. When the index is updated often, the frequent flushing of the cache severely impacts its hit rate, perhaps to the point of rendering caching worthless.

To efficiently invalidate cache entries, we assume that the indexer is able to propagate information to the runtime component upon changes to the index. More concretely, we assume that even though the crawler continuously updates the document corpus, the indexer only generates a new version every Δt time. Upon a new version, we assume that a set of documents D have each been either inserted to or deleted from the index. Note that this simple model subsumes incremental (real-time) indexing, in the sense that the indexer can index every new or removed document by setting Δt to a very small value and having D be a singleton set.

We embody the above idea by introducing a new component to the search engine architecture – the Cache Invalidation Predictor (CIP).

4. CACHE INVALIDATION PREDICTORS

Cache invalidation predictors bridge the indexing and runtime processes of a search engine, which typically do not interact in search engines operating in batch mode, or limit their interaction to synchronization and locking.

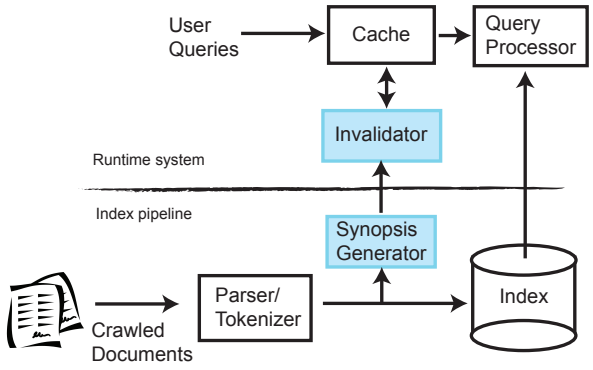

Figure 2: CIP Architecture.

When introducing cache invalidation prediction into a system, the very front end of the runtime system – the cache – needs to become aware of documents coming into the indexing pipeline. We thus envision building a CIP in two major pieces, as depicted in Figure 2:

The synopsis generator: resides in the ingestion pipeline, *e.g.*, right after the *tokenizer*, and is responsible for preparing synopses of the new documents coming in. The synopses may be as robust as the full token stream and other ranking features of each and every incoming document, or as lean as nothing at all (in which case the generator is trivial).

The invalidator: implements an invalidation policy. It receives synopses of documents prepared by the synopsis generator, and through interaction with the runtime system, decides which cached entries to invalidate. The interaction may be complex, such as evaluating each synopsis-query pair, or simplistic (ignoring the synopses altogether).

Section 4.1 describes various pairings of synopsis generators and invalidators, which together constitute a CIP. In each case we note the computational complexities of both components, as well as the communication between them.

4.1 CIP Policies

Our architecture allows composing different synopsis generators with different invalidators, yielding a large variety of behaviors. Below we show how the traditional age-based time-to-live policy (TTL) fits within the framework, and proceed to describe several policies of synopsis generators and invalidators, which we later compose in our experiments.

4.1.1 TTL: Age-based invalidation

Age-based policies consider each cached entry to be valid for a certain amount of time τ after evaluation. Each entry is expired, or invalidated, once its age reaches τ. At the two extremes, $\tau = 1$ implies no caching as results must be recomputed for each and every query. With $\tau = \infty$ no

invalidation ever happens, and results are considered fresh as long as they are in the cache. As the value of τ increases from 1 to ∞, the number of unnecessary invalidations decreases, whereas the number of missed invalidations increases.

TTL-based policies ignore incoming content. In terms of our architecture, the synopsis generator is null in TTL policies, and no communication is required. The invalidator can be realized with a complexity of $\mathcal{O}(1)$ per query.

4.1.2 Synopsis Generation and Invalidation Policies

To improve over TTL, we exploit the fact that the cached results for a given query are its top-k scoring documents. By approximating the score of an incoming document to a query we can try to predict whether it affects its top-k results.

Synopsis generation.

The synopsis generator attempts to send compact representations of a document's score attributes, albeit to unknown queries. Its main output is a vector of the document's top-scoring TF-IDF terms [4] – these are the terms for which the document might score highly for. To control the length of the synopsis, the generator sends a fraction η of each document's top terms in the vector. η can range from zero (empty synopsis) to 1 (all terms, full synopsis). Intuitively, selective (short) synopses will lower the communication complexity of the CIP but will increase its error rate, as less information is available to the invalidator.

Another observation, applicable to document revisions, is that insignificant revisions typically do not affect the rankings achieved by the document. Consequently, cached entries should not be invalidated on account of minor revisions of documents. Hence, we estimate the difference between each document revision and its previously encountered version, and only produce a synopsis if the difference is above a *modification threshold* δ. Concretely, we use the weighted Jaccard similarity [13] as a similarity measure, where the weight of term t in document D is the number of occurrences of t in D. This measure can be efficiently and accurately estimated by using shingles [6]. Increasing δ will result in fewer synopses being produced, thereby lowering the communication complexity of the CIP, at the cost of failing to invalidate cached entries that have become stale.

Invalidation policies.

Once a synopsis is generated, the CIP invalidators make a simplifying assumption that a document (and hence, a synopsis) only affects the results of queries that it matches. While this is true for most synopses and queries, it does not always hold. For example, a document that does not match a query may still change term statistics that affect the scores of documents that do. With this assumption, an invalidator first identifies all queries (and only those) matched by the synopsis. A synopsis *matches* query q if it contains all of q's terms in conjunctive query models, or any term in disjunctive models. Then, the invalidator may invalidate all queries matched by a synopsis (note that match computation can be efficiently implemented with an inverted index over the cached query set). Alternatively, it can apply *score thresholding* – namely, using the same ranking function as the underlying search engine, it computes the score of the synopsis with respect to cached query q, and only invalidates q if the computed score exceeds that of q's last cached result. This score projection procedure, which tries to determine whether a new document is in the top-k results of a cached query, is feasible for many ranking functions, e.g. TF-IDF, probabilistic ranking, etc. However, it is inherently imperfect for an incremental index where cached scores cannot be compared with newly computed ones as the index's term statistics drift. We denote by the indicator variable 1_s whether score thresholding is applied.

η	fraction of top-terms included in synopsis
δ	revision modification threshold for producing a synopsis
1_s	boolean indicating whether score thresholding is applied
τ	time-to-live of a cached entry

Table 1: Summary of parameters.

Similarly to TTL, CIP applies *age-based invalidation* – they invalidate all queries whose age exceeds a certain time-to-live threshold, denoted by τ. This bounds the maximum staleness of the cached results.

Finally, all CIPs invalidate any cached results that include documents that have been deleted. Clearly, all invalidation due to deleted documents are correct.

Table 1 summarizes the parameters of our CIP policies.

4.2 Metrics of Cache Invalidation Predictors

Upon processing a new document set D, a Cache Invalidation Predictor (CIP) makes a decision whether to invalidate or not each cached query. We say CIP is *positive* (p) about query q when CIP estimates that the ingestion of D by the corpus will change q's results, and so q's entry should be invalidated as it is now stale. CIP is *negative* (n) about q when it estimates that q's cached results do not change with the ingestion of document set D.

For each query, we can compare CIP's decision with an oracle that knows exactly if the ingestion of D by the corpus will change q's results or not - as if it had re-run every cached query upon indexing D. This leads to four possible cases (depending on whether CIP or the oracle decide positive or negative for the query). Let us call them $\{pp, pn, np, nn\}$, where the first letter indicates the decision of the CIP and the second the oracle's.

There are two types of errors CIP might make. In a false positive (pn), CIP wrongly invalidates q's results, leading to an unnecessary evaluation of q if it is submitted again. In a false negative (np), CIP wrongly keeps q's results, causing the cache to return stale results whenever q is subsequently submitted until its eventual invalidation. If we have a set of cached queries $\mathcal{Q}$ of size Q, we can compute the total number of queries falling in each one of these categories. Let us call these totals PN and NP respectively.

These two types of errors have very different consequences. The cost of a false positive is essentially computational, whereas false negatives hurt quality of results. Conservative policies, aiming to reduce the probability of users receiving stale results, will focus on lowering false negatives. More aggressive policies will focus on system performance and will tolerate some staleness by lowering false positives. This implies that CIPs should be evaluated along both dimensions - each application will determine the most suitable compromise between false positive and false negatives. We note that modern search engines are conservative, and are willing to devote computational resources to keep their results as fresh as possible ("keeping up with the Web").

False Positive Ratio (FP)	PN/Q
False Negative Ratio (FN)	NP/Q
Stale Traffic Ratio (ST)	$\sum_{q \in S} f_q / F$

Table 2: CIP performance metrics

We use the ratio of false positives and false negatives, denoted FP and FN respectively, as our performance metrics (see Table 2 for definitions). High FP implies many wasteful computation cycles due to unnecessary invalidations. High FN implies many stale results in the cache, leading to potentially many of them being returned to the users.

The metrics above were defined with respect to the contents of the cache given a single document set D. In an incremental setting, a CIP would receive a sequence of document sets, $D_1, D_2, \ldots$. It is important to note that a false positive made by CIP when processing D_t can propagate errors (from the users' standpoint) into the future. Consider a query q, upon which CIP incurs a false negative (np) when processing D_t, thereby leaving q's stale results in the cache. Assume that when processing D_{t+1}, CIP *correctly* labels q as negative (nn) and does not invalidate its results, as the documents in D_{t+1} indeed do not affect q's results. While the predictor made a correct point-in-time decision at time $t+1$, q's cached results remain stale, and any user submitting q until such time when CIP invalidates q will receive stale results. Let S be the set of cached queries whose results are stale. Note that after processing any document set, $|S| \geq NP$ since stale queries may have persisted in the cache from false negatives made on earlier document sets.

False positives and false negatives are asymmetrical also in another aspect: a false positive on query q will incur a single (redundant) re-evaluation of q, so the cost for the engine is irrespective of the query stream. In contrast, the cost of a false negative on q (and any stale query $q \in S$ in general) depends on the frequency of q in the query stream, as the cache returns stale results for each request of q. We therefore define a Stale Traffic ratio metric ST (see Table 2), in which the cost of each stale query $q \in S$ is weighted by its frequency, denoted f_q. The quantity F in the formula of ST is the sum of all query frequencies $F = \sum_{q \in \mathcal{Q}} f_q$.

Note that the metrics above are defined irrespective of the cache replacement policy that may be used. In particular, a CIP false negative on q is harmless if the cache replacement policy evicts q before the next request of q. The interaction between cache invalidation due to the dynamics of the underlying corpus and cache replacement due to the dynamics of the query stream is subject of future work.

5. EXPERIMENTS

This section presents our evaluation framework. We use a large Web corpus and a real query log from the Yahoo! search engine to evaluate our CIP policies. Note that our setup makes several simplifying assumptions to make tractable the problem of simulating a crawler, an indexer, a cache, and a realistic query load interacting in a dynamic fashion.

5.1 Experimental Setup

As a Web-representative dynamic corpus, we use the history log of the (English) Wikipedia[1], the largest time-varying dataset publicly available on the Web. This log contains all revisions of 3,466,475 unique pages between Jan 1, 2006 and Jan 1, 2008. It was constructed from two sources: the latest public dump from the Internet Archive[2], with the information about page creations and updates, and the deletion statistics available from Wikimedia[3].

The initial snapshot on Jan 1, 2006 contained 904,056 individual pages. We processed Wikipedia revisions in single-day batches called *epochs*, each containing the revisions that correspond to one day of Wikipedia history. The average number of revisions per day is 41,851 (i.e., about 4% of the initial corpus), consisting mostly of page modifications (95.22%) and new page creations (4.16%). The (uncompressed) size of the corpus, with all revisions, is 2.8 TB.

We focus on conjunctive queries (the *de facto* standard for Web search) – *i.e.*, documents match a query only when containing all query terms. Our experiments use the open-source Lucene search library as the underlying index and runtime engine[4]. Lucene uses TF-IDF for scoring.

We assess the performance of predictors on a fixed representative set of queries $\mathcal{Q}$, which represents a fixed set of cached queries. The synopsis generator consumes each epoch in turns, sends synopses of its documents to the invalidator, and the invalidator makes a decision on each query $q \in \mathcal{Q}$. We compute the "ground truth" oracle by indexing the epoch in Lucene and running all queries, retrieving the top-10 documents per query. The ground truth oracle is conservative and declares a query as invalid upon *any change* to the ranking of its top-10 results. We record the performance of each CIP relative to the ground truth, and track its set of stale queries. The performance numbers reported in the next section are all averaged, per CIP policy, over a history of 120 consecutive epochs (days) of Wikipedia revisions.

To generate the set of cached queries $\mathcal{Q}$, we performed a uniform sample, with repetitions, of 10,000 queries from the Yahoo! Web search log, sampled from a query log recorded on May 4 and May 5, 2008, which resulted in a user clicking on a page from the `en.wikipedia.org` domain. $\mathcal{Q}$ consists of the 9,234 unique queries in the sample. The multiset of queries was used to derive the frequency f_q of each $q \in \mathcal{Q}$, for computing the stale traffic ratio (ST).

Our choice of working with a fixed query set stems from our desire to isolate the performance of the CIP policies from the effects of a dynamic cache and its parameters (e.g., cache size and replacement policies). The dynamic study, which is plausible and interesting, is left for future research.

5.2 Numerical Results

We start by analyzing the results obtained for three standard policies: no caching, no invalidation (static cache), and TTL caching (invalidating all queries after a fixed period of time). Table 3 reports their performance. Not invalidating entries causes the cache to return stale results. Not caching guarantees that no results are stale, but it also forces the engine to process queries unnecessarily as previous work on caching has shown. Using a TTL value improves the overall situation, since it reduces the amount of stale traffic compared to not invalidating entries, but it still generates a significant number of false positives and negatives. Finally, a basic CIP policy with the following parameters is able to re-

[1] `http://www.wikipedia.org/`

[2] `http://www.archive.org/details/enwiki-20080103`

[3] `http://stats.wikimedia.org`

[4] `http://lucene.apache.org/`

Policy	FP	FN	ST
No Invalidation	0.000	0.108	0.768
No Cache	0.892	0.000	0.000
TTL $\tau = 2$	0.446	0.054	0.055
TTL $\tau = 5$	0.179	0.086	0.175
Basic CIP	0.679	0.001	0.008

Table 3: Baseline CIP comparison.

duce the amount of stale traffic significantly, with very few false negatives - similarly to the "no cache" case - at the cost of many false positives:

Basic CIP: $\tau = \infty, \delta = 0, \eta = 1,$ and $1_s = false$

In words, our Basic CIP does not expire queries ($\tau = \infty$), does not exclude documents based on similarity ($\delta = 0$), does not exclude terms ($\eta = 1$), and does not use score thresholding. The synopsis generator of the Basic CIP essentially sends each document in its entirety to the predictor, which then invalidates each query whose terms appear in conjunction in any synopsis.

Ruling out a cache is ideal with respect to freshness of results, but it is undesirable from a performance perspective. The Basic CIP is able to achieve a similar degree of freshness, while benefiting from cache hits. We next assess how changing the CIP parameters affects both freshness and performance.

Dynamics of stale traffic: Over time, errors due to false negatives accumulate, and imply an increasingly high stale traffic ratio (ST). The impact is most severe for frequent queries. A false negative can be fixed by either (1) a CIP positive, either true or false; or (2) an age threshold expiration. CIP positives depend on the arrival rate of matching documents: if a match never happens after a false negative, then the latter will persist forever. Consequently, it is critical to augment the CIP with a finite age threshold τ, not only to bound the maximum result set age, but also to guarantee that ST converges.

Figure 3 shows how stale traffic evolves over time with three CIP instances. The CIP instances in the figure use a synopsis of the top 20% terms ($\eta = 0.2$), employ score thresholding ($1_s = true$), and have different τ values. For $\tau = \infty$, ST grows, albeit in a declining pace, and eventually exceeds 30% without stabilizing. For $\tau = 5$ and $\tau = 10$, ST stabilizes within a few epochs after the first expiration. Infinite τ is practical only when the predictor's FN ratio is negligible, *e.g.*, with the Basic CIP.

Varying η and τ: Figure 4 depicts the behavior of CIP for different values of synopsis size η and time-to-live τ, also employing score thresholding ($1_s = true$). In this experiment, we create synopses for all document revisions ($\delta = 0$). In addition to plotting the TTL baseline, we show 5 CIP plots, each having a fixed value of τ. The rightmost CIP plot (circle marks) does not apply score thresholding (1_s=false) while the other 4 plots do. The six points in each CIP plot correspond to increments of 0.1 in η, from $\eta = 0.5$ at the top point of each plot to $\eta = 1.0$ at the bottom. The Basic CIP is the bottom point in the rightmost CIP plot.

Score thresholding reduces false positives but increases the false negatives ratio (FN). The τ parameter only affects the positive predictions, hence it has no impact on FN. However, lowering τ reduces stale traffic, as frequent age-based invalidation rectifies false negatives from previous epochs and limits their adverse effect on stale traffic. For example, although the Basic CIP ($\tau = \infty, 1_s = false$) achieves the smallest possible FN (0.08%), there are instances (e.g., $\tau = 2, 1_s = true$) which improve upon it by reducing *both* stale traffic and false positives (0.35% vs 0.89%, and 59.1% vs 67.8%, respectively). In such configurations, false negatives are fixed quickly, causing little cumulative effect.

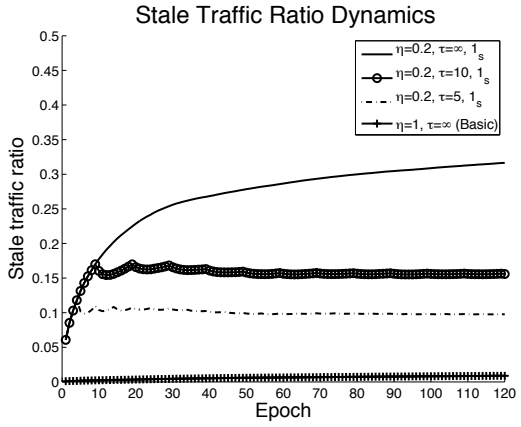

Figure 3: Convergence of stale traffic metric for CIP instantiations. For finite age threshold τ, stale traffic stabilizes shortly after τ. For infinite τ, stale traffic grows throughout the evaluation.

$\delta = 0$	$\delta = 0.005$	$\delta = 0.01$	$\delta = 0.05$	$\delta = 0.1$
100%	69.03%	57.25%	29.25%	20.38%

Table 4: Percentage of transmitted synopses as the modification threshold δ increases.

Finally, shorter synopses (smaller η values) reduce false positives and communication, at the expense of more false negatives, and consequently, higher stale traffic.

Varying τ and δ: Figure 5 evaluates the effect of varying the modification threshold δ. These experiments use complete synopses ($\eta = 1$) and score thresholding ($1_s = true$). Each plot fixes a value of τ, and varies δ.

Increasing the value of δ yields a reduction of FP's at the cost of higher FN's and ST. Additionally, eliminating synopses due to minor revisions reduces the communication overhead between the synopsis generator and the invalidator. This is particularly useful when the two CIP components reside on separate nodes. Table 4 shows how the percentage of generated (and transmitted) synopses drops as the value of δ increases. Note that we compute the communication overhead here by counting the number of synopses.

Best cases: Here we contrast the best individual instances of CIP classes studied in the previous sections against the baseline TTL heuristic. Figure 6 depicts the policy instances that formed the bottom-left envelope of Figure 4 and Figure 5. Our results show that for every point of TTL, there is at least one point of CIP that obtains a significantly lower stale traffic for the same value of false positives. For example, tolerating 6% of stale traffic requires below 20% of false

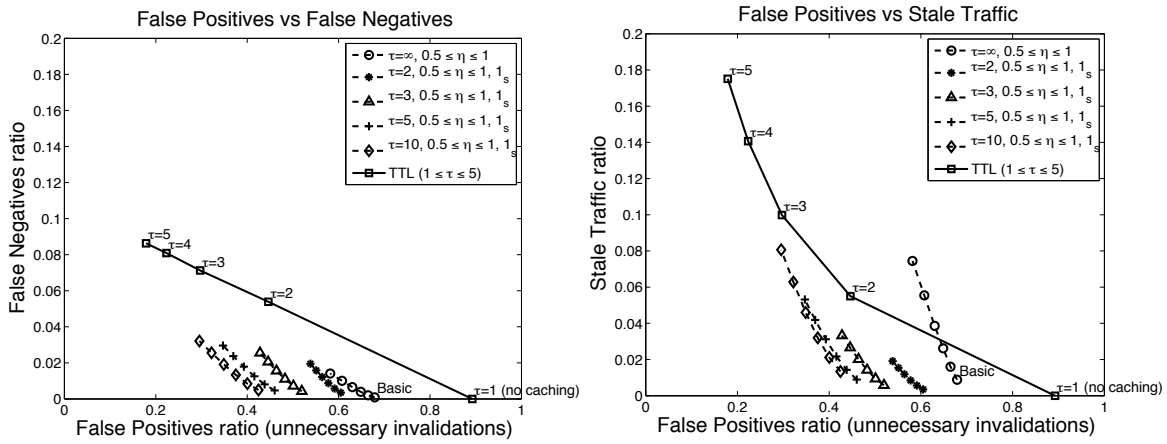

Figure 4: False Negatives (FN, left) and Stale traffic (ST, right) vs. False Positives (FP) curves, for varying 1_s $(false/true)$, τ $(2, 3, 5, 10)$ and η $(50\%, 60\%, 70\%, 80\%, 90\%, 100\%)$. The Basic CIP achieves the optimal FN but a suboptimal ST, due to $\tau = \infty$. Score thresholding (1_s), longer timeouts (τ), and smaller synopses (η) lead to more aggressive policies.

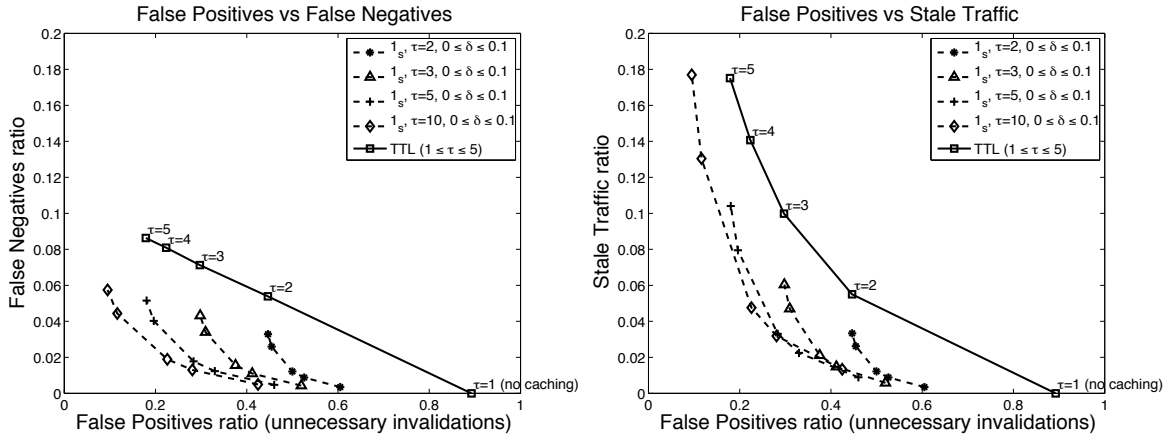

Figure 5: False Negatives (FN, left) and Stale traffic (ST, right) vs. False Positives (FP) curves, for varying τ $(2, 3, 5, 10)$ and δ $(0\%, 0.5\%, 1\%, 5\%, 10\%)$. Higher modification thresholds (increasing δ, from bottom to top of each plot) lead to more aggressive policies.

positives, in contrast with TTL's 44.6%. When high precision is required (low ST), CIP performs particularly well – the number of query evaluations is 30% below the baseline.

6. CONCLUSIONS

Cache invalidation is critical for caching query results over incremental indices. Traditional approaches apply very simple invalidation policies such as flushing the cache upon updates, which induces a significant penalty to cache performance. We presented a cache invalidation predictor (CIP) framework, which invalidates cached queries selectively by using information about incoming documents. Our evaluation results using Wikipedia documents and queries from a real search engine shows that our policies enable a significant reduction to the amount of redundant invalidations (false positives, or FP) required to sustain the desired precision (stale traffic, or ST). More concretely, for every target ST, the reduction of FP compared to the baseline TTL scheme is between 25% and 30%.

The implication of our results to the design of caching systems is the following. False positives impact negatively the cache hit rate as they lead to unnecessary misses in our setting. Consequently, selecting a policy that enables a low ratio of false positives is important for performance. With our CIP policies, it is possible to select a desired ratio of false positives as low as 0.2. Lowering the ratio of false positives, however, causes the ratio of false negatives (and stale traffic) to increase, which is undesirable when the degree of freshness expected for results is high. When designing a caching system, a system architect must confront such a trade-off and choose parameters according to the specific requirements of precision and performance. Our CIP policies enable such choices and improve over previous solutions.

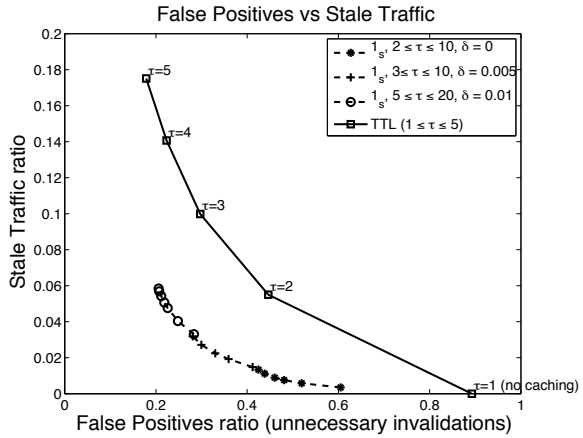

Figure 6: Stale traffic (ST) vs False Positives (FP) for the best cases. We use: $\eta = 1$ − complete synopses, $1_s = true$ − score thresholding, $\delta = (0\%, 0.5\%, 1\%)$ − small modification threshold, and $2 \leq \tau \leq 20$ − a variety of age thresholds.

Acknowledgements

This work has been partially supported by the COAST (ICT-248036) and Living Knowledge (ICT-231126) projects, funded by the European Community.

7. REFERENCES

[1] A. Arasu, J. Cho, H. Garcia-Molina, A. Paepcke, and S. Raghavan. Searching the Web. *ACM Transactions on Internet Technology*, 1(1):2–43, 2001.

[2] Ricardo Baeza-Yates, Aristides Gionis, Flavio P. Junqueira, Vanessa Murdock, Vassilis Plachouras, and Fabrizio Silvestri. Design trade-offs for search engine caching. *ACM Transactions on the Web*, 2(4):1–28, 2008.

[3] Ricardo Baeza-Yates, Flavio Junqueira, Vassilis Plachouras, and Hans F. Witschel. Admission Policies for Caches of Search Engine Results. In *SPIRE*, 2007.

[4] Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison Wesley, New York, NY, 1999.

[5] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. In *WWW'98: Proceedings of the 7th International Conference on the World Wide Web*, pages 107–117, 1998.

[6] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the Web. *Computer Networks and ISDN Systems*, 29(8-13):1157–1166, 1997.

[7] Soumen Chakrabarti. *Mining the Web - Discovering Knowledge from Hypertext Data*. Morgan Kaufmann Publishers, San Francisco, CA, 2003.

[8] Junghoo Cho and Hector García-Molina. The evolution of the Web and implications for an incremental crawler. In *Proc. 26th International Conference on Very Large Data Bases (VLDB2000)*, pages 200–209, 2000.

[9] Anirban Dasgupta, Arpita Ghosh, Ravi Kumar, Christopher Olston, Sandeep Pandey, and Andrew Tomkins. The discoverability of the Web. In *WWW '07: Proceedings of the 16th International Conference on the World Wide Web*, pages 421–430. ACM, 2007.

[10] Tiziano Fagni, Raffaele Perego, Fabrizio Silvestri, and Salvatore Orlando. Boosting the Performance of Web Search Engines: Caching and Prefetching Query Results by Exploiting Historical Usage Data. *ACM Transactions on Information Systems*, 24(1):51–78, 2006.

[11] Marcus Fontoura, Jason Zien, Eugene Shekita, Sridhar Rajagopalan, and Andreas Neumann. High performance index build algorithms for intranet search engines. In *Proc. 30th International Conference on Very Large Data Bases (VLDB 2004)*, pages 1158–1169. Morgan Kaufmann, August 2004.

[12] Qingqing Gan and Torsten Suel. Improved techniques for result caching in Web search engines. In *WWW'09: Proceedings of the 18th International Conference on the World Wide Web*, pages 431–440, April 2009.

[13] Paul Jaccard. Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bulletin de la Société Vaudoise des Sciences Naturelles*, 37:547–579, 1901.

[14] Ronny Lempel and Shlomo Moran. Predictive Caching and Prefetching of Query Results in Search Engines. In *WWW'03: Proceedings of the 12th International Conference on the World Wide Web*, pages 19–28. ACM Press, 2003.

[15] Ronny Lempel and Shlomo Moran. Competitive caching of query results in search engines. *Theoretical Computer Science*, 324(2):253–271, September 2004.

[16] Xiaohui Long and Torsten Suel. Three-level caching for efficient query processing in large Web search engines. In *WWW'05: Proceedings of the 14th International Conference on the World Wide Web*, pages 257–266, May 2005.

[17] Evangelos P. Markatos. On Caching Search Engine Query Results. *Computer Communications*, 24(2):137–143, 2001.

[18] Sergey Melnik, Sriram Raghavan, Beverly Yang, and Hector Garcia-Molina. Building a distributed full-text index for the Web. In *WWW'01: Proceedings of the 10th International Conference on the World Wide Web*, pages 396–406, May 2001.

[19] P. Saraiva, E. Moura, N. Ziviani, W. Meira, R. Fonseca, and B. Ribeiro-Neto. Rank-preserving two-level caching for scalable search engines. In *Proc. 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 51–58, 2001.

[20] Gleb Skobeltsyn, Flavio Junqueira, Vassilis Plachouras, and Ricardo Baeza-Yates. ResIn: a combination of results caching and index pruning for high-performance Web search engines. In *Proceedings of the 31st ACM SIGIR conference*, pages 131–138, 2008.

[21] Ian Witten, Alistair Moffat, and Timoty Bell. *Managing Gigabytes*. Morgan Kaufmann Publishers, Inc., San Francisco, CA, second edition, 1999.