# Topical Result Caching in Web Search Engines

Ida Mele[a], Nicola Tonellotto[b], Ophir Frieder[c], Raffaele Perego[a]

[a] *ISTI-CNR, Pisa, Italy*
[b] *University of Pisa, Pisa, Italy*
[c] *Georgetown University, Washington, DC, USA*

**Abstract**

Caching search results is employed in information retrieval systems to expedite query processing and reduce back-end server workload. Motivated by the observation that queries belonging to different topics have different temporal-locality patterns, we investigate a novel caching model called STD (Static-Topic-Dynamic cache), a refinement of the traditional SDC (Static-Dynamic Cache) that stores in a static cache the results of popular queries and manages the dynamic cache with a replacement policy for intercepting the temporal variations in the query stream.

Our proposed caching scheme includes another layer for topic-based caching, where the entries are allocated to different topics (e.g., weather, education). The results of queries characterized by a topic are kept in the fraction of the cache dedicated to it. This permits to adapt the cache-space utilization to the temporal locality of the various topics and reduces cache misses due to those queries that are neither sufficiently popular to be in the static portion nor requested within short-time intervals to be in the dynamic portion.

We simulate different configurations for STD using two real-world query streams. Experiments demonstrate that our approach outperforms SDC with an increase up to 3% in terms of hit rates, and up to 36% of gap reduction w.r.t. SDC from the theoretical optimal caching algorithm.

*Keywords:* efficiency; caching; topic modeling

## 1. Introduction

Caching is a fundamental architectural optimization strategy [16], and query-result caching is critical for Web search efficiency. Query-result caching, as its name implies, stores the results of some selected user queries in a fast-access memory (cache) for future reuse. When a query is requested and we have a hit in the cache, the cached results are directly returned to the user without reprocessing the request. Result caching improves the main efficiency-performance metrics of search engines, namely, latency and throughput. Indeed, serving the cached results decreases the latency perceived by the user issuing the query as well as avoids the usage of computational resources with a consequent improvement of the search engine throughput.

Another advantage of caching query results is the reduction of energy consumption as cached queries do not need to be reprocessed by the back-end servers. Although other energy-efficiency optimization schemes exist [17, 18, 34, 50], these approaches are complementary to result caching and not a contradictory alternative. Given desired and imposed "Green Policy" restrictions and the significant economic benefits due to the energy conservation, the interest of the search industry in energy saving is high. Energy wise, the cost of a cached query is typically assumed to be nil, while a search of a query costs proportionally to its processing time in combination with the electricity price at the time of processing [46].

*Email addresses:* ida.mele@isti.cnr.it (Ida Mele), nicola.tonellotto@unipi.it (Nicola Tonellotto), ophir@ir.cs.georgetown.edu (Ophir Frieder), raffaele.perego@isti.cnr.it (Raffaele Perego)

The main challenge in query-result caching is the identification of those queries whose results should be cached. However, via query log mining, researchers observed high temporal locality in the query stream, enabling accurate search-engine side caching of *popular* query results, i.e., results of queries frequently requested by different users [45, 53].

The query-result cache can be *static* or *dynamic*. A static cache is periodically populated in an offline manner, with the results of past, most-popular queries. Query popularity is estimated observing previously submitted queries (e.g., previous day or week) in Web search logs; a simplifying, but not always correct, core assumption is that queries popular in the past remain popular in the future. A dynamic cache, as the name suggests, is dynamically updated; when the cache is full, an eviction/replacement policy is applied to decide which cache element must be removed to make space for the new one. The most common replacement policy for dynamic caches is the Least Recently Used (LRU) strategy: every time a query is submitted, the cache is updated, keeping track of what query was used and when; if necessary, the cache entry used least recently is evicted to vacate space for the new entry. The LRU strategy is effective without global knowledge and captures the "bursty" behavior of the queries by keeping recent queries in the cache and replacing those queries that are not requested for a long period of time.

Static and dynamic caches can be combined together. Fagni et al. [22] proposed a Static-Dynamic Cache (SDC) where the cache space is divided into two portions. The static portion stores results of the most popular queries, such as "microsoft," "youtube," or "facebook." The dynamic portion maintains currency by applying the LRU strategy. This hybrid approach has proved successful in improving the performance of query-result caching with respect to both static and dynamic caching solutions in isolation. Despite its good performance, SDC suffers from some issues. Static caching captures highly frequent queries, while LRU caching captures bursts of recently submitted queries. That is, static caching captures past queries that are popular over a relatively large time span (e.g., days or weeks) while LRU caching might fail to capture such long-term temporal locality, but does capture short-term popularity. However, a query might not be sufficiently globally popular to be cached in the static cache and not be requested so frequently to be kept in the dynamic cache, but it might become relatively popular in a specific time interval. For example, a query on a specific topic, such as weather forecast, is typically submitted in the early morning hours or at the end of a work day, but relatively seldom in the remaining hours of a day.

We design a cache for query results that can adapt the cache-space utilization to the popularity of the various topics represented in the query stream. The intuition behind our approach is that queries can be grouped based on broad topics (e.g., the queries "forecast" and "storm" belong to the topic `weather`, while queries "faculty" and "graduate" to the topic `education`), and queries belonging to different topics might have different temporal-locality patterns. We assume that the *topic popularity* is represented by the number of distinct queries belonging to the topic; to capture the specific locality patterns of each topic we split the cache entries among the different topics proportional to their popularity. This provides queries belonging to frequently requested topics greater retention probability in the LRU cache.

As an illustrative example, consider a cache with size 2 and the query stream `abcadeafg`, where query `a` is about a specific topic. A classical LRU strategy will get a 0% hit rate (all queries will cause a miss). Instead, using 1 entry for the topic cache and 1 entry for the LRU cache we will get a 22.2% hit rate (the first occurrence of `a` causes a miss, the other two occurrences will cause two hits in the topic cache).

The topic cache can be combined with a static cache in different configurations. We propose to improve the SDC approach by adding yet an additional cache space partition that stores results of queries based on their topics [23]. We call our approach Static-Topic-Dynamic cache (STD) and in Fig. 1 is shown an example of it. To detect the query topics and incorporate them in the caching strategy we rely on the standard topic modeling approach called Latent Dirichlet Allocation (LDA) [12]. LDA gets as input a document collection and returns lists of keywords representing the topics discussed in the collection. Each document in our setting consists of the query keywords and the textual content of their clicked results. Given the topics, the queries can be classified into topical categories, and we estimate the topic popularity by observing the number of distinct queries belonging to that topic.

For our experiments, we use real-world query logs from AOL and Microsoft search engines, and we observe that modifying the caching strategy to include the topic partition increases the cache hit rate by greater than 3 percent with a consequent improvement of the query processing performance. This increase

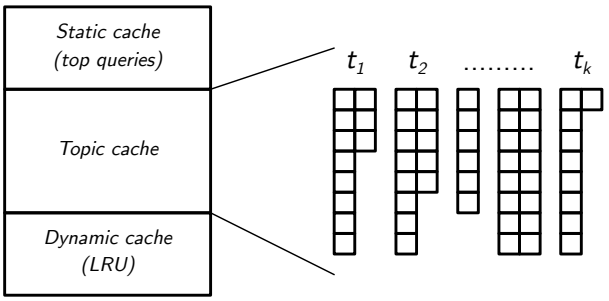



Figure 1: Example of STD cache.

greatly reduces the gap w.r.t. Bélády's optimal caching policy [11], measured as the difference between the hit rate of Bélády's policy and the hit rate of SDC/STD policies. While the SDC gap is ∼10%, the STD gap is ∼6%, with a relative improvement of more than 35% when no cache admission policy is applied. This result is significant for query-result caching, considering that SDC is the state-of-the-art and other attempts to enhance it resulted in small improvements [30].

Compared to a traditional SDC cache, we observe that our caching technique captures moderately popular queries not captured by the static cache; additionally we capture requests repeated within a large time interval that are likewise not captured by the dynamic cache. Some examples of queries that cause a miss in SDC but not in STD are *first bank, texas state bank, first national, renasant bank* for the topic `banking`, *education world, everyday math, french-english dictionary* for the topic `education` and *florida department of health, arthritis foundation, st vincents hospital, st marys hospital* for the topic `health`, mentioning just a few.

The remainder of this paper is structured as follows: Section 2 provides background information and discusses the related work in Web search engine caching, while Section 3 introduces our STD cache model and different configurations of the topic-based cache, followed by how the topics are distilled. Then, Section 4 describes and analyzes the query logs and document collections used for the assessment of the performance of our STD cache(s). Section 5 details the experimental settings and reports on the results of our comprehensive evaluation. Finally, we conclude our investigation in Section 6.

## 2. Related Work

As background, we overview related efforts investigating query log mining, caching solutions in Web search engines, and query topic distillation.

*Query Log Mining.* Information extracted from query logs (e.g., search keywords, users' IDs, query timestamps, and clicked results) allows us to understand the behavior of users interacting with the search engine and, consequently, to enhance the effectiveness and efficiency of the retrieval systems. This desire for user behavioral understanding motivated researchers to investigate the mining of Web search logs in many different directions [48].

Raghavan and Sever [43] performed a first study on how to use past knowledge to improve the effectiveness of information retrieval. They proposed to improve the result retrieval for queries similar to a *query base* which is obtained from past optimal queries mined from query logs. Another approach exploits information extracted from query logs (e.g., query frequencies and distribution) for enhancing the performance of search engines, such as designing new caching strategies [22, 31] and assessing the performance of caching techniques [37]. Other researchers analyzed query logs observing high correlations among query terms [47] and common searching behavior among Web users (e.g., short queries) [27, 49]. A further analysis focusing on the temporal locality of queries was presented by Beitzel et al. [9]. The authors analyzed one week of data, grouping queries per hours and per topic to see how their popularity changes during the day. They observed that the query repetition rate is constant during the day, even though each popular query does

not appear frequently in every hour. This result confirms the intuition that queries grouped by topic show some specific temporal localities in the stream.

*Caching in Web Search Engines.* Modern search engines are large-scale distributed systems where the inverted index is partitioned and stored among multiple back-end machines, each one running a search node. In addition, there is a front-end machine hosting the *broker* for scheduling the queries among the different search nodes [8]. When the query is submitted, the front-end machine sends its keywords to the different back-end machines, each one responsible of searching a given portion of the index. Then, it collects the results to create the Search Engine Result Page (SERP).

In modern search engines, query processing represents one of the major performance bottlenecks, so caching can help to speed up the search engine performance as well as to reduce the latency perceived by the users. Caching can be applied at different granularity including query results [31], posting lists of query terms [45], and posting list intersections [35]. Saraiva et al. [45] proposed a two-level architecture where the front-end machine caches the results of popular queries, while the back-end machines have a cache for the posting lists of most frequently requested terms. This architecture benefits from both types of caching. Indeed, caching query results is faster since future requests of a query whose results are cached can be served immediately, without further processing, but it has a lower hit rate since a hit in cache occurs only with exact-matching queries. On the other hand, caching posting lists entails a better hit rate since the term overlap is greater, but the query needs to be processed by the back-end servers and the only saving is in the number of I/O operations over the disk storing the inverted index. Two-level caching was also studied by Baeza-Yates et al. [6] in the static setting and by Altingovde et al. [2] for dynamic setting.

Three-level architectures store in the additional level of the cache the precomputed intersections of posting lists [35, 51, 54]; this improves the processing of more complex queries such as AND or phrase queries. Ozcan et al. [40] presented an even more sophisticated cache consisting of five levels for storing query results, precomputed scores, posting lists, precomputed intersections of posting lists, and documents. The caching approach is based on a greedy heuristic which selects the best item to cache. Each level of the cache has a priority queue, and the items are ordered in the queue according to their gains reflecting the frequency of the item's past accesses as well as its processing cost and storage overhead. We focus on a cache employed at the level of the front-end machine to store the results of most frequently requested queries. Our cache for query results can be used in combination with these multi-level cache architectures, as well. Improving the hit rate of the query-result cache is in fact beneficial for the lower layers of the caching system that provide an efficient processing of queries resulting in misses in the top-level cache.

The query-result cache enhances the responsiveness of the search engine and saves resources. Depending on the space available, the cache can keep the entire HTML page (*SERP cache*) [22, 31, 37] or the URLs and/or snippets used for reconstructing it (*DocID cache*) [45]. Caching the whole page of results requires more space, and the same page cannot be used for different but similar queries; however, it is faster because when the query is requested, its cached SERP can be returned promptly. On the other hand, the *DocID cache* uses the cache space in a more efficient way, but it is slower because even when we have a cache hit, additional time is needed to reconstruct the SERP. In [2], the authors proposed two levels of caching: first level for the SERPs and the second one for result documents of SERPs evicted from the first level cache. This allows to reconstruct the pages missing from the first level cache on the front-end server without processing the query on the back-end machines.

The query-result cache can be managed in a *static* or *dynamic* way. A static cache is periodically (every day or week) populated with the results of frequent queries. Indeed, several authors observed that query frequency follows an inverse power law, which means that most of the queries are submitted a few times (e.g., they are long, with typos, or very rare), but a small portion of queries are requested several times and shared by different users [22, 31, 53]. This justifies the use of a server-side static cache storing results of these queries. The cache is updated with the results of most popular queries observed in the previous period and for refreshing the results. A dynamic cache changes the content of entries according to the query stream, and it applies a replacement policy for deciding which element must be evicted when the cache is full. This policy tries to minimize the number of misses, and a popular one is Least Recently Used (LRU) which evicts the item not requested for the longer period of time. Markatos [37] compared the hit rates

achieved by LRU and its variations (e.g., Segmented LRU and LFU) over a query stream from the Excite query log. He observed that the LRU policy (and its variations) is very effective in query-result caching as it captures the temporal changes in the stream of requests. Although his work did not propose any novel caching algorithm, it showed the potentiality of caching in Web search. Static and dynamic caches can be combined to improve the hit rate. Fagni et al. in [22] presented Static-Dynamic Cache (SDC), consisting of a static portion for storing the results of most frequent queries and a dynamic portion for which an LRU-like approach is applied. Combined or not with prefetching strategies, SDC outperforms purely static or dynamic caching policies. We present a further improvement of SDC by adding another layer which is made of several topic-dependent LRU caches.

To improve cache performance, the system can apply an admission policy which accepts to the cache only those queries with a high re-submission probability. The idea behind these policies is that caching search results of queries that are not requested anymore or are requested after a long time (and generally after their eviction from the cache) should be avoided. Indeed, these queries (also called "polluting" queries [5]) only waste space in the cache; thus, ignoring them increases the likelihood of retaining results of other queries in the dynamic portion of the cache (e.g., the LRU queue). These policies mostly rely on features extracted from the queries or from usage information [5, 24, 30]. Baeza-Yates et al. proposed an admission policy for restricting infrequent and too long queries from entering the cache [5]. In [30], Kucukyilmaz et al. proposed a machine learning approach for predicting the next queries to cache. Their approach is based on features extracted from the query, index, term frequency, query frequency, and user session. This policy on top of SDC yields to a small improvement of the hit rate compared to the one achieved with our novel caching model. Recent works employed machine learning to predict queries not worthy to be cached [41] or to adjust the time-to-live values of queries [1]. Alternatively, the policy can favor queries that are more costly to process [3]. Other works focus on predicting stale queries [28] and use the information extracted from both queries and session features to increase the hit rate of the cache. We do not investigate novel admission policies; rather we use them in coordination with our caching model as they allow to enhance the input stream of queries. We followed [5] and used the *stateful* and *stateless* features of queries for implementing the admission policy. The former is given by statistical information computed over the past stream (e.g., the number of times the query was submitted in the past). The latter is based on the query characteristics (e.g., the length of the query). The idea is to avoid the caching of infrequently requested queries; long queries are typically rare and are unlikely to be resubmitted. Moreover, we computed an upper bound of the best hit ratio that can be achieved using an admission policy with a cache of a given size. For this reason we used the Bélády's optimal caching policy [11] that assumes to know the future and evicts the results of the query not requested for the longest time (representing the best replacement policy) and an oracle which knows the future and does not admit in the cache the singleton queries (representing the best admission policy).

Orthogonal research works focus on caching single results (e.g., URLs and snippets) that are used for reconstructing the final page of results of queries (SERP). Cambazoglu et al. [15] present techniques for computing result pages of unseen queries starting from the cached results of previously requested query. Ceccarelli et al. investigate the problem of caching query-based snippets [19]. In [4], the *stochastic query covering* approach is presented for selecting the set of documents to store in the cache. These documents serve many queries and can be used for fast retrieval as well as for approximating the results of a query when the connection between the front-end server with the back-end machines is too slow or unavailable. We do not aim at improving document caching as our strategy for query-result caching is orthogonal to it and can be employed together with a document cache in a two-level fashion architecture as in [2]. So, the first level caches the SERPs of queries, while the second level caches the documents used for reconstructing the SERPs. When one of the SERP is evicted from the first level cache, the system may decide to cache its documents in the second level cache, so that queries evicted from the first level cache have a second chance to be served by cached documents stored in the second level cache.

Another line of research focuses on index tiering [29] as well as on document replication [21]; these techniques can be employed in synergy with server-side caching with the purpose of improving the performance of distributed search engines.

Personalization is often employed by modern search engines to improve the effectiveness of query results, presenting result rankings with a bias depending on user interests [48]. Usually personalization is

implemented at server or client side by re-ranking search results according to a specific user profile built automatically [33]. At server side, unbiased result caching is still useful, since the personalization re-ranking can be performed on a list of unbiased results previously cached. Alternatively, personalization is implemented at the client side, by re-ranking the unbiased results returned by the Web search engine [13]. In doing so, no user information is stored at the server side and the users' privacy is not at stake. Client-side personalization is orthogonal to query result caching at the server side since any performance boosts in terms of latency and throughput in the client-server interaction is beneficial in the same way.
Another scenario in which query result caching can be critical and, in some cases, difficult to exploit is that of search systems where the underlying document collection changes very frequently, such as in e-commerce sites where new pages are continuously added or stale pages removed. In this case, it is likely that the cache contains references to deleted documents, and effective techniques to identify and invalidate query results that have become stale in the cache must be put in place [7].

*Query-Topic Distillation.* Two challenging problems in query log mining are query classification and query-topic distillation. Flat and hierarchical taxonomies were proposed for classifying the user queries, even though these taxonomies are limited to some specific domains and provide a broad categorization [14, 49]. Query-topic distillation/detection can improve quality of the Web search [10]. It allows to distill the multiple possible topics behind an ambiguous/broad query which is beneficial for better understanding the query intention and improving the automatic reformulation of queries [26]. Nevertheless, query-topic detection is not easy due to the shortness and lack of context of queries. One possible approach for understanding the topic of a query is enriching its search keywords with the content (from snippet or page) of its top results [26]. Another solution consists of using the content of only the clicked results. As explained in Sec. 3, we opted for this second solution since the user click is a strong indicator of the relevance of the document to the query. Once the search queries are enriched with page content, the resulting document collection can be used to discover the topics. Text clustering techniques or topic modeling approaches can be applied. Topic models are commonly used as they discover the topics discussed in large document collections [12, 32, 44]. We opt to use a topic-model approach (e.g., LDA [12]) as it is completely unsupervised and domain independent; additionally, it leverages the word co-occurrences, providing better results compared to a basic text clustering approach [36].

## 3. Methodology

We propose a new query-result caching strategy based on user search topics. We first describe our caching architecture, including possible implementation configurations, and continue by discussing our query topic extraction approach. We assume that the cache stores the SERPs of the queries; onward we use *query results* or just *results* to refer to the content of a cache entry. Note that regardless of query length or complexity, caching the SERPs (e.g., top-10 results) require the similar storage as a SERP corresponding to one HTML page of results.

### *3.1. Topic-based Caching*

Given the total number $N$ of cache entries available for storing the results of past queries, we propose a Static-Topic-Dynamic (STD) cache that includes the following components:

- a *static* cache $\mathcal{S}$ of size $|\mathcal{S}| = f_s \cdot N$ entries, used for caching the results of the most frequently requested queries. The static cache $\mathcal{S}$ is updated periodically with the fresh results of the top frequent $|\mathcal{S}|$ queries submitted in the previous time frame (e.g., the previous week or month). This static cache is expected to serve very popular queries such as navigational ones (e.g., "google" and "facebook");
- a *topic* cache $\mathcal{T}$ of size $|\mathcal{T}| = f_t \cdot N$ entries, which is in turn partitioned in $k$ topic-based sections $\mathcal{T}.\tau$, with $\tau \in \{t_1, t_2, \ldots, t_k\}$, where $k$ is the number of distinct topics. Each section $\mathcal{T}.\tau$ is considered as an independent cache, managed with some caching policy (e.g., LRU or SDC), and aimed at capturing the specific temporal locality of the queries belonging to a given topic, i.e., queries more frequent

in specific time intervals or with periodic "burstiness" (e.g., queries on weather forecasting, typically issued in the morning, or queries on sport events, typically issued in the weekend);

- a *dynamic* cache $\mathcal{D}$ of size $|\mathcal{D}| = f_d \cdot N$. The dynamic cache $\mathcal{D}$ is managed using some replacement policy, such as LRU. It is expected to store the results of "bursty" queries (i.e., queries requested frequently for a short period of time) that are not captured neither by $\mathcal{S}$ nor $\mathcal{T}$ as they are not sufficiently popular or are unassigned to any of the $k$ topics. Queries cannot be assigned to a topic for two reasons: (i) the query was never seen before, hence the topic classifier fails to detect its topics, or (ii) even though it was already submitted in the past, no topic was assigned to it due to a very low classification confidence (see Sec. 3.3).

The parameters $f_s$, $f_t$, and $f_d$ denote the fractions of entries $N$ devoted to the static, topic, and dynamic caches, respectively, so that $f_s + f_t + f_d = 1$. Note that, if $f_t = 0$, our STD cache becomes the classical SDC cache. The number of entries in each section $\mathcal{T}.\tau$ of the topic cache can be fixed, i.e., $|\mathcal{T}.\tau| = |\mathcal{T}|/k$, for every $\tau \in \{t_1, \ldots, t_k\}$, or chosen on the basis of the popularity of the associated topic (observed in a past query stream). In the latter case, we model the topic popularity as the number of distinct queries in the topic since estimating this number allows us to assign to the topic a number of entries proportional to its requested queries. This entails a more efficient utilization of the cache space since queries belonging to a popular topic have greater chances to be retained in the cache as their topic receives more entries as compared to other queries belonging to unpopular topics.

In Fig. 2 we show an example of how the proposed cache can be employed in a search engine system. The management of the STD cache is reported in Alg. 1. When a query with its topic $\tau \in \{t_1, \ldots, t_k\}$ arrives, the cache manager first checks if the query is in the static cache $\mathcal{S}$. If so, we have a hit; otherwise, if the query has a topic handled by the cache, the manager checks the topic-specific section of the topic cache $\mathcal{T}.\tau$, updating the topic cache with its specific replacement policy, if necessary, and producing a hit or a miss if the query was cached or not. If the query was not assigned to any topic, the dynamic cache $\mathcal{D}$ is responsible for managing the query and producing a hit or a miss. We note that the pseudo-code does not detail the retrieval of the query results from the cache or its processing on the inverted index of the search engine in case of a hit or miss, respectively.

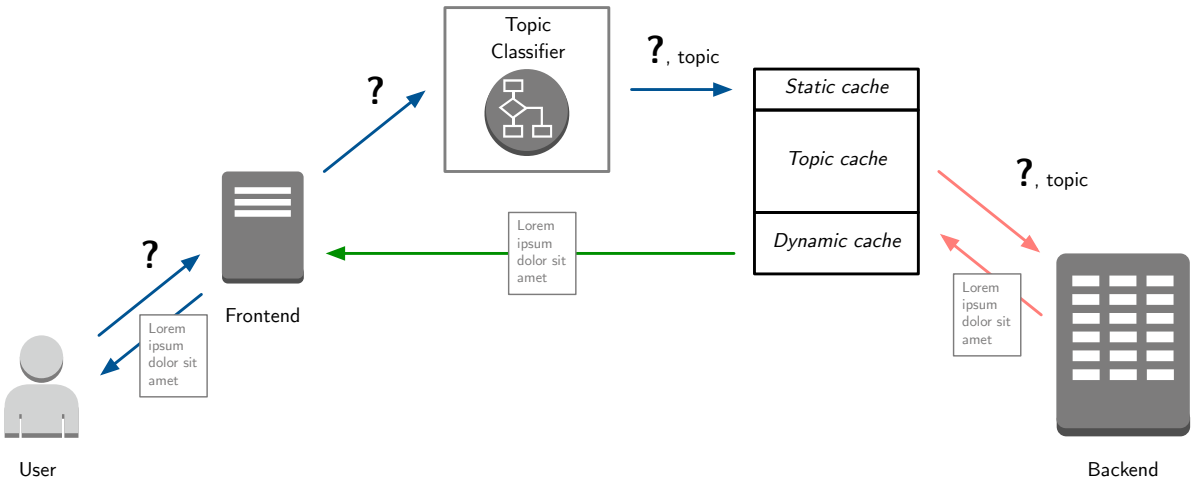



Figure 2: Example of a search engine using the STD cache. The question mark represents the user query. In case of a hit in the cache, the results are returned to the user immediately (green arrow). Otherwise, a cache miss is encountered, and the request is sent to the back-end server (red arrows).

**Algorithm 1: The STD cache management process.**

**Input :** A query $q$ and its topic $\tau$
**Output:** Hit or miss and updated STD cache

```
if q in S then
    return hit
if τ in {t_1, ..., t_k} then
    if q in T.τ then
        update T.τ and return hit
    else
        update T.τ and return miss
if q in D then
    update D and return hit
else
    update D and return miss
```

Note that cache misses incur different costs since some queries are more expensive to process than others (in term of time and resources). For our research purpose of comparing different caching architectures, our performance analysis focuses on the hit rate (considering all the misses with the same cost) and the processing time of cache-miss queries. In other efforts that focus on determining which element must be evicted from the dynamic cache or admitted to the static cache, the cost of the misses is also taken into account [3, 24, 41]. Anyway, these strategies, based on how costly is the computation of query results, can be used in synergy with our caching architecture to improve the overall performance.

### *3.2.* STD *Cache Configurations*

Our proposed STD cache model can be implemented in different ways, depending on several parameters such as the values of $f_s$, $f_t$ and $f_d$, the number of entries assigned to each topic in the topic cache, the replacement policy adopted, and so on. In the following, we illustrate some of these implementations (see Fig. 3), which will be part of our experimental evaluation (see Sec. 5).

- STD with topic cache managed by LRU with fixed size ($\text{STD}^{\text{f}}_{\text{LRU}}$). This cache includes the static, topic, and dynamic caches discussed above. The topic cache entries are divided equally among the different topics (i.e., without taking into account the topic popularity), and each topic cache section is managed according to the LRU replacement policy.
- STD with topic cache managed by LRU with variable size ($\text{STD}^{\text{v}}_{\text{LRU}}$). This cache is similar to the previous one with the difference that each topic has a number of entries proportional to its popularity. We quantified this topic popularity as the number of distinct queries that belong to the topic in the training set of the query log.
- STD with topic cache managed by SDC with variable entry size ($\text{STD}^{\text{v}}_{\text{SDC}}$). This cache is similar to the $\text{STD}^{\text{v}}_{\text{LRU}}$ cache, but now the topic cache is managed by SDC instead of LRU. Each topic gets a given number of entries proportional to its topic popularity, and all topic cache sections are split in a static and dynamic cache. The fraction of entries allocated to the static portion of these caches is a constant fraction of the topic cache entries and denoted with $f_t^s$. The remaining entries, allocated for the topic, are managed by LRU.
- Topic-only cache managed by SDC with variable entry size ($\text{T}^{\text{v}}_{\text{SDC}}$). This is an alternative version of the previous implementation since the queries with no topic are managed as queries belonging to an additional topic $k+1$. This means that instead of having a predefined size for static and dynamic caches, the number of entries would depend on the number of queries without a topic.

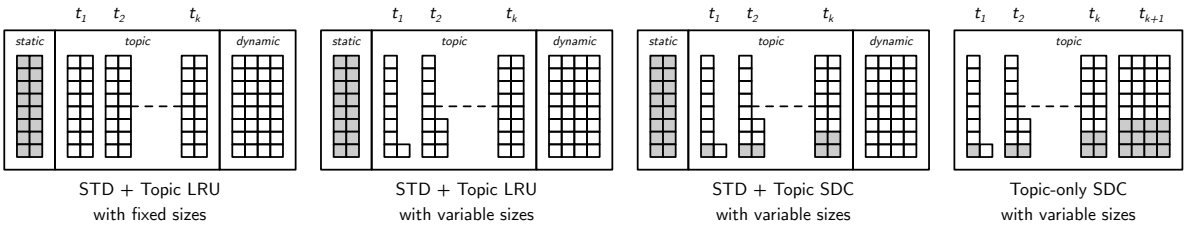



Figure 3: Different Configurations of a STD cache.

### *3.3. Modeling Queries as Topics*

Query topic categorization of user queries is well addressed within Web companies to increase effectiveness, efficiency, and revenue potential in general-purpose Web search engines [10]. Unfortunately, we cannot rely on the query topic characterization provided by the search engine industry since this information is missing from the query logs we use. Thus, to distill the topics of the queries in our query logs, we rely on LDA topic modeling.

*Latent Dirichlet Allocation (LDA) [12].* LDA is an unsupervised approach not requiring any prior knowledge of the domain for discovering the latent topics. Given a collection of documents and the number $k$ of topics, it returns $k$ lists of keywords, each representing a latent topic. Let $\theta_d$ be the per-document topic distribution, which is assumed to be drawn from a Dirichlet distribution with hyper-parameter $\alpha$. The documents are a mixture of topics, and the multinomial random variable $z_{d,n}$ of a topic to appear in position $n$ of document $d$ is conditioned on $\theta_d$. Words appearing in a document are selected according to another multinomial distribution with hyper-parameter $\beta$, conditioned on the chosen topic. In this way, each word has a probability that depends on its likelihood to appear in the document relevant to the topic. In summary, LDA can be seen as a generative process where documents are generated sequentially as reported in Alg. 2.

**Algorithm 2: The LDA generative process.**

```
for each document d do
    Draw θ_d ~ Dir(α)
    for each word position n do
        Draw a topic z_{d,n} ~ Multinomial(θ_d)
        Draw a word w_{d,n} ~ Multinomial(β|z_{d,n})
```

By inverting the generative process, it is possible to infer the topics from the words appearing in the documents. So, given a document $d$, we have to compute the posterior distribution of the hidden variables $\mathbf{z}_d$ and $\theta_d$ as follows:

$$p(\theta_d, \mathbf{z}_d|\mathbf{w}_d, \alpha, \beta) = \frac{p(\theta_d, \mathbf{z}_d, \mathbf{w}_d|\alpha, \beta)}{p(\mathbf{w}_d|\alpha, \beta)},$$

where the vector $\mathbf{w}_d$ represents the words observed in $d$, while the vector $\mathbf{z}_d$ represents the positions of words in $d$. Both vectors have the same size, equal to the length of $d$. Statistical inference techniques, such as Gibbs sampling [25], are employed to learn the underlying topic distribution $\theta_d$ of each document.

*Finding Latent Topics from Query-Document Pairs.* Given a training query log, we aim at learning a query topic classifier based on LDA. Since queries are short and lack context, it is difficult to train the model accurately. To circumvent this problem, we enrich the queries with the content of their clicked pages whose URLs are available in the training query log. We thus create a collection of query-document samples made of queries plus the text of their clicked results gathered from the Web. In case, for a given query, the user

did not click any results, or the clicked URL was not available any longer, we remove the corresponding query from the set. Given a query-document pair, we use this content as a proxy of the query, and train LDA to learn the topics discussed in the collection of query-document pairs.

The trained LDA classifier returns a distribution of topics for each query-document pair. Since we assume that a query can be assigned to only one topic, we always choose, for the pair, the topic with the highest probability. In our experiments, we use different query logs w.r.t. the training one, and we assume the LDA classifier is able to classify only queries already seen in the training query log since for new queries we will lack the content of clicked pages.

*Query Topic Assignment.* Once we know the topics of the proxy query-documents, we must assign a single topic to each query. Since the same query might appear in different query-document pairs, possibly assigned to different topics, we must decide which one of these topics should be associated with the query. To this end, we adopted a simple voting scheme that assigns to each query the topic of the query-document that got more clicks by the users. In doing so, we leverage the strong signal coming from clicks about the relevance of a document and its topic to the information need expressed by the query. Also, it allows us to estimate the most popular topic that can be assigned to ambiguous queries (i.e., queries with more than one meaning that have more possible topics).

*Estimating Topic Popularity.* In some of the implementations of our topic cache (i.e., $\mathsf{STD}^{\mathsf{v}}_{\mathsf{LRU}}$, $\mathsf{STD}^{\mathsf{v}}_{\mathsf{SDC}}$, $\mathsf{T}^{\mathsf{v}}_{\mathsf{SDC}}$), we assign to each topic an amount of cache entries proportional to the topic popularity. Similarly to the static cache, where past popular queries are assumed popular in the future, we assume that popular topics observed in the past remain popular in the future; so they get more entries in the topic cache $\mathcal{T}$.

We quantified this topic popularity as the number of distinct queries that belong to the topic $q_\tau$. Note that this statistic is computed over the training period. More in detail, let $|\mathcal{T}|$ be the size of the topic cache, $q$ be the number of distinct queries in the training set, each topic $\tau$ will get a number of entries $|\mathcal{T}.\tau|$ equal to

$$|\mathcal{T}.\tau| = \left\lfloor \frac{|\mathcal{T}|}{q} \cdot q_\tau \right\rceil$$

where the symbol $\lfloor x \rceil$ is the nearest integer to $x$. For example, suppose we have a topic cache with size $|\mathcal{T}| = 5$ and 9 distinct queries observed in the training: 6 for the topic `weather` and 3 for `education`, we will have $|\mathcal{T}.\mathsf{weather}| = \lfloor 3.33 \rceil = 3$ and $|\mathcal{T}.\mathsf{education}| = \lfloor 1.66 \rceil = 2$.

## 4. Dataset Description

*Query logs.* For our experiments, we used two real-world query logs, namely, the AOL and MSN query logs.

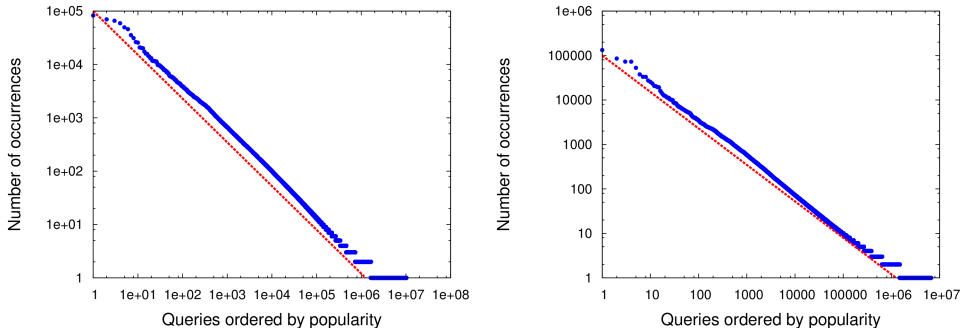
The AOL[1] query log [42] consists of about $36M$ query records, submitted by $650K$ users over a period of three months (from March to May 2006). Each record in the dataset is made of the user ID, the query keyword(s), the timestamp, the rank and the URL of the domain of the clicked result. The last two fields represent a click-through event, and they are present only if the user clicked on a search result. About $19.4M$ records have a domain URL, and the number of distinct URLs is $1.6M$. Notice that, when a user clicked on more than one result in the same search session, there are repeated records for that query (one for each clicked result). Hence, for our experiments, we recreated the query stream by removing the duplicated records representing multiple clicks for the same query, and we kept only the "first query" of the sequence. The final query stream, used for the simulation of the cache, consists of $20M$ queries.

The MSN [2] query log [20] consists of about $14.9M$ records storing the timestamp, the search keyword(s), the query id, the session id, and the result count. About $8.8M$ records have a clicked URL for a total of $3.4M$ distinct URLs.

For both datasets, we preprocessed the queries by removing special characters and converting them to lowercase. After the preprocessing, the number of distinct queries is $9.3M$ and $6.2M$ for AOL and MSN,

[1] Although controversial, the AOL dataset is widely used in academic research [48].

[2] The MSN dataset has restriction as it was released only to the participants of the workshop [20].




(a) AOL (b) MSN

Figure 4: Query logs: Distribution of the query frequencies.

respectively. The distribution of the query popularity follows a power law in both query logs as shown in Fig. 4. We plot the distribution using a log-log scale; the queries are ordered along the $x$-axis by popularity, while the number of occurrences of the query (i.e., query frequency) is shown on the $y$-axis.

We partitioned both query logs into two portions, one for training purposes (e.g., cache initialization) and one for testing the performance of the cache. We sorted the query logs by time and split them into two fractions: $X$ for the training set and $100 - X$ for the test set with $X = 30\%, 50\%$, and $70\%$. We present the results for the 70%-30% split for caching without admission policies, but we observed similar results for the other training-test splits. In this case, the training (resp. test) set contains $6.7M$ (resp. $3.2M$) unique queries for AOL and $4.5M$ (resp. $2.1M$) unique queries for MSN. We also tried different splitting of the datasets, such as weekly splits, but the average results were not much different from the ones reported in the next section.

*Document Collections.* Given the URLs from both query logs, we collected the associated pages from the Web, and gathered $1M$ documents for AOL and $2.1M$ documents for MSN. Then, we extracted and pre-processed the text (e.g., stop-word removal, lemmatization, and stemming). We removed overly short and long documents (less than 5 and more than 100K words). Lastly, we enriched the documents with the corresponding query keywords.

*LDA Topics.* To train the LDA model, we subsampled the documents from the training period and used $500K$ documents from AOL and $350K$ documents from MSN query log. Notice that this training is only used for learning the topics discussed in the collection (e.g., sport, politics, weather). After the topic detection, given a query and its clicked result, we use LDA to predict the topic of the query.

We removed frequent and rare words from the dictionary and set the number $k$ of topics to discover to 500, estimated empirically. The approach is probabilistic; hence, the topic detection can change with different collections and different number of topics. We tried other configurations, changing the subsets of documents in the training set and using different values of $k$ (e.g., 50, 100, and 500); we observed that the impact on the caching performance was negligible. Some of the topic keywords, extracted from the AOL dataset, are shown in Table 1.

In Fig. 5, we report the distribution of topics (the number of queries per topic) extracted from both datasets. It is worth noting that the topic portion of STD cache exploits the subset of queries in the test set stream having a known topic. These queries are necessarily among those already encountered in the training set stream and successfully classified. The test queries that were not assigned to a topic compete instead for the use of the static and dynamic portions of the STD cache. The percentage of queries in the test set with a topic is 65% for AOL and 58% for the MSN query log.

Table 1: Some topics and their keywords extracted from the AOL search queries and the page content of their clicked results.

| Topic | Topic Keywords |
|---|---|
| **shopping** | shop, order, item, ship, gift, custom, sale, return, account, cart |
| **university** | student, program, faculti, campu, graduat, research, academ, alumni, colleg, univers |
| **weather** | weather, forecast, snow, storm, rain, wind, winter, radar, flood, cold |
| **movies** | movi, comic, news, star, theater, review, marvel, film, seri, comedi |
| **cooking** | recip, cook, bean, chicken, chef, salad, cake, flavor, potato, rice |
| **travelling** | travel, trip, destin, flight, vacat, book, deal, airlin, hotel, search |

Notice that the behavior of the cache may change depending on the value of $k$ as it reflects the number of partitions of the topical cache. For our experiments, we estimated the value of $k$ by keeping the other parameters of LDA unchanged and checking the log-likelihood of the predicted topics. This approach was already tested for detecting unknown events in news streams [38, 39]. For both datasets, the best value of $k$ was 500. Also, we tried other values, such as 50 and 100, and we did not observe a large difference in the hit rates proving the robustness of the proposed cache. However, the performance consistently declined with different values of $k$. In particular, too small values of $k$ (e.g., $k < 50$ for our data) would mean a few too general topics (a lot of queries would be assigned to each topic). The extreme case is $k = 1$, where the STD cache has only 3 portions: the static cache, the (one) topic cache, and the dynamic cache. This configuration fails to benefit from the temporal patterns observed for queries belonging to the different topics as there is only a single topic. On the other hand, too large values of $k$ (e.g., $k > 500$ for our data) results in too many very narrow topics (only a few queries would be assigned per topic). This fails to leverage the repetition of queries inside a topic cache partition as the entire topical portion would be too shattered.

## 5. Experiments

We ran our experiments using the AOL and MSN query logs described in Sec. 4. For the caching simulations, we considered the scenario of storing the query results in the cache (e.g., the first SERP). We took as input the stream of queries. If the query is found in the cache we have a cache hit; otherwise, there is a cache miss. In case of miss, if the query is not filtered out by the admission policy implemented (if any) and the dynamic portion of the cache is full, the eviction policy is applied to vacate space for the query results.

*Experimental Setting.* For our experiments, we set the cache size $N$ to different values: $64K$, $128K$, $256K$, $512K$, and $1024K$. The datasets are split into training and test sets. Training set data are used for three

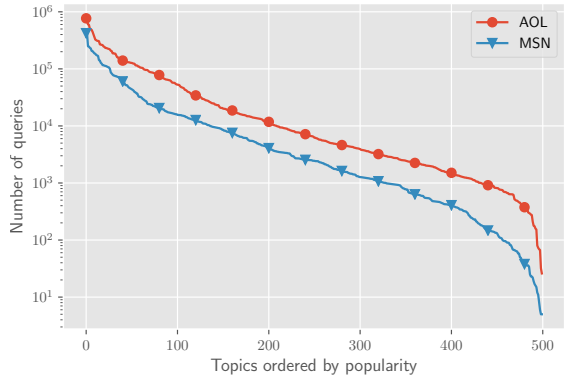



Figure 5: Distribution of topic popularities (# queries per topic) for AOL and MSN query logs.

purposes: (1) learning the frequency of the queries and loading popular queries in the static cache(s), (2) training the LDA topic classifier from the queries (and clicked documents) and estimating their popularity for balancing the entries of the topic cache(s), and (3) warming up the LRU cache(s). We assess the cache performance in terms of hit rate, namely the number of cache hits in the test set divided by the number of queries in the test set.

For our experiments, we considered the following cache organizations:

- SDC: we used as baseline the traditional static and dynamic cache, where the dynamic portion is managed by LRU. This approach is considered the state-of-the-art technique for query-result caching as also confirmed by recent works on caching that used SDC as a principal baseline (e.g., [30]).
- $\mathsf{STD}^{\mathsf{f}}_{\mathsf{LRU}}$: the STD cache where the topic cache is managed by LRU and all topics receive the same amount of entries.
- $\mathsf{STD}^{\mathsf{v}}_{\mathsf{LRU}}$: the STD cache where the topic cache is managed by LRU and the topics receive an amount of entries which is proportional to the topic popularity, as explained in Sec. 3.
- $\mathsf{STD}^{\mathsf{v}}_{\mathsf{SDC}}$: the STD cache where the topic cache is managed by SDC and whose size depends on the popularity of the topic. Compared to the previous two configurations, this cache has another parameter $f_t^s$ representing the static fraction of the SDC used inside the topic cache. In our tests, we include two different implementations of this configuration. In the first implementation, the static cache $\mathcal{S}$ stores only the frequent queries with no topic (C1) as the popular queries assigned to a topic would be stored in the static portion of the corresponding topic cache. In the second implementation (C2), the $\mathcal{S}$ stores all the top queries (with or without the topic). For popular queries with topic, the algorithm checks if they are already in $\mathcal{S}$. If not, it stores them in the $f_t^s$ fraction of entries of the corresponding SDC used in topic cache.
- $\mathsf{T}^{\mathsf{v}}_{\mathsf{SDC}}$: The cache entries are divided proportionally to the topic popularity and the no-topic queries that belong to an additional topic $\tau = t_{k+1}$.

For the baseline SDC cache and the proposed STD cache configurations, the static parameter $f_s$ varies from 0.0 to 1.0 with step of 0.1, while the other parameters ($f_t$ and $f_d$) are tuned based on the remaining size of the whole cache, e.g., $N \cdot (1 - f_s)$. Regarding $\mathsf{STD}^{\mathsf{v}}_{\mathsf{SDC}}$, the fraction of static of the SDC caches used in the topic portion, $f_t^s$, is the same for all the topics. We also experimented with variable $f_t^s$ estimates *per topic*, but the overall experimental results are similar to those achieved with a fixed $f_t^s$, and we do not report them here.

*Experimental Limitations.* The two query logs used for our experiments only span a few months/weeks, resulting in the following evaluation limitations.

- Both query logs are small, around $20M$ queries for AOL and $14M$ for MSN. We overcome this lack of data by keeping the overall size of the cache small. Maintaining a realistic ratio of query volume to cache size better simulates real-world search engine performance as search engines receive tens of thousands of queries per second and that volume of queries compete for access to the cache. A similar decision was taken by Fagni et al. in [22]. They used cache sizes from $50K$ to $300K$ entries for testing logs from $2M$ to $7M$ of queries. Indeed, with such small query logs, a cache with several millions of entries would be able to host all of the requests, and the hit rate would be unrealistically high (i.e., only cold-start misses would occur).
- The two query logs span a short period of time, AOL covers 3 months (from March to May 2006) and MSN only 1 month (May 2006). For this reason, we could not run experiments to show the change of topics over time. Anyway, a topic modeling approach (e.g., LDA) can be trained on different time periods to discover different topics. The corresponding topical portion of the cache can be updated periodically (e.g., during the refreshing of the static portion of the cache) to capture these variation in the topics of queries.

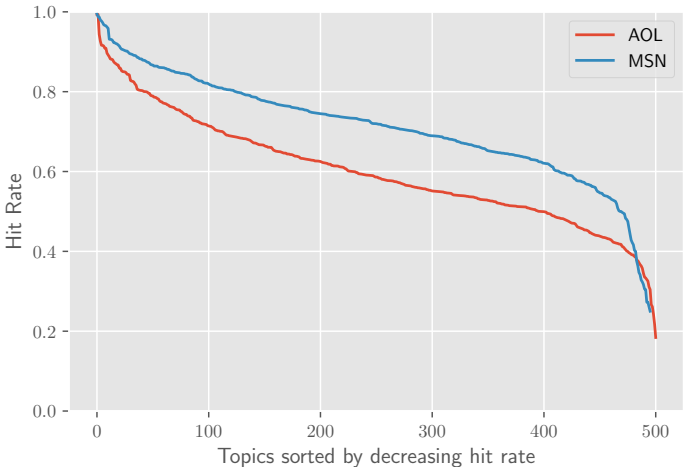



Figure 6: Topic hit rates registered by the different topic portions in the $\mathsf{T}^{\mathsf{v}}_{\mathsf{SDC}}$ configuration. The red curve is for the AOL query log, and the blue curve corresponds to the MSN query log.

- Due to the lack of data, we experimented with the splitting of our logs. For some experiments, we used less training data so as to retain a longer stream on which to test our caching strategy (e.g., caching with admission policy). A test set with 30% of the query stream would results in too few queries if we consider that some of them (e.g., singleton or rare queries) are not allowed to enter in the cache by the admission policy. As previously mentioned, when using a few queries for assessing the caching performance, the hit rate can be unrealistically high as the cache is sufficiently large to allocate all the requests, and we want to avoid this situation.

*Experimental Results.* We investigate the following research questions:

**RQ1.** For a given cache size, is our proposed STD cache able to improve the hit rate performance metric w.r.t. SDC and, if so, adopting which configuration and optimal parameter values?

**RQ2.** Given the best STD competitor identified in RQ1, what is the impact of the other configuration parameters? In particular, given a static fraction $f_s$, what is the impact of the topic and dynamic caches of STD w.r.t. the dynamic cache of SDC?

**RQ3.** How large are the hit rate improvements of the best STD configuration w.r.t. SDC, measured in term of the distance with the hit rate of a theoretical optimal caching strategy?

**RQ4.** Do the query admission policies affect the caching performance? How do these policies impact performance gains of STD over SDC?

To address RQ1, we assume a cache with a given number of entries (e.g., $N$ is defined by the system administrator) and aim to discover the best cache configuration and parameters in terms of hit rate. Table 2 reports the best hit rates obtained with SDC (our baseline) and the other topic-caching strategies for different cache sizes. For each caching strategy, we report also the values of the $f_s$, $f_t$, $f_d$, and $f_t^s$ that achieved the best hit rates. As shown, not all parameters are used by all the cache configurations, so for those caches where the parameter is not needed we use the symbol $-$. For each cache size, the best hit rates are highlighted in bold. Since for the $\mathsf{T}^{\mathsf{v}}_{\mathsf{SDC}}$ cache there is no separation among static, topic and dynamic caches, and all topic caches are treated as small SDC portions, we complete our analysis on the behavior of topical caching by showing the hit rates achieved by the different topic portions in Fig. 6. Note that the topics are shown ordered by decreasing hit rate.

Our experiments show that with both datasets the STD caches always perform better than SDC in terms of hit rate. In particular, the $\mathsf{STD}^{\mathsf{v}}_{\mathsf{SDC}}$ (C2) configuration performs always better than the others.

Table 2: Best hit rates for SDC and different topic-cache strategies for the AOL and MSN datasets. The parameters $f_s$, $f_t$, and $f_d$ denote the fractions of the total cache devoted to the static, topic, and dynamic caches, respectively. The parameter $f_t^s$ denotes the percentage of static cache in the topic cache. The best hit rates are highlighted in bold.

| **Cache Size** | **Strategy** | AOL | | | | | MSN | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **Hit Rate** | $f_s$ | $f_t$ | $f_d$ | $f_t^s$ | **Hit Rate** | $f_s$ | $f_t$ | $f_d$ | $f_t^s$ |
| 64K | $\mathsf{SDC}$ | 33.70% | 0.9 | – | – | – | 45.23% | 0.9 | – | – | – |
| | $\mathsf{STD}^{\mathsf{f}}_{\mathsf{LRU}}$ | 36.91% | 0.9 | 0.07 | 0.03 | – | 46.57% | 0.8 | 0.13 | 0.07 | – |
| | $\mathsf{STD}^{\mathsf{v}}_{\mathsf{LRU}}$ | **37.34%** | 0.9 | 0.07 | 0.03 | – | 47.09% | 0.9 | 0.05 | 0.05 | – |
| | $\mathsf{STD}^{\mathsf{v}}_{\mathsf{SDC}}$ (C1) | 36.21% | 0.8 | 0.16 | 0.04 | 90% | 46.00% | 0.1 | 0.72 | 0.18 | 90% |
| | $\mathsf{STD}^{\mathsf{v}}_{\mathsf{SDC}}$ (C2) | **37.34%** | 0.8 | 0.16 | 0.04 | 60% | **47.15%** | 0.8 | 0.13 | 0.07 | 60% |
| | $\mathsf{T}^{\mathsf{v}}_{\mathsf{SDC}}$ | 33.16% | – | – | – | 90% | 42.30% | – | – | – | 80% |
| 128K | $\mathsf{SDC}$ | 37.58% | 0.9 | – | – | – | 48.15 % | 0.9 | – | – | – |
| | $\mathsf{STD}^{\mathsf{f}}_{\mathsf{LRU}}$ | 40.89% | 0.9 | 0.08 | 0.02 | – | 49.73% | 0.9 | 0.07 | 0.03 | – |
| | $\mathsf{STD}^{\mathsf{v}}_{\mathsf{LRU}}$ | **41.19%** | 0.9 | 0.07 | 0.03 | – | 50.04% | 0.9 | 0.07 | 0.03 | – |
| | $\mathsf{STD}^{\mathsf{v}}_{\mathsf{SDC}}$ (C1) | 40.08% | 0.8 | 0.16 | 0.04 | 90% | 49.13% | 0.1 | 0.72 | 0.18 | 90% |
| | $\mathsf{STD}^{\mathsf{v}}_{\mathsf{SDC}}$ (C2) | **41.19%** | 0.9 | 0.08 | 0.02 | 20% | **50.08%** | 0.8 | 0.16 | 0.04 | 80% |
| | $\mathsf{T}^{\mathsf{v}}_{\mathsf{SDC}}$ | 37.49% | – | – | – | 90% | 46.08% | – | – | – | 90% |
| 256K | $\mathsf{SDC}$ | 41.25% | 0.9 | – | – | – | 50.77% | 0.9 | – | – | – |
| | $\mathsf{STD}^{\mathsf{f}}_{\mathsf{LRU}}$ | 44.57% | 0.9 | 0.08 | 0.02 | – | 52.50% | 0.9 | 0.08 | 0.02 | – |
| | $\mathsf{STD}^{\mathsf{v}}_{\mathsf{LRU}}$ | **44.80%** | 0.9 | 0.08 | 0.02 | – | **52.63%** | 0.9 | 0.07 | 0.03 | – |
| | $\mathsf{STD}^{\mathsf{v}}_{\mathsf{SDC}}$ (C1) | 43.63% | 0.7 | 0.24 | 0.06 | 90% | 51.94% | 0.1 | 0.72 | 0.18 | 90% |
| | $\mathsf{STD}^{\mathsf{v}}_{\mathsf{SDC}}$ (C2) | **44.80%** | 0.9 | 0.08 | 0.02 | 20% | **52.63%** | 0.9 | 0.07 | 0.03 | 30% |
| | $\mathsf{T}^{\mathsf{v}}_{\mathsf{SDC}}$ | 41.70% | – | – | – | 90% | 49.46% | – | – | – | 90% |
| 512K | $\mathsf{SDC}$ | 44.52% | 0.9 | – | – | – | 52.91% | 0.9 | – | – | – |
| | $\mathsf{STD}^{\mathsf{f}}_{\mathsf{LRU}}$ | 47.74% | 0.9 | 0.07 | 0.03 | – | 54.60% | 0.9 | 0.08 | 0.02 | – |
| | $\mathsf{STD}^{\mathsf{v}}_{\mathsf{LRU}}$ | 48.06% | 0.8 | 0.16 | 0.04 | – | **54.83%** | 0.8 | 0.16 | 0.04 | – |
| | $\mathsf{STD}^{\mathsf{v}}_{\mathsf{SDC}}$ (C1) | 46.93% | 0.7 | 0.15 | 0.15 | 80% | 54.43% | 0.2 | 0.64 | 0.16 | 90% |
| | $\mathsf{STD}^{\mathsf{v}}_{\mathsf{SDC}}$ (C2) | **48.08%** | 0.8 | 0.16 | 0.04 | 20% | **54.83%** | 0.8 | 0.16 | 0.04 | 10% |
| | $\mathsf{T}^{\mathsf{v}}_{\mathsf{SDC}}$ | 45.60% | – | – | – | 90% | 52.52% | – | – | – | 90% |
| 1024K | $\mathsf{SDC}$ | 47.37% | 0.7 | – | – | – | 54.93% | 0.9 | – | – | – |
| | $\mathsf{STD}^{\mathsf{f}}_{\mathsf{LRU}}$ | 50.31% | 0.8 | 0.13 | 0.07 | – | 56.57% | 0.9 | 0.07 | 0.03 | – |
| | $\mathsf{STD}^{\mathsf{v}}_{\mathsf{LRU}}$ | 50.90% | 0.6 | 0.32 | 0.08 | – | 56.86% | 0.7 | 0.24 | 0.06 | – |
| | $\mathsf{STD}^{\mathsf{v}}_{\mathsf{SDC}}$ (C1) | 49.98% | 0.4 | 0.40 | 0.20 | 80% | 56.72% | 0.2 | 0.64 | 0.16 | 80% |
| | $\mathsf{STD}^{\mathsf{v}}_{\mathsf{SDC}}$ (C2) | **51.01%** | 0.5 | 0.40 | 0.10 | 70% | **56.92%** | 0.5 | 0.40 | 0.10 | 50% |
| | $\mathsf{T}^{\mathsf{v}}_{\mathsf{SDC}}$ | 49.33% | – | – | – | 80% | 55.16% | – | – | – | 90% |

As expected, $\mathsf{STD}^{\mathsf{f}}_{\mathsf{LRU}}$ performs worse than $\mathsf{STD}^{\mathsf{v}}_{\mathsf{LRU}}$ as it gives to each topic the same number of entries instead of allocating the topic cache entries proportionally to the topic popularity. Moreover, $\mathsf{STD}^{\mathsf{v}}_{\mathsf{SDC}}$ (C1) cache exhibits lower hit rates compared to $\mathsf{STD}^{\mathsf{v}}_{\mathsf{SDC}}$ (C2) and $\mathsf{STD}^{\mathsf{v}}_{\mathsf{LRU}}$ caches. Analyzing the cache misses encountered with (C1), we see that this reduction of performance is due to the fact that the static cache $\mathcal{S}$ of (C1) hosts only the results of *no-topic* queries. Some of these queries may be not very popular; hence, storing them in the static fraction causes a lower hit rate in static with a reduction of the overall performance. In particular, this phenomenon is more evident when $f_s$ increases since we are allocating more space to $\mathcal{S}$ and, at some point, also infrequent *no-topic* queries are selected just to fill in the space. Nevertheless, (C2) does not suffer from this since it stores in $\mathcal{S}$ the frequent queries (with or without topic), allowing a better utilization of the static fraction of the whole cache. We can also observe that the $\mathsf{T}^{\mathsf{v}}_{\mathsf{SDC}}$ cache has lower performance than the other $\mathsf{STD}$ configurations. In most of the cases it performs close to $\mathsf{SDC}$, and for small caches it does not improve the baseline. Anyway, we believe that its results allow us to better understand the benefit of using a topic cache together with static and dynamic caches. In $\mathsf{STD}^{\mathsf{f}}_{\mathsf{LRU}}$, $\mathsf{STD}^{\mathsf{v}}_{\mathsf{LRU}}$, and $\mathsf{STD}^{\mathsf{v}}_{\mathsf{SDC}}$ the amount of entries dedicated to the *no-topic* queries is limited by the parameter $f_d$. Hence, there is a fair division of the cache space among the queries belonging to a topic and those that could not be classified. On the other hand, in $\mathsf{T}^{\mathsf{v}}_{\mathsf{SDC}}$ the *no-topic* queries are treated as queries belonging to an extra topic (i.e., $t_{k+1}$); thus, the amount of entries is proportional to the popularity of the $(k+1)$-th topic, penalizing the other $k$ topics. Since in our data most of the queries are not classified, this leads to an unbalanced splitting of the space between the *no-topic* queries and the others. In particular, for both query logs we observed that small- and medium-size caches (e.g., up to 256K entries) have more than 90% of their entries used by no-topic queries, leaving a few entries for each topic cache (e.g., on average 40 entries per topic in caches of 256K total entries). This affects the overall hit rate which is smaller as compared to the best $\mathsf{STD}$ approach. For example, with cache size of 256K entries, we measured ∼41% vs. ∼44% hit rates for AOL and ∼49% vs. ∼52% hit rates for MSN (see Table 2). For large caches (e.g., with 1024K entries), the no-topic cache gets around 50% of the entries, and each topic has on average 820 entries resulting in better hit rates (the cache benefits from the topic splitting of the entries). Indeed, as observed from Table 2, the hit rate degradation compared to the best $\mathsf{STD}$ approach is smaller, i.e., ∼49% vs. ∼51% for AOL and ∼55% vs ∼56% for MSN.

We investigated the reasons why our $\mathsf{STD}$ cache outperforms the $\mathsf{SDC}$ baseline. Since the higher $\mathsf{STD}$ hit rate is due to less misses encountered, we analyzed the average distance of misses in the test streams (*avg. miss distance*). This distance is defined as the number of queries between two misses that were caused by the same query, e.g., for the stream `abcadafga` and a cache of size 2 the misses caused by `a` have an average distance of 2. For this experiment, we considered caches with 1024K entries and $f_s = 0.5$, as it gave the best hit rate performance for $\mathsf{STD}^{\mathsf{v}}_{\mathsf{SDC}}$ (the best configuration). We separately identified the average miss distance of its dynamic cache from the average miss distance of topic caches in $\mathsf{STD}^{\mathsf{v}}_{\mathsf{SDC}}$. Notice that the static cache does not impact on the analysis since it is populated by the same top-frequent queries for both $\mathsf{STD}$ and $\mathsf{SDC}$ caches. The results are shown in Fig. 7. The curves represent the average miss distances for the topic caches sorted by decreasing values, and we use it as a proxy of temporal locality. On the left we have large distances, which means that a miss occurred only when the repeated requests of that query were far away from each other. Notice that the number of topics can be lower than 500 since for some topics there are no misses. The average miss distances for the dynamic caches in $\mathsf{SDC}$ and $\mathsf{STD}$ are constant as they are topic-independent. These two average miss distances are lower when compared to those reported for most of the topic caches. It confirms that a $\mathsf{LRU}$ dynamic cache captures the repeated requests only if they are close to each other (small average miss distance). On the other hand, topic caches have large average miss distances. So, the advantage of a topic cache with space divided in a proportional way among the topics is that it even serves requests distant from each other on a per-topic base, i.e., with different temporal localities.

We also analyzed the time savings. Table 3 shows the saving in time for the best $\mathsf{STD}^{\mathsf{v}}_{\mathsf{SDC}}$ (C2) configuration and for the traditional $\mathsf{SDC}$. For both datasets, we measured the time (in hours) for processing the queries that cause a miss in the test set. The query processing times were computed using the partitioned Elias-Fano index and MaxScore for the ranking [52]. The queries were processed in disjunctive mode, and the top 1000 documents according to BM25 returned as results. As expected, the $\mathsf{STD}^{\mathsf{v}}_{\mathsf{SDC}}$ (C2) caching strategy allows to save time for processing queries as it has less misses in the test set.

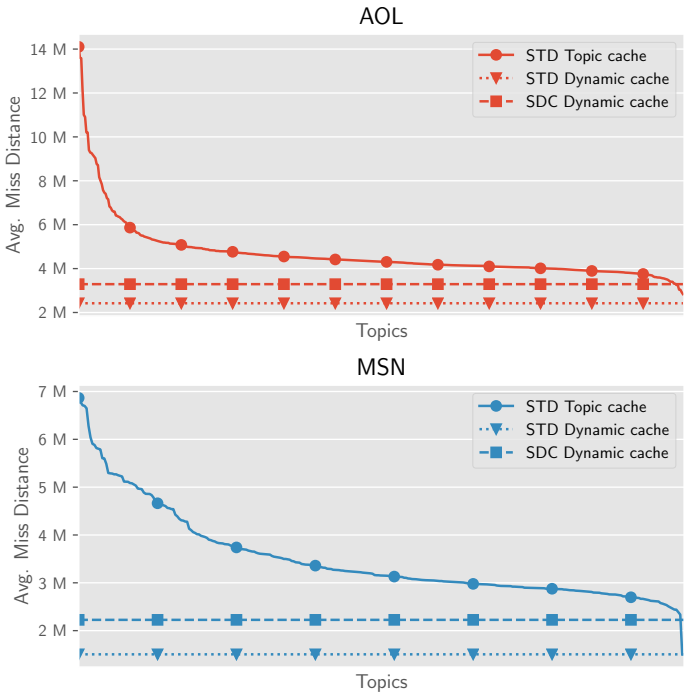



Figure 7: Per-topic average miss distances for AOL and MSN query logs.

Table 3: Time (in hours) for processing the queries that cause a miss in the test set. We report values for both AOL and MSN datasets, using different cache sizes and comparing SDC against the best configuration of $\text{STD}^{\text{v}}_{\text{SDC}}$ (C2). The gain column represents the amount of time spared by the STD approach compared to SDC.

| **Cache Size** | **Strategy** | AOL | | MSN | |
|---|---|---|---|---|---|
| | | **Miss Queries** | **Gain** | **Miss Queries** | **Gain** |
| 64K | SDC | 48.06 h | 2.78 h | 27.78 h | 0.86 h |
| | $\text{STD}^{\text{v}}_{\text{SDC}}$ (C2) | 45.28 h | | 26.92 h | |
| 128K | SDC | 45.83 h | 2.77 h | 26.69 h | 1.02 h |
| | $\text{STD}^{\text{v}}_{\text{SDC}}$ (C2) | 43.06 h | | 25.67 h | |
| 256K | SDC | 43.33 h | 2.77 h | 25.58 h | 1.00 h |
| | $\text{STD}^{\text{v}}_{\text{SDC}}$ (C2) | 40.56 h | | 24.58 h | |
| 512K | SDC | 41.39 h | 2.78 h | 24.64 h | 1.03 h |
| | $\text{STD}^{\text{v}}_{\text{SDC}}$ (C2) | 38.61 h | | 23.61 h | |
| 1024K | SDC | 39.44 h | 2.77 h | 23.72 h | 1.03 h |
| | $\text{STD}^{\text{v}}_{\text{SDC}}$ (C2) | 36.67 h | | 22.69 h | |

The overall gain is more than 2 hours and ∼1 hour for the AOL and MSN datasets, respectively.

To conclude on RQ1, the experimental results confirmed that, on equal cache sizes, the STD approach can improve SDC, resulting in an improvement of the hit rate up to ∼3.6% for AOL and ∼2% for MSN. As confirmed by simulations on two real-world query logs, the best configuration is $\text{STD}^{\text{v}}_{\text{SDC}}$ (C2), although $\text{STD}^{\text{v}}_{\text{LRU}}$ performs close. Moreover, this performance improvement is justified by the analysis of the avgerage miss distances. In fact, in STD the misses occurring in the topic caches are caused by repeated requests that are much more distant in the query stream as compared to the misses that are encountered in SDC.

To address RQ2, and see if the improvement of $\text{STD}^{\text{v}}_{\text{SDC}}$ (C2) over SDC is consistent, we compared their

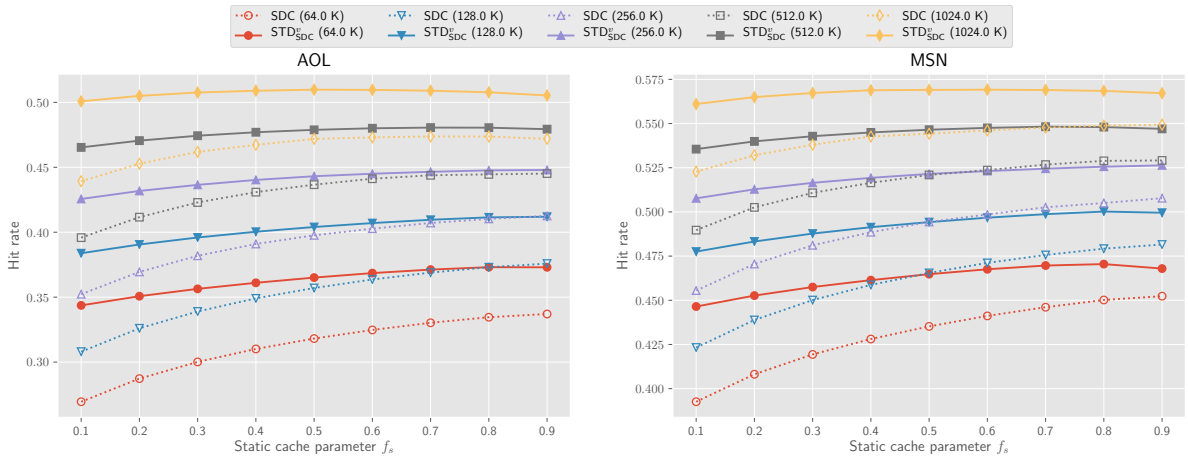



Figure 8: Hit rates of SDC and $\text{STD}^{v}_{\text{SDC}}$ for different values of $N$ and of $f_s$, for the AOL (left) and MSN (right) query logs. The remaining non-static cache space is divided in 80% for topic and 20% for dynamic caches, respectively.

hit rates varying the cache size and the value of $f_s$. Since the size of the static portion changes with $f_s$, we split the remaining $N \cdot (1 - f_s)$ entries between the topic and dynamic caches using different proportions. Here, we report the results obtained with splitting $80\% - 20\%$ (and $20\% - 80\%$) for the topic and dynamic caches, respectively, while the $f_t^s$ parameter was set to 40%.

In Fig. 8, we illustrate the hit rates for the two approaches, using dashed lines for SDC and solid lines for $\text{STD}^{v}_{\text{SDC}}$ (C2). Notice that we omit the hit rates for $f_s = 0.0$ and $f_s = 1.0$ as they correspond to completely dynamic and static caches, and the performance among the approaches are the same. If we observe the red curves for $N = 64K$, SDC hit rates (dashed lines) are always lower than $\text{STD}^{v}_{\text{SDC}}$ hit rates (solid line). The gap of hit rates between these two caching approaches goes from ∼7% for $f_s = 0.1$ to ∼3% for $f_s = 0.9$ for AOL and ∼5% for $f_s = 0.1$ to ∼2% for $f_s = 0.9$ for MSN. As expected, the maximum improvement is registered for lower values of $f_s$, since the impact of a topic plus a dynamic cache of STD over the only dynamic cache of SDC is more evident. We can conclude that the $\text{STD}^{v}_{\text{SDC}}$ (C2) cache always outperforms the SDC cache, with an average gap of 5% for AOL and 3% for MSN, and a maximum gap of more than 5% on both query logs. Consistent results were observed for other parameter values. In Fig. 9 we show the hit rates inverting the sizes of topic and dynamic caches (i.e., non-static entries are 20% for the topic cache and 80% for the dynamic cache). Hence, for a cache of $1024K$ entries and $f_s = 50\%$, the remaining non-static entries are $512K$, and they are assigned for 20% to the topic cache (102 entries) and 80% to the dynamic cache (410 entries). As expected, the hit rates are a little bit lower (especially for small caches) as this parameter configuration penalizes the topic cache. By using less entries for topics, the cache does not benefit from the topical proportions of entries. Anyway, this result shows how the hit rate curves tend to follow the same behavior observed in Fig. 8 (i.e., STD outperforms SDC for different cache sizes).

To answer RQ3, we compared the best hit rates achieved with STD and SDC against the best hit rate that can be achieved with an optimal cache policy. We used the Bélády's optimal algorithm (also known as the clairvoyant algorithm), which always evicts the element that will not be requested for the longest time. In practice, the clairvoyant algorithm assumes knowledge of future requests and optimizes based on such [11]. It provides an upper bound of the performance over which no other caching strategy can improve. We computed the gaps between Bélády hit rates and the ones achieved with the best SDC and STD configurations. The results are reported in Table 4. As we can see, the hit rates of STD are always higher than those of SDC. We report the gap between STD and SDC in the 7th column of the table and the average gap is 3.6% for AOL and 1.9% for MSN (averaged over the size of the cache). Moreover, the hit rates of STD are very close to Bélády hit rates for all cache sizes (the gap is reported in the 6th column of Table 4). The average gap between the hit rates of STD and Bélády is 6.70% for AOL and 5.06% for MSN (averaged over the size of the cache). On the other hand, the distance between SDC hit rate and Bélády

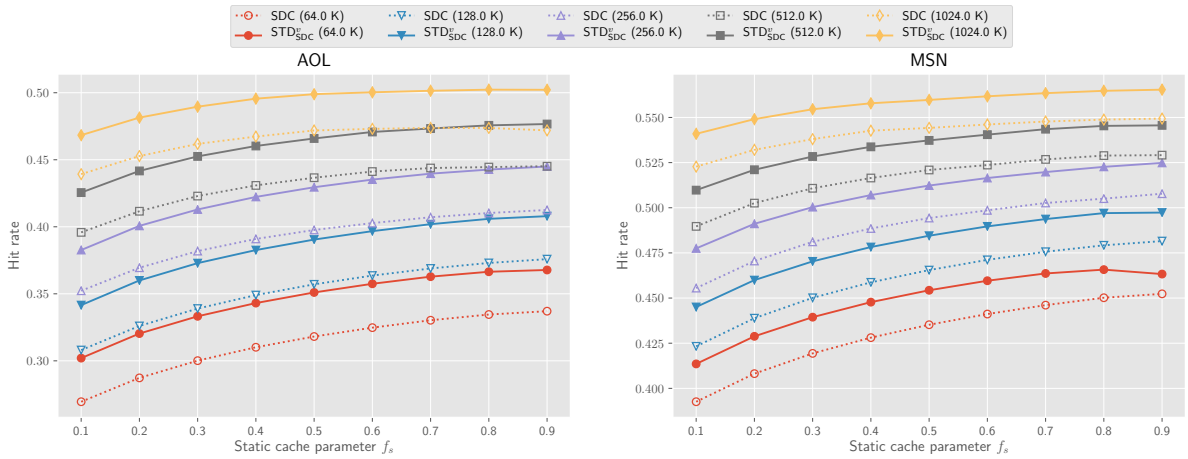



Figure 9: Hit rates of SDC and STD$^{\nu}_{SDC}$ for different values of $N$ and of $f_s$, for the AOL (left) and MSN (right) query logs. The remaining non-static cache space is divided in 20% for topic and 80% for dynamic caches, respectively.

Table 4: Gap between the best hit rates achieved with SDC and STD caches w.r.t. Bélády and each other, plus the gap reduction w.r.t. Bélády for the best hit rates achieved with SDC and STD caches.

| **Cache Size** | **Bélády** | **Best** SDC | **Best** STD | **Gap** SDC **w.r.t. Bélády** | **Gap** STD **w.r.t. Bélády** | **Gap** STD **w.r.t.** SDC | **Gap Reduction** |
|---|---|---|---|---|---|---|---|
| | | | AOL | | | | |
| 64K | 43.67% | 33.70% | 37.34% | 9.97% | 6.33% | 3.64% | 36.51% |
| 128K | 47.68% | 37.58% | 41.19% | 10.10% | 6.49% | 3.61% | 35.74% |
| 256K | 51.67% | 41.25% | 44.80% | 10.42% | 6.87% | 3.55% | 34.06% |
| 512K | 54.88% | 44.52% | 48.08% | 10.36% | 6.80% | 3.56% | 34.36% |
| 1024K | 58.06% | 47.37% | 51.01% | 10.69% | 7.05% | 3.64% | 34.05% |
| | | | MSN | | | | |
| 64K | 51.54% | 45.23% | 47.15% | 6.31% | 4.39% | 1.92% | 30.42% |
| 128K | 54.73% | 48.15% | 50.08% | 6.58% | 4.65% | 1.93% | 29.33% |
| 256K | 57.98% | 50.77% | 52.63% | 7.21% | 5.35% | 1.86% | 25.80% |
| 512K | 61.33% | 52.91% | 54.83% | 8.42% | 6.50% | 1.92% | 22.80% |
| 1024K | 61.34% | 54.93% | 56.92% | 6.41% | 4.42% | 1.99% | 31.04% |

hit rate is bigger (see the 5th column). The average gap between them is 10.30% for AOL and 6.98% for MSN. To quantify the gap reduction, we computed the relative delta between the two gaps (see the 8th column). The relative delta provides an indication on how much STD improves SDC w.r.t. Bélády hit rate. To conclude on RQ3, STD hit rates achieve a significant gap reduction w.r.t. SDC from the theoretical optimal hit rate, which is up to 36.51% for AOL and up to 31.04% for MSN.

Lastly, we addressed RQ4 by implementing two admission policies to be used in conjunction with the SDC cache and the different STD configurations. The first admission policy tries to predict queries that are not worthy, namely should not be cached, also known as *polluting queries* [5]. The second uses an oracle to determine the singleton queries that are not going to be requested in the future, hence they are not admitted in the cache. This provides an upper bound of the performance as no other admission policy can do better than the oracle which is able to see the future and determine no longer requested queries.

*Caching Without Polluting Queries.* Some queries are not worthy to be cached as they are not requested again or they are requested after a long period, generally after their eviction from the cache. To improve caching algorithms, researchers have tried to predict the singleton/polluting queries which are not admitted in the cache, with the purpose of preserving space for other, more promising, queries. Baeza-Yates et al. [5]

Table 5: Hit rates achieved with SDC and STD caches with an admission policy that does not admit the polluting queries into the cache. The best hit rates are highlighted in bold.

| **Cache Size** | SDC | $\text{STD}^{\text{f}}_{\text{LRU}}$ | $\text{STD}^{\text{v}}_{\text{LRU}}$ | $\text{STD}^{\text{v}}_{\text{SDC}}$ **(C1)** | $\text{STD}^{\text{v}}_{\text{SDC}}$ **(C2)** | $\text{T}^{\text{v}}_{\text{SDC}}$ |
|---|---|---|---|---|---|---|
| | | | AOL | | | |
| 64K | 37.98% | 41.53% | 41.85% | 40.07% | **41.87%** | 37.05% |
| 128K | 40.40% | 44.08% | 44.38% | 43.28% | **44.45%** | 41.21% |
| 256K | 41.48% | 44.86% | **45.52%** | 44.96% | 45.51% | 43.84% |
| 512K | 42.58% | 45.74% | **46.72%** | 46.39% | 46.59% | 45.74% |
| 1024K | 43.91% | 46.69% | **48.02%** | 47.82% | 47.88% | 47.43% |
| | | | MSN | | | |
| 64K | 48.35% | 49.85% | 50.27% | 49.06% | **50.29%** | 45.47% |
| 128K | 48.83% | 50.82% | **51.01%** | 50.57% | 50.97% | 48.70% |
| 256K | 49.22% | 51.16% | **51.63%** | 51.40% | 51.61% | 50.45% |
| 512K | 49.86% | 51.58% | **52.45%** | 52.33% | 52.39% | 51.56% |
| 1024K | 50.84% | 52.21% | **53.27%** | 53.19% | 53.20% | 52.84% |

analyzed stateless and stateful features for admitting queries in the cache. The stateless features do not require statistics from previous queries and can be computed on-the-fly based on the query characteristics (e.g., the query length as the number of words or characters). The assumption is that too long queries are not requested by many users, so very likely they will not appear in the future and are not worth caching. On the other hand, short queries are more popular and appear several times in the stream, so they are more likely to be requested while their results are still in the cache. The stateful features are based on historical statistics computed over the query log, such as the number of times the query has been already requested. Other more sophisticated admission policies are based on machine learning (e.g., regression models) that predict the next query request [30]. They rely on several features of queries (e.g., query length, presence of typos), index (e.g., length of posting lists of most common or uncommon terms), query and term frequencies as well as user's sessions (e.g., user logged in, clickthrough rate). The intent is to admit to the static cache only those queries with high expected frequency and to the dynamic cache the queries with high probability to be resubmitted. The authors tried different cache configurations and showed an improvement of 0.66% for static cache and of 0.47% for SDC when their admission policy is used. We did not implement this admission policy as we lack information for index and session features, but our improvement was higher even with the admission policy based on query features (e.g., [5]). Besides, in the next paragraph, we show hit rates of STD and SDC using an oracle that prevents singleton queries from entering the cache, and this can be seen as a performance upper bound for admission techniques.

Following [5], we implemented an admission strategy that accepts a query in the cache based on stateless and stateful features. In particular, the query is cached only if it satisfies the following conditions:

- it has been requested in the training period at least X times (stateful feature);
- the number of terms in the query is less than Y (stateless features);
- the number of characters is less than Z (stateless features);

For the experiments discussed in this paper, we set $X = 3$, $Y = 5$, and $Z = 20$. Similar results were achieved with other threshold values proposed in [5]. As explained before, for these experiments we decide to use the split $30\% - 70\%$ of data for a realistic simulation of larger caches.

The results of these experiments are reported in Table 5. As we can see, the admission policy improves the hit rates of both SDC and STD caches. Note that for all configurations of cache sizes and datasets, STD outperforms SDC.

Table 6: Caching with an admission policy that removes the polluting queries: gap between the best hit rates achieved with SDC and STD caches w.r.t. Bélády and each other, plus the gap reduction w.r.t. Bélády hit rate for the best hit rates achieved with SDC and STD caches.

| Cache Size | Bélády | Best SDC | Best STD | Gap SDC w.r.t. Bélády | Gap STD w.r.t. Bélády | Gap STD w.r.t. SDC | Gap Reduction |
|---|---|---|---|---|---|---|---|
| | | | AOL | | | | |
| 64K | 46.37% | 37.98% | 41.87% | 8.39% | 4.50% | 3.89% | 46.36% |
| 128K | 49.13% | 40.40% | 44.45% | 8.73% | 4.68% | 4.05% | 46.39% |
| 256K | 50.90% | 41.48% | 45.52% | 9.42% | 5.38% | 4.04% | 42.88% |
| 512K | 51.29% | 42.58% | 46.72% | 8.71% | 4.57% | 4.14% | 47.53% |
| 1024K | 51.30% | 43.91% | 48.02% | 7.39% | 3.28% | 4.11% | 55.61% |
| | | | MSN | | | | |
| 64K | 52.81% | 48.35% | 50.29% | 4.46% | 2.52% | 1.94% | 43.49% |
| 128K | 54.40% | 48.83% | 51.01% | 5.57% | 3.39% | 2.18% | 39.13% |
| 256K | 55.11% | 49.22% | 51.63% | 5.89% | 3.48% | 2.41% | 40.91% |
| 512K | 55.11% | 49.86% | 52.45% | 5.25% | 2.66% | 2.59% | 49.33% |
| 1024K | 55.11% | 50.84% | 53.27% | 4.27% | 1.84% | 2.43% | 56.90% |

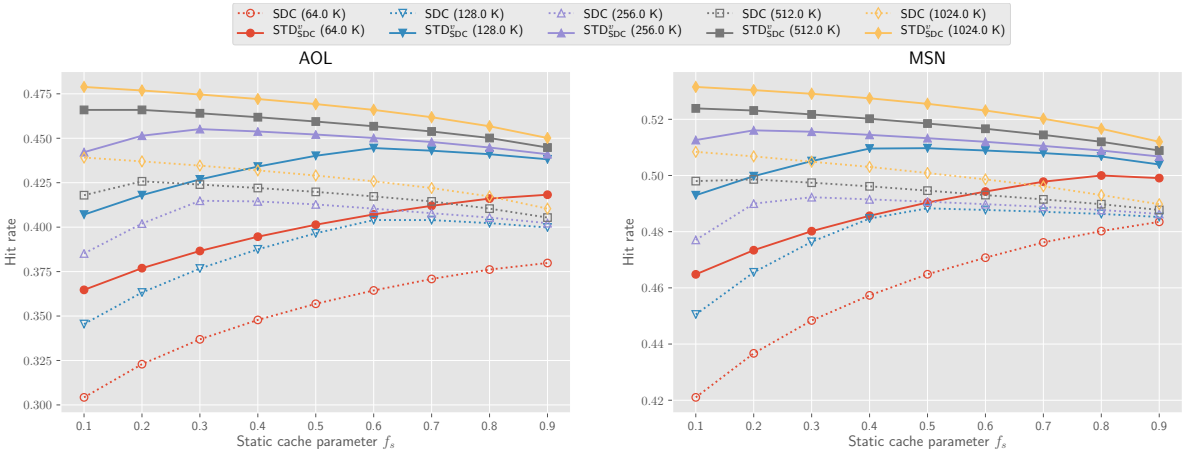



Figure 10: Hit rates of SDC and STD$^{v}_{SDC}$, using the admission policy that removes polluting queries, for different values of $N$ and of $f_s$, for the AOL (left) and MSN (right) query logs.

Table 6 reports the gaps between hit rates achieved by SDC and by the best STD configuration with respect to the hit rates of Bélády algorithm. In all these experiments, we restrict polluting queries from entering the cache. We also computed the gap between SDC and STD (see 7th column of the table) and the gap improvement w.r.t. Bélády (see 8th column of the table). We observe that STD continues to improve SDC with larger gaps compared to the caches without admission policy (see the 7th column of Table 4), reaching peaks of 4.14% for AOL and of 2.59% for MSN. The gap improvement w.r.t. Bélády is more than 55% for both datasets.

In Fig. 10, we show the hit rates for SDC and STD with the admission policy that does not allow caching of polluting queries. We remind the reader that the dashed lines are for SDC and solid lines for STD$^{v}_{SDC}$ (C2), and we show results for the static fraction $f_s$ varying from 0.1 to 0.9. Each pair of curves represent a different size of the cache. As we can see, for the same cache size and $f_s$ value, the STD$^{v}_{SDC}$ (C2) hit rate curve is always higher than the SDC curve.

*Caching Without Singleton Queries.* Since STD uses topical information to better cache query results, one could surmise that the STD cache capitalizes on non-singleton queries as they were observed in the training

Table 7: Hit rates achieved with SDC and STD caches with a perfect admission policy that does not allow the singleton queries. The best hit rates are highlighted in bold.

| **Cache Size** | SDC | $\text{STD}^{\text{f}}_{\text{LRU}}$ | $\text{STD}^{\text{v}}_{\text{LRU}}$ | $\text{STD}^{\text{v}}_{\text{SDC}}$ **(C1)** | $\text{STD}^{\text{v}}_{\text{SDC}}$ **(C2)** | $\text{T}^{\text{v}}_{\text{SDC}}$ |
|---|---|---|---|---|---|---|
| | | | AOL | | | |
| 64K | 57.27% | 59.88% | 60.47% | 58.47% | **60.48%** | 54.31% |
| 128K | 63.47% | 65.72% | 66.38 % | 65.01% | **66.39%** | 61.05% |
| 256K | 70.28% | 71.27% | 72.81% | 71.92% | **72.83%** | 68.03% |
| 512K | 77.80% | 76.32% | **79.87%** | 79.45% | 79.84% | 75.84% |
| 1024K | 86.66% | 82.27% | **87.73%** | 87.62% | 87.69% | 84.49% |
| | | | MSN | | | |
| 64K | 67.85% | 68.85% | 69.41% | 68.06% | **69.44%** | 63.54% |
| 128K | 71.64% | 72.83% | **73.24%** | 72.37% | 73.22% | 68.49% |
| 256K | 75.60% | 76.16% | **77.41%** | 76.80% | 77.35% | 73.14% |
| 512K | 80.89% | 79.54% | **82.80%** | 82.55% | 82.72% | 78.72% |
| 1024K | 87.18% | 83.70% | **87.90%** | 87.75% | 87.85% | 85.33% |

period and consequently have a topic assigned. To show that the advantage encountered with STD is irrespective of singleton queries, we experimented with an admission policy that restricts singleton queries from entering the cache.

A singleton query, by definition, is requested only once; hence, caching its results wastes space as the query will not be requested again. We implemented an oracle (i.e., an algorithm that knows the future) which, given as input the query stream, returns the singleton queries (i.e., queries that appear only once in the stream). This is obviously not feasible in practice but does provide an optimal environment to compare STD and SDC. Then, we ran our experiments not admitting in the caches these singleton queries.

As we did for the experiment on polluting queries, also for this experiment we decided to use the split $30\% - 70\%$, and we show the results in Table 7. Again, the STD approach always improves SDC in terms of hit rates. Overall, the hit rates obtained without singleton queries are much higher as compared to those of Table 2 since not caching singleton queries spares space for future worthy queries.

In Table 8 we report the gaps between the hit rates of SDC and best STD caches with respect to Bélády as well as the gap between SDC and best STD (see the 7th column of the table). We observe that the gaps are smaller compared to the gaps of caches without admission policies (see the 7th column of Table 4), especially for large caches. This is expected, as both caches are advantaged by removing singleton queries. The gap reduction is also smaller than the one observed in the other experiments as SDC and STD perform close to the optimum. Nevertheless, this experiment proved that STD still has higher hit rates as compared to SDC. These results show how STD improves SDC even in extreme cases where the caching approaches have the advantage of not storing singleton queries. Moreover, STD does not benefit by not storing the non-singleton queries more than SDC could do. Finally, this result is a good indication that SDC even with the best admission strategy (i.e., the oracle in our case) on the top of it cannot perform better than STD.

In Fig. 11 we show the hit rates for SDC and STD with the admission policy that does not allow caching of singleton queries. Also, for this comparison we can see that the performance of $\text{STD}^{\text{v}}_{\text{SDC}}$ (C2) are always better than SDC, although the gap is lower because removing all singleton requests is a huge advantage for both caching strategies.

## 6. Conclusions

We presented a novel cache model, Static-Topic-Dynamic (STD) cache, which leverages query topics to better utilize cache space, yielding improved hit rates. Compared to the traditional SDC cache, the STD cache stores queries belonging to a given topic in a dedicated portion of the cache where for each topic the

Table 8: Caching with an admission policy that removes the singleton queries: Gap between the best hit rates achieved with SDC and STD caches, plus the gap reduction w.r.t. Bélády hit rate for the best hit rates achieved with the admission policy.

| Cache Size | Bélády | Best SDC | Best STD | Gap SDC w.r.t. Bélády | Gap Best STD w.r.t. Bélády | Gap SDC and Best STD | Gap Reduction |
|---|---|---|---|---|---|---|---|
| | | | AOL | | | | |
| 64K | 68.76% | 57.27% | 60.48% | 11.49% | 8.28% | 3.21% | 27.93% |
| 128K | 75.04% | 63.47% | 66.39% | 11.57% | 8.65% | 2.92% | 25.23% |
| 256K | 81.09% | 70.28% | 72.83% | 10.81% | 8.26% | 2.55% | 23.59% |
| 512K | 85.26% | 77.80% | 79.87% | 7.46% | 5.39% | 2.07% | 27.74% |
| 1024K | 87.88% | 86.66% | 87.73% | 1.22% | 0.15% | 1.07% | 87.70% |
| | | | MSN | | | | |
| 64K | 75.10% | 67.85% | 69.44% | 7.25% | 5.66% | 1.59% | 21.93% |
| 128K | 79.69% | 71.64% | 73.24% | 8.05% | 6.45% | 1.60% | 19.87% |
| 256K | 83.76% | 75.60% | 77.41% | 8.16% | 6.35% | 1.81% | 22.18% |
| 512K | 85.21% | 80.89% | 82.80% | 4.32% | 2.41% | 1.91% | 44.21% |
| 1024K | 88.21% | 87.18% | 87.90% | 1.03% | 0.31% | 0.72% | 69.90% |

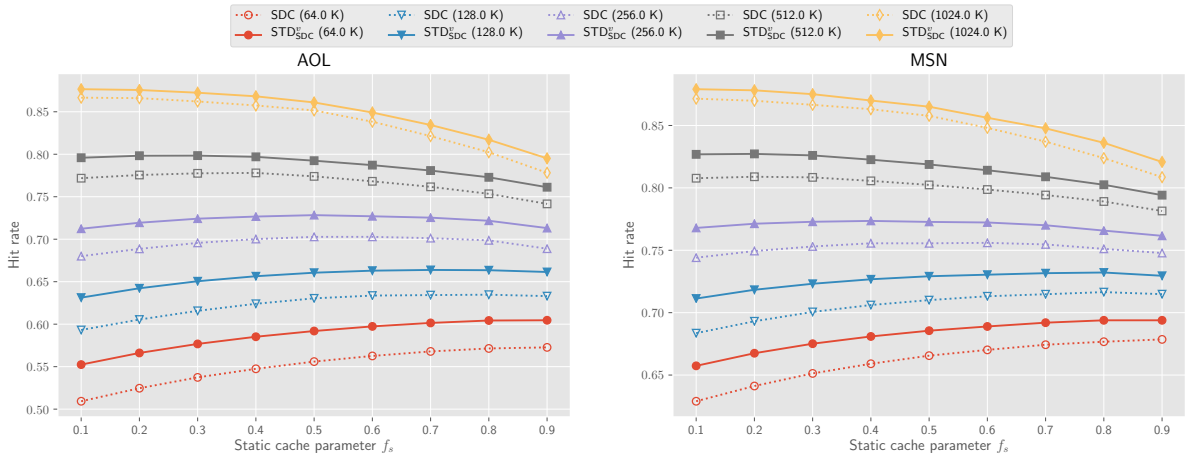



Figure 11: Hit rates of SDC and $\text{STD}^{\text{v}}_{\text{SDC}}$, using the admission policy that does not allow singleton queries, for different values of $N$ and of $f_s$, for the AOL (left) and MSN (right) query logs.

number of entries available is proportional to the topic popularity. This allows to capture queries that are frequently requested at large intervals of time and would be evicted in a cache only managed by the LRU policy. Extensive experiments conducted with two real-world query logs show that STD increases the cache hit rate by more than 3% over SDC. Such improvements result in a hit rate gap reduction w.r.t. Bélády's optimal caching policy [11] of up to ∼36% over SDC, depending on the query log and the total size of the cache. The improvement is higher when an admission policy for not storing polluting queries is employed.

It is worth noting that the greater hit rate achieved by the proposed query-result cache does not require specific investments by the search engine companies. The query topic classification service is in general already deployed for other purposes [10], while our caching solution is managed entirely by software and can be easily implemented and deployed in existing Web search systems.

In the future, we would like to investigate if and how query topic classification performance impacts the cache hit ratio. We assigned topics to queries by means of a LDA classification technique trained on a portion of the query logs. We believe that improving coverage and accuracy of query classification might be beneficial for the STD cache hit ratio. As future work, we plan to employ other topic-modeling techniques which are tailored for short text [44]. We would also like to use this cache in synergy with other caches, e.g., those storing posting lists of frequently requested terms.

## 7. Acknowledgements

We thank Roberto Trani (ISTI-CNR) for his help in measuring the query processing times. Also, we thank the anonymous reviewers who have provided constructive comments for improving our paper.

## References

[1] Sadiye Alici, Ismail Sengör Altingövde, Rifat Ozcan, Berkant Barla Cambazoglu, and Özgür Ulusoy. Adaptive time-to-live strategies for query result caching in web search engines. In *Proceedings of the 34th European Conference on Advances in Information Retrieval*, ECIR '12, pages 401–412, Berlin, Heidelberg, 2012. Springer-Verlag.

[2] Ismail Sengör Altingövde, Rifat Ozcan, Berkant Barla Cambazoglu, and Özgür Ulusoy. Second chance: a hybrid approach for dynamic result caching in search engines. In *Proceedings of the 33rd European Conference on Information Retrieval Research*, ECIR '11, pages 510–516, Berlin, Heidelberg, 2011. Springer-Verlag.

[3] Ismail Sengör Altingövde, Rifat Ozcan, and Özgür Ulusoy. A cost-aware strategy for query result caching in web search engines. In *Proceedings of the 31th European Conference on Information Retrieval Research*, ECIR '09, pages 628–636, Berlin, Heidelberg, 2009. Springer-Verlag.

[4] Aris Anagnostopoulos, Luca Becchetti, Ilaria Bordino, Stefano Leonardi, Ida Mele, and Piotr Sankowski. Stochastic query covering for fast approximate document retrieval. *ACM Transactions on Information Systems*, 33(3):11:1–11:35, 2015.

[5] Ricardo Baeza-Yates, Flavio Junqueira, Vassilis Plachouras, and Hans Friedrich Witschel. Admission policies for caches of search engine results. In Nivio Ziviani and Ricardo Baeza-Yates, editors, *String Processing and Information Retrieval*, pages 74–85, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[6] Ricardo Baeza-Yates and Felipe Saint-Jean. A three level search engine index based in query log distribution. In Mario A. Nascimento, Edleno S. de Moura, and Arlindo L. Oliveira, editors, *String Processing and Information Retrieval*, pages 56–65, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[7] Xiao Bai and Flavio P. Junqueira. Online result cache invalidation for real-time web search. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, pages 641–650, New York, NY, USA, 2012. ACM.

[8] Luiz André Barroso, Jeffrey Dean, and Urs Hölzle. Web search for a planet: The Google cluster architecture. *IEEE Micro*, 23(2):22–28, 2003.

[9] Steven M. Beitzel, Eric C. Jensen, Abdur Chowdhury, David Grossman, and Ophir Frieder. Hourly analysis of a very large topically categorized web query log. In *Proceedings of the 27th International Conference on Research and Development in Information Retrieval*, SIGIR '04, pages 321–328, New York, NY, USA, 2004. ACM.

[10] Steven M. Beitzel, Eric C. Jensen, Ophir Frieder, David Grossman, David D. Lewis, Abdur Chowdhury, and Aleksandr Kolcz. Automatic web query classification using labeled and unlabeled training data. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, pages 581–582, New York, NY, USA, 2005. ACM.

[11] Laszlo A. Bélády, Randolph A. Nelson, and Gerald S. Shedler. An anomaly in space-time characteristics of certain programs running in a paging machine. *Communications of the ACM*, 12(6):349–353, June 1969.

[12] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet Allocation. *Journal on Machine Learning Research*, 3:993–1022, 2003.

[13] Oisín Boydell and Barry Smyth. Capturing community search expertise for personalized web search using snippet-indexes. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, CIKM '06, pages 277–286, New York, NY, USA, 2006. ACM.

[14] Andrei Z. Broder, Marcus Fontoura, Evgeniy Gabrilovich, Amruta Joshi, Vanja Josifovski, and Tong Zhang. Robust classification of rare queries using web knowledge. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 231–238, New York, NY, USA, 2007. ACM.

[15] Berkant Barla Cambazoglu, Ismail Sengör Altingövde, Rifat Ozcan, and Özgür Ulusoy. Cache-based query processing for search engines. *ACM Transactions on the Web*, 6(4):14:1–14:24, 2012.

[16] Berkant Barla Cambazoglu and Ricardo A. Baeza-Yates. *Scalability Challenges in Web Search Engines*. Synthesis Lectures on Information Concepts, Retrieval, and Services. Morgan & Claypool Publishers, 2015.

[17] Matteo Catena, Craig Macdonald, and Nicola Tonellotto. Load-sensitive cpu power management for web search engines. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 751–754, New York, NY, USA, 2015. ACM.

[18] Matteo Catena and Nicola Tonellotto. Energy-efficient query processing in web search engines. *IEEE Transactions on Knowledge and Data Engineering*, 29(7):1412–1425, 2017.

[19] Diego Ceccarelli, Claudio Lucchese, Salvatore Orlando, Raffaele Perego, and Fabrizio Silvestri. Caching query-biased snippets for efficient retrieval. In *Proceedings of the 14th International Conference on Extending Database Technology, Uppsala, Sweden, March 21-24, 2011*, EDBT '11, pages 93–104, 2011.

[20] Nick Craswell, Rosie Jones, Georges Dupret, and Evelyne Viegas, editors. *Proceedings of the 2009 workshop on Web Search Click Data, WSCD@WSDM 2009, Barcelona, Spain, February 9, 2009*. ACM, 2009.

[21] Caio Moura Daoud, Edleno Silva de Moura, Andre Carvalho, Altigran Soares da Silva, David Fernandes, and Cristian Rossi. Fast top-k preserving query processing using two-tier indexes. *Information Processing & Management*, 52(5):855 – 872, 2016.

[22] 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.

[23] Ophir Frieder, Ida Mele, Raffaele Perego, and Nicola Tonellotto. Cache optimization via topics in web search engines, Dec. 10 2019. US Patent 10,503,792.

[24] Qingqing Gan and Torsten Suel. Improved techniques for result caching in web search engines. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 431–440, New York, NY, USA, 2009. ACM.

[25] Thomas L. Griffiths and Mark Steyvers. Finding scientific topics. *Proceedings of the National Academy of Sciences*, 101(suppl 1):5228–5235, 2004.

[26] Xuefeng He, Jun Yan, Jinwen Ma, Ning Liu, and Zheng Chen. Query topic detection for reformulation. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 1187–1188, New York, NY, USA, 2007. ACM.

[27] Bernard J. Jansen and Amanda Spink. How are we searching the World Wide Web? A comparison of nine search engine transaction logs. *Information Processing & Management*, 42(1):248 – 263, 2006. Formal Methods for Information Retrieval.

[28] Simon Jonassen, Berkant Barla Cambazoglu, and Fabrizio Silvestri. Prefetching query results and its impact on search engines. In *Proceedings of the 35th International ACM SIGIR conference on research and development in Information Retrieval, SIGIR '12, Portland, OR, USA, August 12-16, 2012*, pages 631–640, 2012.

[29] Enver Kayaaslan, Berkant Barla Cambazoglu, and Cevdet Aykanat. Document replication strategies for geographically distributed web search engines. *Information Processing & Management*, 49(1):51–66, January 2013.

[30] Tayfun Kucukyilmaz, Berkant Barla Cambazoglu, Cevdet Aykanat, and Ricardo Baeza-Yates. A machine learning approach for result caching in web search engines. *Information Processing & Management*, 53(4):834–850, July 2017.

[31] Ronny Lempel and Shlomo Moran. Predictive caching and prefetching of query results in search engines. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, pages 19–28, New York, NY, USA, 2003. ACM.

[32] Ximing Li, Ang Zhang, Changchun Li, Jihong Ouyang, and Yi Cai. Exploring coherent topics by topic modeling with term weighting. *Information Processing & Management*, 54(6):1345 – 1358, 2018.

[33] Fang Liu, Clement Yu, and Weiyi Meng. Personalized web search by mapping user queries to categories. In *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, CIKM '02, pages 558–565, New York, NY, USA, 2002. ACM.

[34] David Lo, Liqun Cheng, Rama Govindaraju, Luiz A. Barroso, and Christos Kozyrakis. Towards energy proportionality for large-scale latency-critical workloads. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 301–312, June 2014.

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

[36] Yue Lu, Qiaozhu Mei, and ChengXiang Zhai. Investigating Task Performance of Probabilistic Topic Models: an Empirical Study of PLSA and LDA. *Information Retrieval*, 14(2):178–203, 2011.

[37] Evangelos P. Markatos. On caching search engine query results. *Computer Communications*, 24(2):137–143, 2001.

[38] Ida Mele, Seyed Ali Bahrainian, and Fabio Crestani. Linking news across multiple streams for timeliness analysis. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017*, pages 767–776, 2017.

[39] Ida Mele, Seyed Ali Bahrainian, and Fabio Crestani. Event mining and timeliness analysis from heterogeneous news streams. *Inf. Process. Manage.*, 56(3):969–993, 2019.

[40] Rifat Ozcan, Ismail Sengör Altingövde, Berkant Barla Cambazoglu, Flavio P. Junqueira, and Özgür Ulusoy. A five-level static cache architecture for web search engines. *Information Processing and Management*, 48(5):828 – 840, 2012.

[41] Rifat Ozcan, Ismail Sengör Altingövde, Berkant Barla Cambazoglu, and Özgür Ulusoy. Second chance: A hybrid approach for dynamic result caching and prefetching in search engines. *ACM Transactions on the Web*, 8(1):3:1–3:22, 2013.

[42] Greg Pass, Abdur Chowdhury, and Cayley Torgeson. A picture of search. In *Proceedings of the 1st International Conference on Scalable Information Systems (InfoScale '06)*, New York, NY, USA, 2006. ACM.

[43] Vijay V. Raghavan and Hayri Sever. On the reuse of past optimal queries. In *Proceedings of the 18th International Conference on Research and Development in Information Retrieval*, SIGIR '95, pages 344–350, New York, NY, USA, 1995. ACM.

[44] Junaid Rashid, Syed Muhammad Adnan Shah, and Aun Irtaza. Fuzzy topic modeling approach for text mining over short text. *Information Processing & Management*, 56(6):102060, 2019.

[45] Patricia Correia Saraiva, Edleno Silva de Moura, Rodrigo C. Fonseca, Wagner Meira Jr., Berthier A. Ribeiro-Neto, and Nivio Ziviani. Rank-preserving two-level caching for scalable search engines. In *Proceedings of the 24th International Conference on Research and Development in Information Retrieval*, SIGIR '01, pages 51–58, New York, NY, USA, 2001. ACM.

[46] Fethi Burak Sazoglu, Berkant Barla Cambazoglu, Rifat Ozcan, Ismail Sengor Altingovde, and Özgür Ulusoy. A financial cost metric for result caching. In *Proceedings of the 36th International Conference on Research and Development in Information Retrieval*, SIGIR '13, pages 873–876, 2013.

[47] Craig Silverstein, Hannes Marais, Monika Henzinger, and Michael Moricz. Analysis of a very large web search engine query log. *SIGIR Forum*, 33(1):6–12, September 1999.

[48] Fabrizio Silvestri. Mining query logs: Turning search usage data into knowledge. *Foundations and Trends in Information Retrieval*, 4(1-2):1–174, 2009.

[49] Amanda Spink, Dietmar Wolfram, Major B. J. Jansen, and Tefko Saracevic. Searching the web: the public and their queries. *Journal of the American Society for Information Science and Technology*, 52(3):226–234, 2001.

[50] Amin Teymorian, Ophir Frieder, and Marcus A. Maloof. Rank-energy selective query forwarding for distributed search systems. In *Proceedings of the 22Nd ACM International Conference on Information & Knowledge Management*, CIKM '13, pages 389–398, New York, NY, USA, 2013. ACM.

[51] Gabriel Tolosa, Esteban Feuerstein, Luca Becchetti, and Alberto Marchetti-Spaccamela. Performance improvements for search systems using an integrated cache of lists + intersections. *Information Retrieval Journal*, 20(3):172–198, 2017.

[52] Nicola Tonellotto, Craig Macdonald, and Iadh Ounis. Efficient query processing for scalable web search. *Foundations and Trends in Information Retrieval*, 12(4–5):319–492, 2018.

[53] Yinglian Xie and David O'Hallaron. Locality in search engine queries and its implications for caching. In *Proceedings of the 21st Joint Conference of the IEEE Computer and Communications Societies (Infocom '02)*, pages 1238–1247. IEEE, 2002.

[54] Wanwan Zhou, Ruixuan Li, Xinhua Dong, Zhiyong Xu, and Weijun Xiao. An intersection cache based on frequent itemset mining in large scale search engines. In *Proceedings of the 3rd IEEE Workshop on Hot Topics in Web Systems and Technologies, HotWeb 2015, Washington, DC, USA, November 12-13, 2015*, pages 19–24, 2015.