Query Forwarding in Geographically Distributed Search Engines

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

Emre Varol
Bilkent University
Computer Engineering Dept.
Ankara, Turkey
evarol@cs.bilkent.edu.tr

Enver Kayaaslan
Bilkent University
Computer Engineering Dept.
Ankara, Turkey
enver@cs.bilkent.edu.tr

Cevdet Aykanat
Bilkent University
Computer Engineering Dept.
Ankara, Turkey
aykanat@cs.bilkent.edu.tr

Ricardo Baeza-Yates
Yahoo! Research
Barcelona, Spain
rbaeza@acm.org

ABSTRACT

Query forwarding is an important technique for preserving the result quality in distributed search engines where the index is geographically partitioned over multiple search sites. The key component in query forwarding is the thresholding algorithm by which the forwarding decisions are given. In this paper, we propose a linear-programming-based thresholding algorithm that significantly outperforms the current state-of-the-art in terms of achieved search efficiency values. Moreover, we evaluate a greedy heuristic for partial index replication and investigate the impact of result cache freshness on query forwarding performance. Finally, we present some optimizations that improve the performance further, under certain conditions. We evaluate the proposed techniques by simulations over a real-life setting, using a large query log and a document collection obtained from Yahoo!.

Categories and Subject Descriptors

H.3.3 [**Information Storage Systems**]: Information Retrieval Systems

General Terms

Algorithms, Design, Performance, Experimentation

Keywords

Search engines, distributed IR, query forwarding, optimization, linear programming, index replication, result caching

1. BACKGROUND

Commercial web search engines of the past relied on a single search site (data center), which processed queries issued from all around the world. This approach had the typical scalability problems in centralized architectures. Moreover, queries issued from distant locations suffered from poor response times as the network latency between the user and the site became an issue. For such queries, either the query processing times had to be shortened, thus degrading the result quality, or users experienced unreasonable response times, which had implications on user satisfaction [16].

At this point, replicating the data (i.e., the web collection and the inverted index built upon it) over multiple, geographically distant search sites emerged as a feasible solution. In this strategy, each geographical region is mapped to a nearby search site. A search site processes over its full web index only the queries originating from the regions assigned to itself[1]. Although this strategy reduces network latencies, the scalability still remains as an issue since the entire web index had to be maintained on all search sites and queries are evaluated over the full web index.

A strategy that contrasts replication is to partition the data disjointly and assign each site only the documents obtained (crawled) from its region [8]. In this strategy, local queries of a region are evaluated over the partial index in the corresponding search site. The underlying assumption here is that users are interested more in documents located in their own region and local documents are more relevant for queries originating from the same region. As queries are now evaluated over small subsets of the web index, gains are possible in query processing time and throughput [8], along with other gains, such as those in web crawling [9].

Unfortunately, although the assumption about having high relevance between the documents and queries of the same region is reasonable, this is not true for all queries as some queries target non-regional documents [4]. This implies that evaluation over a partitioned index will lead to inferior search quality as some relevant, non-local documents are not retrieved. The problem of accessing non-local documents has two immediate solutions: taking the data to where it is sought and/or taking the queries to what they seek. The first is an offline solution that requires partial replication of the popular documents in a region on some non-local search sites. The second is an online solution that requires selective

[1]Herein, we refer to such queries as local queries.

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

forwarding of queries between search sites to extend coverage of search results. Our focus in this paper is on the latter approach, but we briefly touch to the former as well.

Selective query forwarding works as follows. The local search site receives a query and makes a decision about the quality of the locally computed results (relative to globally computed results, which would have been obtained through evaluation over the full index). If it is predicted that the local ranking misses some documents that would have appeared in the global ranking, a forecast is made about which search sites might have those documents. The query is then forwarded to those sites for further processing over non-local indexes and more results are retrieved. Finally, non-local and local results are merged and returned to the user.

The predictions made may lead to false positives (the query is forwarded to a site with no useful results, thus degrading performance) as well as false negatives (the query is not forwarded to a site with useful results, thus degrading the search quality). In this paper, our focus is on query forwarding techniques that preserve the search quality, i.e., those with no false negatives. This requires correctly identifying all search sites that will contribute to the global top k. In the mean time, the number of contacted sites with no useful results should be kept minimal as this has an impact on the performance and overall costs of the search engine.

A recent study [3] has proposed a thresholding technique that preserves the search quality while reducing the number of sites contacted. In this work, we build upon that work and propose a new thresholding algorithm that substantially improves the algorithm in [3] in terms of efficiency. Our algorithm has an offline phase, in which past user query logs are used to create offline queries, for which the maximum possible score attainable on each site is precomputed and globally replicated. In the online phase, this information is used in a linear programming (LP) formulation to set upper bounds on possible non-local site scores for new queries. Forwarding decisions are given based on comparisons between these bounds and the kth top score on the local site.

The following are the contributions of this paper:

- We describe an LP-based thresholding algorithm that significantly outperforms the current state-of-the-art [3].
- We evaluate a heuristic for partial index replication.
- We investigate the impact of result caching and cache freshness on query forwarding performance.
- We present several optimizations that provide further performance improvements under certain conditions.

The rest of the paper is organized as follows. Section 2 presents the considered geographically distributed search engine architecture and the associated query forwarding problem. We describe the proposed thresholding algorithm in Section 3. Experimental framework is given in Section 4. Section 5 provides the performance results. Further optimizations are proposed and evaluated in Section 6. Related work is surveyed in Section 7. We conclude in Section 8.

2. QUERY FORWARDING PROBLEM

2.1 Architecture

We consider a distributed architecture with N geographically distant search sites, where each site is assigned a nearby geographical region and is responsible for crawling and indexing only the documents in its assigned region. That is, the global web index is disjointly (document-based) partitioned into N local indexes, and each local index is uniquely assigned to a search site. Also, each site is assigned the task of generating the top k results for queries issued from its own region and returning these results to its users.

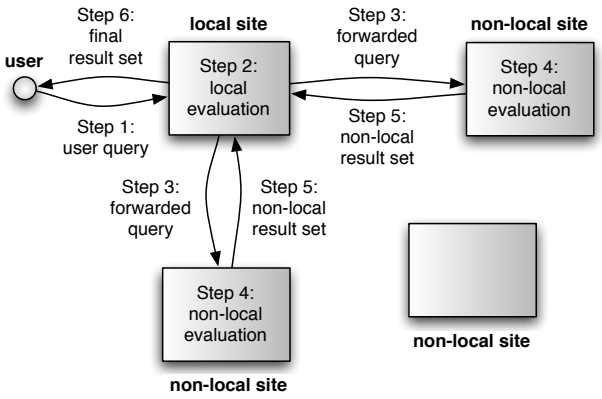

Figure 1: A geographically distributed search engine architecture with query forwarding.

A query is processed as follows (Fig. 1). The query is first issued to the local site, which evaluates the query over its partial index and computes a local top k result set. Then, a check is made to determine whether the global ranking over the full web index would bring results that are of higher quality than those in the local top k set. If it is guaranteed that the local top k set is identical to the global top k set, local results are immediately returned to the user. Otherwise, the local site identifies the non-local sites that store the documents that are missing in the local top k, but may appear in the global top k. The query is forwarded to those sites to retrieve missing results. When a non-local site receives a forwarded query, it processes the query over its own local index and generates a top k set. This result set is then transferred back to the local site, which forwarded the query. In the mean time, the local site waits for replies from all non-local sites that are contacted. Once all remote top k sets are received, they are merged[2] at the local site to generate a global top k set, which is returned to the user.

2.2 Problem

The problem in selective query forwarding is to decide which search sites are likely to contain relevant results for the query and if retrieving those results will improve the results of the local site. If a query is forwarded to a non-local site and none of the returned results get into the final result set, the non-local site becomes unnecessarily burdened. Moreover, a delay is introduced in the query response time. On the other hand, if a non-local site had documents that would have appeared in the global result set but the query is not forwarded to that site, those documents are missed in the final result set and hence the search quality degrades.

The difficulty in the query forwarding problem is in correctly identifying which queries need to be forwarded and to which sites. Herein, we focus on solutions that preserve the search quality, i.e., the final result set returned to the user is guaranteed to be identical to the global top k set computed over the full web index. Hence, our objective is to mini-

[2]Result sets are merged according to document scores. We assume that global collection statistics are available on all sites, and scores generated by different sites are compatible.

mize any form of redundancy and inefficiency incurred by forwarding decisions that do not improve the search quality.

2.3 Performance Metrics

Let Q_{L} and Q_{F} be the sets of locally processed and forwarded queries, respectively. Let F^q denote the set of non-local sites that query q is forwarded to. We employ two performance metrics: the fraction α of locally processed queries

$$\alpha = \frac{|Q_{\mathrm{L}}|}{|Q_{\mathrm{L}}| + |Q_{\mathrm{F}}|}, \tag{1}$$

and the average number β of non-local sites hit per query

$$\beta = \frac{\sum_{q \in Q_{\mathrm{L}} \cup Q_{\mathrm{F}}} |F^q|}{|Q_{\mathrm{L}}| + |Q_{\mathrm{F}}|}. \tag{2}$$

In addition to these metrics, we measure the average query response time and the average query processing workload (relative to query processing over the full index) of the search engine. Since all of our optimizations are quality-preserving, herein, we do not use a result quality metric (e.g., P@k).

Let q denote a query submitted by some user u^q to a local search site $\hat{S}^q$, and let $\tilde{S}_i \in F^q$, where $1 \leq i \leq |F^q|$. Also, let $\bar{I}$, $\hat{I}^q$, and $\tilde{I}_i$ denote the global index, the local index of $\hat{S}^q$, and the local index of $\tilde{S}_i$, respectively. If a query is processed locally, there are mainly two cost components in the query response time[3]. The first cost (steps 1 and 6 in Fig. 1) is the user-to-site network latency $\ell(u^q, \hat{S}^q)$, which is incurred while q is transferred from u^q to $\hat{S}^q$ and also while the final results are transferred from $\hat{S}^q$ to u^q. The second cost (step 2 in Fig. 1) is the computational cost $t(q, \hat{I}^q)$ of processing q over $\hat{I}^q$. The query response time then becomes

$$T_{\mathrm{L}}(q) = 2 \times \ell(u^q, \hat{S}^q) + t(q, \hat{I}^q). \tag{3}$$

If the query is forwarded, then there are two additional costs. The first cost (steps 3 and 5 in Fig. 1) is the site-to-site network latency $\ell(\hat{S}^q, \tilde{S}_i)$, which is incurred while q is transferred from $\hat{S}^q$ to $\tilde{S}_i$ and also while non-local results are transferred from $\tilde{S}_i$ to $\hat{S}^q$. The second cost (step 4 in Fig. 1) is the non-local query processing cost $t(q, \tilde{I}_i)$, i.e., the cost of processing q remotely on $\tilde{I}_i$. The response time becomes

$$T_{\mathrm{F}}(q) = T_{\mathrm{L}}(q) + \max_{\tilde{S}_i \in F^q} (2 \times \ell(\hat{S}^q, \tilde{S}_i) + t(q, \tilde{I}_i)). \tag{4}$$

In Eq. (4), we take the maximum of all remote response times since queries are transferred to non-local sites at the same instant and the highest remote response time determines the waiting time of the local site. We can now compute the average query response time T_{avg} as

$$T_{\mathrm{avg}} = \frac{\sum_{q \in Q_{\mathrm{L}}} T_{\mathrm{L}}(q) + \sum_{q \in Q_{\mathrm{F}}} T_{\mathrm{F}}(q)}{|Q_{\mathrm{L}}| + |Q_{\mathrm{F}}|}. \tag{5}$$

Let $W(q, I)$ represent the workload[4] incurred to the search engine when evaluating q over an index I. For a given query set Q, we compute the relative workload W_{rel} as the ratio of the total workload incurred in our architecture to the workload incurred by evaluation over the full index, i.e.,

$$W_{\mathrm{rel}} = \frac{\sum_{q \in Q} (W(q, \hat{I}^q) + \sum_{\tilde{S}_i \in F^q} W(q, \tilde{I}_i))}{\sum_{q \in Q} W(q, \bar{I})}. \tag{6}$$

[3]We assume that result merging as well as various other costs are negligible, as this is the case for low k and N values.

[4]Herein, we approximate the workload incurred by a query as the sum of inverted list lengths of all terms in the query. Interested readers may refer to [11] for other possibilities.

3. THRESHOLDING ALGORITHM

3.1 Preliminaries

We assume that queries are processed over the inverted index in the AND mode [15], which is often the case in practice. In this mode, only the documents that contain all query terms are retrieved. The score of a document is computed simply by summing the term scores, indicating the relevance of the term to the document (e.g., BM25)[5]. Optionally, document-specific scores (e.g., PageRank) may be added to the final score. The technique proposed herein is applicable only to the former type of scoring. Extending it to cover the latter type requires further research.

As our aim is to preserve the search quality of a centralized architecture, a query q should be forwarded to any non-local site that would have at least one result in the global top k set. A non-local site $\tilde{S}$ can contribute to this set only if the top score $s(q, 1, \tilde{S})$ it computes for q is larger than the kth score $s(q, k, \hat{S})$ that local site $\hat{S}$ computes[6]. Obviously, it is not possible to know $s(q, 1, \tilde{S})$ before evaluating q on $\tilde{S}$. A simple but effective technique [3] for deciding whether q should be forwarded to $\tilde{S}$ is based on computing an upper-bound $m(q, \tilde{S})$ for $s(q, 1, \tilde{S})$ and comparing this bound against $s(q, k, \hat{S})$. If $m(q, \tilde{S}) \leq s(q, k, \hat{S})$ holds, it is guaranteed that $\tilde{S}$ has no better documents than those in $\hat{S}$ and there is no need to forward q to $\tilde{S}$. Otherwise, $\tilde{S}$ may have better documents, and q has to be forwarded to $\tilde{S}$. As the gap between $m(q, \tilde{S})$ and $s(q, 1, \tilde{S})$ increases, the query is more likely to be forwarded to a site with no useful documents. Hence, the objective in this thresholding technique is to compute the $m(q, \tilde{S})$ value as tight as possible, i.e., this bound should be as close as possible[7] to $s(q, 1, \tilde{S})$.

3.2 LP Formulation

Assume that we have the precomputed $s(q'_i, 1, \tilde{S})$ value for every query q'_i in a set $Q' = \{q'_1, \ldots, q'_m\}$ of m offline queries. Each $q'_i = \{t^i_1, \ldots, t^i_{n_i}\}$ is composed of n_i unique terms. We are given an online query $q = \{t_1, \ldots, t_n\}$ at local site $\hat{S}$ with a kth local score of $s(q, k, \hat{S})$. Given these, we formulate the problem of computing a tight $m(q, \tilde{S})$ value as a linear programming (LP) problem as follows. We first introduce a real-valued variable x_j for each term $t_j \in q$. We then find every offline query $q' \in Q'$ such that q' is a proper subset of q, i.e., $q' \subset q$. For every such q', we introduce an inequality

$$\sum_{t_j \in q'} x_j \leq s(q', 1, \tilde{S}), \ \forall q' \text{ s.t. } q' \in Q' \text{ and } q' \subset q, \tag{7}$$

which always holds. We also have the set of inequalities

$$x_j \geq 0, \ \forall t_j \text{ s.t. } t_j \in q, \tag{8}$$

which guarantee that the top scores for single-term queries (i.e., query terms) are always non-negative. After this setting, the thresholding problem reduces to finding

$$m(q, \tilde{S}) = \max \sum_{t_j \in q} x_j \tag{9}$$

subject to the linear constraints given in Eqs. 7 and 8 via linear programming. In practice, there exist well-known, efficient LP solvers for this and similar problems.

[5]Refer to Section 6.1 in [3] for more background on scoring.

[6]We omit superscripts on symbols for better readability.

[7]However, the inequality $m(q, \tilde{S}) \geq s(q, 1, \tilde{S})$ always holds.

We now illustrate the formulation by an example. Let $q = \{t_1, t_2, t_3, t_4\}$ and $Q' = \{q'_1, q'_2, \ldots, q'_7\}$ with $q'_1 = \{t_1\}$, $q'_2 = \{t_2\}$, $q'_3 = \{t_3\}$, $q'_4 = \{t_4\}$, $q'_5 = \{t_1, t_2\}$, $q'_6 = \{t_2, t_3\}$, and $q'_7 = \{t_2, t_3, t_4\}$. Assume that precomputed top scores are $s(q'_1, 1, \tilde{S}) = 9.7$, $s(q'_2, 1, \tilde{S}) = 8.1$, $s(q'_3, 1, \tilde{S}) = 3.2$, $s(q'_4, 1, \tilde{S}) = 4.9$, $s(q'_5, 1, \tilde{S}) = 4.2$, $s(q'_6, 1, \tilde{S}) = 4.7$, and $s(q'_7, 1, \tilde{S}) = 5.1$. These lead to the following set of inequalities

$$x_1 \leq 9.7; \quad x_2 \leq 8.1; \quad x_3 \leq 3.2; \quad x_4 \leq 4.9;$$
$$x_1 + x_2 \leq 4.2; \quad x_2 + x_3 \leq 4.7; \quad x_2 + x_3 + x_4 \leq 5.1;$$
$$x_1 \geq 0; \quad x_2 \geq 0; \quad x_3 \geq 0; \quad x_4 \geq 0.$$

The maximum score satisfying these constraints is found as $m(q, \tilde{S}) = 9.3$ ($x_1 = 4.2$, $x_2 = 0$, $x_3 = 0.2$, $x_4 = 4.9$).

3.3 Query Forwarding Algorithm

The forwarding algorithm contains an offline and an online phase. In the offline phase, offline queries are generated first. This can be done in different ways (see Section 4.4), e.g., synthetically by combining popular terms in the document collection or by extracting popular queries in past user query logs. Assuming such a set is available, the top scores are then computed for every query in this set over all local indexes. The computed values are replicated on all sites.

In the online phase, a query q is processed as follows. First, we evaluate q locally on $\hat{S}$ and record $s(q, k, \hat{S})$. If a query term $t_j \in q$ does not appear in any of the offline queries, q is forwarded to every non-local site (case `F-MissingInfo`[8]) as it is impossible to compute any score bounds. For a non-local site $\tilde{S}$, if there is a subquery $q' \subset q$ for which $m(q, \tilde{S}) = 0$, q is not forwarded to $\tilde{S}$ (case `L-ZeroThreshold`). Finally, for each non-local site $\tilde{S}$ for which no decision is yet made, we separately solve the LP formulation of the previous section. If $m(q, \tilde{S}) > s(q, k, \hat{S})$ holds, q is forwarded to $\tilde{S}$ (case `F-HighLPBound`); otherwise, it is not (case `L-LowLPBound`). We note that, in our LP formulation, it is possible to capture the `F-MissingInfo` case simply by introducing an equation $x_j \leq \infty$ for every term $t_j \in q$ such that $t_j \notin q'$ for $\forall q' \in Q'$.

4. EXPERIMENTAL FRAMEWORK

4.1 Setup

We simulate a geographically distributed search engine architecture using two different setups. The first setup, referred to as `Europe`, consists of five search sites, located in Berlin (Germany), Madrid (Spain), Paris (France), Rome (Italy), and London (UK). The second setup, referred to as `World`, contains five relatively distant search sites, located in Canberra (Australia), Brazil (Brazil), Ottawa (Canada), Berlin (Germany), and Mexico City (Mexico). These two setups represent search architectures where the network latencies between the sites are low and high, respectively.

For both setups, we approximate the site-to-site network latency between any two sites by taking into account the speed of light on copper wire (200,000 km/s) and the bird-fly distances between the cities that the sites are located. To approximate the user-to-site latency between a site and its users, we take an average over the latencies between the capital city where the site is located and the most populated five cities in the respective country. Computing latencies this way is reasonable as latencies are known to correlate well with geographical distance [12], and our data transfer costs are negligible as transferred result sets are very small. However, this approach ignores queuing delays and the fact that network connections are not necessarily on straight lines. Therefore, using several, geographically distant computers, we measured real network latencies and obtained a mapping from predicted latencies to actual values through regression (Fig. 2). All predicted values are converted to final, more accurate latency values via this mapping. Table 1 displays some statistics about the latency values used in our setups.

[8]We will use these labels later while discussing Fig. 10.

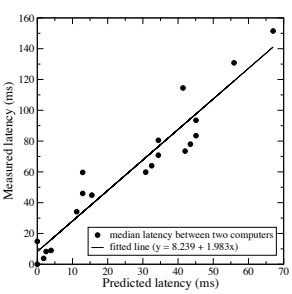

Figure 2: Predicted versus measured network latencies.

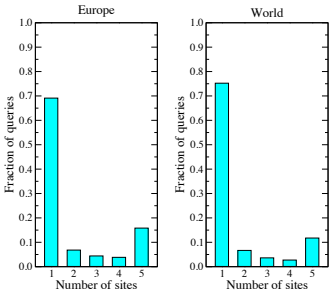

Figure 3: Fraction of queries that appear on n distinct search sites.

Table 1: Network latency statistics (in ms)

	Site-to-site latencies			User-to-site latencies		
	Min.	Avg.	Max.	Min.	Avg.	Max.
`Europe`	12.0	19.6	26.9	10.6	11.5	12.6
`World`	43.9	109.8	172.8	9.0	16.3	23.9

4.2 Dataset

As the global document collection, we use a large crawl of the Web (about 200 million documents). This collection is obtained through various cleansing and filtering steps. Hence, it is high-quality and its documents have high potential to appear in real-life search results. Then, using a proprietary classifier[9], a home country is predicted for every document, and disjoint subsets of documents are assigned to search sites (some documents are not assigned to any site). Finally, separate indexes are built on each subcollection.

For each site, we extract consecutive queries (about 19 million queries in total) from the query logs of the Yahoo! web search engine. Queries are passed through cleansing steps, such as case-folding, stop-word elimination, term uniquing, and reordering of query terms in alphabetical order. We omit queries issued by clicking on the next link and use only first page requests[10]. The query set of each site is separately sorted in increasing order of arrival times. The last quarter of each query set is used in the online phase. The rest are used in the offline phase. In our sample log, most queries are regional and occur in one site (Fig. 3).

4.3 Simulations

We compute thresholds and local top k results using a modified version of Terrier. In simulations, we assume that each search cluster node builds an index on three million documents. The total number of processors available to the overall search system is determined accordingly. Each site is assigned a number of processors proportional to its index size. Therefore, query processing times are comparable

[9]This is a production-level classifier that uses features such as language, IP, and domain to identify documents' regions.

[10]Next page requests may be handled by prefetching of result pages [14]. This is beyond the scope of this paper.

for search sites[11]. Query evaluation is simulated via a detailed simulator, which computes a separate response time for each query, using Eqs. (3) and (4). In query processing, we assume a processing cost of 200ns per posting. This is an average value obtained from Terrier, but we observed it to correlate well with real search engine timings. We also assume a 20ms overhead for preprocessing, per query.

4.4 Offline Query Generation

Our LP-based solution is applicable to online queries of arbitrary length and is especially suitable for long queries. However, our query set is composed of web queries, which are very short in nature. Using long queries in the offline query set does not bring much additional performance benefit[12]. Therefore, in our offline query set, we consider only single- and two-term queries[13]. This approach also reduces the storage requirement for the precomputed scores and their offline computation cost. Moreover, if we assume that an offline query can be accessed in $O(1)$-time using a hash table, the computational cost of accessing offline queries that are proper subsets of the online query becomes much lower as this can be done in $O(|q|^2)$-time instead of $O(2^{|q|})$-time.

In this work, we generate two offline query sets, referred to as `D1` and `D2`. `D1` contains all terms (i.e., queries of length one) in the vocabulary of the collection. `D2` contains all possible pairs of vocabulary terms (i.e., queries of length two). Obviously, the latter may not be feasible in a practical setting, but we still prefer to experiment with this set to observe the degree of benefit that our thresholding algorithm may provide. We also generate two more query sets, `Q1` and `Q2`, using the query log. `Q1` contains, as an offline query, all the terms in the vocabulary of the query log. `Q2` contains subqueries of length two in each individual query in the log (but, not across the entire vocabulary as we do for `D2`).

For performance evaluation, we use selected combinations (unions) of the above-mentioned sets: `Q1`, `D1`, `Q1-Q2`, `D1-Q2`, and `D1-D2`. We omit combinations `Q2`, `D2`, `Q1-D1`, and `Q1-D2` as they are less meaningful or useful. In our experiments, we use the `D1` set as our baseline as this is identical to the technique discussed in [3] (see the discussion in Section 7). To measure the best possible performance, we also consider an `Oracle` algorithm that has no false positives, i.e., it forwards a query to only the non-local sites with positive contribution to the final result set. Due to space limitations, occasionally, we display the results for only a single setup (often, `Europe`).

5. PERFORMANCE

5.1 Effect of Offline Query Set

Fig. 4 shows the fraction of locally processed queries for different offline query sets, as k varies. It is interesting to observe that the `Q1-Q2` set outperforms the baseline (`D1`) although it has fewer queries. When the baseline is combined with the term pairs extracted from the query log (i.e., `D1-Q2`), for $k = 10$, about 9.1% more queries are processed locally (10.2% for the `World` setup). The impractical `D1-D2` set performs quite close to the `Oracle` algorithm, which processes about 40% of the queries locally, for $k = 10$.

[11]We assume that indexes are entirely kept in main memory.
[12]A similar issue is mentioned before in the context of caching intersections of posting lists [15].
[13]We implemented an LP solver, tailored to our purposes.

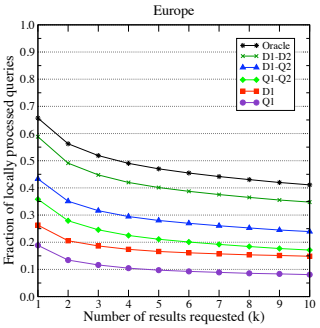

Figure 4: Fraction of locally processed queries.

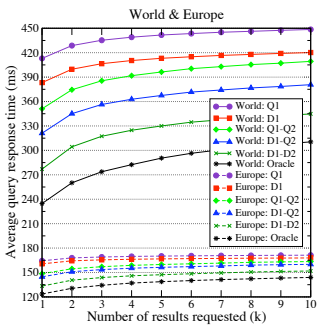

Figure 5: Average query response times.

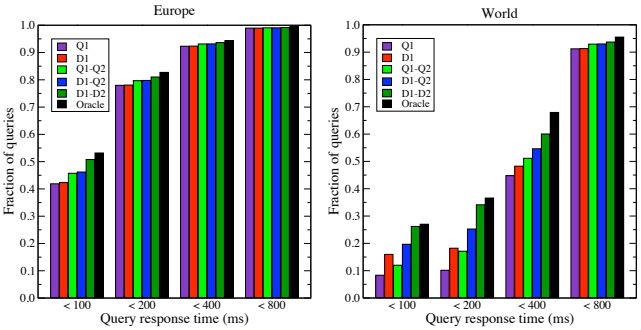

Figure 6: Fraction of queries that are answered under a certain response time (for $k = 10$).

The increase in the fraction of locally processed queries leads to a reduction in average query response times (Fig. 5). However, the distribution of response times is also important, i.e., we should check what fraction of queries can be processed under a given threshold time. It is empirically shown that after a certain response time threshold, users become frustrated and URL click-through rates go down, leading to financial losses for the search engine [16]. It is also shown that, given additional time for query processing, it is possible to improve the quality of search results [8]. In our simulations, we observe that the response time is a more critical issue for the `World` setup than `Europe` (Fig. 6). For `Europe`, only less than 10% of the queries cannot be answered under 400ms, whereas this rate varies in the 40%–55% range for the `World` setup, depending on the offline query set used.

According to Fig. 7, our technique achieves considerable reduction in the average number of non-local sites contacted. Additionally, in Fig. 8, we show the average number of sites that are active in processing a query. The second excludes any site that does not process the query on its local index. This may happen due to absence of a query term in the site's index, which explains the overlap of curves in Fig. 8 for `Q1` and `D1` as well as `Q1-Q2` and `D1-Q2`. Fig. 9 shows the average relative workload values as computed by Eq. (6). We observe that, for $k = 10$, there is about 16% reduction in the workload when queries are evaluated over our architecture (assuming `D1-Q2`) relative to query evaluation over the full index (this goes up to 20% for the `World` setup).

Fig. 10 shows the average outcome of a forwarding decision for a (query, site) pair (recall the discussion in Section 3.3). In the figure, we observe the following: Since `Q1` and `Q1-Q2` sets miss some terms in the collection vocabulary, about 10% of test queries had to be forwarded to all non-local sites (case `F-MissingInfo`). Most of the improvement

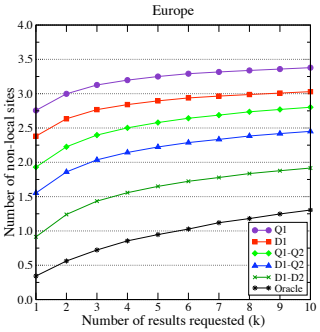

Figure 7: Average number of non-local sites a query is forwarded to.

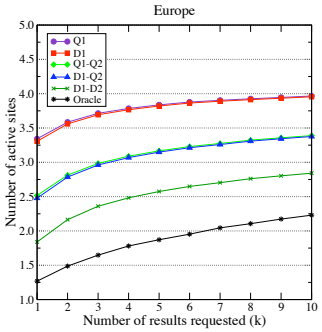

Figure 8: Average number of active sites (search backends) per query.

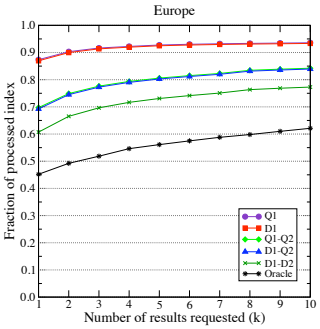

Figure 9: Fraction of the evaluated index relative to the global index.

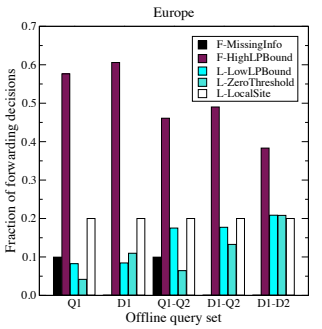

Figure 10: Dissection of forwarding decisions into various possible cases.

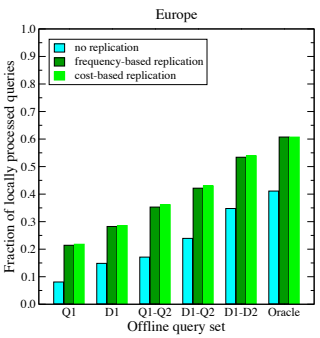

Figure 11: Impact of replication on locally processed query rate.

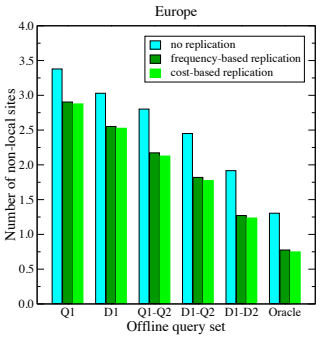

Figure 12: Impact of replication on non-local sites per query.

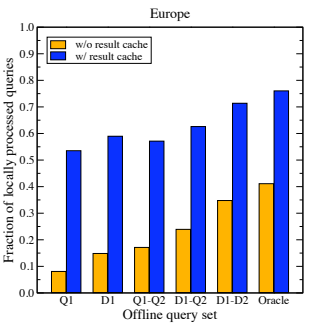

Figure 13: Impact of result caching on locally processed query rate.

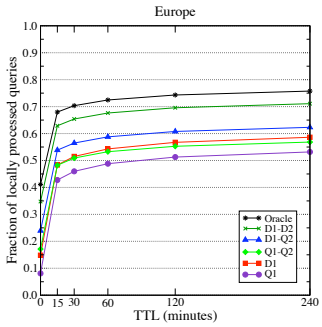

Figure 14: Impact of TTL on locally processed query rate.

over the baseline is due to the proposed LP solution (cases `F-HighLPBound` and `L-LowLPBound`), which uses term pairs (e.g., see `D1` versus `D1-Q2`). The fraction of `L-ZeroThreshold` cases correlates with the size of the offline query set used.

5.2 Effect of Partial Index Replication

Replicating globally popular documents on all sites leads to a high reduction in the forwarded query count [3]. The algorithm in [3] (herein, we call it `R-freq`) sorts the documents globally in decreasing number of occurrences in the top 200 results of training queries. A certain fraction of the most frequent documents are then replicated and indexed on all sites. This type of replication has two benefits for thresholding algorithms: local kth scores get higher as local sites have more documents, and thresholds computed for offline queries on non-local sites get lower as they are now computed over fewer documents. Both imply less forwarding.

A possible improvement over `R-freq` is to incorporate the storage cost incurred on the index due to replication of the document[14]. This is a greedy algorithm (`R-cost`) that tries to optimize per-byte benefit at any step by prioritizing documents according to the ratio of their occurrence frequencies and storage costs. In related experiments, we compute the occurrence frequencies using the top 10 results of training queries and then replicate on all sites 0.5% of documents with the highest benefit. According to Figs. 11 and 12, surprisingly, the improvement achieved by `R-cost` over `R-freq` is minor (0.3%–0.9% increase in the rate of locally processed queries and 0.9%–2.8% decrease in the number of non-local sites contacted per query). This is mainly because `R-cost` fills the given replication budget with small documents that have low past occurrence frequencies. Although past occurrences correlate well with future occurrences at high frequency values, there is little correlation at low occurrence frequencies[15]. This limits the performance of `R-cost`.

[14]We estimate this cost by document's unique term count.

5.3 Effect of Result Caching and TTL

Search engines cache the results of frequent and/or recent queries to reduce the query workload on backend clusters and improve query response times [2]. Queries that result in a hit in the cache are served by the cache. In our context, result caching has a significant impact on the number of forwarded queries. With result caching, the fraction of queries that can be locally processed increases by 35%–45%, depending on the offline query set used (Fig. 13). We also observe that more informative query sets receive a lower benefit. This is because, under result caching, only the queries that are seen for the first time (i.e., compulsory cache misses) are subject to forwarding. Most cache misses are tail queries, which are long. As we will see in Section 6.3, long queries are much less likely to be forwarded, and hence having more information in bound computations becomes less important.

The above discussion holds for search engines with indexes that are periodically rebuilt (e.g., once a week). If, however, there are incremental updates on the index, result cache entries may become stale. In practice, a quick solution is to associate a fixed time-to-live (TTL) value t with every cache entry [7]. A cache entry that is not refreshed for at

[15]An interesting discussion and possible solutions are available in the context of result caching [11]. But, application of those techniques are beyond the scope of our paper.

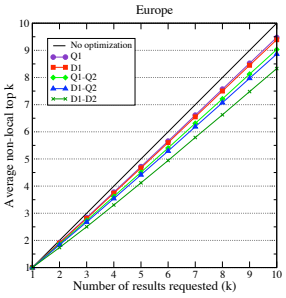

Figure 15: Number of local versus non-local results requested.

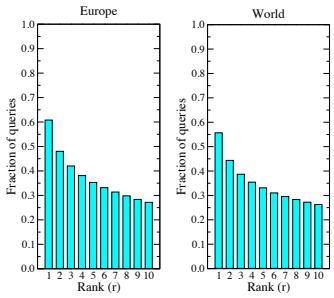

Figure 16: Fraction of queries where all results up to the rth rank are local.

Table 2: Fraction of queries according to length

	Query length				
Setup	1	2	3	4	>4
`Europe`	0.330	0.365	0.194	0.074	0.038
`World`	0.288	0.369	0.211	0.085	0.046

least t units of time is said to be expired, and any request for the entry is treated as a cache miss. In our context, such requests are subject to forwarding. According to Fig. 14, the performance saturates very quickly with increasing TTL[16], due to the power-law distribution of query frequencies [2].

6. FURTHER OPTIMIZATIONS

We now describe various techniques that improve performance under certain conditions. Some of the optimizations in distributed IR are also applicable to our setting, e.g., non-local computations can be early terminated simply by transferring $s(q,k,\hat{S})$ values together with the query. However, here, we skip such techniques and focus on those that are more meaningful in a geographically distributed setting.

6.1 Non-local Top k Optimization

If a query is forwarded to a non-local site, the top k results are requested. However, we note that it suffices to request $k-r$ results, where r is the lowest rank such that $m(q,\tilde{S}) < s(q,r,\hat{S})$ holds. Hence, for some queries, it is possible to request fewer documents and reduce the number of remotely computed snippets, in addition to other savings in score computations. For $k=10$, the saving is in the 5.3%–16.7% range, depending on the offline query set (Fig. 15).

6.2 Early Result Presentation

Search results are typically displayed in pages, containing 10 results. An interesting optimization is to show the user the local site's search results without waiting for replies of non-local sites. If it later turns out that $s(q,k,\hat{S}) \geq s(q,1,\tilde{S}_i)$ for every non-local site $\tilde{S}_i$, the query becomes served at the speed of a local query. Otherwise, non-local results are merged as a background job and the user's screen is refreshed with the correct results[17]. In our case, for about a quarter of queries, all top 10 results come from the local site. For the top result, this is so for more than half of queries (Fig. 16).

[16]We use small TTL values that are suitable to our sample query set. In practice, the TTL values are around a day [7].

[17]Some vertical search sites use similar optimizations (e.g., `http://www.kayak.com`). The impact of this kind of result presentation on user satisfaction is open to investigation.

Table 3: Average saving (in ms) in query response saving by early query forwarding

	Offline query set				
Setup	`Q1`	`D1`	`Q1-Q2`	`D1-Q2`	`D1-D2`
`Europe`	12.1	13.4	12.1	13.4	13.4
`World`	15.7	18.1	15.7	18.1	18.1

Table 4: Query response time (in ms) as the number of non-local sites a query is forwarded to varies

	Number of non-local sites				
Setup	0	1	2	3	4
`Europe`	87.3	152.0	157.1	161.5	190.2
`World`	121.4	376.6	435.1	484.5	524.1

6.3 Early Query Forwarding

We note that if $m(q,\tilde{S}) > \sum_{t\in q} s(t, \lfloor (k-1)/|q| \rfloor + 1, \hat{S})$ holds[18], q can be immediately forwarded to $\tilde{S}$ without waiting for completion of the local evaluation, which determines $s(q,k,\hat{S})$. This way, it becomes possible to overlap local query evaluation with network transfer. This approach requires precomputing and storing $s(t, \lfloor (k-1)/|q| \rfloor + 1, \hat{S})$ values for all terms in the collection vocabulary. Since the stored value depends on query length, covering all query lengths (assuming k is fixed to 10) requires storing all $s(t,r,\hat{S})$ values for $r \in \{1,2,3,4,5,k\}$. If long queries (according to Table 2, less than 15% of queries have more than 3 terms) are ignored, it suffices to store the scores only for $r \in \{4,5,k\}$. Herein, we only store $s(t,k,\hat{S})$ values and observe the impact on queries with a single term (about one-fifth of forwarded queries have one term, as seen in Fig. 17). For affected queries, the response time saving is up to 18ms (Table 3).

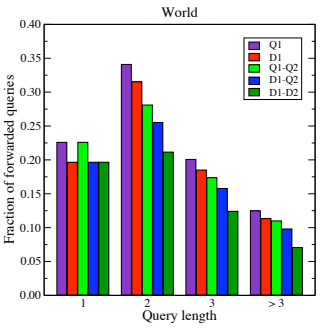

Figure 17: Fraction of forwarded queries as query length varies.

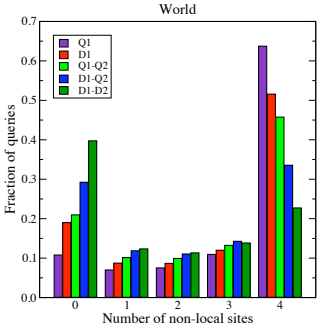

Figure 18: Fraction of queries forwarded to n non-local sites.

6.4 Remote Result Preparation

If the query is forwarded to only a single non-local site, the final results can be created and returned to the user by the non-local site as there are only two sets to be merged. This approach requires transferring the local top k result set together with the query, but may reduce the overall network latency in returning results to the user. Table 4 shows that there is a correlation between the average query response time and the number of non-local sites a query is forwarded to. The gap between the response time of local and forwarded queries is clearly seen. Our optimization, however, is applicable to a limited set of queries. According

[18]The right-hand side is an upper bound on $s(q,k,\hat{S})$.

Table 5: Average saving (in ms) in query response time by remote result preparation

	Offline query set				
Setup	Q1	D1	Q1-Q2	D1-Q2	D1-D2
Europe	10.8	10.8	10.4	10.4	10.4
World	16.6	16.6	16.2	16.2	16.2

to Fig. 18, about only 10% of our queries are forwarded to a single non-local site. For such queries, the saving in query response time is between 10ms and 16ms (Table 5).

7. RELATED WORK

Although there is much research on distributed IR [1], little research is done on multi-site, distributed search engines [3, 8]. Cambazoglu et al. present cost models and results, showing the potential of multi-site architectures for efficiency and relevance improvements [8]. Baeza-Yates et al. develop analytical models to compare operational costs of multi-site search systems against centralized systems [3].

The work in [3] is the closest to ours in that it also proposes an algorithm that tries to increase the number of locally processed queries by a thresholding technique, based on precomputation of maximum score contributions for all terms in the global vocabulary and replicating this information on all search sites. This way, it becomes possible to locally compute the maximum score a document can get on a non-local site, simply summing the maximum possible scores for query terms without evaluating the query remotely at all. It turns out that this technique is a limited case of our general solution (the same as using D1 in our setting).

We note that the query forwarding problem we deal with is somewhat different than the collection selection problem in federated IR [6]. In our system, queries are evaluated over a local index and some of them are forwarded. In federated IR, all queries are forwarded without any evaluation on a local index. We further note that P2P search systems [17] are also very different due to existence of a very high number of peers, their volatile nature, and limited availability. Baeza-Yates et al. [5] describe an architecture in which the global index is split into two tiers. In this architecture, queries are evaluated on one or two tiers, based on the decision of a machine-learned corpus predictor. In that architecture, documents are split into tiers by their importance or location.

Finally, Das et al. employ an LP-based solution in the context of databases for top k computation using materialized views [10]. Kumar et al. apply a similar technique to generalize top k thresholding algorithms by using precomputed intersections of posting lists [13]. To our knowledge, there is no work applying a similar technique in our context.

8. CONCLUSIONS

We showed that the fraction of locally processed queries in a multi-site search engine can be significantly increased by using an LP-based thresholding technique, and results are further improved by caching and replication. There are three research directions surfaced by our work. First, the applicability of our techniques to other problems (e.g., tiering [5]) should be investigated. Second, a trade-off analysis is needed between forwarding performance and offline query generation and storage overheads. Finally, the freshness of precomputed score thresholds needs further research.

9. ACKNOWLEDGEMENT

This publication is based on work performed in the framework of the Project COAST-ICT-248036, funded by the European Community, and partially supported by the Scientific and Technological Research Council of Turkey under grant EEEAG-109E019 and COST Action IC080 ComplexHPC.

10. REFERENCES

[1] R. Baeza-Yates, C. Castillo, F. Junqueira, V. Plachouras, and F. Silvestri. Challenges on distributed web retrieval. In *23rd Int'l Conf. on Data Engineering*, pages 6–20, 2007.

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

[3] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Plachouras, and L. Telloli. On the feasibility of multi-site web search engines. In *Proc. 18th ACM Conf. on Information and Knowledge Management*, pages 425–434, 2009 (best paper).

[4] R. Baeza-Yates, C. Middleton, and C. Castillo. The geographical life of search. In *Proc. 2009 IEEE/WIC/ACM Int'l Joint Conf. on Web Intelligence and Intelligent Agent Technology*, pages 252–259, 2009.

[5] R. Baeza-Yates, V. Murdock, and C. Hauff. Efficiency trade-offs in two-tier web search systems. In *Proc. 32nd Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 163–170, 2009.

[6] J. P. Callan, Z. Lu, and W. B. Croft. Searching distributed collections with inference networks. In *Proc. 18th Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 21–28, 1995.

[7] B. B. Cambazoglu, F. P. Junqueira, V. Plachouras, S. Banachowski, B. Cui, S. Lim, and B. Bridge. A refreshing perspective of search engine caching. In *19th Int'l Conf. on World Wide Web*, 2010 (accepted).

[8] B. B. Cambazoglu, V. Plachouras, and R. Baeza-Yates. Quantifying performance and quality gains in distributed web search engines. In *Proc. 32nd Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 411–418, 2009.

[9] B. B. Cambazoglu, V. Plachouras, F. Junqueira, and L. Telloli. On the feasibility of geographically distributed web crawling. In *Proc. 3rd Int'l Conf. on Scalable Information Systems*, 2008.

[10] G. Das, D. Gunopulos, N. Koudas, and D. Tsirogiannis. Answering top-k queries using views. In *Proc. 32nd Int'l Conf. on Very Large Data Bases*, pages 451–462, 2006.

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

[12] B. Huffaker, M. Fomenkov, D. J. Plummer, D. Moore, and K. Claffy. Distance metrics in the internet. In *Proc. Int'l Telecommunications Symposium*, 2002.

[13] R. Kumar, K. Punera, T. Suel, and S. Vassilvitskii. Top-k aggregation using intersections of ranked inputs. In *Proc. 2nd ACM Int'l Conf. on Web Search and Data Mining*, pages 222–231, 2009.

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

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

[16] E. Schurman and J. Brutlag. Performance related changes and their user impact. In *Velocity: Web Performance and Operations Conf.*, 2009.

[17] C. Tang, Z. Xu, and M. Mahalingam. Peersearch: Efficient information retrieval in peer-to-peer networks. In *Proc. of HotNets-I, ACM SIGCOMM*, 2002.